Oracle® Communications Unified Inventory Management

Web Services Developer's Guide

Release 8.0.1
G50248-01
April 2026

ORACLE

Oracle Communications Unified Inventory Management Web Services Developer's Guide, Release 8.0.1

G50248-01

Copyright © 2014, 2026, Oracle and/or its affiliates.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software, software documentation, data (as defined in the Federal Acquisition Regulation), or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software," "commercial computer software documentation," or "limited rights data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle®, Java, MySQL, and NetSuite are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Contents

Copyright © 2014, 2026, Oracle and/or its affiliates.

Copyright © 2014, 2026, Oracle and/or its affiliates.

3 Working with the Network Resource Management Web Service

Copyright © 2014, 2026, Oracle and/or its affiliates.

Copyright © 2014, 2026, Oracle and/or its affiliates.

4 Developing Custom SOAP Web Services

Copyright © 2014, 2026, Oracle and/or its affiliates.

5 Developing Custom REST Web Services

Copyright © 2014, 2026, Oracle and/or its affiliates.

6 Deploying, Testing, and Securing UIM Web Services

Copyright © 2014, 2026, Oracle and/or its affiliates.

About This Content

This guide describes the Oracle Communications Unified Inventory Management (UIM) Web Services. The information provided in this guide includes the UIM Web Service framework that supports web services, the various UIM Web Services that are available, and how to create custom web services. When creating custom web services, you can use Oracle Communications Service Catalog and Design - Design Studio, which is an Eclipse-based integrated development environment. This guide includes references to both Design Studio and UIM, and often directs the reader to see the Design Studio Help and the UIM Help for instructions on how to perform specific tasks. This guide includes examples used in given situations. The guidelines and examples may not be applicable in every situation.

Audience

This guide is intended for developers who have a working knowledge of web services in general, and who understand XML, Ant and Java development, including JPA, standard Java practices, and J2EE principles.

You should read *UIM Concepts* before reading this 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.

Copyright © 2014, 2026, Oracle and/or its affiliates.

1

Web Services Overview

This chapter provides introductory information about the Oracle Communications Unified Inventory Management (UIM) Web Services.

Web services support interoperable machine-to-machine interaction over a network. Web services are APIs that can be accessed over a network, and run on a remote system hosting the requested services. Web service operations are described by the Web Service Definition Language (WSDL).

> **Note**
>
> In this guide, *UIM_CONFIG_PATH* represents the directory for UIM configuration files, as follows:
>
> - In a traditional environment, *UIM_CONFIG_PATH* is *UIM_Home*/**config**. *UIM_Home* is *Oracle_home*/**user_projects/domains**/*domain_name*/**UIM**, where *domain_name* is the domain name you supplied when installing UIM.
> - In a cloud native environment, *UIM_CONFIG_PATH* is *UIM_IMAGE_BUILDER_TOOLKIT*/**staging/cnsdk/uim-model/UIM/config**. You can obtain UIM_IMAGE_BUILDER_TOOLKIT from Oracle Software Delivery Cloud. To modify configuration files in UIM cloud native, see "Customizing UIM Configuration Properties" in *UIM Cloud Native Deployment Guide* and "Administering a UIM Cloud Native Deployment" in *UIM System Administrator's Guide*.
>
> In this guide, *UIM_SDK_Home* represents the directory where you extract UIM_SDK.zip, as follows:
>
> - In a traditional environment, the files present in *UIM_SDK_Home* are also present in *UIM_Home.*
> - When working with UIM web services or developing custom web services, see "Using Design Studio to Extend UIM" in *UIM Developer's Guide*.

About UIM Web Service Standards and Specifications

Table 1-1 lists the UIM web service standards and specifications.

Table 1-1 UIM Web Service Standards and Specifications

Standard and Specification	Version Release	Description	Compliance
JAX-RPC	1.1	XML <--> Java binding specification	Compliant.
JSR-181	NA	Java web service annotations	Deprecated. Uses basic annotations for inter operability.

Copyright © 2014, 2026, Oracle and/or its affiliates.

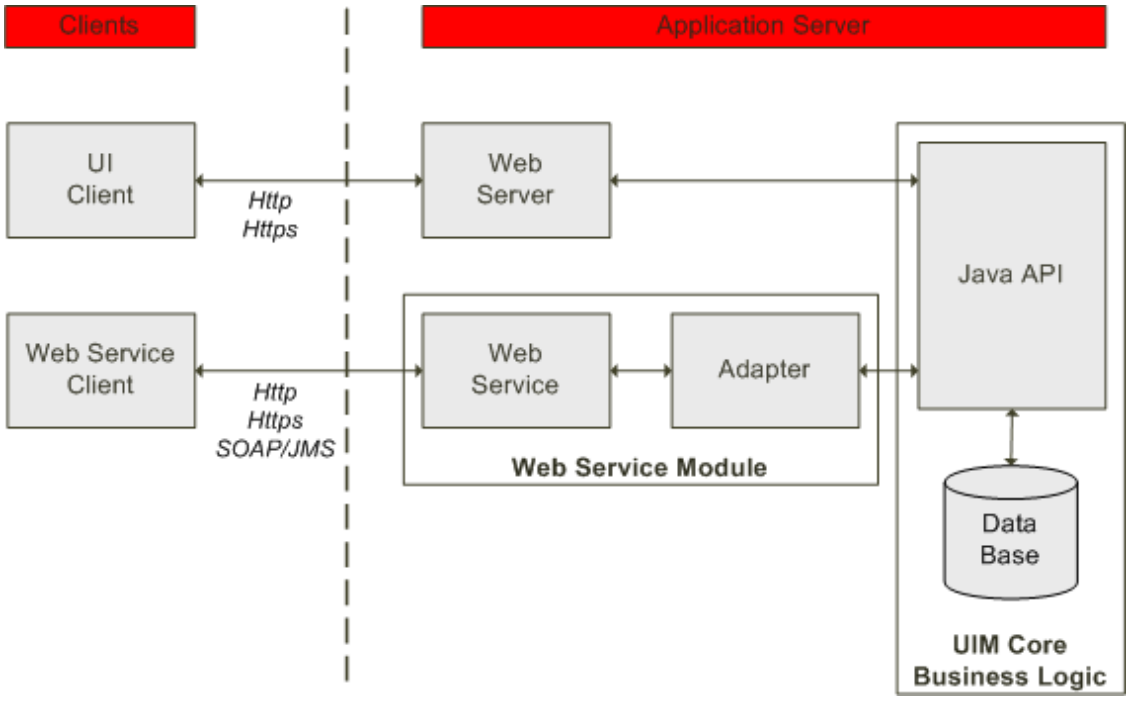
Table 1-1 (Cont.) UIM Web Service Standards and Specifications

Standard and Specification	Version Release	Description	Compliance
SOAP	1.1	Simple Object Access Protocol (Also referred to as Service Oriented Architecture Protocol.)	Compliant. Uses XML/SOAP/HTTP and XML/SOAP/JMS.
Transport Protocols	HTTP 1.0, HTTPS 1.0 (HTTP 1.1), JMS 1.1	NA	NA
WSDL	1.1	Web Service Definition Language	Compliant.
XML	1.1	NA	Compliant. Uses XML/SOAP/HTTP and XML/SOAP/JMS.

About the UIM Web Services Framework

Figure 1-1 shows the different paths traveled by a call originating from the UIM UI client, and a call originating from outside UIM that is then processed by the web service client.

Figure 1-1 UIM Web Services



The path of the web service includes:

- Web Service Client

Copyright © 2014, 2026, Oracle and/or its affiliates.

This represents the web service user (client, web service client, or customer). Web service operations are called by sending SOAP messages over HTTP or HTTPS, or by posting SOAP messages on a UIM-defined JMS message queue. See "About Message Queues" for more information.

- Web Service Module

  This represents all the sub-modules required for implementing a web service, including the web service, the web service framework, WSDL interfaces, and WSDL implementations. The web service module is deployed as a WAR file.

  See "About the UIM Web Service Module" for more information.

- UIM business logic

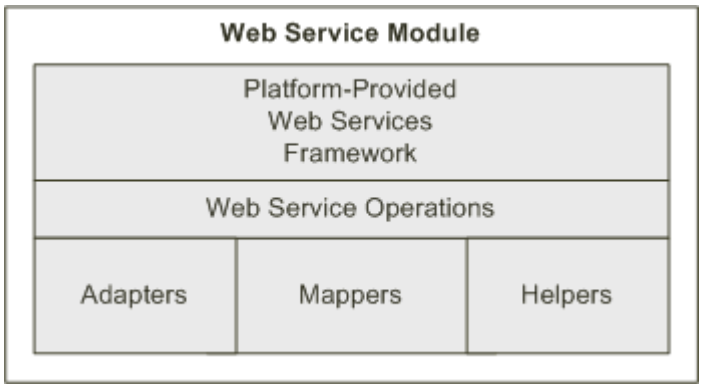
  This represents all the sub-modules required for attaining business functionality. This includes the Java API, the Java API framework, business logic, and persistence framework.

  Details of the UIM business logic are not within the scope of this document.

