Oracle® Communications Network Integrity

UIM Integration Cartridge Guide

Release 7.5
G13605-01
December 2024

Oracle Communications Network Integrity UIM Integration Cartridge Guide, Release 7.5

G13605-01

Contents

4 About Cartridge Modeling

5 About Design Studio Construction

6 Working with the UIM Sample Web Service

Preface

This guide describes the functionality and design of the Oracle Communications Network Integrity 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 either to build or to extend similar cartridges.

The developers should have a good working knowledge of specifications, Network Integrity, Oracle Communications Unified Inventory Management (UIM), and Oracle Communications Design Studio for both UIM and Network Integrity.

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

- *Network Integrity Concepts*
- *Network Integrity Developer's Guide*
- *Network Integrity Installation Guide*

This guide assumes that you are familiar with the following:

- Oracle Communications Design Studio
- Oracle Communications Information Model

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.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve.

Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

1
Overview

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

About the UIM Integration Cartridge

The UIM Integration cartridge provides discrepancy detection, discrepancy resolution, and import functionality for integration with Oracle Communications Unified Inventory Management (UIM). The UIM Integration cartridge supports logical device and physical device hierarchies.

The cartridge contains an abstract import action that is extended by the deployable Import action. The abstract import action supports extensibility where custom scan parameters are required for selective import of data. The deployable import action has scan parameters to control import of logical device and physical device trees, and to specify matching criteria for devices.

The import action allows logical device and physical device trees in UIM to be imported to Network Integrity for comparison of objects with discovered data.

The cartridge contains an abstract discrepancy detection action that supports UIM-specific behavior like matching by nativeEMSName and excluding Device Interface Configuration. The discrepancy detection action provides the mechanism to allow a filtered comparison of logical and physical trees between data that is discovered and data that is imported from UIM.

The cartridge contains an abstract discrepancy resolution action that supports resolution of both logical device and physical device trees. The discrepancy resolution action allows the discovered logical device and physical device trees to be created and updated in UIM. For more information about discrepancy detection and discrepancy resolution, see "Using Design Studio to Extend Network Integrity" in *Network Integrity Developer's Guide*.

About the UIM Sample Web Service

The UIM Sample web service enables Network Integrity to communicate with UIM and perform Create, Read, Update, and Delete operations on entity instances in UIM.

See "Working with the UIM Sample Web Service " for more information about the UIM Sample Web Service and web service operations.

About Cartridge Dependencies

This section provides information about dependencies that the UIM Integration cartridge has on other entities.

The **Network_Integrity_Cartridge_Projects\UIM_Integration_Cartridge** project is dependent on the **UIM_Cartridge_Projects\ora_ni_uim_webservice** project being available in Oracle Communications Design Studio for compilation if you intend to modify the web service.

The **UIM_Integration_Cartridge** project contains the **buildUimClient.xml** file, which is dependent on the **UIM_Cartridge_Projects\ora_ni_uim_webservice\wsdl\NI_Uim.wsdl** file.

The **buildUimClient.xml** file must have target **ALL** run if you are modifying the web service and must be successful before you compile the UIM Integration cartridge.

Run-Time Dependencies

For the UIM Integration cartridge to work at run time:

- You must deploy the Address_Handlers and ora_ni_uim_ocim cartridges to Network Integrity.
- UIM must be installed and be accessible to Network Integrity.

The following component must be deployed to UIM:

- UIM Integration web service
- ora_ni_uim_ocim

Design-Time Dependencies

The UIM Integration cartridge has the following dependencies:

- NetworkIntegritySDK
- ora_ni_uim_ocim

Opening the Cartridge Files in Design Studio

To review and extend the UIM Integration cartridge, download the Oracle Communications Network Integrity UIM Integration Cartridge software from the Oracle software delivery website:

https://edelivery.oracle.com/

The software contains the Network Integrity UIM Integration Cartridge ZIP file, which has the following structure:

- **\UIM_Cartridge_Projects**
- **\Network_Integrity_Cartridge_Projects**

> **Note:**
>
> These are Java projects that can be loaded into Eclipse and that contain identical software. These folders are duplicated depending on which Network Integrity software cartridges you want to use.

The **Network_Integrity_Cartridge_Projects\UIM_Integration_Cartridge** project contains the extendable Design Studio files.

You can find the WSDL and schema files within the Java projects.

The WSDL and schemas are located in the following paths:

- **ora_ni_uim_webservice\wsdl\NI_Uim.wsdl**
- **ora_ni_uim_webservice\wsdl\schemas**
- **ora_ni_uim_webservice\wsdl\referenceschemas**

See "About Network Integrity" in *Network Integrity Concepts* for guidelines and best practices for extending cartridges.

See "Using Design Studio to Extend Network Integrity" in *Network Integrity Developer's Guide* for information about opening files in Design Studio.

Building and Deploying the Cartridge

See "Getting Started with Design Studio for Network Integrity (1)" in *SCD Design Studio Modeling Network Integrity* for information about building and deploying cartridges.

2

About the Cartridge Components

This chapter describes the components of Oracle Communications Network Integrity UIM Integration cartridge.

The UIM Integration cartridge is composed of the following actions, each containing processors:

- Abstract Import from UIM Action
- Import from UIM Action
- Abstract Incremental Import from UIM
- Incremental Scan Parameter UIM Initializer
- Abstract Detect UIM Discrepancies Action
- Abstract Resolve in UIM Action

Abstract Import from UIM Action

The Abstract Import from UIM action imports data from Oracle Communications Unified Inventory Management (UIM). This is an abstract action without any scan parameters. Using various filter criteria, this action interacts with the UIM Integration web service to find devices and then to import and model logical devices and physical devices.

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

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

Figure 2-1 shows the processor workflow of the Abstract Import from UIM action.

Figure 2-1 Abstract Import from UIM Action Processor Workflow

Import UIM Initializer

This processor outputs variable filters of type DeviceFilter with default values populated; uimImportContext of type UIMImportContext, which contains a context used by multiple processors; and uimLogicalDeviceIDs and uimPhysicalDeviceIDs of type Set.

Table 2-1 shows the available filter properties with default values. The Effective Type column indicates how the value is used, including whether wildcards are supported. Note that the value Text* in the Effective Type column is multi-value and a comma-separated list.

Table 2-1 Filter Properties

Property Name	Java Type	Effective Type	Default	Notes
logicalDeviceName	String	Text*	N/A	Supports comma-separated list for multiple values.
physicalDeviceName	String	Text*	N/A	Supports comma-separated list for multiple values.
logicalDeviceSpecification	String	Text	N/A	The specification name(s) for a logical device. Supports comma-separated list for multiple values.
physicalDeviceSpecification	String	Text	N/A	The specification name(s) for a physical device. Supports comma-separated list for multiple values.

Table 2-1 (Cont.) Filter Properties

Property Name	Java Type	Effective Type	Default	Notes
logicalDeviceInventoryState	String	Enum	N/A	Available values: • INSTALLED • UNAVAILABLE
physicalDeviceInventoryState	String	Enum	N/A	Available values: • INSTALLED • UNAVAILABLE
networkLocationEntityCode	String	Text*	N/A	Supports comma-separated list for multiple values. This can be used to search for devices at a particular location, a specific device at a specific location, and all devices with a particular entity code across multiple network locations.
queryLogicalDevices	boolean	Boolean	true	If false, it does not perform a logical device query.
queryPhysicalDevices	boolean	Boolean	true	If false, it does not perform a physical device query.
importRelatedPhysicalOrLogicalDevice	boolean	Boolean	true	If false, it imports devices that explicitly match filter criteria. It does not automatically import related device for a matched device. For example, if you want to import only logical devices, set Query Physical Devices to "false" and Import Related Physical or Logical Device to "false".
logicalDeviceCharacteristics	Map<String, String>	Text, Text* (name, value)	null	A map of characteristic name to characteristic value.This allows searching on multiple, arbitrary characteristic name/value pairs. An extending action typically does not allow you to enter the name. The name is set internally for a field that specifies the value (for example, Mgmt IP Address).
physicalDeviceCharacteristics	Map<String, String>	Text, Text* (name, value)	null	A map of characteristic name to characteristic value.This allows searching on multiple, arbitrary characteristic name/value pairs. An extending action typically does not allow you to enter the name. The name is set internally for a field that specifies the value (for example, Mgmt IP Address).

Logical Device UIM Finder

This processor uses the filters uimImportContext, uimLogicalDeviceIDs, and uimPhysicalDeviceIDs as input parameters; performs one or more Logical Device find operations through the UIM web service API; and updates uimLogicalDeviceIDs and uimPhysicalDeviceIDs.

This processor uses the following filter properties:

- logicalDeviceName
- logicalDeviceSpecification
- logicalDeviceInventoryState
- networkLocationEntityCode
- queryLogicalDevices
- importRelatedPhysicalOrLogicalDevice
- logicalDeviceCharacteristics

Physical Device UIM Finder

This processor uses the filters uimImportContext, uimLogicalDeviceIDs, and uimPhysicalDeviceIDs as input parameters; performs one or more physical device find operations through the UIM web service API; and updates uimLogicalDeviceIDs and uimPhysicalDeviceIDs.

This processor uses the following filter properties:

- physicalDeviceName
- physicalDeviceSpecification
- physicalDeviceInventoryState
- queryPhysicalDevices
- importRelatedPhysicalOrLogicalDevice
- physicalDeviceCharacteristics

Logical Device UIM MultiThread Importer

This processor uses the filters uimImportContext, uimLogicalDeviceIDs as input parameters, imports and models the specified device(s). It stores the results associated with logical devices in Network Integrity. Instead of using hardcoded specifications for logical device and device interface, the UIM specification name is mapped to a Network Integrity specification name. When creating a device interface, this processor instantiates either DeviceInterface or MediaInterface based on the type received from UIM.

