Oracle® Communications Network Integrity

RESTCONF-Driven Network Discovery and UIM Integration Cartridge Guide

Release 8.0
G34940-01
October 2025

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

G34940-01

Contents

7 About Design Studio Extension

About This Content

This guide describes the functionality and design of the Oracle Communications Network Integrity RESTCONF 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 Representational State Transfer Configuration Protocol (RESTCONF) Network Configuration Protocol 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 RESTCONF Network Discovery and UIM Integration cartridge.

About the RESTCONF Network Discovery Cartridge

The Representational State Transfer Configuration (RESTCONF) Network Discovery cartridge supports network discovery of devices using the RESTCONF protocol. It generates a representation of the network infrastructure that comprises both logical and physical device hierarchies. The logical hierarchy includes logical devices, child interfaces, sub-interfaces, and device interface configurations. The physical hierarchy includes physical devices, 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 RESTCONF Network Discovery cartridge, make sure your network devices support the RESTCONF protocol and Yet Another Next Generation (YANG) models. Network Integrity relies on standard RESTCONF/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 RESTCONF Protocol

The RESTCONF protocol, standardized by the Internet Engineering Task Force (IETF), allows users to install, modify, and delete the configuration of network devices. RESTCONF uses JSON-based encoding for both configuration data and protocol messages. It performs its operations through HTTP or HTTPS Representational State Transfer Configuration (REST) calls. The table below describes the main HTTP methods RESTCONF uses and their purposes.

Table 1-1 Main HTTP Methods of RESTCONF Protocol

HTTP Method	RESTCONF Purpose	Description
OPTIONS	Discover supported methods	Returns allowed HTTP methods for a resource
GET	Retrieve data or metadata	Fetches resource state/config; supports query parameters like **content**, **depth**, **fields**, **filter**, etc.
HEAD	Retrieve header-only metadata	Same as GET but without message body
POST	Create resources or invoke a Remote Procedure Call (RPC) or action	Creates list entries or triggers actions and returns `201 Created`; and returns conflicts if the object already exists.

Table 1-1 (Cont.) Main HTTP Methods of RESTCONF Protocol

HTTP Method	RESTCONF Purpose	Description
PUT	Create or replace a resource	Replaces the entire target with the configuration in the message.
PATCH	Merge updates into existing resource	Partial update (merge); supports both plain and YANG patches; returns `200 OK` (with body) or `204 No Content`.
DELETE	Remove a resource	Deletes target, returns a `204 No Content` on success, or fails if the resource doesn't exist.

> **Note**
>
> NI supports **GET** operation only with the query parameter `fields`.

The following examples show sample RESTCONF Request and responses.

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

```
http://100.76.169.40:1443/restconf/data/ietf-interfaces:interfaces
```

Example 2: Sample RESTCONF Response.

```
{
    "ietf-interfaces:interfaces": {
        "interface": [
            {
                "name": "Ethernet0/0",
                "type": "iana-if-type:ethernetCsmacd",
                "enabled": true,
                "ietf-ip:ipv4": {
                    "address": [
                        {
                            "ip": "192.0.2.105",
                            "netmask": "255.255.255.0"
                        }
                    ]
                },
                "ietf-ip:ipv6": {}
            },
            {
                "name": "Ethernet0/1",
                "type": "iana-if-type:ethernetCsmacd",
                "enabled": true,
                "ietf-ip:ipv4": {
                    "address": [
                        {
                            "ip": "192.0.2.102",
                            "netmask": "255.255.255.0"
                        },
                        {
```

```
                            "ip": "192.0.2.100",
                            "netmask": "255.255.255.0"
                        }
                    ]
                },
                "ietf-ip:ipv6": {}
            },
            {
                "name": "Ethernet0/2",
                "type": "iana-if-type:ethernetCsmacd",
                "enabled": true,
                "ietf-ip:ipv4": {
                    "address": [
                        {
                            "ip": "192.0.2.100",
                            "netmask": "255.255.255.0"
                        }
                    ]
                },
                "ietf-ip:ipv6": {}
            },
            {
                "name": "Ethernet0/3",
                "type": "iana-if-type:ethernetCsmacd",
                "enabled": true,
                "ietf-ip:ipv4": {
                    "address": [
                        {
                            "ip": "192.0.2.100",
                            "netmask": "255.255.255.0"
                        }
                    ]
                },
                "ietf-ip:ipv6": {
                    "address": [
                        {
                            "ip": "2001:db8:0:1::1",
                            "prefix-length": 64
                        },
                        {
                            "ip": "2001:db8:5eff:fe00:5301",
                            "prefix-length": 64
                        }
                    ]
                }
            },
            {
                "name": "Null0",
                "type": "iana-if-type:other",
                "enabled": true,
                "ietf-ip:ipv4": {},
                "ietf-ip:ipv6": {}
            }
        ]
    }
}
```