About the UIM Web Service Module

Figure 1-2 shows the web service module in more detail.

Figure 1-2 Web Service Module



The web service module includes:

- Platform-provided Web Services Framework

  This represents the web service framework provided by Java EE platforms, such as Oracle WebLogic Server.

- Web Service Operations

  This represents the Java web service implementation class. This is the entry point to a UIM web service. The web service operations are Java representations of the WSDL.

- Adapters

  The web service operations layer calls the adapters, which direct the calls and collect data from the appropriate UIM API managers. Transaction handling is performed in the adapters.

- Mappers

Copyright © 2014, 2026, Oracle and/or its affiliates.

Mapper classes convert data from XML to Java, and from Java to XML. Specifically, data elements of an incoming XML request are converted to data attributes of a Java class so the data can be processed. When processing is done, the data attributes of the Java class are converted to data elements of an outgoing XML response. Mapper classes are typically called by the adapter code.

- Workers

  Worker classes assist the working logic of the adapters.

The web service operations, adapter, mapper, and worker classes are further explored in "Developing Custom SOAP Web Services".

About Message Queues

The UIM installation provides the following message queues to use when calling the Inventory Web Services, which includes the UIM Service Fulfillment Web Service and the Network Resource Management Web Service, both of which are packaged in the **InventoryWS.war** file:

- inventoryWSQueue
- inventoryWSQueueAlternate

Two message queues are provided for efficient processing of web service calls. For example, you may have all web service operation calls except ProcessInteraction use inventoryWSQueue, and have ProcessInteraction use inventoryWSQueueAlternate because the ProcessInteraction operation takes longer to run than the other operations.

About Message Queues for Custom Web Services

The UIM installation also provides the following message queue to use when calling custom web services packaged in the provided **custom.ear** file:

- inventoryCustomWSQueue

> **Note**
>
> If you package your custom web service in an Enterprise Archive (EAR) file other than the provided **custom.ear** file, you must create your own message queue, create a custom listener class, and configure the class to listen to the queue. See "Packaging the WAR File in the EAR File" for more information.

About Transaction Handling

The adapter layer performs transaction handling. Transactions are started only if the thread is not already within a transaction.

About Exception Stacktraces

Exception stacktraces are available in the WebLogic server logs. Exception stacktraces are not available in the UIM web service responses.

About UIM Web Services

UIM provides the following web services:

Copyright © 2014, 2026, Oracle and/or its affiliates.

- The Service Fulfillment Web Service defines operations that enable you to create and modify business interactions, through which you can create and modify services, service configurations, and service configuration items. See "Working with the Service Fulfillment Web Service" for more information.
- The Network Resource Management Web Service defines operations that enable you to create, find, update, and delete various entities in UIM. The web service also enables you to reserve and unreserve various resource entities, and also update reservations. Lastly, the web service enables you to import and export various entities into and out of UIM. See "Working with the Network Resource Management Web Service" for more information.
- UIM also provides a way for you to develop, build, and deploy custom web services. The UIM Reference Web Service defines a single operation to create a Logical Device. This web service serves as an example to reference when developing custom web services. See "Developing Custom SOAP Web Services" for more information.

> **Note**
>
> The deprecated Reference Web Service operations are removed. The Service Fulfillment Web Service operations replace these deprecated operations. See "Working with the Service Fulfillment Web Service" for more information.

- The Cartridge Management Web Service defines various operations that support deploying cartridges. The Cartridge Deployer Tool and Oracle Communications Service Catalog and Design - Design Studio use this web service to manage cartridges. The Cartridge Management Web Service is deployed as an installation step and is displayed on the Oracle WebLogic Server Administration Console.
- The NFV Orchestration RESTful APIs define operations that enable you to create, implement, and manage the life cycles of network services and their deployment as interconnected virtual network functions (VNFs) on virtual resources. Refer to the *UIM NFV Orchestration Implementation Guide* for more information.

Copyright © 2014, 2026, Oracle and/or its affiliates.

2

Working with the Service Fulfillment Web Service

This chapter provides information about the Oracle Communications Unified Inventory Management (UIM) Service Fulfillment Web Service.

About the Service Fulfillment Web Service

> **Note**
>
> For this web service, you need an understanding of the following subjects described in "About Unified Inventory Management" in *UIM Concepts*:
>
> - Planning (business interactions, business interaction items, and engineering work orders)
> - Services (services, service configurations, and service configuration items)
> - Life-cycle management
> - Service fulfillment

The Service Fulfillment Web Service enables an external system to create new business interactions and change existing business interactions in UIM. Similarly, this web service enables an external system to create new engineering work orders and change existing engineering work orders. The Service Fulfillment Web Service also enables you to create multiple pending configurations for a service. See "About Unified Inventory Management" in *UIM Concepts* for more information.

The Service Fulfillment Web Service also enables you to disconnect a service. When you disconnect a service using the Service Fulfillment Web Service, all the configuration versions on the service transition to Canceled status, and an additional configuration version is created with the current date after the service configuration version that is in Completed status.

> **Note**
>
> Engineering work orders are a type of business interaction. They are based on a special Business Interaction specification and have the same supported Service Fulfillment Web Service functionality as business interactions.

Through business interactions, an external system can manage services, resources associated with services, and relationships between services.

The Service Fulfillment Web Service enables you to:

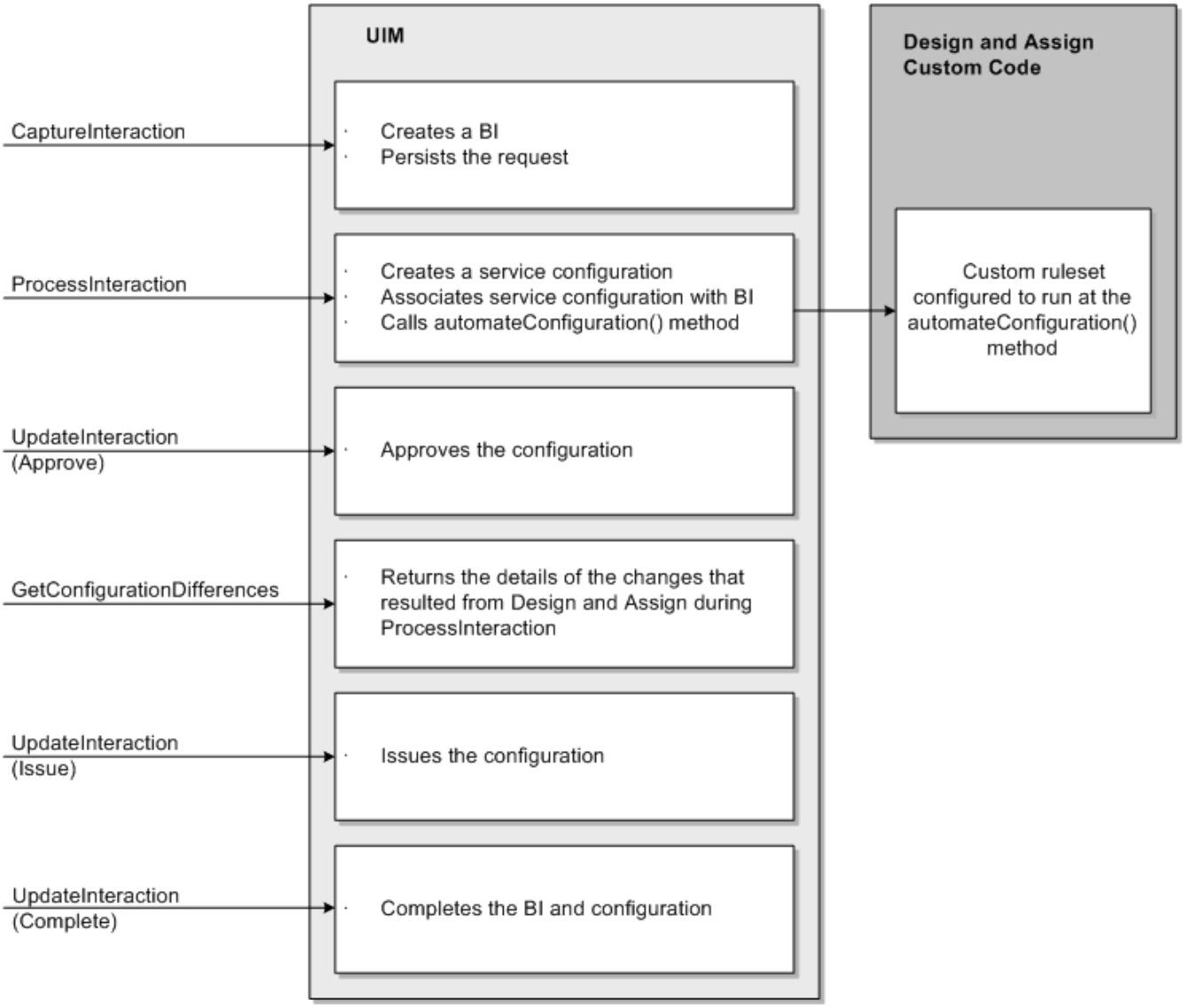
- Plan the addition, change, or disconnection of a service through a business interaction

Copyright © 2014, 2026, Oracle and/or its affiliates.