This processor verifies whether a scan is configured with the Parallel Process option enabled. If not, it skips the process.

This processor uses WebLogic's Managed Executor Service work manager concept to process it in parallel. For more information, see **Working with application context work-managers** in "Using Design Studio to Extend Network Integrity" in *Developer's Guide*.

Physical Device UIM MultiThread Importer

This processor uses the filters, uimImportContext, uimLogicalDeviceIDs, and uimPhysicalDeviceIDs as input parameters, imports and models the specified device(s). It stores the results associated with physical devices in Network Integrity. Instead of using hardcoded specifications for physical entities, the UIM specification name is mapped to a Network Integrity specification name.

This processor verifies whether a scan is configured with the Parallel Process option enabled. If not, it skips the process.

This processor uses weblogic's Managed Executor Service work manager concept to process it in parallel. For more information, see **Working with application context work-managers** "Using Design Studio to Extend Network Integrity" in *Developer's Guide*.

Logical Device UIM Importer

This processor uses the filters uimImportContext, uimPhysicalDeviceIDs, and uimLogicalDeviceID (from uimLogicalDeviceIDs For loop) as input parameters; imports and models the specified device; and outputs ldev (the modeled logical device) and uimLDev (the UIM web service logical device).

Instead of using hardcoded specifications for logical device and device interface, the UIM specification name is mapped to a Network Integrity specification name.

When creating a device interface, this processor instantiates either DeviceInterface or MediaInterface based on the type received from UIM.

Linked Physical Device UIM Importer

This processor uses the filters uimImportContext, uimPhysicalDeviceIDs, ldev, and uimLDev as input parameters; imports and models the specified physical device; outputs pdev (the modeled physical device).

Instead of using hardcoded specifications for physical entities, the UIM specification name is mapped to a Network Integrity specification name.

This processor removes the processed physical device from uimPhysicalDeviceIDs.

Logical Device UIM Persister

This processor stores the results associated with logical devices in Network Integrity.

Physical Device UIM Importer

This processor uses the filters uimImportContext and uimPhysicalDeviceID (from uimPhysicalDeviceIDs For loop) as input parameters, imports and models the specified physical device, and outputs pdev (the modeled physical device).

Instead of using hardcoded specifications for physical entities, the UIM specification name is mapped to a Network Integrity specification name.

Linked Logical Device UIM Importer

This processor uses the filters, uimImportContext, pdev, uimLogicalDeviceIDs, and uimPDev as input parameters, imports and models the specified logical device. Instead of using hardcoded specifications for logical entities, the UIM specification name is mapped to a Network Integrity specification name.

Physical Device UIM Persister

This processor stores results associated with physical devices in Network Integrity.

Import from UIM Action

The Import from UIM action extends the Abstract Import from UIM action by adding scan parameters. This action initializes filters from the scan parameters.

The import functionality is implemented to:

- Retrieve all the logical device IDs that match the filter, and the physical device IDs of associated devices.
- Retrieve all the physical device IDs that match the filter, and the logical device IDs of associated devices.
- Iterate over each logical device ID to:
  - Retrieve and model the logical device
  - Retrieve and model the associated physical device (if any)
  - Persist the logical and physical device trees
- Iterate over each physical device ID not already processed to:
  - Retrieve and model the physical device
  - Retrieve and model the associated logical device (if any)
  - Persist the logical and physical device trees

This import action extends the Abstract Import from UIM action and inherits all its processors. See "Abstract Import from UIM Action" for more information.

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

1. Import UIM Initializer (inherited)
2. Scan Parameter UIM Initializer
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)

Figure 2-2 shows the workflow of processors of the Import from UIM action.

Figure 2-2 Import from UIM Action Processor Workflow

Scan Parameter UIM Initializer

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 Groups".

Abstract Incremental Import from UIM

The Abstract Incremental Import from UIM action extends the Abstract Import from UIM action by adding scan parameters.

This action extends the Abstract Import from UIM action and inherits all its processors. For more information, see "Abstract Import from UIM Action".

The Abstract Incremental Import from UIM action contains the following processors run in the following order:

1. Import UIM Initializer (inherited)
2. ME Names Collector
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)
13. UpdateNotificationStatus

Figure 2-3 shows the workflow of processors of the Abstract Incremental Import from UIM action.

Figure 2-3 Abstract Incremental Import from UIM action

ME Names Collector

This processor uses the filter incrementalimportScanParams as input, collects managed element names from NMS notifications and outputs meNames.

UpdateNotificationStatus

This processor uses incrementalimportScanParams and meNames as input and updates NMS notifications with appropriate status.

Incremental Import from UIM

The Incremental Import from UIM action extends the Abstract Incremental Import from UIM action by adding scan parameters.

The incremental import functionality is implemented to:

- Retrieve all the logical device IDs from the NMS notifications with status ‘INITIAL’ and the physical device IDs of associated devices.
- Retrieve all the physical device IDs from the NMS notifications with status ‘INITIAL’ and the logical device IDs of associated devices.
- Iterate over each logical device ID to:
  - Retrieve and model the logical device.
  - Retrieve and model the associated physical device (if any).
  - Persist the logical and physical device trees.
- Iterate over each physical device ID not already processed to:
  - Retrieve and model the physical device.
  - Retrieve and model the associated logical device (if any).
  - Persist the logical and physical device.
- Update the NMS notifications with status ‘IMPORTED’ for imported devices.

The Incremental Import from UIM action extends the Abstract Import from UIM action and inherits all its processors. See "Abstract Import from UIM Action" for more information. The Abstract Incremental Import from UIM action contains the following processors run in the following order.

1. Import UIM Initializer (inherited)
2. Incremental Scan Parameter UIM Initializer
3. ME Names Collector (inherited)
4. Logical Device UIM Finder (inherited)
5. Physical Device UIM Finder (inherited)
6. Logical Device UIM MultiThread Importer (inherited)
7. Physical Device UIM MultiThread Importer (inherited)
8. Logical Device UIM Importer (inherited)
9. Linked Physical Device UIM Importer (inherited)
10. Logical Device UIM Persister (inherited)
11. Physical Device UIM Importer (inherited)
12. Linked Logical Device UIM Importer (inherited)
13. Physical Device UIM Persister (inherited)
14. UpdateNotificationStatus (inherited)

Figure 2-4 shows the workflow of processors of the Incremental Import from UIM action.

Figure 2-4 Incremental Import from UIM Action

Incremental Scan Parameter UIM Initializer

This processor uses incrementalimportScanParams as input, and sets filter values based on the scan parameters.

Abstract Detect UIM Discrepancies Action

The Abstract Detect UIM Discrepancies action detects discrepancies between discovered data and the data imported from UIM.This discrepancy detection action extends the Detect Discrepancies action (from the NetworkIntegritySDK cartridge) and inherits all its processors. For information about the inherited processors in this action, see "Using Design Studio to Extend Network Integrity" in *Network Integrity Developer's Guide*.

The Abstract Detect UIM Discrepancies action contains the following processors run in the following order:

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

Figure 2-5 illustrates the processor workflow of the Abstract Detect UIM Discrepancies action.

Figure 2-5 Abstract Detect UIM Discrepancies Action Processor Workflow

UIM Discrepancies Filter Initializer

This processor implements the following filters:

- Ignore the ID field on all entities.
- Treat Media Interface and Device Interface as the same objects.
- Ignore DeviceInterfaceConfigurationItem.
- Instead of the name field, use the nativeEMSName field as the comparator across all entities to determine whether the objects from discovery and import are same.
- On a logical device, ignore the networkLocationEntityCode and deviceIdentifier fields.

Abstract Resolve in UIM Action

The Abstract Resolve in UIM action resolves discrepancies between Network Integrity discovery data and UIM import data by constructing and updating logical device and physical device trees in UIM. The implementation instantiates a class called BaseResolutionElement, which acts as a triage system. Handlers registered into BaseResolutionElement deal with particular entities.

When you submit one or more discrepancies to be resolved to UIM, the batch of discrepancies is sent to the BaseResolutionElement, which sets the order and priority of the discrepancies. BaseResolutionElement then calls entity handlers to dispatch the resolution to UIM. The entity handlers use ora_ni_uim_webservice to communicate with UIM.

In UIM, you can create an entity only if the specification on which the entity is based already exists. The NI discrepancy detection process uses entity specifications to discern information about entities based on them.

When modeling your inventory solution, Oracle recommends that you follow a consistent naming convention when defining entity specifications within both the Import and Discovery processors to allow the discrepancy resolution process to resolve entity discrepancies more efficiently.

The Network Integrity discrepancy resolution process creates the entities in a hierarchical fashion within UIM and each entity in the hierarchy is created using a separate transaction. If any failure occurs during the discrepancy resolution process, the process stops and displays an error message.

Table 2-2 lists the handlers called by BaseResolutionElement and their corresponding discrepancy types and definitions.