About YANG Models

YANG is a data modeling language standardized by the IETF. 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.

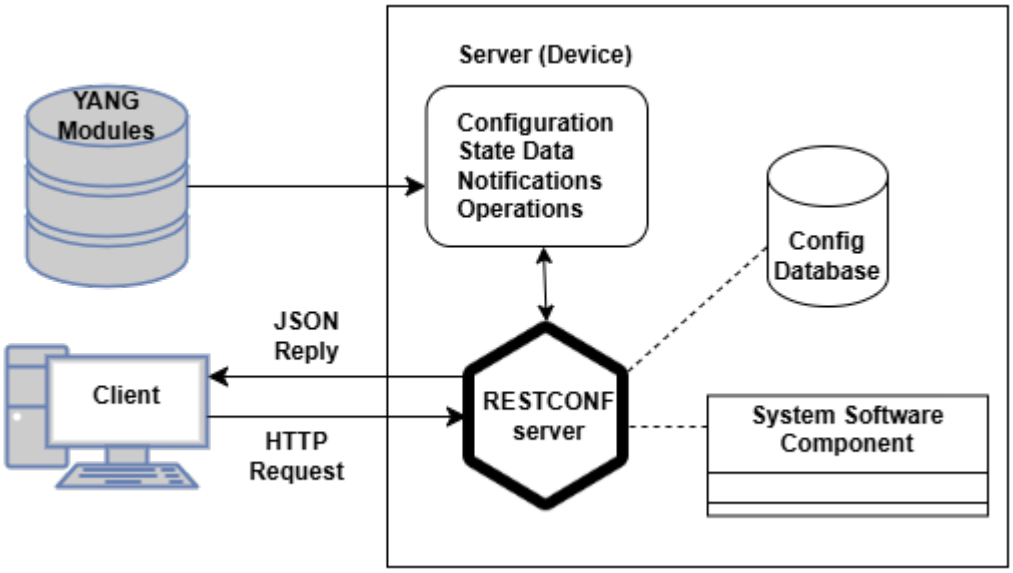
A YANG specification is called a YANG module, and a group of modules is often referred to as a YANG model. A YANG model serves as the schema of a network device, similar to a database schema or blueprint. This schema outlines the structure and types of data the system can exchange—such as possible configuration 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 RESTCONF and YANG

Figure 1-1 shows the architecture for network management made using RESTCONF 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 RESTCONF and YANG Models



The sequence of events for the typical client server interaction is as follows:

- The client builds and sends an operation defined in the YANG module, as HTTP call with appropriate method and payload.

- The server receives and parses the payload and verifies the contents of the request against the data model defined in the YANG module.
- The server performs the requested operation, and builds the response, containing the response, any requested data, and any errors.
- The server sends the response, encoded in XML/JSON.
- The client receives and parses the JSON element.
- The client inspects the response and processes it as needed.