- Process business interactions to move planned services into current inventory or change existing services in current inventory, and, through custom rulesets and custom code, create or change service configuration items and allocate resources for services in current inventory
- Retrieve business interactions
- Transition business entities through their respective life-cycle states within the context of a business interaction
- Retrieve configurations
- Retrieve configuration differences
- Transition services and service configurations through their respective life-cycle states

Figure 2-1 provides an overview of the Service Fulfillment Web Service and its intended usage.

Figure 2-1 Service Fulfillment Web Service Overview



Copyright © 2014, 2026, Oracle and/or its affiliates.

The following describes the steps for the Service Fulfillment Web Service usage illustrated in Figure 2-1:

1. Call the CaptureInteraction operation. The request provides details regarding the action to take on a service, and includes a list of parameters that provide information UIM needs to provision the service. UIM creates a business interaction and persists the request. The business interaction ID (or external ID representing the business interaction) is returned to the external system.
2. Call the ProcessInteraction operation. The request includes the business interaction ID. This ID is the external ID representing the business interaction. UIM uses the business interaction ID to retrieve the business interaction and the persisted captureInteractionRequest data.

   UIM evaluates the service action on the persisted captureInteractionRequest:

   - When the service action is **create**, UIM creates a new service and service configuration.
   - When the service action is **change**, UIM finds the service based on the service ID and creates a new service configuration.

   A new service configuration is associated to the business interaction. UIM then calls the automateConfiguration() method. You must extend this empty method through a custom ruleset. This method is intended to design and assign the service. See "Overview" in *UIM Developer's Guide* for more information on custom rulesets.

   Within the automateConfiguration ruleset, you can access the following:

   - Business interaction entity
   - Service configuration entity
   - List of parameters from captureInteractionRequest
   - UIM APIs to call and perform various functions, such as unassigning existing resources on the configuration or creating new resources to assign to the configuration.

   After resources are assigned to the configuration, the assignments on the service configuration are set to **Pending** status.
3. Call the UpdateInteraction operation to approve the configuration. This updates the status of the assignments on the service configuration from **Pending** to **Approved**.
4. Call the GetConfigurationDifferences operation to get details of changes that resulted from the design and assign during ProcessInteraction.
5. Call the UpdateInteraction operation to issue the configuration. This updates the status of the assignments on the service configuration from **Approved** to **Issued**.
6. Call the UpdateInteraction operation to complete the configuration. This updates the status of the assignments on the service configuration from **Issued** to **Completed**, and also updates the business interaction status to **Completed**.

About Business Interactions and Services

Only business interactions that support services and service configurations can be added through the web service. However, after the business interaction is created in UIM, you can use the UI to add business interaction items of any type.

Even though business interactions support only services through the web service, services have service configurations, which can have child configurations, and the web service can support these child configurations. For example, a service configuration may have a child

Copyright © 2014, 2026, Oracle and/or its affiliates.

configuration that is a service, logical device, logical device account, network, pipe (representing a pipe or channelized connectivity), or place configuration, and these configurations can be added through the customized ProcessInteraction operation as children of a service configuration. Child configurations can also be retrieved through the GetConfiguration operation, and retrieved and compared through the getConfigurationDifferences operation.

> **Note**
>
> The configuration-specific operation sections of this chapter apply to all configurable entities: service, logical device, logical device account, network, pipe (representing a pipe or channelized connectivity), and place entities.
>
> A configurable place is actually a GeographicSite specialization of the abstract Place entity; GeographicSite is the only specialization of the Place entity that is configurable. See *Oracle Communications Information Model Reference* for more information.

About Engineering Work Orders

Engineering work orders are related to and share functionality with business interactions. They are based on special Business Interaction specification that you must install by deploying the **ora_uim_workorder** base cartridge. Engineering work orders have the same supported functionality as business interactions with the Service Fulfillment Web Service.

See "Unified Inventory Management Installation Overview " in *UIM Installation Guide* for more information about installing base cartridges. See "About Unified Inventory Management" in *UIM Concepts* for more information about engineering work orders and business interactions.

About the Web Service Packaging

The Service Fulfillment Web Service is packaged in the **inventory.ear** file, within the **InventoryWS.war** file. When the installer deploys the **inventory.ear** file, the Service Fulfillment Web Service is automatically deployed and ready to use.

> **Note**
>
> The **InventoryWS.war** file also includes all of the Network Resource Management Web Service operations. See "Working with the Service Fulfillment Web Service" for information about these operations.
>
> The Service Fulfillment Web Service is no longer packaged within the **UIMServiceFulfillment.war** file. This was previously deprecated and is now removed. The URI for the HTTP protocol is */InventoryWS/InventoryWSHTTP* and for JMS protocol is */InventoryWS/InventoryWSJMS*.

About the WSDL and Schema Files

The Service Fulfillment Web Service is defined by the **InventoryWS.wsdl** file and is supported by several schema files. The WSDL file and supporting schema files are located in the *UIM_SDK_Home***/webservices/schema_inventory_webservice.zip** file.

Copyright © 2014, 2026, Oracle and/or its affiliates.

About the WSDL File

Within the ZIP file, the WSDL file is located in the **ora_uim_webservices/wsdl** directory. The WSDL file defines the CaptureInteraction, ProcessInteraction, GetInteraction, UpdateInteraction, GetConfiguration, GetConfigurationDifferences, and UpdateConfiguration operations. Each web service operation defines a request, a response, and the possible faults that can be thrown. For example, the WSDL file defines the following for the CaptureInteraction operation:

- CaptureInteractionRequest
- CaptureInteractionResponse
- CaptureInteractionFault
- InventoryFault
- ValidationFault

The request, response, and faults each define an XML structure that is defined in the supporting schema files. The following excerpts show how an XML structure defined in the WSDL correlates to the supporting schema files.

For example, the WSDL file defines and references the biws namespace (in bold):

```
xmlns:biws="http://xmlns.oracle.com/communications/inventory/webservice/
businessinteraction"
.
.
.
targetNamespace
.
.
.
<xsd:import namespace="http://xmlns.oracle.com/communications/inventory/webservice/
businessinteraction" schemaLocation="./schemas/InteractionMessages.xsd"/>
.
.
.
<wsdl:message name="CaptureInteractionRequest">
    <wsdl:part name=
        captureInteractionRequest" element="biws:captureInteractionRequest"/>
</wsdl:message>
```

This tells you that the captureInteractionRequest XML structure is defined in the schema file that defines the specified namespace as its target namespace. A search for the specified namespace reveals that **InteractionMessages.xsd** defines the referenced namespace as its target namespace.

After you determine which schema file defines the XML structure that the WSDL file references, you can navigate through the schema files to determine child XML structures and elements.

About the Schema Files

Several schema files support the Service Fulfillment Web Service. These schemas are categorized as reference schemas, web service schemas, and business schemas.

Copyright © 2014, 2026, Oracle and/or its affiliates.

Reference Schemas

Within the ZIP file, the reference schemas are located in the **ora_uim_webservices/wsdl/referenceSchemas** directory. The reference schemas define common elements used by more than one operation. So, the elements are defined in one place and then referenced.

The reference schemas are:

- InventoryCommon.xsd
- InventoryFaults.xsd
- FaultRoot.xsd

Web Service Schemas

Within the ZIP file, the web service schemas are located in the **ora_uim_webservices/wsdl/schemas** directory. The web service schemas define elements specific to the web service, such as the request structures, the response structures, and any fault structures.

The web service schemas are defined in the following files:

- InteractionMessages.xsd
- ConfigurationMessages.xsd

> **Note**
>
> The web service schemas use the **type-mapping.xsdconfig** file to map XML namespaces to Java packages.

Business Schemas

Within the ZIP file, the business schemas are located in the **ora_uim_business/schemas** directory. Each web service operation wraps a call (or multiple calls) to the UIM business layer, which is exposed through APIs. The wrapped APIs are the same APIs that the UIM UI calls in response to user input. The business layer APIs are based on functional area, as are the business schemas.

The business schemas are:

- Activity.xsd
- BusinessInteraction.xsd
- Configuration.xsd
- Connectivity.xsd
- CustomNetworkAddress.xsd
- CustomObject.xsd
- Entity.xsd
- InventoryGroup.xsd
- IPAddress.xsd
- LogicalDevice.xsd

Copyright © 2014, 2026, Oracle and/or its affiliates.

- MediaStream.xsd
- Network.xsd
- NetworkAddress.xsd
- Number.xsd
- Party.xsd
- PhysicalDevice.xsd
- Place.xsd
- Property.xsd
- PropertyLocation.xsd
- Role.xsd
- Service.xsd
- Specification.xsd
- Structure.xsd
- TNBlockModelType.xsd

> **Note**
>
> The API schemas use the **xmlbeans-mapping.xsdconfig** file to map XML namespaces to Java packages.