Table 2-2 Handler Discrepancy Types

Handler	Discrepancy Type
Logical Device Handler	Entity+ (Entity is missing from UIM) Attribute Value Mismatch (Entity has field value that differs) Assoc+ (Peer entities are missing association between them) Assoc- (Peer entities not discovered to be peers in discovery)
Device Interface Handler	Entity+ Entity- (Entity is in UIM but not discovered) Attribute Value Mismatch Assoc+ Assoc-
Physical Device Handler	Entity+ Attribute Value Mismatch Assoc+ Assoc-
Equipment Handler	Entity+ Entity- Attribute Value Mismatch Assoc+ Assoc-
Equipment Holder Handler	Entity+ Entity- Attribute Value Mismatch Assoc+ Assoc-
Physical Port Handler	Entity+ Entity- Attribute Value Mismatch Assoc+ Assoc-

The Abstract Detect UIM Discrepancies action contains the following processors run in the following order:

1. UIM Resolution Framework Initializer
2. UIM Resolution Initializer

3. UIM Resolution Framework Dispatcher

Figure 2-6 illustrates the processor workflow of the Abstract Resolve in UIM action.

Figure 2-6 Abstract Resolve in UIM Action Processor Workflow

UIM Resolution Framework Initializer

The UIM Resolution Framework Initializer processor instantiates BaseResolutionElement and the web service connection class.

BaseResolutionElement evaluates discrepancies, which are then processed serially to UIM. Ordering is enforced to ensure, for example you create a Logical Device before you create a DeviceInterface.

This processor produces uimResolutionContext of type UimResolutionContext as output.

This processor configures a default specification mapper that does not map and return the specification name that is passed in.

Table 2-3 shows the order of execution of discrepancies.

Table 2-3 Discrepancy Execution Order

Discrepancy Type	Entity Type
Attribute Value Mismatch	All
Assoc-	DeviceInterfacePhysicalPort
Assoc-	LogicalDevicePhysicalDevice

Table 2-3 (Cont.) Discrepancy Execution Order

Discrepancy Type	Entity Type
Entity-	DeviceInterface Equipment EquipmentHolder PhysicalPort
Entity-	LogicalDevicePhysicalDevice
Entity+	LogicalDevicePhysicalDevice
Entity+	DeviceInterface Equipment EquipmentHolder PhysicalPort
Assoc+	LogicalDevice PhysicalDevice
Assoc+	DeviceInterface PhysicalPort

UIM Resolution Initializer

This processor registers the following handlers required for physical and logical device resolution:

- LogicalDeviceHandler
- PhysicalPortHandler
- DeviceIntefaceHandler
- PhysicalDeviceHandler
- EquipmentHandler
- EquipmentHolderHandler

UIM Resolution Framework Dispatcher

This processor uses the registered entity handlers to trigger BaseResolutionElement to evaluate and treat discrepancies.

The entity handler creates a web service message and populates data into the body of the message. The entity handler connects to UIM. UIM receives the web service message. If a success message is returned, the discrepancy is marked Success. If a fail or exception message is returned, the discrepancy is marked Fail.

> **Note:**
>
> There is no transaction handling across the web service. If a discrepancy is successful, a subsequent discrepancy failure in the batch does not roll back the former. Each discrepancy is an independent event.

The position of the object being treated in the hierarchy is significant. For example, for an **Entity+** discrepancy on a logical device, you create a logical device and all child interfaces. Even if the creation of one of the child entities fails, the remaining entities continue to be created, but a Failed message is sent to the discrepancy resolution result.

For an **Entity+** discrepancy on a device interface, the handler creates the device interface and all of its child sub device interfaces.

Supported Creation Scenarios in UIM

This section describes the following creation scenarios, which are supported in UIM:

- Creation of a Logical Device and a Physical Device
- Creation of a Logical Device
- Creation of a Device Interface
- Creation of a Physical Device
- Creation of an Equipment
- Creation of an Equipment Holder
- Creation of a Physical Port
- Creation of an Association Between Logical Device and Physical Device
- Creation of an Association Between Device Interface and Physical Port

Creation of a Logical Device and a Physical Device

If UIM does not contain pre-existing logical devices and physical devices, Network Integrity displays **Entity+** for the root logical device and the root physical device.

Resolution of these two entities creates the entire logical device tree. The logical device tree attempts to create associations to the physical device tree (which does not exist), but the creation of association fails.

Resolution creates the physical device tree. The physical device tree creates associations to the logical device tree. The creation of the association is succeeds, and the tree is complete.

Creation of the logical device tree includes logical device and child device interfaces or sub interfaces.

Creation of the physical device tree includes physical device and child equipment, equipment holders, and physical ports.

Creation of any object in the tree is dependent on the ID not being occupied. If the ID is occupied, then the object is not created. If any objects fail to be created, the root object which initiated the creation is marked **Fail** or **Partial Fail** in the log file. However, it is possible that some parts of the tree are completely created and available in UIM.

Successfully created objects are not rolled back when creation of an object fails because of an occupied ID. The subsequent import, discovery, and discrepancy detection actions indicate which objects exist and which objects do not exist, allowing you to continue resolution operations.

Before objects are created in UIM, no search is carried out to see if the object with the same name or nativeEmsName already exists in UIM. UIM does not validate uniqueness of name. When you find an object with the name you need and which does not have child objects, use it. Do not create a new object with the same name.

Custom handling is required when two objects are found with the same name; when an object is found with child objects, and when an object is found in a tree.

Creation of a Logical Device

When a logical device is discovered, and when it does not exist in UIM, Network Integrity displays **Entity+** for the logical device.

Resolution creates the logical device and child objects. As each object is created, it is associated to the physical device tree.

Creation of a Device Interface

When a logical device exists in UIM without a device interface, Network Integrity displays **Entity+** for the device interface.

Resolution creates the device interface and child objects. As each object is created, it is associated to the physical port object.

Creation of a Physical Device

When a logical device tree exists in UIM, and when the physical device does not exist, Network Integrity displays **Entity+** for the physical device.

Resolution creates the physical device and child objects. As each object is created, it is associated to the logical device tree.

Creation of an Equipment

When a parent of an equipment exists in UIM, but the equipment does not exist, Network Integrity displays **Entity+** for the equipment.

Resolution creates the equipment and child objects. As each object is created, it is associated to the logical device tree.

Creation of an Equipment Holder

When a parent (an equipment) of an equipment holder exists in UIM, but the equipment holder does not exist, Network Integrity displays **Entity+** for the equipment holder.

Resolution creates the equipment holder and child objects of the equipment holder. As each object is created, it is associated to the logical device tree.

Creation of a Physical Port

When a parent (an equipment) of a physical port exists in UIM, but the physical port does not exist, Network Integrity displays **Entity+** for the physical port.

Resolution creates the physical port. The physical port is associated to the device interface.

Creation of an Association Between Logical Device and Physical Device

When a logical device and physical device exist in UIM, but the association between them does not exist, Network Integrity displays two instances of **Assoc+**: one from the physical device and the other from the logical device.

Resolution of **Assoc+** creates the logical-device-to-physical-device association. Resolution then creates the physical-device-to-logical-device association. The web service detects the association and returns a Success message.

Creation of an Association Between Device Interface and Physical Port

When a device interface and physical port exist in UIM, but the association between them does not exist, Network Integrity displays two instances of **Assoc+**: one from the device interface and the other from the physical port.

Resolution of **Assoc+** creates the device-interface-to-physical-port association. Resolution then creates the physical-port-to-device-interface association. The web service detects the association and returns a success message.

Teardown, Deletion, and Removal Scenarios in UIM

This section describes the following teardown, deletion, and removal scenarios, which are supported in UIM:

- Teardown of Association Between Device Interface and Physical Port
- Teardown of Association Between Logical Device and Physical Device
- Deletion of a Physical Port
- Deletion of an Equipment Holder
- Deletion of a Device Interface
- Removal of an Equipment from a Physical Device Tree

Teardown of Association Between Device Interface and Physical Port

When a device interface and physical port exist in UIM with incorrect associations between them, Network Integrity displays two instances of **Assoc-**: one from the device interface and the other from the physical port.

Resolution of **Assoc-** creates the device-interface-to-physical-port association. Resolution then deletes the physical-port-to-device-interface association. The web service detects that the association does not exist and returns a Success message.

Teardown of Association Between Logical Device and Physical Device

When a logical device and physical device exist in UIM with incorrect associations between them, Network Integrity displays two instances of **Assoc-**: one from the logical device and the other from the physical device.

Resolution of **Assoc-** creates the logical-device-to-physical-device association. Resolution then deletes the physical-device-to-logical-device association. The web service detects that the association does not exist and returns a Success message.

Deletion of a Physical Port

When a physical port (a child of an equipment) exists in UIM, but it is not discovered, Network Integrity displays **Entity-**.

Resolution of **Entity-** deletes the physical port.

> **Note:**
>
> A physical port does not exist in isolation. It must exist under a parent object.

Deletion of an Equipment Holder

When an equipment holder (a child of an equipment) exists in UIM, but it is not discovered, Network Integrity displays **Entity-**.

Resolution of **Entity-** deletes the equipment holder.

> **Note:**
>
> An equipment holder does not exist in isolation. It must exist under a parent object.

Deletion of a Device Interface

When a device interface (a child of a logical device or of a device interface) exists in UIM, but it is not discovered, Network Integrity displays **Entity-**.

Resolution of **Entity-** deletes the device interface.

> **Note:**
>
> A device interface does not exist in isolation. It must exist under a parent object.

Removal of an Equipment from a Physical Device Tree

When an equipment (a child of another equipment or of a physical device) exists in UIM, but is not discovered, Network Integrity displays **Entity-**.

Resolution of **Entity-** unlinks the equipment from the parent.

> **Note:**
>
> An equipment can exist in isolation. This equipment continues to exist in UIM until an administrator manually deletes it.