RESTCONF Communication Between Network Integrity and Devices

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

When Network Integrity interacts with the devices using RESTCONF protocol, it does the following:

1. The client builds and sends a request using the YANG container selected in the YANG processor as an HTTP REST call.
2. The client receives the response from the device and decodes it.

About Supported Routing Protocols

This section provides information about the different routing protocol details that the RESTCONF 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 the 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 the two 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. This 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 RESTCONF 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 RESTCONF Network Discovery cartridge has the following dependencies:

- Address_Handlers
- NetworkIntegritySDK
- ora_ni_uim_yang_model
- ora_uim_model
- UIM_Integration_Cartridge
- yang-generic-model-generator-8.0.0.jar
- yang-vendor-model-generator-8.0.0.jar

About the RESTCONF YANG Model

The RESTCONF YANG model used in the RESTCONF 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 RESTCONF 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 RESTCONF YANG model in the cartridge.

The project hierarchy is as shown below:

```
yang_sources_generator
    yangs-restconf
      src/main/Restconf-yang
      pom.xml
   yangs-restconf
        generic
            pom.xml
        vendor
            pom.xml
     pom.xml
pom.xml
```

To generate Java classes for other YANG files:

1. Provide the required YANG files from the discovered devices along with all the dependent YANG files in **yang_sources_generator/yangs-restconf/generic/genric_yangs** folder (for vendor, use the appropriate vendor folder).
2. Run the command `mvn clean install` in the **yang_sources_generator/yangs-restconf/generic** directory.
   Running this command generates the updated yang-generic-model-generator-8.0.0.jar file in the **yang_sources_generator/yangs-restconf/generic/target** directory. In the case of other vendor devices, an updated vendor JAR file is generated in the **yang_sources_generator/yangs-restconf/vendor/target** directory.

Opening Cartridge Files in Design Studio

To use the RESTCONF Network Discovery and UIM Integration cartridge, you must first download the Oracle Communications RESTCONF Network Discovery and UIM Integration cartridge software from the Oracle software delivery web site:

https://edelivery.oracle.com

The software contains the cartridge ZIP file, which contains the following:

- UIM_Cartridge_Projects
- Network_Integrity_Cartridge_Projects
- **Address_Handlers.iar**
- **Restconf_Network_Discovery_Cartridge.iar**

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

Building and Deploying the Cartridge

See *Design Studio 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 RESTCONF Network Discovery and UIM Integration cartridge.

This cartridge contains the following actions:

- Abstract Discover Restconf Devices
- Discover Restconf Devices
- Import Restconf Device from UIM
- Detect Restconf Device Discrepancies
- Reconcile Restconf Device Discrepancies

Abstract Discover Restconf Devices

This is an abstract action that can be extended in Oracle Communications Service Catalog and Design - Design Studio to discover specified network objects.

The Abstract Discover Restconf Devices action contains the following processors run in the following order:

- Restconf Property Initializer
- Generic Restconf Yang Collector Processor
- Generic Restconf Data Discovery
- Vendor-Based Restconf Yang Collector Processor
- Vendor Restconf Data Discovery
- Restconf Logical Data Modeler
- Restconf Physical Data Modeler

Restconf Property Initializer

The Restconf Property Initializer processor sets the properties needed to select and run the appropriate processors, using the parameters from the **RestconfParameters** scan parameter group.

Generic Restconf Yang Collector Processor

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

Generic Restconf Data Discovery

The Generic Restconf Data Discovery Processor sends RESTCONF requests to retrieve attribute values from a device, collects the response, and generates Java binding objects. It runs when the **deviceType** parameter is set to **Generic**.

Vendor-Based Restconf Yang Collector Processor

The Vendor-based Restconf Yang Collector Processor collects variables from a device for discovery. It uses attributes from vendor-specific YANG models required to retrieve device details. This processor runs when the **deviceType** parameter is set to **Cisco IOS XE 1691**. For more information, see About Poll Lists.