CaptureInteraction

The CaptureInteraction operation plans the addition, change, or disconnection of a service through a business interaction. Business interactions are used for planning inventory resources, prior to making the inventory resources available in current inventory.

captureInteractionRequest defines one order per request. The order can define multiple line items, and multiple child orders. Each child order is defined by the same structure as the order on the request. Each child order can define multiple line items and multiple child orders, and so forth.

captureInteractionRequest must specify an interaction action of CREATE or CHANGE. The interaction action is defined as an enumeration in the **BusinessInteraction.xsd** schema file. The enumeration defines several actions, but CREATE and CHANGE are the only valid actions for CaptureInteraction.

When captureInteractionRequest specifies the CREATE interaction action, it creates a business interaction to contain the order information sent in the request, and creates an attachment that contains the entire <interaction> element from captureInteractionRequest. CaptureInteraction then associates the attachment to the business interaction.

When captureInteractionRequest specifies the CREATE interaction action, you can specify an external ID for the business interaction. The external ID must be unique within UIM, and the calling system is responsible for enforcing the uniqueness; UIM does not enforce uniqueness on external IDs. When an external ID is specified, UIM captures it and stores it with all of the other request data. A subsequent request can then specify a CHANGE interaction action and supply the external ID to identify the business interaction to be changed.

Copyright © 2014, 2026, Oracle and/or its affiliates.

When captureInteractionRequest specifies the CHANGE interaction action, the request must provide either the external ID or the business interaction ID to indicate the business interaction to change. If the request provides an external ID, CaptureInteraction assumes the external ID was supplied when the business interaction was created. CaptureInteraction then retrieves the business interaction and updates it with the order information sent in the request. CaptureInteraction also creates another attachment with a higher sequence number that contains the entire <interaction> element from captureInteractionRequest, and associates the attachment to the business interaction.

You can view the XML that is contained in the attachment from within the UIM UI. If a business interaction has multiple sequence numbers for an attachment, you can view all of them in UIM.

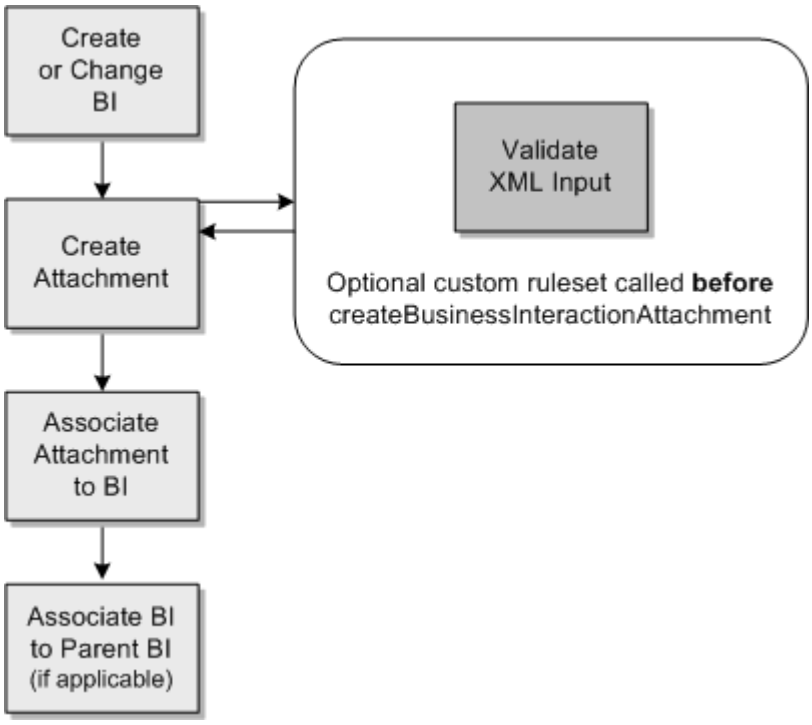
Associating Business Interactions

In UIM, business interactions can be associated with one another. captureInteractionRequest defines an element that enables you to associate one or more child business interactions to the business interaction you are creating or changing. Furthermore, you can associate one or more child business interactions to each child business interaction, which would be the grandchild business interactions to the business interaction you are creating or changing, and so forth.

CaptureInteraction Logic Flow

Figure 2-2 shows what occurs when the CaptureInteraction operation is called. A business interaction is represented as BI in the figure.

Figure 2-2 CaptureInteraction Logic Flow



In Figure 2-2, the **Validate XML Input** box represents the custom ruleset that you can configure to run before the creation of the attachment. See "Validating Input Data" for more information.

Copyright © 2014, 2026, Oracle and/or its affiliates.

CaptureInteraction wraps the BusinessInteraction.captureInteraction() API method. The API method defines two arguments: the parent business interaction, and the XML. When CaptureInteraction calls the API method, the parent business interaction argument is always null. If the interaction action is CREATE, the API method creates a business interaction, creates an attachment, and associates the attachment to the business interaction. If the interaction action is CHANGE, the API method changes the business interaction, creates an attachment with a higher sequence number, and associates the attachment to the business interaction.

If the business interaction defines a child business interaction, the API method is called from within itself. In this scenario, the parent business interaction argument is no longer null. As a result, after the business interaction is created, and the attachment is created and associated, the business interaction is associated to the parent business interaction that was specified by the argument. For example, a request defines one new business interaction that has one child business interaction. CaptureInteraction calls the API method with a parent business interaction argument of null. Business interaction A is created. The attachment is created and associated to business interaction A. Because the parent business interaction argument is null, the **Associate BI to Parent BI** box does nothing. Next, the first (in this example, the only) child business interaction is processed and calls the API method with a parent business interaction argument (business interaction A). Business interaction B is created. The attachment is created and associated to business interaction B. Because the parent business interaction argument is not null, business interaction B is associated to the parent business interaction argument that was supplied (business interaction A).

Validating Input Data

You can validate the request input data through custom code. The custom code can reside in a ruleset, or in Java code that the ruleset calls. You can configure your ruleset to run at a provided base extension point that defines the createBIAttachment() method. By configuring your ruleset to run **before** this method, your custom validations run before the attachment is created. If the validation fails, the session rolls back and the business interaction that was created is not committed.

To validate input data:

1. In Design Studio, create an Inventory project.
2. Open the Project editor.
3. Click the **Dependency** tab.
4. Add the **ora_uim_base_extpts** cartridge to the list of dependencies.
5. Save the project.
6. Create a ruleset.

   Write your custom validations in the ruleset or in Java code that the ruleset calls. For information about writing custom rulesets, see "Overview" in *UIM Developer's Guide*.
7. Save the ruleset.
8. Create a ruleset extension point and configure it as follows:
   a. In **Ruleset**, select your ruleset.
   b. In **Point**, select the **BusinessInteractionManager_createBusinessInteractionAttachment** base extension point.
   c. In **Placement**, select **BEFORE**.
   d. Save the ruleset extension point.

Copyright © 2014, 2026, Oracle and/or its affiliates.

9. Open the Service Order base specification.

   This step assumes you are using the Service Order base specification, and have copied it from the **ora_uim_basespecifications** cartridge into your project for modifying. If you are not using the Service Order base specification for your Business Interaction specification, open the Business Interaction specification that you are using. See "Business Interaction" for more information about the Service Order base specification.

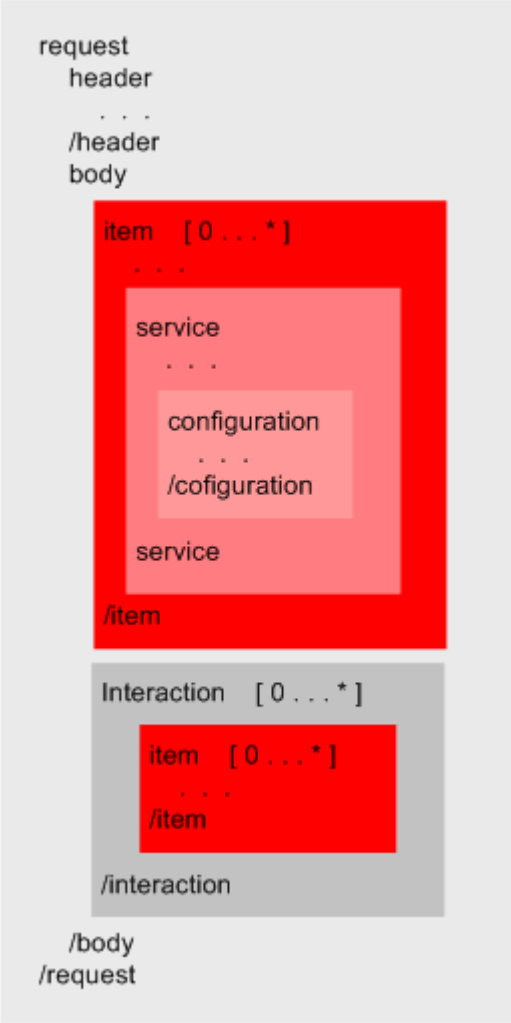
10. Click the **Rules** tab.
11. Click **Select**.
12. Select your ruleset extension point.
13. Click **OK**.
14. Save the Business Interaction specification.
15. Build the project.
16. Deploy the resultant cartridge.

captureInteractionRequest

Figure 2-3 shows the high-level content of captureInteractionRequest. Each request defines a single interaction, which specifies the data used to create the business interaction. The interaction defines a header and a body. The body defines a sequence of items: each item defines a service, and each service defines a service configuration. The body also defines a sequence of interactions, which specifies the data used to create any child business interactions.