Mismatched Data Scenarios

The following fields are ignored for mismatched data:

- ID, which is not a discovered field
- nativeEmsName, which is used for matching objects, and not treated for mismatch
- DeviceInterfaceConfigurationItem, which exists only in Network Integrity, but not in UIM

The following sections describe mismatched data scenarios:

- Mismatch of Logical Device Data
- Mismatch of Device Interface Data
- Mismatch of Physical Device Data
- Mismatch of Equipment Data
- Mismatch of Equipment Holder Data
- Mismatch of Physical Port Data

Mismatch of Logical Device Data

When a logical device in UIM has data that does not match the discovered data, Network Integrity displays **Mismatch**.

Resolution updates the mismatched logical device attribute in UIM and sets the value in UIM to the discovered value.

Mismatch of Device Interface Data

When a device interface in UIM has data that does not match the discovered data, Network Integrity displays **Mismatch**.

Resolution updates the mismatched device interface attribute in UIM and sets the value in UIM to the discovered value.

Mismatch of Physical Device Data

When a physical device in UIM has data that does not match the discovered data, Network Integrity displays **Mismatch**.

Resolution updates the mismatched physical device attribute in UIM and sets the value in UIM to the discovered value.

Mismatch of Equipment Data

When an equipment in UIM has data that does not match the discovered data, Network Integrity displays **Mismatch**.

Resolution updates the mismatched equipment attribute in UIM and sets the value in UIM to the discovered value.

Mismatch of Equipment Holder Data

When an equipment holder in UIM has data that does not match the discovered data, Network Integrity displays **Mismatch**.

Resolution updates the mismatched equipment holder attribute in UIM and sets the value in UIM to the discovered value.

Mismatch of Physical Port Data

When a physical port in UIM has data that does not match the discovered data, Network Integrity displays **Mismatch**.

Resolution updates the mismatched physical port attribute in UIM and sets the value in UIM to the discovered value.

Working with Foreign IDs

A foreign ID is an ID that is available in UIM, but it does not originate from Network Integrity. This ID is auto-generated in UIM. The ID can also be entered manually for an entity created in UIM, rather than through resolution.

For mismatch scenarios and teardown scenarios (**Entity-** or **Assoc-**), there is no distinction between working with foreign IDs and Network Integrity-generated IDs. For creation scenarios (**Entity+** and **Assoc+**), there are precautions to creating objects if those objects must have associations to pre-existing objects in UIM with foreign IDs. The following sections describe the resolution behavior.

Scenario 1: Physical Device Tree Uses Foreign IDs

The discovery action discovers a logical device tree and physical device tree and builds the associations.

In the following example, LD indicates a logical device; PD indicates a physical device; E indicates an equipment; DI indicates a device interface; PP indicates a physical port.

- Discovery assigns IDs generated by Network Integrity to all objects:
  - LD (x) and PD (y)
  - DI (x1) E (y1)
  - DI (x2) and PP (y2)
- The import action, which uses foreign IDs, imports only the physical device tree; the logical device tree does not exist.
- Network Integrity generates: PD (z), E (z1), PP (z2), x, xN ID, y, and yN ID.
- UIM generates z and zN ID (foreign IDs).
- Discrepancy detection generates:
  - **Entity+** for missing logical devices in UIM
  - **Assoc+** for missing associations between physical devices and logical devices in UIM
  - **Assoc+** for missing associations between physical ports and device interfaces in UIM

Use Case 1: Submitting Entity+ Discrepancies Only

When you submit only **Entity+** discrepancies for treatment:

1. Network Integrity creates the logical device using the ID x it generated.
2. Network Integrity attempts to create the association between LD(x) and PD(y). Because PD(y) does not exist in UIM, the **Entity+** resolution is marked **Partial Fail**. PD(z) is not accessible in the discrepancy object at the time of resolution.
3. Network Integrity attempts to create DI(x1) and succeeds.
4. Network Integrity attempts to create DI(x2) and succeeds. The association to PP(y2) fails. A partial failure occurs.
5. The resolution on the **Entity+** is marked **Fail** (**Entity+** creates the root object and all child objects).

6. The logical device tree exists in UIM without associations to the physical device tree. The failure or partial failure is valid.
7. Run the import, discovery, and resolution actions. **Assoc+** is treated and the trees are synchronized.

Use Case 2: Submitting Entity + and Assoc+ Discrepancies Together

When you submit **Entity+** and **Assoc+** together for treatment:

1. Network Integrity creates the logical device using the ID x it generated.
2. Network Integrity attempts to create the association between LD(x) and PD(y). Because PD(y) does not exist in UIM, the **Entity+** resolution is marked **Partial Fail**. PD(z) is not accessible in the discrepancy object at the time of resolution.
3. Network Integrity attempts to create DI(x1) and succeeds.
4. Network Integrity attempts to create DI(x2) and succeeds. The association to PP(y2) fails. A partial failure occurs.
5. The resolution on **Entity+** is marked **Fail** (**Entity+** creates the root object and all child objects).
6. Network Integrity treats **Assoc+** on PD(z). Network Integrity succeeds in associating the physical device to the logical device.
7. Network Integrity treats **Assoc+** on PP(z2). Network Integrity succeeds in associating the physical port to the device interface.
8. The two **Assoc+** discrepancies are marked **Successful**.
9. The logical device tree is present in UIM with associations to the physical device tree. Treatment of **Assoc+** completes the tree lineage.
10. Run the import, discovery, and resolution actions.

    Network Integrity does not show any discrepancies.

Scenario 2: Logical Device Tree Uses Foreign IDs

This scenario is a reversal of Scenario 1. Use cases 1 and 2 described in Scenario 1 are true for this scenario if the logical device and physical device scenarios outlined in "Scenario 1: Physical Device Tree Uses Foreign IDs" are reversed.

In this scenario, the import action imports only the logical device tree; the physical device tree does not exist; the logical device tree uses foreign IDs; discrepancy detection generates **Entity+** for the missing physical devices in UIM.

Handling of foreign IDs may take up to two passes of import, discovery, and resolution actions to completely synchronize Network Integrity with UIM. If you require handling of foreign IDs in a single pass, modify the resolution to consider the auxiliary object (associated object).

Swapping Cards

This section describes swapping cards on a physical device tree.

Swapping of cards is not supported out-of-the-box, but it can be done with custom handling. This example uses manual IDs so that creation of an entity with the same ID fails.

In this example, A(1), B(2), C(3), D(4), E(5) and F(6) represent cards on devices.

1. Discover the logical and physical trees for a and create them in UIM.

2. Swap two cards on the device: C(3) and F(6). Discovery shows devices in the following order:
   - A(1)
   - B(2)
   - F(6)
   - D(4)
   - E(5)
   - C(3)
3. Import data from UIM.
4. Discover the device again.

   The following discrepancies are displayed:
   - **Entity-** on B (indicating card C is missing in discovery, but present in import)
   - **Entity+** on B (indicating card F is present in discovery, but missing in import)
   - **Entity-** on E (indicating card F is missing in discovery, but present in import)
   - **Entity+** on E (indicating card C is present in discovery, but missing in UIM)
5. Submit resolutions.
   - **Entity-** is prioritized before **Entity+**.
   - **Entity-** unlinks F(6) and C(3) from the parent in UIM and succeeds.
   - **Entity+** attempts to create C(3) and F(6).
   - **Entity+** fails because C(3) and F(6) exist. Because C(3) and F(6) exist, the required IDs are occupied. **Entity+** fails.
6. Run the import and discovery actions again.

   Discrepancy detection displays a pair of **Entity+** discrepancies.
7. Submit the resolutions.
   - **Entity+** attempts to create C(3) and F(6).
   - **Entity+** fails because C(3) and F(6) exist. Because C(3) and F(6) exist, the required IDs are occupied. **Entity+** fails.
8. Delete C(3) and F(6) manually from UIM.
9. Run the import, discovery, and resolution actions.
10. Submit the identified discrepancies for resolution.

    Resolution creates C(3) and F(6) and links them to the tree.
11. At this point, custom handling is required. Modify resolution handling to search for objects by name or by nativeEmsName before they are created. If the objects are found, use them. Do not create new objects.

    See "Creation of a Logical Device and a Physical Device" for further information.

Running Multiple Scenarios Simultaneously

When you select all discrepancies for resolution, Network Integrity and UIM incorporate prioritization to ensure correct ordering, and create all selected events in the following order:

- **Mismatch**
- **Assoc-**
- **Entity-**
- **Entity+**
- **Assoc+**

When you select a batch of discrepancies, ensure that dependencies between the discrepancies in the selected batch meet. If a dependency is not met, then resolution fails. For example, when an equipment exists in an equipment holder, but it does not belong there, unlink the equipment (**Entity-**) before you create, and link the correct one (**Entity+**).

3
Using the Cartridge

This chapter provides instructions for using the Oracle Communications Network Integrity UIM Integration cartridge in Network Integrity and Oracle Communications Design Studio.

About Wild Card Searching

Network Integrity UI search and Oracle Communications Unified Inventory Management (UIM) UI search support wildcards and all searches are case-insensitive.

Table 3-1 shows the special search characters supported in the Network Integrity search panels.

Table 3-1 Supported Special Search Characters

Special Character	Meaning	Example
%	Match 0 or more characters	abc% is equivalent to "Starts With abc" %xyz is equivalent to "Ends With xyz" %lmn% is equivalent to "Contains lmn"
*	same as %	abc* is equivalent to "Starts With abc" *xyz is equivalent to "Ends With xyz" *lmn* is equivalent to "Contains lmn"
_	Match a single character	Among other things, a_c will match a2c and abc
\	Used to match special characters	\& matches & * matches * _ matches _ \\ matches \

A % or * wildcard at the start and/or end of a search string is always used to map the remaining string with the appropriate qualifier (Begins With, Ends With, and Contains).