Vendor Restconf Data Discovery

The Vendor RESTCONF Data Discovery Processor sends RESTCONF requests to retrieve attribute values from a device, collects the response, and generates Java binding objects. It runs when the **deviceType** parameter is set to **Cisco IOS XE 1691**.

Restconf Logical Data Modeler

The Restconf Logical Data Modeler Processor builds the logical data model from data collected by the YANG collector processor. It creates a hierarchical structure of the logical devices and its child interfaces.

Restconf Physical Data Modeler

The Restconf Physical Data Modeler Processor builds the physical data model from data collected by the Yang collector processor. It creates a hierarchy of the physical device, equipment, equipment holders, and physical ports.

Discover Restconf Devices

The Discover Restconf Devices action scans devices and builds physical and logical hierarchies based on the discovery. It also models associations between the physical and logical layers.

This action extends the Abstract Restconf Discovery action. It contains the following processors run in the following order:

- Restconf Property Initializer (inherited)
- Generic Rest Yang Collector Processor (inherited)
- Generic Restconf Data Discovery (inherited)
- Vendor based Restconf Yang Collector Processor (inherited)
- Vendor Restconf Data Discovery (inherited)
- Restconf Logical Data Modeler (inherited)
- Restconf Physical Data Modeler (inherited)

Import Restconf Device from UIM

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

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

1. Import UIM Initializer (inherited)
2. ProcessRestconfImportScanInput
3. Logical Device UIM Finder (inherited)
4. Physical Device UIM Finder (inherited)
5. Logical Device UIM MultiTheard Importer (inherited)
6. Physical Device UIM MultiTheard 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)

ProcessRestconfImportScanInput

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 Scan Parameter Group.

Detect Restconf Device Discrepancies

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

This discrepancy detection 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. Restconf Discrepancy Filters Initializer
3. Discrepancy Detector (inherited)

Restconf 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.

Reconcile Restconf Device Discrepancies

The Resolve Restconf 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 Restconf 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 YANG processor that collects details of variables to be discovered from device.

About the Generic REST YANG Collector Processor Poll List

The Generic REST YANG Collector Processor poll list:

- **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 Restconf Yang Collector Processor Poll List

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

- **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
```

- **openconfig-bgp.yang**

```
/bgp:bgp/neighbors/neighbor/neighbor-address
/bgp:bgp/neighbors/neighbor/state/session-state
/bgp:bgp/neighbors/neighbor/transport/config/local-address
```

- **openconfig-lldp.yang**

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

- **openconfig-platform.yang**

```
/platform:components/component/name
/platform:components/component/state/description
/platform:components/component/state/id
/platform:components/component/state/mfg-name
/platform:components/component/state/part-no
/platform:components/component/state/serial-no
/platform:components/component/state/type
/platform:components/component/state/version
/platform:components/component/subcomponents/subcomponent/name
```

- **Cisco-IOS-XE-ospf-oper.yang**

```
/Cisco-IOS-XE-ospf-oper:ospf-oper-data/ospf-state/ospf-instance/af
/Cisco-IOS-XE-ospf-oper:ospf-oper-data/ospf-state/ospf-instance/ospf-area/
area-id
/Cisco-IOS-XE-ospf-oper:ospf-oper-data/ospf-state/ospf-instance/ospf-area/
ospf-interface/dr
/Cisco-IOS-XE-ospf-oper:ospf-oper-data/ospf-state/ospf-instance/ospf-area/
ospf-interface/name
/Cisco-IOS-XE-ospf-oper:ospf-oper-data/ospf-state/ospf-instance/ospf-area/
ospf-interface/ospf-neighbor/neighbor-id
/Cisco-IOS-XE-ospf-oper:ospf-oper-data/ospf-state/ospf-instance/ospf-area/
ospf-interface/state
/Cisco-IOS-XE-ospf-oper:ospf-oper-data/ospf-state/ospf-instance/router-id
```

- **Cisco-IOS-XE-native.yang**