Copyright © 2014, 2026, Oracle and/or its affiliates.

Figure 2-3 Request Content



Example 2-1 is a condensed version of captureInteractionRequest that highlights the main content to better understand CaptureInteraction. The example is numbered so that information describing the example can be referenced.

Example 2-1 omits the following:

- Namespaces, and assumes that they are properly defined
- Elements such as notes, start and end dates, effective dates, and descriptions
- Structures that detail an external ID, specification, configuration, and configurationItem
- Structures and elements within party and place, which are designated with ". . ."

> **Note**
>
> CaptureInteraction, ProcessInteraction, GetInteraction, and UpdateInteraction all use the same structure for the request and for the response. The only difference is the actual request/response name (line 01 and line 71).

Copyright © 2014, 2026, Oracle and/or its affiliates.

Example 2-1 Condensed captureInteractionRequest

```
<captureInteractionRequest>
    <invbi:interaction>
        <invbi:header>
            <invbi:specification/>
            <invbi:action/>
            <invbi:id/>
            <invbi:name/>
            <invbi:externalIdentity/>
            <invbi:state/>
        </invbi:header>
        <invbi:body>
            <invbi:item>
                <invbi:externalIdentity>
                <invbi:action/>
                <invbi:service>
                    <invsvc:specification/>
                    <invsvc:id/>
                    <invsvc:action/>
                    <invsvc:name/>
                    <invsvc:externalIdentity/>
                    <invsvc:state/>
                    <invsvc:place>
                        . . .
                        <invplace:service>
                            . . .
                            <invsvc:party>
                                . . .
                                <invparty:service>
                                    . . .
                                    <invsvc:configuration/>
                                </invparty:service>
                            </invsvc:party>
                            <invsvc:configuration/>
                        </invplace:service>
                    </invsvc:place>
                    <invsvc:party>
                        . . .
                        <invparty:service>
                            . . .
                            <invsvc:place>
                                . . .
                                <invplace:service>
                                    . . .
                                    <invsvc:configuration/>
                                </invplace:service>
                            </invsvc:place>
                            <invsvc:configuration/>
                        </invparty:service>
                    </invsvc:party>
                    <invsvc:configuration/>
                </invbi:service>
                <invbi:parameter>
                    <invbi:name/>
                    <invbi:value/>
                </invbi:parameter>
            </invbi:item>
            <invbi:interaction>
                </invbi:header>
                    <invbi:specification/>
                    <invbi:action/>
```


Copyright © 2014, 2026, Oracle and/or its affiliates.

```
61                      <invbi:id/>
62                      <invbi:name/>
63                      <invbi:externalIdentity/>
64                      <invbi:state/>
65                  </invbi:header>
66              </invbi:interaction>
67          </invbi:body>
68      </interaction>
69      <executeProcess/>
70      <responseLevel/>
71  </captureInteractionRequest>
```

Throughout Example 2-1, the <specification> element that is shown is actually a structure that defines the following elements:

Example 2-2 Specification Structure

```
<invbi:specification>
    <invent:entityNote/>
    <invspec:name/>
    <invspec:entityClass/>
    <invspec:description/>
    <invspec:startDate/>
    <invspec:endDate/>
</invbi:specification>
```

Within the specification structure, the <name> element is the name of a specification. This <name> element is not to be confused with the <name> element that is specified for the business interaction (line 07) or for the service (line 19). For example, a request that specifies the CREATE interaction action must supply the business interaction specification name (within the specification structure on line 04), and the name of the business interaction being created by the request (line 07). Similarly, a request that specifies the **add** service action must supply the Service specification name (within the specification structure on line 16), and the name of the service being created by the request (line 19).

Within the specification structure, the <entityClass> element is defined as an enumeration in the **Specification.xsd** schema file. The enumeration values reflect UIM entity specification types, such as BusinessInteraction, Service, Equipment, and so forth. The Service Fulfillment Web Service does not use the <entityClass> element, so the request does not need to specify it.

Business Interaction

captureInteractionRequest captures one interaction per request (lines 02 through 68). For each interaction, the request captures one or more items (lines 12 through 56), and one or more child interactions (lines 57 through 66).

When calling CaptureInteraction, the request must specify an interaction action (line 05) of CREATE or CHANGE. The interaction <action> element is defined as an enumeration in the **BusinessInteraction.xsd** schema file.

If the interaction action is CREATE, the request must provide an arbitrary name for the business interaction (line 07) being created, and the business interaction specification name (within line 04) upon which the business interaction is being based. (The specification name is typically **Service Order**, which is the business interaction specification provided in the **ora_uim_basespecifications** cartridge.) The request can optionally provide an external ID for the business interaction. You do not need to provide the specification entityClass enumeration value of **BusinessInteraction**; this is assumed based on the placement of the specification structure within the <interaction> element.

Copyright © 2014, 2026, Oracle and/or its affiliates.

If the interaction action is CHANGE, the request must provide the external ID (within line 08) or the business interaction ID (line 06) to indicate the business interaction to change, and the actual changes.

Business Interaction Item

captureInteractionRequest captures one or more items per interaction. Example 2-1 shows just one item (lines 12 through 56). To include multiple items, replicate the item and place it between lines 56 and 57.

Each item defines an interaction action (line 14), which must be **ADD** regardless of the request.

> **Note**
>
> The interaction action must be **ADD**. It cannot be another action value and it cannot be left blank. If the interaction action is not **ADD**, the operation errors.

Business Interaction Item Parameter

Each business interaction item optionally specifies one or more input parameters. For these parameters, the complex type ParameterType is defined as the following:

```
<xsd:element maxOccurs="unbounded" minOccurs="0"
    name="parameter" type="invbi:ParameterType">
</xsd:element>
```

The value element definition within ParameterType is the following:

```
<xs:element name="value" type="xsd:anyType">
```

For the value element with the anyType declaration, you can use any valid XML schema type that provides the following:

- An entity in UIM
- A type of StructuredType, defined in the business schema file **Structure.xsd**

Use the StructuredType complex type to pass entities with multiple property values. Example 2-3 provides sample XML for using StructuredType with one level of property information.

Example 2-3 Parameter Sample using StructuredType with One Level

```
<invbi:parameter>
    <invbi:name>StructuredType Parameter</invbi:name>
    <invbi:value xsi:type="invstruc:StructuredType">
        <invstruc:name>CPE Device 1</invstruc:name>
        <invstruc:property>
        <invprop:name>CPE_MAC</invprop:name>
        <invprop:value xsi:type="xs:string">01-23-45-67-89-ab</invprop:value>
        </invstruc:property>
        <invstruc:property>
        <invprop:name>CPE_Model</invprop:name>
        <invprop:value xsi:type="xs:string">PBS</invprop:value>
        </invstruc:property>
        <invstruc:property>
        <invprop:name>CPE_Brand</invprop:name>
```


Copyright © 2014, 2026, Oracle and/or its affiliates.

```
        <invprop:value xsi:type="xs:string">Motorola</invprop:value>
        </invstruc:property>
        <invstruc:property>
        <invprop:name>CPE_SerialNumber</invprop:name>
        <invprop:value xsi:type="xs:string">4TUI-632552</invprop:value>
        </invstruc:property>
     </invbi:value>
</invbi:parameter>
```

If you have properties with hierarchical information, you create the parameter list with the StructuredType including a hierarchy. Example 2-4 provides sample XML for using StructuredType including a child element representing hierarchical property information.

Example 2-4 Parameter Sample using StructuredType with Hierarchy

```
<invbi:parameter>
    <invbi:name>Structured Param </invbi:name>
    <invbi:value xsi:type="invstruc:StructuredType">
        <invstruc:name>CPE Device</invstruc:name>
        <invstruc:property>
        <invprop:name>CPE_MAC</invprop:name>
        <invprop:value xsi:type="xs:string">1.2.3.4</invprop:value>
        </invstruc:property>
        <invstruc:property>
        <invprop:name>CPE_Model</invprop:name>
        <invprop:value xsi:type="xs:string">MI6</invprop:value>
        </invstruc:property>
        <invstruc:property>
        <invprop:name>CPE_Brand</invprop:name>
        <invprop:value xsi:type="xs:string">Motorola</invprop:value>
        </invstruc:property>
        <invstruc:property>
        <invprop:name>CPE_SerialNumber</invprop:name>
        <invprop:value xsi:type="xs:string">838373723</invprop:value>
        </invstruc:property>
        <invstruc:child xsi:type="invstruc:StructuredType">
            <invstruc:name>Channel Pack</invstruc:name>
            <invstruc:property>
            <invprop:name>CPE_MAC</invprop:name>
            <invprop:value xsi:type="xs:string">1.2.3.5</invprop:value>
            </invstruc:property>
            <invstruc:property>
            <invprop:name>code</invprop:name>
            <invprop:value xsi:type="xs:string">code value A</invprop:value>
            </invstruc:property>
        </invstruc:child>
    </invbi:value>
</invbi:parameter>
```

Service

Each item defines a service, as shown in Example 2-1 (lines 15 through 51). For each service, you must supply a valid Service specification name (within line 16) from which to create an instance of the specification in UIM.