The * wildcard is always mapped to the % wild card for use by UIM. Similarly * is mapped to * for use by UIM.

Embedded wildcards (for example, A%Z) are passed directly to the UIM search operations. The result is the same as when performing the equivalent search from the UIM UI. In particular, wildcard matches are not supported for equals (A%Z will not match A2Z) but are supported for other qualifiers (A%Z% will match A2Z, AtoZ!, etc.).

Wildcards at start and/or end of value are mapped to one of the existing search qualifiers as described in the following table. In all cases, any leading or trailing wildcard is removed leaving "Something" as the value for search.

Table 3-2 shows the qualifiers for the wildcard pattern.

Table 3-2 Wildcard Pattern Qualifiers

Wildcard Pattern	Qualifier
Something%	BEGINS_WITH_IGNORE_CASE
%Something	ENDS_WITH_IGNORE_CASE
%Something%	CONTAINS_IGNORE_CASE
Something	EQUALS_IGNORE_CASE

Multi-valued fields result in multiple searches. For example, if logicalDeviceName is Device1, Device2 and networkLocationEntityCode is Place1*, Place2* then 4 searches are performed (Device1/Place1*, Device1/Place2*, Device2/Place1*, and Device2/Place2*).

The comma can be escaped. A value 'a,b' searches for a, then b while a value 'a\,b' searches once for 'a,b'.

Creating an Import from UIM Scan

The Import from UIM scan imports devices from UIM and models them into physical device and logical device models.

To create an Import from UIM import scan:

1. Create a new scan.

   See "Using Network Integrity" in *Network Integrity Online Help* for more information.
2. On the **General** tab, do the following:
   - From the **Scan Action** list, select **Import From UIM**.

     The **Scan Type** field displays **Import**.
   - In the Scan Action Parameters section, enter filter values.
3. Make any other required configurations.
4. Save and run the scan.

Creating a Reconciliation Solution

This section describes how to build a deployable cartridge that includes discovery, discrepancy detection, and reconciliation actions for integration with UIM.

Before you perform this procedure, ensure that the Network Integrity UIM Integration cartridge and its dependencies are imported into Design Studio.

To create a reconciliation solution:

1. In Design Studio, create a new UIM Model cartridge project and add Logical Device and Physical Device specifications.
2. Create a new Network Integrity cartridge project.
3. In the Network Integrity cartridge project, create a new discovery action.
4. Make the Network Integrity cartridge project dependent on the UIM cartridge project.
5. Add the logical and physical devices specifications from the UIM Model cartridge project to the model collection in the Network Integrity cartridge project.

6. Add the model collection to the discovery action.
7. Create a discovery processor to model the new specifications.
8. Create a new discrepancy detection action.
9. Extend the Abstract Detect UIM Discrepancies action.
10. Set the result source to the new discovery action.
11. Create a new discrepancy resolution action.
12. Extend the Abstract Resolve in UIM action.
13. Set the result source to the new discovery action.
14. Set the Resolution Action label to Correct in UIM.

Populating UIM with Discovered Data

To populate UIM with discovered network data:

1. In Network Integrity, configure a discovery scan with the **Discrepancy Detection** option enabled.
2. Run the discovery scan.

   The scan generates discrepancies for each missing logical and physical entity.
3. Submit the discrepancy resolutions for the entities you want to import to UIM.
4. Verify that UIM is populated with the submitted data.
5. In Network Integrity, configure and run an import scan.

   The import scan imports data from UIM.
6. Rerun the discovery scan with the **Discrepancy Detection** option enabled.

   The scan does not detect any discrepancies as UIM contains the discovered data.

Performing Ongoing Reconciliation with UIM

Before you perform this procedure, ensure that:

- UIM is running
- UIM Integration and Network Integrity discovery cartridges are deployed
- The Import Systems tab is configured for UIM
- Import scans are configured
- Discovery scans are configured with Discrepancy Detection enabled

To perform ongoing reconciliation with UIM:

1. In Network Integrity, run the import scan.
2. Run the discovery scan.
3. Select the discrepancies that are found during the discovery scan and correct them in UIM.
4. Submit the discrepancies for resolution.
5. Run the import scan.

   The import scan imports data from UIM.

6. Run the discovery scan.

   The scan does not detect any discrepancies.

About UIM Auto-Termination

UIM supports auto-termination of connectivity to device interfaces. For this feature, UIM expects a complete subinterface hierarchy under a STM interface.

To create STM interfaces in UIM for the purpose of auto-termination, you must ensure that the discovered interface hierarchy is complete before performing the Correct in UIM action for entity- discrepancies of logical devices or STM device interfaces.

To discover the complete interface hierarchy, configure the discovery scan with the **Collect CTP** scan parameter set to **Potential**.

> **Note:**
>
> If you run a discovery scan with the **Collect CTP** scan parameter value set to **Potential** and upload discrepancies to UIM, all future discovery scans with a different **Collect CTP** parameter value generate entity- discrepancies. You can ignore these discrepancies.

4

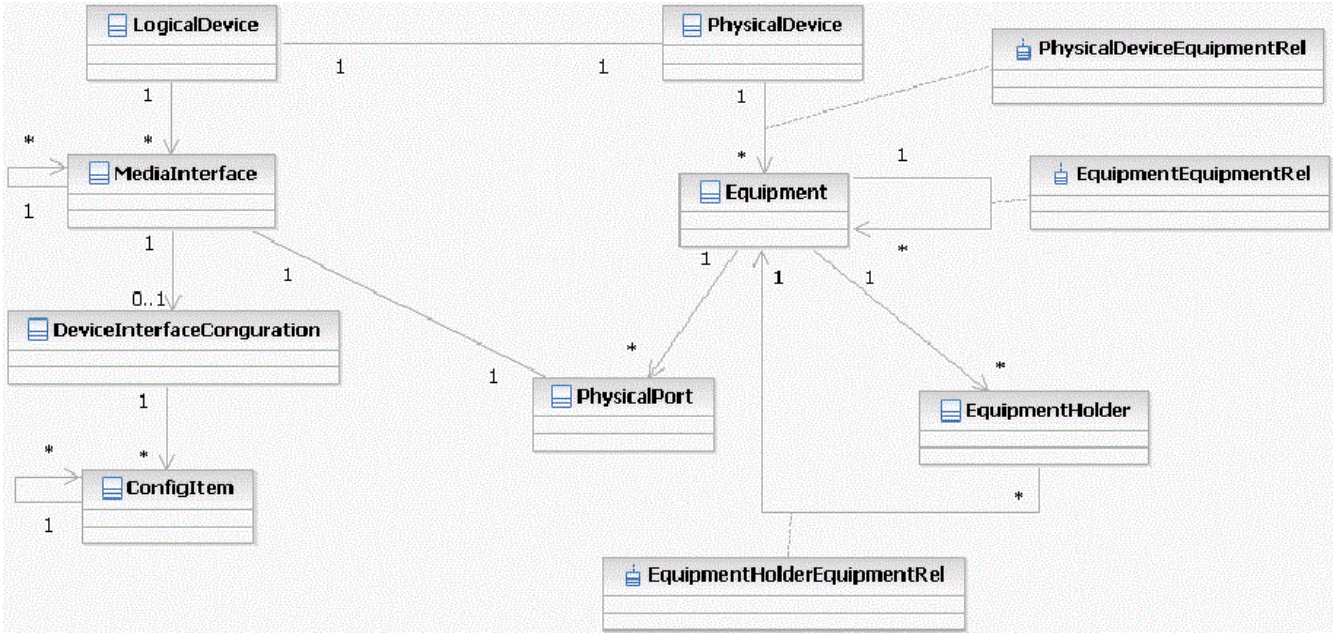
About Cartridge Modeling

This chapter describes how the Oracle Communications Network Integrity UIM Integration cartridge is modeled in Oracle Communications Design Studio.

UIM Integration Cartridge UML Representation

Figure 4-1 displays a Unified Modeling Language (UML) diagram depicting the object relationship being rendered.

Figure 4-1 UIM Integration Cartridge UML Representation



Oracle Communications Information Model Information

Oracle Communications Unified Inventory Management (UIM) does not support Device Interface Configuration and Media Interface entities. The Device Interface Configuration entity is excluded from integration. MediaInterface in Network Integrity is mapped to DeviceInterface in UIM.

The Media Interface entity may be realized as either DeviceInterface or MediaInterface in Network Integrity. It can be realized only as DeviceInterface in UIM.

Attributes that are defined in the Oracle Communications Information Model, supported by Network Integrity, but not supported by UIM are implemented as characteristics in UIM and are mapped from attribute to characteristic and from characteristic to attribute. These characteristics are not mandatory in UIM. If they are not present, the value is empty on import and is not set during resolution. All other attributes are mapped directly.

Device Hierarchy

The UIM Integration cartridge supports both logical device and physical device hierarchies. The cartridge supports physical-to-logical mapping relationships (physical-device-to-logical-device mapping and physical-port-to-device-interface mapping).The cartridge supports both Device Interface and Media Interface entities. In UIM, Media Interface is represented as Device Interface with additional characteristics.

Characteristics

Table 4-1 shows characteristics included in the **ora_ni_uim_ocim** cartridge.