```
/Cisco-IOS-XE-native:native/hostname
/Cisco-IOS-XE-native:native/location
```

4 Using the Cartridge

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

Creating a Discover RESTCONF Devices scan

The RESTCONF Network Discovery and UIM Integration cartridge allows you to create a Discover Restconf Devices scan.

To create a Discover Restconf 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, do the following, select **Discover Restconf 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:
   - In the **Device Type** field, select type of device to discover.
   - In the **Authentication** field, select type of HTTP Authentication protocol.
   - In the **SSL field**, **enable** or **disable** SSL. If you enable SSL, then import the certificate of the RESTCONF server (device) into either Java's trust store or the server's trust store.
   - In the **Username** field, enter username for the authentication protocol.
   - In the **Password** field, enter password for the authentication protocol.
   - In the **Restconf Port** field, provide the port for connection.
   - In the **Context root** field, provide the leading url for Restconf call.
   - In the **Connect Timeout** field, provide the connection timeout value in milliseconds.
   - In the **Read Timeout** field, provide the read timeout value in milliseconds.
   - In the **Field Filtering** field, **enable** or **disable** field filtering for REST response.
4. Provide device IP Address in the **Scope** field.
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 RESTCONF Network Discovery and UIM Integration cartridge allows you to populate UIM with network data discovered by the Discover Restconf 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 Restconf Devices**.
      The **Scan Type** field displays **Discovery**.
   b. Select **Detect Discrepancies**.
   c. In the **Scan Action Parameters** area, make any necessary configurations.
3. Provide device IP Address in the **Scope** field.
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 Restconf Device Discrepancies in UIM**.
6. Click **Submit**.

Import Reconciled Data from UIM

To create an Import Restconf 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 Restconf Device from UIM**.
      The **Scan Type** field displays *Import*.
   b. In the **Scan Action Parameters**, add any filters if required.
3. Make any other required configurations.
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 RESTCONF Network Discovery 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 RESTCONF Network Discovery and UIM Integration 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**

Equipment, Equipment Holders, and Physical Ports are established and seeded with data sourced from the following YANG file:

- **ietf-hardware.yang**

Vendor-Based Device

The data sourced from the **Cisco-IOS-XE-native.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 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**
- **openconfig-bgp.yang**
- **openconfig-lldp.yang**
- **Cisco-IOS-XE-ospf-oper.yang**

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

- **openconfig-platform.yang**

Oracle Communications Information Model Information

All the entities used in RESTCONF 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.

This cartridge supports the **Generic Media** configuration.

Specifications

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

You must first model inventory (UIM) 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_device cartridge. These cartridges 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

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 rate code.
400GigE_interface	ora_ni_uim_yang_model	Used to model device interfaces with 400GigE rate code.
200GigE_interface	ora_ni_uim_yang_model	Used to model device interfaces with 200GigE rate code.
100GigE_interface	ora_ni_uim_yang_model	Used to model device interfaces with 100GigE rate code.
50GigE_interface	ora_ni_uim_yang_model	Used to model device interfaces with 50GigE rate code.
25GigE_interface	ora_ni_uim_yang_model	Used to model device interfaces with 25GigE rate code.
15GigE_interface	ora_ni_uim_yang_model	Used to model device interfaces with 15GigE rate code.
10GigE_interface	ora_ni_uim_yang_model	Used to model device interfaces with 10GigE rate code.
5GigE_interface	ora_ni_uim_yang_model	Used to model device interfaces with 5GigE rate code.

Table 5-3 (Cont.) Device Interface Specifications

Specification	Cartridge	Intended Usage
1GigE_interface	ora_ni_uim_yang_model	Used to model device interfaces with 1GigE rate code.
100M_interface	ora_ni_uim_yang_model	Used to model device interfaces with 100M rate code.