Copyright © 2014, 2026, Oracle and/or its affiliates.

> **Note**
>
> The <service> element (line 15) is actually defined as a choice in the **BusinessInteraction.xsd** schema file, with the choices being service, connectivity, and entity. However, the service choice is the only choice you can use in the request. The connectivity choice is not supported by the Service Fulfillment Web Service, and the entity choice is not used in the request; it is only used in the response.

Each service also defines a service action (line 18). The service action is not an enumeration, as are the interaction action and item action. Rather, there are several predefined service actions that UIM code recognizes. You can extend the list of service actions and their corresponding processes through custom rulesets. Service actions are further explored in "Customizing ProcessInteraction".

Specifically, there are some service actions that the core code recognizes. Through a custom ruleset, you can extend the input service actions and map each custom service action to one of the service actions that the core code recognizes.

The service may also specify a place (lines 22 through 35) or a party (lines 36 through 49) to associate to the service.

The request and response use the same structure. Most of the elements are used only by the response, so there are numerous elements that are not used by the request. For example, a service and configuration for the place (lines 24 through 34), a service and configuration for the party (lines 38 through 48), and the configuration for the service itself (line 50).

Associated Business Interaction

captureInteractionRequest captures one or more child interactions per interaction. Example 2-1 shows just one child interaction (lines 57 through 66). To include multiple interactions, replicate the child interaction (lines 57 through 66) and place it between lines 66 and 67.

ExecuteProcess Element

The <executeProcess> element (line 69) is defined after the interaction and applies to the interaction. This element is defined as a Boolean and is used only by CaptureInteraction. When the value of <executeProcess> is **true**, CaptureInteraction executes and, upon completion, ProcessInteraction executes. This eliminates the need to place two separate web service calls; one for CaptureInteraction and one for ProcessInteraction. When the value of <executeProcess> is **false**, just CaptureInteraction executes. The default value is **false**.

ResponseLevel Element

captureInteractionRequest, processInteractionRequest, getInteractionRequest, and updateInteractionRequest define the <responseLevel> element (line 70). This element specifies an enumeration value, as defined by the **InteractionResponseLevelEnum** enumeration in the **InteractionMessages.xsd** schema file.

Depending on the enumeration value specified in the request, the level of information returned by the response can vary:

- INTERACTION

  Returns just the interaction information.

- INTERACTION_ITEM

Copyright © 2014, 2026, Oracle and/or its affiliates.

Returns the interaction and item information.

- INTERACTION_ITEM_ENTITY

  Returns the interaction, item, and entity information.

- INTERACTION_ITEM_ENTITY_CONFIGURATION (default option)

  Returns the interaction, item, entity, and configuration information.

- INTERACTION_ITEM_ENTITY_CONFIGURATION_EXPANDED

  Returns the interaction, item, entity, configuration, and any child configurations.

captureInteractionResponse

captureInteractionResponse returns a varying level of information based on the <responseLevel> value the request specifies. See "ResponseLevel Element" for more information.

captureInteractionResponse always includes the business interaction ID and the current business interaction state. If a new business interaction is created, the business interaction ID generated by UIM is returned. If an existing business interaction is changed, the business interaction ID sent with the request is returned. The valid business interaction states are CREATED, IN_PROGRESS, COMPLETED, or CANCELLED, as defined by an enumeration in the **BusinessInteraction.xsd** schema file.

captureInteractionResponse returns an error when:

- The request specifies an interaction action of CREATE with a business interaction ID that already exists.
- The request specifies an interaction action of CHANGE with a business interaction ID (or external ID) that does not exist.
- An optional extension point is used to validate the input, and the associated ruleset logs an error. For example, the XML input does not validate.

ProcessInteraction

The ProcessInteraction operation moves planned services into current inventory. The planned service is represented by the <interaction> element, which is stored in UIM as a business interaction attachment, having been placed there by CaptureInteraction.

ProcessInteraction retrieves the business interaction and the attachment with the highest sequence number and, based on the items defined for the interaction, processes each item. Each item creates or updates a service, including any default service configuration items defined by the specified Service Configuration specification.

ProcessInteraction also calls the following methods per service configuration item:

- BusinessInteractionManager.getEntityAction()
- BaseConfigurationManager.automateConfiguration()

The **ora_uim_baseextpts** cartridge provides base extension points for both of these methods, which you can use to run custom code that maps custom service actions and that creates or updates service configuration items. This topic is further explored in "ProcessInteraction Logic Flow" and "Customizing ProcessInteraction".

When calling ProcessInteraction, the request must specify the external ID or the business interaction ID to indicate the business interaction to process.

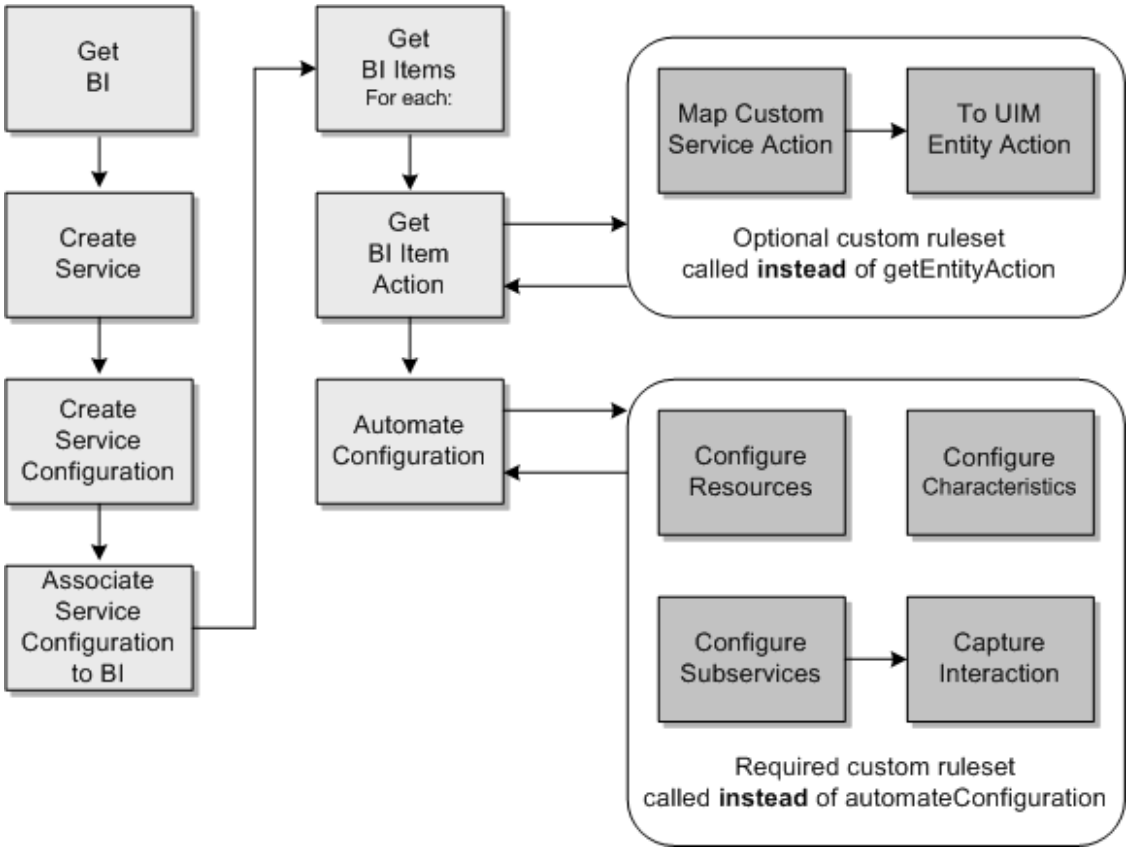
Copyright © 2014, 2026, Oracle and/or its affiliates.

The request can specify whether to process the entire business interaction, or just specific business interaction items. If the request specifies the external ID or business interaction ID only, the entire business interaction is processed; if the request specifies the external ID or business interaction ID and specific business interaction items, only the specified business interaction items are processed.

ProcessInteraction Logic Flow

Figure 2-4 shows what occurs when processInteractionRequest specifies a CREATE interaction action. A business interaction is represented as BI in the figure.

Figure 2-4 ProcessInteraction Logic Flow



In Figure 2-4, the light gray boxes represent the work performed by ProcessInteraction, prior to calling the custom ruleset. ProcessInteraction handles the processing of the business interaction. The dark gray boxes represent the work performed by custom code, which handles:

- Mapping custom service actions to UIM entity actions (optional)

  If your implementation uses only existing service actions, this custom code is not needed; if your implementation defines additional custom service actions, this custom code is required.

- Processing business interaction items (required)

  The processing of the business interaction items involves customizations that are necessary to meet the business requirements of providing the specific type of service. This

Copyright © 2014, 2026, Oracle and/or its affiliates.