Table 4-1 Characteristics in the ora_ni_uim_ocim Cartridge

Characteristic	UI Label	Field Type	Notes
nativeEmsName	Native EMS Name	Text	N/A
nativeEmsAdminServiceState	Native EMS Admin Service State	Text	Available values: • UNKNOWN • IN_SERVICE • OUT_OF_SERVICE • TESTING • IN_MAINTAINANCE
nativeEmsServiceState	Native EMS Service State	Text	Available values: • UNKNOWN • IN_SERVICE • OUT_OF_SERVICE • TESTING • IN_MAINTAINANCE
mtuCurrent	MTU Current	Text	N/A
mtuSupported	MTU Supported	Text	N/A
nativeEmsConnectorPresent	Native EMS Connector Present	Check box	N/A

Logical Mapping

The UIM Integration cartridge supports the following logical mappings.

- Logical Device
- Device Interface

Logical Device

Table 4-2 shows characteristics for the Logical Device specification.

Table 4-2 Logical Device Characteristics

Characteristic	Information Model Support	Field Type
nativeEmsName	Static	Text

Table 4-2 (Cont.) Logical Device Characteristics

Characteristic	Information Model Support	Field Type
nativeEmsAdminServiceState	Static	Text
nativeEmsServiceState	Static	Text
physicalLocation	Static	Text

Device Interface

Table 4-3 shows characteristics for the Device Interface specification.

Table 4-3 Device Interface Characteristics

Characteristic	Information Model Support	Field Type
nativeEmsName	Static	Text
nativeEmsAdminServiceState	Static	Text
nativeEmsServiceState	Static	Text
physicalLocation	Static	Text
ifType	Static	Text
minSpeed	Static	Text
maxSpeed	Static	Text
nominalSpeed	Static	Text
physicalAddress	Static	Text
mtuCurrent	Static	Text
mtuSupported	Static	Text
nativeEmsConnectorPresent	Static	Check box

Physical Mapping

The UIM Integration cartridge supports the following physical mappings.

- Physical Device
- Equipment
- Equipment Holder
- Physical Port

Physical Device

Table 4-4 shows characteristics for the Physical Device specification.

Table 4-4 Physical Device Characteristics

Characteristic	Information Model Support	Field Type
nativeEmsName	Static	Text

Equipment

Table 4-5 shows characteristics for the Equipment specification.

Table 4-5 Equipment Characteristics

Characteristic	Information Model Support	Field Type
nativeEmsName	Static	Text
physicalLocation	Static	Text

Equipment Holder

Table 4-6 shows characteristics for the Equipment Holder specification.

Table 4-6 Equipment Holder Characteristics

Characteristic	Information Model Support	Field Type
nativeEmsName	Static	Text
physicalLocation	Static	Text

Physical Port

Table 4-7 shows characteristics for the Physical Port specification.

Table 4-7 Physical Port Characteristics

Characteristic	Information Model Support	Field Type
nativeEmsName	Static	Text
physicalLocation	Static	Text
serialNumber	Static	Text
physicalAddress	Static	Text

5

About Design Studio Construction

This chapter describes how the Oracle Communications Network Integrity UIM Integration cartridge is built from the Oracle Communications Design Studio perspective.

Actions

The following tables describe the Design Studio construction of actions and associated components in the UIM Integration cartridge.

> **Note:**
>
> Parameter values are case-sensitive and must be entered in capital letters when commands are run from a command-line interface.

Table 5-1 Describes how actions in the UIM Integration cartridge are constructed in Design Studio.

Table 5-1 Design Studio Construction of Actions

Action	Result Category	Address Handler	Scan Parameter Groups	Processors
Abstract Import from UIM	Device	N/A	Parallel Process Parameters See "Using Design Studio to Extend Network Integrity" in *Network Integrity Developer's Guide* for more information.	• Import UIM Initializer • Logical Device UIM Finder • Physical Device UIM Finder • Logical Device UIM MultiThread Importer • Physical Device UIM MultiThread Importer • Logical Device UIM Importer • Linked Physical Device UIM Importer • Logical Device UIM Persister • Physical Device UIM Importer • Linked Logical Device UIM Importer • Physical Device UIM Persister
Import from UIM	Device	N/A	UIMImportParameters. See Table 5-6.	• Processors inherited from the Abstract Import from UIM action • Scan Parameter UIM Initializer
Abstract Detect UIM Discrepancies	Device	N/A	N/A	• Processors inherited from the Detect Discrepancies action • UIM Discrepancies Filter Initializer
Abstract Resolve in UIM	Device	N/A	N/A	• UIM Resolution Framework Initializer • UIM Resolution Initializer • UIM Resolution Framework Dispatcher

Table 5-1 (Cont.) Design Studio Construction of Actions

Action	Result Category	Address Handler	Scan Parameter Groups	Processors
Abstract Incremental Import from UIM	Device	N/A	N/A	• Processors inherited from the Abstract Import from UIM action • ME Names Collector • UpdateNotificationStatus
Incremental Import from UIM	Device	N/A	UIMIncrementalImportParameters. See Table 5-5	• Processors inherited from the Abstract Incremental Import from UIM action. • Incremental Scan Parameter UIM Initializer

Characteristics

Oracle Communications Unified Inventory Management (UIM) requires certain characteristics to model attributes that are native in Network Integrity. Some of these characteristics are part of UIM Integration, delivered in the **ora_ni_uim_ocim** cartridge. Other attributes are part of UIM, delivered in the **ora_uim_model** cartridge.

These characteristics have the **Ignore Characteristic in Network Integrity** tag. They appear as characteristics in UIM, but not in Network Integrity. The UIM Integration cartridge needs to handle these characteristics differently.

Table 5-2 shows UIM integration characteristics.

Table 5-2 UIM Integration Characteristics

Characteristic	Source	Enumerations
nativeEmsName	ora_ni_uim_ocim	N/A
nativeEmsAdminServiceState	ora_ni_uim_ocim	Available values: • UNKNOWN • IN_SERVICE • OUT_OF_SERVICE • TESTING • IN_MAINTAINANCE
nativeEmsServiceState	ora_ni_uim_ocim	Available values: • UNKNOWN • IN_SERVICE • OUT_OF_SERVICE • TESTING • IN_MAINTAINANCE
physicalLocation	ora_uim_model	N/A
ifType	ora_uim_model	N/A
minSpeed	ora_uim_model	N/A
maxSpeed	ora_uim_model	N/A
nominalSpeed	ora_uim_model	N/A
physicalAddress	ora_uim_model	N/A
mtuCurrent	ora_ni_uim_ocim	N/A

Table 5-2 (Cont.) UIM Integration Characteristics

Characteristic	Source	Enumerations
mtuSupported	ora_ni_uim_ocim	N/A
nativeEmsConnectorPresent	ora_ni_uim_ocim	N/A
serialNumber	ora_uim_model	N/A

To integrate Network Integrity cartridges with UIM and to include the attributes listed in Table 5-2, add the characteristics listed in the following table to your UIM cartridge specifications.

Table 5-3 displays characteristics that entities should include for integration with UIM.

Table 5-3 Characteristics for UIM Integration

UIM Entity	Characteristics to Include
LogicalDevice	• nativeEmsName • nativeEmsAdminServiceState • nativeEmsServiceState • physicalLocation
DeviceInterface	• nativeEmsName • nativeEmsAdminServiceState • nativeEmsServiceState • physicalLocation • ifType • minSpeed • maxSpeed • nominalSpeed If the Interface is to mimic a MediaInterface in Network Integrity, the following will also be required: • physicalAddress • mtuCurrent • mtuSupported • nativeEmsConnectorPresent
PhysicalDevice	nativeEmsName
Equipment	• nativeEmsName • physicalLocation
EquipmentHolder	• nativeEmsName • physicalLocation
PhysicalPort	• nativeEmsName • physicalLocation • serialNumber • physicalAddress

Scan Parameter Groups

The Import from UIM action uses the UIMImportParameters scan parameter group. Table 5-4 outlines the Design Studio construction of this scan parameter group.

Table 5-4 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 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
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

The Incremental Import from UIM action uses the UIMIncrementalImportParameters scan parameter group. Table 5-5 outlines the Design Studio construction of this scan parameter group.

Table 5-5 UIMIncrementalImportParameters Scan Parameter Group Design Studio Construction

Characteristic Name	Parameter Type	Description	UI Label
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	Checkbox	Use this box to indicate whether to import physical devices. By default, this box is checked in the UI.	Import Physical Devices
nmsNotificationCircle	String	Use this field to provide NMS circle/oss name	Nms Notification Circle
nmsNotificationVendor	String	Use this field to provide NMS vendor name	Vendor
nmsNotificationCount	Int	Use this field to provide how many NMS notifications to be fetched	Nms Notification Count

Processors

Table 5-6 describes how processors are constructed in Design Studio.