10M_interface	ora_ni_uim_yang_model	Used to model device interfaces with 10M rate code.
subInterfaceSpecification	ora_ni_uim_yang_model	Used to model all sub interfaces.

Table 5-3 shows the characteristics applied to deviceInterfaceSpecification specification and all device interface specifications that have rate code.

Table 5-4 Device Interface Item Characteristics fordeviceInterfaceSpecification 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	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 the A end
AEndInterfaceIPAddress	String	Text	A-end interface IP Address
AEndInterfaceName	String	Text	Name of interface on the A end
ZEndDeviceName	String	Text	Name of device on the Z end
ZEndInterfaceIPAddress	String	Text	Z-end interface IP Address
ZEndInterfaceName	String	Text	Name of interface on the Z end

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

Characteristics	Field Type	Field Content	Description
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 (for example, Down, DR, BDR)
ospfIfType	String	Text	Interface type (for example, broadcast, point-to-point)
ospfNbrIpAddr	String	Text	IP address of the adjacent OSPF neighbor
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 (for example, Down, DR, BDR)
ospfIfType	String	Text	Interface type (for example, broadcast, point-to-point)
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.

Table 5-6 shows the characteristics applied to Generic_IPAddress Device Interface Configuration Item

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" and "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.

The physicalDeviceSpecification is available in ora_ni_uim_yang_model.

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
nativeEmsName	String	Text
vendorName	String	Text

Equipment Holder

Table 5-12 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.

Table 5-13 (Cont.) Physical Port Specifications

Specification	Cartridge	Intended Usage
400GigE_Port	ora_ni_uim_yang_model	Used to model physical ports mapped with 400GigE device interfaces.
200GigE_Port	ora_ni_uim_yang_model	Used to model physical ports mapped with 200GigE device interfaces.
100GigE_Port	ora_ni_uim_yang_model	Used to model physical ports mapped with 100GigE device interfaces.
50GigE_Port	ora_ni_uim_yang_model	Used to model physical ports mapped with 50GigE device interfaces.
25GigE_Port	ora_ni_uim_yang_model	Used to model physical ports mapped with 25GigE device interfaces.
15GigE_Port	ora_ni_uim_yang_model	Used to model physical ports mapped with 15GigE device interfaces.
10GigE_Port	ora_ni_uim_yang_model	Used to model physical ports mapped with 10GigE device interfaces.
5GigE_Port	ora_ni_uim_yang_model	Used to model physical ports mapped with 5GigE device interfaces.
1GigE_Port	ora_ni_uim_yang_model	Used to model physical ports mapped with 1GigE device interfaces.
100M_Port	ora_ni_uim_yang_model	Used to model physical ports mapped with 100M device interfaces.
10M_Port	ora_ni_uim_yang_model	Used to model physical ports mapped with 10M device interfaces.

Table 5-14 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 customer to decide if they want to enrich the technology pack to provide visual effects to the UIM GUI for a given Equipment entity.

Field Mapping

The RESTCONF 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 RESTCONF Network Discovery and UIM Integration cartridge from the Oracle Communications Service Catalog and Design - Design Studio perspective.

Model Collections

Table 6-1 lists the model collection used in the RESTCONF Network Discovery and UIM Integration cartridge.

Table 6-1 RESTCONF Network Discovery Cartridge Model Collection

Specification	Information Model Entity Type	Intended Usage/Notes
logicalDeviceSpecification	LogicalDevice	Represents the 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.) RESTCONF 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 RESTCONF 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 RESTCONF Network Discovery Cartridge supports the discovery action Discover Restconf Devices which is used to discover devices in the network.

Table 6-2 Discover Restconf Devices

Result Category	Address Handler	Scan Parameters	Model	Processors
Device	IPAddressHandler	RestconfParameters • username • password • port • deviceType • authentication • connectTimeout • ssl • readTimeout • fieldFiltering • contextRoot	Restconf_Discovery	• Restconf Property Initializer • Generic Rest Yang Collector Processor • Generic Restconf Data Discovery • Vendor Based Rest Yang Collector Processor • Vendor Restconf Data Discovery • Restconf Logical Data Modeler • Restconf Physical Data Modeler

Scan Parameter Group

The scan parameter groups used in this cartridge are listed in Scan Parameter Group.