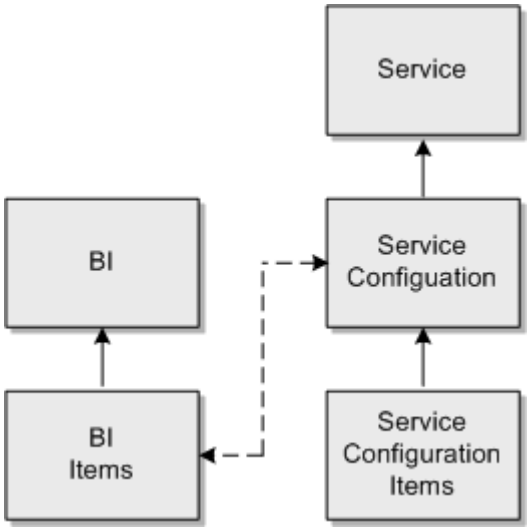
custom code must process service actions, custom service actions, and custom parameters, and calls the appropriate UIM API methods to create the service in UIM.

Service Configuration Association

Regarding the **Associate Service Configuration to BI** box in Figure 2-4: A service configuration is indirectly associated to a business interaction through the business interaction items. This association is shown by the dotted line in Figure 2-5. To associate the service configuration to the business interaction, ProcessInteraction:

- Creates business interaction items based on the items for the interaction in the request
- Associates the business interaction items to the service configuration

Figure 2-5 Association of Service Configuration to BI



Customizing ProcessInteraction

Customizing ProcessInteraction involves the following:

- Modeling the Service in Design Studio
- Customizing Service Actions
- Customizing the Automation of Service Configurations

An example of customizing ProcessInteraction is described in "ProcessInteraction Example".

Modeling the Service in Design Studio

Before you begin customizing ProcessInteraction, you must model your service within an Inventory project in Design Studio. For example, your Inventory project must define specifications that describe your service, service configuration, and service configuration items that fulfill the service in UIM.

See "About Unified Inventory Management" in *UIM Concepts* for information about services and service configurations, and see "*SCD Design Studio Modeling Inventory*" for information about modeling services.

Copyright © 2014, 2026, Oracle and/or its affiliates.

Customizing Service Actions

captureInteractionRequest defines a service action for each service in the request. The service action is not an enumeration; rather, there are several predefined actions (called entity actions) that UIM code recognizes and processes.

The UIM-defined entity actions are:

- create
- change
- delete
- disconnect
- suspend
- resume
- no_action

> **Note**
>
> The **no_action** service action prevents the creation of new service configurations as a part of process interaction flow in UIM. This service action merges the changes into existing service configurations that are in progress. For example, you can use this service action in Revision orders where the service is still in **Pending** and the existing service configurations should be updated than creating new ones.

The web service recognizes two additional entity actions that enable the web service to perform additional functionality. They are:

- suspendWithConfiguration
- resumeWithConfiguration

For example, the suspend entity action suspends a service but does not touch the service configuration. The suspendWithConfiguration entity action suspends a service and creates a new service configuration version. Similarly, the resume entity action resumes a suspended service but does not modify the service configuration. The resumeWithConfiguration entity action resumes a suspended a service and creates a new service configuration version. (For either action, if an existing service configuration version does not exist, an error is thrown because the service configuration must already exist if you are suspending or resuming it.)

> **Note**
>
> All of the entity actions are case sensitive.

Customizations are based on the service action (Example 2-1, line 18) and parameters (Example 2-1, lines 52 through 55). So, you must establish a finite list of service actions and parameters that can be specified in the request, which can then be recognized and processed by the custom code.

To customize service actions:

1. Determine the finite list of service actions and parameters to process.

Copyright © 2014, 2026, Oracle and/or its affiliates.

If your finite list of service actions includes only UIM-defined entity actions (and no custom service actions), you do not need to perform this procedure.

2. In Design Studio, open the Project editor.

   This is the same project that contains the specifications you created to model your service.

3. Click the **Dependency** tab.
4. Add the **ora_uim_base_extpts** cartridge to the list of dependencies.
5. Save the project.
6. Create a ruleset.

   Write custom code that maps your custom service actions to entity actions. The custom code can be in the ruleset, or in Java code that the ruleset calls. For information about writing custom rulesets, see "Overview" in *UIM Developer's Guide*.

7. Save the ruleset.
8. Create a ruleset extension point and configure it as follows:
   a. In **Ruleset**, select your ruleset.
   b. In **Point**, select the **BusinessInteractionManager_getEntityAction** base extension point.
   c. In **Placement**, select **INSTEAD**.
   d. Save the ruleset extension point.
9. Open your Service specification.
10. Click the **Rules** tab.
11. Click **Select**.
12. Select your ruleset extension point.
13. Click **OK**.
14. Save the Service specification.
15. Build the project.
16. Deploy the resultant cartridge.

Customizing the Automation of Service Configurations

To customize the automation of service configurations:

1. In Design Studio, open the Project editor.

   This is the same project that contains the specifications that you created to model your service.

2. Click the **Dependency** tab.
3. Add the **ora_uim_base_extpts** cartridge to the list of dependencies.
4. Save the project.
5. Create a ruleset.

   Write custom code that processes the business interaction items, evaluates the mapped entity actions and custom parameters, and calls the appropriate API methods to create the service in UIM. See "Developing the Custom Code" for more information.

Copyright © 2014, 2026, Oracle and/or its affiliates.

The custom code can be in the ruleset, or in Java code that the ruleset calls. For information about writing custom rulesets, see "Overview" in *UIM Developer's Guide*.

6. Save the ruleset.
7. Create a ruleset extension point and configure it as follows:
   a. In **Ruleset**, select your ruleset.
   b. In **Point**, select the **BaseConfigurationManager_automateConfiguration** base extension point.
   c. In **Placement**, select **INSTEAD**.
   d. Save the ruleset extension point.
8. Open your Service Configuration specification.
9. Click the **Rules** tab.
10. Click **Select**.
11. Select your ruleset extension point.
12. Click **OK**.
13. Save the Service Configuration specification.
14. Build the project.
15. Deploy the resultant cartridge.

Developing the Custom Code

ProcessInteraction triggers events that result in a call to custom code that automates service configurations by calling API methods to fulfill the service in UIM.

See "Overview" in *UIM API Overview* for code examples that show how to call the UIM API methods from within custom code.

The following information pertains to the custom code:

- The custom code must handle and process the XML payload based on the domain-specific business rules and models.
- The custom code must handle the creation or deletion of any dependent resources.
- The custom code must handle auto-design for new orders and auto-redesign for change orders.
- The custom code can assume the service and service configuration are already created; the purpose of the custom code is to manage the resources and characteristics.
- When modifying a subservice with parent input only:

  The business interaction attachment typically may not contain specific change request information for a subservice that was created when fulfilling the requested service. For example, a voice mail service created by UIM to fulfill the request for a Mobile GSM service with a voice mail feature. In this scenario, the voice mail service is a subservice assigned to the Mobile GSM service. When the subservice requires a change, the change request and service action are often submitted for the parent service, and not for the subservice.

  In such scenarios, the web service operation has to identify that the **change** service action is for the subservice, and process the change for the subservice. For example, if the custom code needs to act on a subservice, it can build a request based on the subservice,

Copyright © 2014, 2026, Oracle and/or its affiliates.

call CaptureInteraction, and recursively call ProcessInteraction until it returns the **no action** entity action.

ProcessInteraction Example

The following list describes some project content your implementation may require to run ProcessInteraction.

- Numerous custom specifications and characteristics

  At a minimum, your project needs to define a Service specification, a Service Configuration specification, and resource-specific specifications, such as Telephone Number, Physical Device, Logical Device, and so forth. Your project may also require characteristics in which to store resource-specific data.

- AUTOMATE_MY_CONFIGURATION.ruleset

  This is a custom ruleset that is the entry point into the custom code. The ruleset calls the AutomateMyConfiguration() method, which is defined in a custom Java class. In this example, the custom Java class is named MyConfigurationManagerImpl.java, which is also described in this list.

- AUTOMATE_MY_CONFIGURATION_EXT.rst

  This is a custom ruleset extension point that associates the AUTOMATE_MY_CONFIGURATION custom ruleset to the UIM-provided BaseConfigurationManager_automateConfiguration extension point and configures the custom ruleset to run **instead** of the method that the extension point defines (the BaseConfigurationManager.automateConfiguration() method).

- MAP_MY_SERVICE_ACTION.ruleset

  If you defined custom service actions, this is a custom ruleset that evaluates custom service actions specified in the request and maps them to an entity action that is recognizable to UIM. In this example, there are five custom service actions, so this ruleset evaluates the five custom service actions and maps each one to the appropriate entity action. The entity actions are defined in the Service Fulfillment Web Service code, as described in "Customizing ProcessInteraction".

  Table 2-1 provides an example of mapping custom service actions to UIM entity actions.

Table 2-1 Example Mapping of Custom Service Actions

Custom Service Action	UIM Entity Action
createMyService	create
updateMyService	change
changeAddToMyService	change
disconnectMyService	disconnect
suspendMyService	suspend

- MAP_MY_SERVICE_ACTION_EXT.rst

  This is a custom ruleset extension point that associates the MAP_MY_SERVICE_ACTION custom ruleset to the UIM-provided BusinessInteractionManager_getEntityAction extension point and configures the custom ruleset to run **instead** of the method that the extension point defines (BusinessInteractionManager.getEntityAction() method).