Table 5-6 Design Studio Construction of Processors

Processor	Variable
Import UIM Initializer	Input: N/A Output: • filters • uimImportContext • uimLogicalDeviceIDs • uimPhysicalDeviceIDs
Scan Parameter UIM Initializer	Input: filters Output: N/A
Logical Device UIM Finder	Input: • filters • ldev • uimLDev • uimImportContext • uimLogicalDeviceIDs • uimPhysicalDeviceIDs
Physical Device UIM Finder	Input: • filters • uimImportContext • uimLogicalDeviceIDs • uimPhysicalDeviceIDs
Logical Device UIM Importer	Input: • filters • uimImportContext • uimLogicalDeviceID • uimPhysicalDeviceIDs Output: • ldev • uimLDev
Linked Physical Device UIM Importer	Input: • filters • uimImportContext • uimLogicalDeviceID • uimPhysicalDeviceIDs • ldev • uimLDev Output: pDev
Logical Device UIM Persister	Input: N/A Output: N/A
Physical Device UIM Importer	Input: • uimImportContext • uimPhysicalDeviceID Output: pDev
Physical Device UIM Persister	Input: N/A Output: N/A
UIM Discrepancies Filter Initializer	Input: N/A Output: N/A

Table 5-6 (Cont.) Design Studio Construction of Processors

Processor	Variable
Discrepancy Detector	Input: N/A Output: N/A
UIM Resolution Framework Initializer	Input: N/A Output: • baseResolutionElement • uimResolutionContext
UIM Resolution Initializer	Input: • baseResolutionElement • uimResolutionContext Output: N/A
UIM Resolution Framework Dispatcher	Input: • baseResolutionElement • uimResolutionContext Output: N/A
Logical Device UIM MultiThread Importer	Input: • filters • uimImportContext • uimLogicalDeviceIDs
Physical Device UIM MultiThread Importer	Input: • filters • uimImportContext • uimLogicalDeviceIDs • uimPhysicalDeviceIDs
Linked Logical Device UIM Importer	Input: • filters • uimImportContext • uimLogicalDeviceIDs • pdev • uimPDev
ME Names Collector	Input: incrementalimportScanParams Output: meNames
UpdateNotificationStatus	Input: • incrementalimportScanParams • meNames
Incremental Scan Parameter UIM Initializer	Input: incrementalimportScanParams

6
Working with the UIM Sample Web Service

This chapter provides information about the Oracle Communications Network Integrity UIM Sample Web Service.

About the NI UIM Client

This section provides instructions for building the Oracle Communications Unified Inventory Management (UIM) Sample web service client JAR file. This file is used by the UIM Integration cartridge. This client JAR is available in the ora_ni_uim_webservice project.

Generating the NI UIM Client JAR File

Building the UIM Sample web service client JAR is required only when you want to modify the web service and have the UIM Integration cartridge use it. To use the web service in client software, you must generate the **NI_UimClient.jar** file.

To generate the JAR file:

1. Set up the *host*.properties file. See "Installing the UIM Sample Web Service" for instructions.
2. Execute the **All** target using the **buildUimClient.xml** ANT file.
3. Copy the **NI_UimClient.jar** file from the **webarchive\ora_ni_uim_webservices_cartproj** folder into the UIM Integration cartridge **lib** directory.

The **NI_UimClient.jar** file is generated, which can then be used in client software.

UIM Connection Client Example

Example 6-1 shows the code that you can use to connect the NI UIM Client to UIM.

Example 6-1 Client UIM Connection Code

```
String userId = dis.getUsername();
String passWd = dis.getPassword();
if (userId == null || passWd == null) {
    logger.warning("Configuration error: Username/Password values are required for UIM
Inventory System.");
    throw new ProcessorException(
            "Configuration error: Username/Password values are required for UIM
Inventory System.");
}

try {
    logger.finest("invoke new NI_Uim_Impl");

    NI_Uim service = new NI_Uim_Impl();
    if (service == null) {
        logger.severe("UIM Web Service initialization error: NI_Uim == null");
        throw new ProcessorException("UIM Web Service initialization error: NI_Uim ==
null");
    }
```

```
    logger.finest("invoke service.getNI_UimHTTPPort");

    NI_UimPort port = service.getNI_UimHTTPPort(userId.getBytes(), passWd.getBytes());
    if (port == null) {
        logger.severe("UIM Web Service initialization error: NI_UimPort == null");
        throw new ProcessorException("UIM Web Service initialization error: NI_UimPort
== null");
    }

    logger.finer("UIM WS Endpoint = " + dis.getAddress());
    logger.finest("Setting endpoint on the WS port.");
    Stub stub = (Stub) port;
    stub._setProperty(Stub.ENDPOINT_ADDRESS_PROPERTY, dis.getAddress());

    // Success. Save the connection objects.
    //
    uim_service = service;
    uim_port = port;
}
```

When **uim_port** is established, the web service calls are available through uim_port.

> **Note:**
>
> For the full source code for this fragment, view `oracle.communications.integrity.mibiiuimcartridge.resolutionprocessors.resolutionframeworkinitializer.uimWebService`.

About the UIM Sample Web Service Operations

This section provides information about web service operations for the UIM Sample web service provided in the **ora_ni_uim_webservice** project.

The web service assumes that all specifications comply with the Oracle Communications Information Model. The web service assumes that all specifications in UIM that the web service interacts with contain the characteristics listed in Table 6-1.

Table 6-1 UIM Sample Web Services Entity Characteristics Requirements

Entity	Characteristics
PhysicalDevice	nativeEMSName
Equipment	physicalLocation nativeEMSName
EquipmentHolder	nativeEMSName physicalLocation
PhysicalPort	nativeEmsName physicalLocation physicalAddress serialNumber

Table 6-1 (Cont.) UIM Sample Web Services Entity Characteristics Requirements

Entity	Characteristics
LogicalDevice	nativeEMSName nativeEMSAdminServiceState nativeEMSServiceState physicalLocation
DeviceInterface	nativeEMSName nativeEMSAdminServiceState nativeEMSServiceState physicalLocation ifType minSpeed maxSpeed nominalSpeed
MediaInterface	nativeEMSName nativeEMSAdminServiceState nativeEMSServiceState physicalLocation ifType minSpeed maxSpeed nominalSpeed mtuCurrent mtuSupported nativeEMSConnectorPresent physicalAddress

Installing the UIM Sample Web Service

Before you install the UIM Sample Web Service, ensure that the following are installed:

- Oracle Communications Service Catalog and Design - Design Studio for UIM. See *SCD Design Studio Installation Guide.*
- Oracle WebLogic Server and UIM, or UIM resources, in an environment accessible to Design Studio. See "Unified Inventory Management Installation Overview " in *UIM Installation Guide* for information about installing UIM. See *UIM Developer's Guide* for information about installing UIM resources.

To install the UIM Sample Web Service on UIM:

1. Rename the **etc\COMPUTERNAME.properties** file to *host***.properties**, where *host* is the environment name of the computer.

> **Note:**
>
> Obtain the environment name for the computer by entering the **hostname** command at the prompt. The output from this command is the computer environment name.

This enables the loaded Java project to reflect the environment.

2. Open the *host***.properties** file and set the property fields to reference your WebLogic domain and UIM instance.
3. Set the READ.TIMEOUT value in milliseconds with the desired value. This value describes the duration after which the NI-UIM webservice requests will time-out if no response is received.
4. Save and close the file.
5. (Optional) If you wish to develop the web service in Design Studio, set the following Java Build Path Variables:
   - UIM_LIB: Example **C:/wls1036/user_projects/domains/UIM/UIM/lib**
   - POMS_LIB: Example **C:/OracleCommunications/commsplatform/ws**
   - POMS_PLIB: Example **C:/OracleCommunications/POMSClient/lib**
   - FMW_LIB: Example **C:/wls1036/modules**
6. In the **build.xml** file, run the following ANT targets to get the web service ready to build and deploy:
   - clean
   - extract.ear
7. Edit the **META-INF\application.xml** file and add the following code fragment:

```
<module>
  <web>
    <web-uri>NI_Uim.war</web-uri>
    <context-root>NI_Uim</context-root>
  </web>
</module>
```

8. In the **build.xml** file, run the following ANT targets to build the web service and update the EAR file:
   - generate-from wsdl
   - build-service
   - update.ear
9. Redeploy **custom.ear** using the WebLogic console.
10. (Optional) If you are installing the Network Integrity UIM Sample Web Service on a UIM instance that is *not* located on the same server as Design Studio, do not run **update.ear**. For information about installing UIM on a remote server, see *UIM Developer's Guide*.

Configuring the Network Integrity UI for the UIM Sample Web Service

To configure the Network Integrity UI for the UIM Sample Web Service:

1. Log in to Network Integrity.

2. Click **Manage Import System**. The Import System Details page is displayed.
3. In the **Name** field, enter the name of the import system.
4. In the **Address** field, enter the following address:

   `http://UIM_IPAddress:UIM_Port/NI_Uim/NI_UimHTTP`

   If UIM is installed with SSL enabled, the UIM URL requires HTTPS.
5. In the **User Name** field, enter the UIM user name.
6. In the **Password** field, enter the password of UIM.

> **Note:**
>
> You must configure the WebLogic domain to accept UIM Sample Web Service connections using HTTPS. See "Unified Inventory Management Installation Overview " in *UIM Installation Guide* and "Unified Inventory Management System Administration Overview" in *UIM System Administrator's Guide* for more information.

About CRUD Operations

The UIM Sample Web Service provides Create, Read, Update, and Delete (CRUD) operations on the following entities:

- Logical Device Entity
- Device Interface Entity
- Media Interface Entity
- Physical Device Entity
- Equipment Entity
- Equipment Holder Entity
- Physical Port Entity
- Channelized Connectivity Entity

The UIM Sample Web Service provides create-association operations between peer entities using:

- linkLogicalPhysicalDeviceRequest: This operation supports horizontal associations. The operation provides associations between the logical device and the physical device.
- linkLogicalPhysicalInterfaceRequest: This operation supports horizontal associations. The operation provides associations between the interface and the physical ports.

The UIM Sample Web Service provides create-parent-and-child associations between entities using **link**.