Table 6-3 Scan Parameter Group

Scan Parameter Group Name	Parameters
RestconfParameters	• username • password • port • deviceType • authentication • connectTimeout • ssl • readTimeout • fieldFiltering • contextRoot

Discovery Processors

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

Table 6-4 List of Discovery Processors

Processor Name	Variable
Restconf Property Intializer	Input: N/A Output: deviceType
Generic Rest Yang Collector Processor	Input: The directory path to YANG files Output: genericRestYangCollectorProcessorProcessorResponseType The collection of attributes to be discovered from the device
Generic Restconf Data Discovery	Input: genericRestYangCollectorProcessorProcessorResponseType Output: genericbindings Map of response Java objects
Vendor Based Rest Yang Collector Processor	Input: The directory path to YANG files Output: vendorbasedRestYangcollectorProcessorProcessorResponseType The collection of attributes to be discovered from the device
Vendor Restconf Data Discovery	Input: vendorbasedRestYangcollectorProcessorProcessorResponseType Output: vendorBindings Map of response Java objects
Restconf Logical Data Modeler	Input: • genericbindings • vendorBindings Output: • logicalDevice

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

Processor Name	Variable
Restconf Physical Data Modeler	Input: • genericbindings • vendorBindings Output: physicalDevice

Import Action

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

Table 6-5 Import Restconf Device from UIM

Result Category	Scan Parameters	Model	Processors
Device	UIMImportParameters	Restconf_Network_Discovery_Cartridge	ProcessRestconfImportScanInput This action extends the Abstract Import from UIM action. For more information about the processors in this action, see *Network Integrity UIM Integration Cartridge Guide*.

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	Checkbox	Use this box to indicate whether to import logical devices. By default, this box is checked in the UI.	Import Logical Devices
importPhysicalDevices	Checkbox	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 in case multiple specifications 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

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

Characteristic Name	Parameter Type	Description	UI Label
physicalDeviceSpecification	String	The specification name(s) for physical devices. This field supports wildcard characters. Values are comma separated in case multiple specifications given.	Physical Device Specification

Discrepancy Detection Action

The Detect Restconf Device Discrepancies action is used to perform Discrepancy detection.

Table 6-7 Detect Restconf Device Discrepancies

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

Discrepancy Resolution Action

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

Table 6-8 Reconcile Restconf Device Discrepancies

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

7

About Design Studio Extension

This chapter provides scenarios for the extensibility of Network Integrity using Oracle Communications Service Catalog and Design - Design Studio.

Adding a New Device Type under Generic

To discover and model a new device type:

1. Collect all the YANG files supported by the device and place them in the YANG directory path set in Design Studio preferences.
2. Generate the Restconf-yang-model-8.0.0.jar with new YANG files using the yang_sources_generator project and copy the JAR file to the **Restconf_Network_Discovery_Cartridge/lib** folder.

   For more information about generating the Restconf-yang-model, see About the RESTCONF YANG Model.
3. Create a new processor of the type **yangprocessor** in the cartridge. Select all attributes that you want to be discovered from device.
4. Create a new **discoveryprocessor**, which accepts **yangprocessor** output as input. Refer to the **VendorRestconfDataDiscoveryProcessorImpl** class to collect response from the device and generate Java objects.
5. Update the Restconf Logical Device Modeler and Restconf 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 RESTCONF Network Discovery Cartridge

This section provides information about known issues in the RESTCONF 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._native.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.rev180522._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 RESTCONF 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 RESTCONF 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 RESTCONF 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 RESTCONF 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 RESTCONF Network Discovery and UIM Integration cartridge.