The UIM Sample Web Service provides delete associations between parent and child entities and delete associations between peer entities. For these operations, the UIM Sample Web Service uses **unLink**, which deletes vertical and horizontal associations.

The UIM Sample Web Service provides operations to retrieve an entire device tree using:

- getPhysicalDeviceTree
- getLogicalDeviceTree

About Find Qualifiers

The UIM Sample Web Service defaults to the following attribute values for find-entity operations (see findLogicalDeviceRequest and findPhysicalDeviceRequest) if these qualifiers are not already set:

- id field: EQUALS
- Name: BEGINS_WITH
- Mgmt Ip Address: EQUALS

About the <specType> Entity

The **<specType>** element is commonly used throughout the UIM Sample Web Service examples (see Example 6-2). The **<specType>** element is used to identify the entity type. This stages the operations to be executed for a particular Oracle Communications Unified Inventory Management (UIM) API. Set the **<name>** element to the name of the specification you are using.

Example 6-2 <specType> Element

```
<specType>
      <name>?</name>
      <entityType>PHYSICALDEVICE</entityType>
</specType>
```

Supported Entity Types

The **<entityType>** element supports the following values:

- LOGICALDEVICE
- DEVICEINTERFACE
- MEDIAINTERFACE
- PHYSICALDEVICE
- EQUIPMENT
- EQUIPMENTHOLDER
- PHYSICALPORT

Response Messages

The following response messages can appear:

- SUCCESS: This message appears with a return value and indicates that an operation has succeeded and is committed to UIM.
- FAILURE: This error message indicates that the operation failed but the UIM rollback was successful.
- Exception messages indicate that an operation failed and the UIM rollback failed. This throws exceptionsInventoryFaultType and ValidationFaultType.

Use the following code pattern to inspect a response and detect an operation FAILURE.

```
CreateLogicalDeviceRequestType createLogicalDeviceRequestType = new
CreateLogicalDeviceRequestType();

createLogicalDeviceRequestType.setLogicalDevice(wsLogicalDevice);

UIMWebService.checkResponse(uimPort.createLogicalDevice(createLogicalDeviceRequestType));
```

UIM Sample Web Service Entity Operations

Table 6-2 describes the files included in the UIM Sample Web Service project.

Table 6-2 UIM Sample Web Service Project Files

File	Description
NI-Uim-soapui-project.xml	SoapUI project file that contains sample requests and responses.
WSDL-Documentation.html	Generated WSDL documentation that shows all the available operations. A short description of each operation is provided.

Table 6-3 shows how the UIM Sample Web Service processes various entities.

Table 6-3 Entity Operation Coding Patterns

Entity Operations	Signature Patterns
createLogicalDeviceRequest createDeviceInterfaceRequest createMediaInterfaceRequest createPhysicalDeviceRequest createEquipmentRequest createEquipmentHolderRequest createPhysicalPortRequest	Web service: • Passes in a fully specified entity • Returns the ID of each entity if successful
updateLogicalDeviceRequest updateDeviceInterfaceRequest updateMediaInterfaceRequest updatePhysicalDeviceRequest updateEquipmentRequest updateEquipmentHolderRequest updatePhysicalPortRequest	Web service: • Passes in a fully specified entity; however, all parameters must be specified, not just those parameters being updated • Returns the ID of each entity if successful
findLogicalDeviceRequest findDeviceInterfaceRequest findMediaInterfaceRequest findPhysicalDeviceRequest findEquipmentRequest findEquipmentHolderRequest findPhysicalPortRequest	Web service: • Passes in filters and filter qualifiers • Returns 0, 1, or *n* entity IDs • Checks for null ID elements when 0 entity IDs are returned • Returns ID. For finddevice operations, it may also return the IDs of related devices.

Table 6-3 (Cont.) Entity Operation Coding Patterns

Entity Operations	Signature Patterns
getLogicalDeviceRequest getDeviceInterfaceRequest getMediaInterfaceRequest getPhysicalDeviceRequest getEquipmentRequest getEquipmentHolderRequest getPhysicalPortRequest	Web service: • Passes in the ID of the entity to be retrieved • Returns the fully specified entity
link	Web service: • Passes in the parent and child ID • Returns the parent ID if successful
unlink	Web service: • Passes in the parent and child IDs • Assumes the parent ID is the logical entity and the child ID is the physical entity when used to delete associations between peer LogicalDevice to PhysicalDevice, and peer DeviceInterface to PhysicalPort • Returns the parent ID if successful
getPhysicalDeviceTree getLogicalDeviceTree	Web service: • Passes in the ID • Returns the entire device tree. See the schemas to understand how this works with unlimited number of descendent child objects.

Note:

Oracle Communications Network Integrity does not support delete operations on the PhysicalDevice, LogicalDevice, and Equipment entities. Log in to UIM as administrator to delete these entities.

Enabling Debugging for the Web Service

To enable debugging for the UIM Sample Web Service, which is displayed on the Oracle WebLogic Server console:

1. Open the *domain***UIM\config\loggingconfig.xml** file.
2. Add the following text to the file:

```
<logger name="oracle.communications.inventory.webservice.adapter.ni"
additivity="false">
      <level value="debug" />
      <appender-ref ref="stdout"/>
      <appender-ref ref="rollingFile"/>
</logger>
```

UIM Sample Web Service to Update Logical Devices Code Example

Example 6-3 shows a fragment of code used to update a logical device using the UIM Sample Web Service. The fragment shows how handling is accomplished for the UIM Sample Web Service.

Example 6-3 Update Logical Device Coding Fragment

```
String errorMessageTask = "";
try {
    if (connect() == null) {
        failAll(context, attributeValueMismatchList, UIM_CONNECTION_FAILURE);
        return;
    }

    errorMessageTask = "findLogicalDevice";
    Range range = new Range();
    range.setStartRange(0);
    range.setEndRange(2);
    FindLogicalDeviceRequestType findLogicalDeviceRequestType = new
FindLogicalDeviceRequestType();
    findLogicalDeviceRequestType.setId(importedLogicalDevice.getId());
    findLogicalDeviceRequestType.setIdQualifier(SearchQualifier.equals);
    findLogicalDeviceRequestType.setRange(range);

    GetIdsResponseType uimLogicalDevices =
uimPort.findLogicalDevice(findLogicalDeviceRequestType);

    if (uimLogicalDevices.getId() != null) {
        String[] Ids = uimLogicalDevices.getId();
        if (Ids.length != 1) {
            String msg = "Found " + Ids.length + " logicalDevices matching
importedLogicalDevice.getId()="
                    + importedLogicalDevice.getId() + ", expecting 1";
            logger.log(Level.SEVERE, msg);
            failAll(context, attributeValueMismatchList, msg);
            return;
        }
    }

    errorMessageTask = "getLogicalDevice";
    GetLogicalDeviceRequestType getLogicalDeviceRequestType = new
GetLogicalDeviceRequestType();
    getLogicalDeviceRequestType.setId(importedLogicalDevice.getId());
    GetLogicalDeviceResponseType getLogicalDeviceResponseType = uimPort
            .getLogicalDevice(getLogicalDeviceRequestType);

    NILogicalDevice wsLogicalDevice = getLogicalDeviceResponseType.getLogicalDevice();

    for (DisDiscrepancy discrepancy : attributeValueMismatchList) {

        String attrName = discrepancy.getAttributeOrRelationshipName();

        if (attrName.equals(NAME)) {
            wsLogicalDevice.setName(discoveredLogicalDevice.getName());
        } else if (attrName.equals(DESCRIPTION)) {
            wsLogicalDevice.setDescription(discoveredLogicalDevice.getDescription());
        } else if (attrName.equals(LogicalDeviceHelper.MGMT_IP_ADDRESS)) {
```

```
            copyCharacteristic(discoveredLogicalDevice, wsLogicalDevice,
LogicalDeviceHelper.MGMT_IP_ADDRESS);
        } else if (attrName.equals(LogicalDeviceHelper.NATIVE_EMS_ADMIN_SERVICE_STATE)) {
            wsLogicalDevice.setNativeEmsAdminServiceState(LogicalDeviceHelper
                    .convertEmsServiceState(discoveredLogicalDevice.getNativeEmsAdminServ
iceState()));
        } else if (attrName.equals(LogicalDeviceHelper.NATIVE_EMS_SERVICE_STATE)) {

wsLogicalDevice.setNativeEmsServiceState(LogicalDeviceHelper.convertEmsServiceState(disco
veredLogicalDevice
                    .getNativeEmsServiceState()));
        } else if (attrName.equals(LogicalDeviceHelper.SYS_OBJECT_ID)) {
            copyCharacteristic(discoveredLogicalDevice, wsLogicalDevice,
LogicalDeviceHelper.SYS_OBJECT_ID);
        } else {
            copyCharacteristic(discoveredLogicalDevice, wsLogicalDevice, attrName);
        }
    }
    UpdateLogicalDeviceRequestType updateLogicalDeviceRequestType = new
UpdateLogicalDeviceRequestType();
    updateLogicalDeviceRequestType.setLogicalDevice(wsLogicalDevice);

    errorMessageTask = "updateLogicalDevice";
    checkResponse(uimPort.updateLogicalDevice(updateLogicalDeviceRequestType));
    passAll(context, attributeValueMismatchList);
} catch (Exception e) {
    failAll(context, attributeValueMismatchList, errorMessageTask + LOCAL_ENTITY +
discoveredLogicalDevice.getId(), e);
}
logger.exiting(LOG_MYCLASSNAME, "handleAttributeValueMismatch");
```