- MyConfigurationManagerImpl.java

Copyright © 2014, 2026, Oracle and/or its affiliates.

This is custom Java code that contains a series of **if else** statements that evaluate the mapped entity action. For each entity action, the code calls another method within the same class. Within each of these methods, the finite set of parameters that are valid for the specific service action that was mapped to the entity action is evaluated.

From there, the custom code calls various API methods to perform the work required to realize any service in UIM.

- Any additional custom rulesets and ruleset extension points

  When the custom code calls API methods, the existing API functionality may need to be extended to realize a service in UIM. So, your project may also have to define any needed rulesets that can be configured to run **before** or **after** the API methods that the custom code calls.

processInteractionResponse

processInteractionResponse returns a varying level of information based on the <responseLevel> value the request specifies. See "ResponseLevel Element" for more information.

ProcessInteraction returns an error when:

- It cannot find the business interaction specified by the calling system.
- The calling system specifies an input item entity other than Service.
- Any errors thrown by the custom code that ProcessInteraction calls.

GetInteraction

The GetInteraction operation retrieves a business interaction based on an external ID or business interaction ID. The data returned in the response depends on when GetInteraction is called and on the <responseLevel> value getInteractionRequest specifies.

When GetInteraction is called before ProcessInteraction, the response returns only the business interaction data. In this scenario, service data is not returned because ProcessInteraction has not yet processed the business interaction into current inventory, so there is no service data in UIM yet.

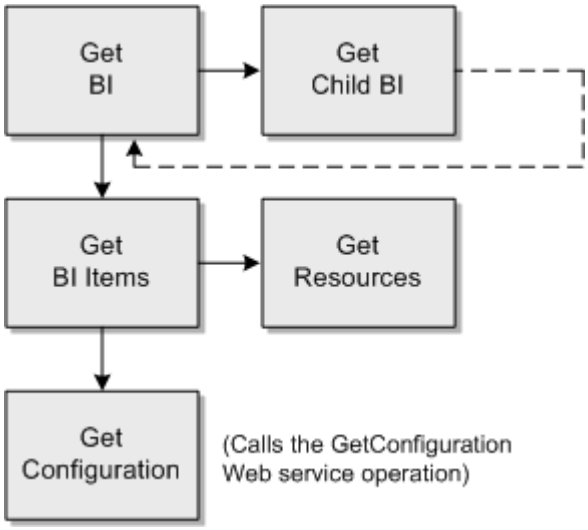
When GetInteraction is called after ProcessInteraction, the response returns the business interaction data and service data. In this scenario, service data is returned because ProcessInteraction has processed the business interaction into current inventory, so there is service data in UIM to retrieve. The level of detail of service data returned by the response depends on the <responseLevel> value getInteractionRequest specifies. See "ResponseLevel Element" for more information.

GetInteraction Logic Flow

Figure 2-6 shows what occurs when the GetInteraction operation is called. A business interaction is represented as BI in the figure.

Copyright © 2014, 2026, Oracle and/or its affiliates.

Figure 2-6 GetInteraction Logic Flow



getInteractionResponse

getInteractionResponse returns a varying level of information based on when the operation is called and on the <responseLevel> value the request specifies. See "ResponseLevel Element" for more information.

GetInteraction returns an error when:

- The request does not specify an external ID or business interaction ID upon which to base the retrieval
- It cannot find the business interaction specified in the request

UpdateInteraction

The UpdateInteraction operation transitions UIM business entities through their respective life-cycle states within the context of a business interaction.

When calling UpdateInteraction, the request must specify an external ID or business interaction ID and an interaction action of **APPROVE**, **ISSUE**, **CANCEL**, or **COMPLETE**. If you want to change the effective date of the configuration version, the request must specify an effectiveDate and an interaction action of **CHANGE**.

Interaction actions are defined by the BusinessInteractionActionEnum enumeration in the **BusinessInteraction.xsd** schema file. This enumeration defines several actions, but only the **APPROVE**, **ISSUE**, **CANCEL**, **COMPLETE**, or **CHANGE** actions are valid for UpdateInteraction.

UpdateInteraction uses the business interaction ID to find the service and service configuration, and performs the specified action for the service and service configuration. For example, if the interaction action is **APPROVE**, it approves the service and service configuration associated to the business interaction and performs the action recursively to any child business interactions.

UpdateInteraction does not cascade to child entities assigned to configuration items. For example, if the business interaction is associated to a service configuration, and the service configuration has a service configuration item with a child service assigned to it,

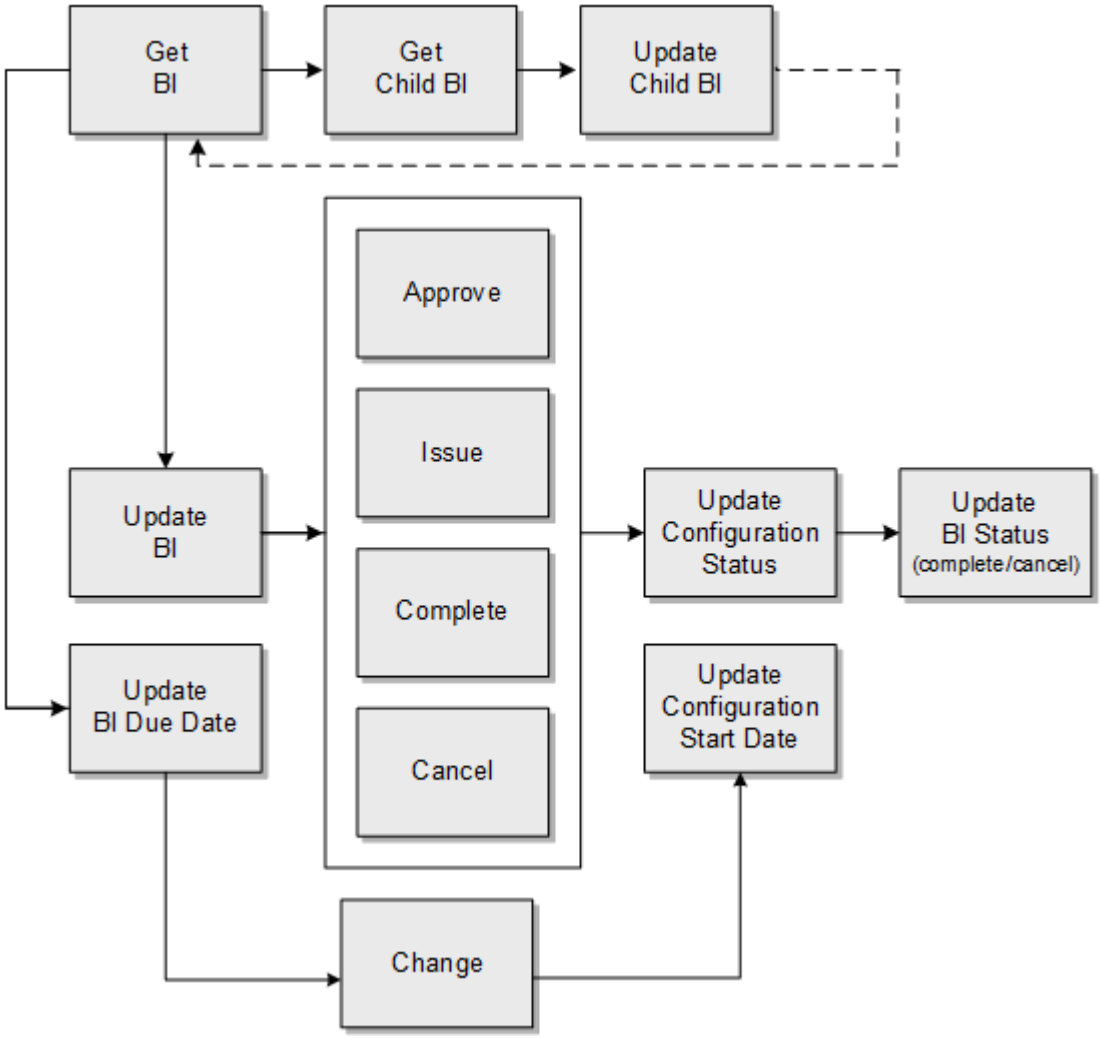
Copyright © 2014, 2026, Oracle and/or its affiliates.

UpdateInteraction does not apply the action to the service configuration item child service status.

UpdateInteraction Logic Flow

Figure 2-7 shows what occurs when the UpdateInteraction operation is called. A business interaction is represented as BI in the figure.

Figure 2-7 UpdateInteraction Logic Flow



updateInteractionResponse

updateInteractionResponse returns a varying level of information based on the <responseLevel> value the request specifies. See "ResponseLevel Element" for more information.

UpdateInteraction returns an error when:

- It cannot find the business interaction specified by the calling system
- The request specifies a value for <item> other than <service>

Copyright © 2014, 2026, Oracle and/or its affiliates.

GetConfiguration

The GetConfiguration operation retrieves one of the following, based on a search option specified in the request:

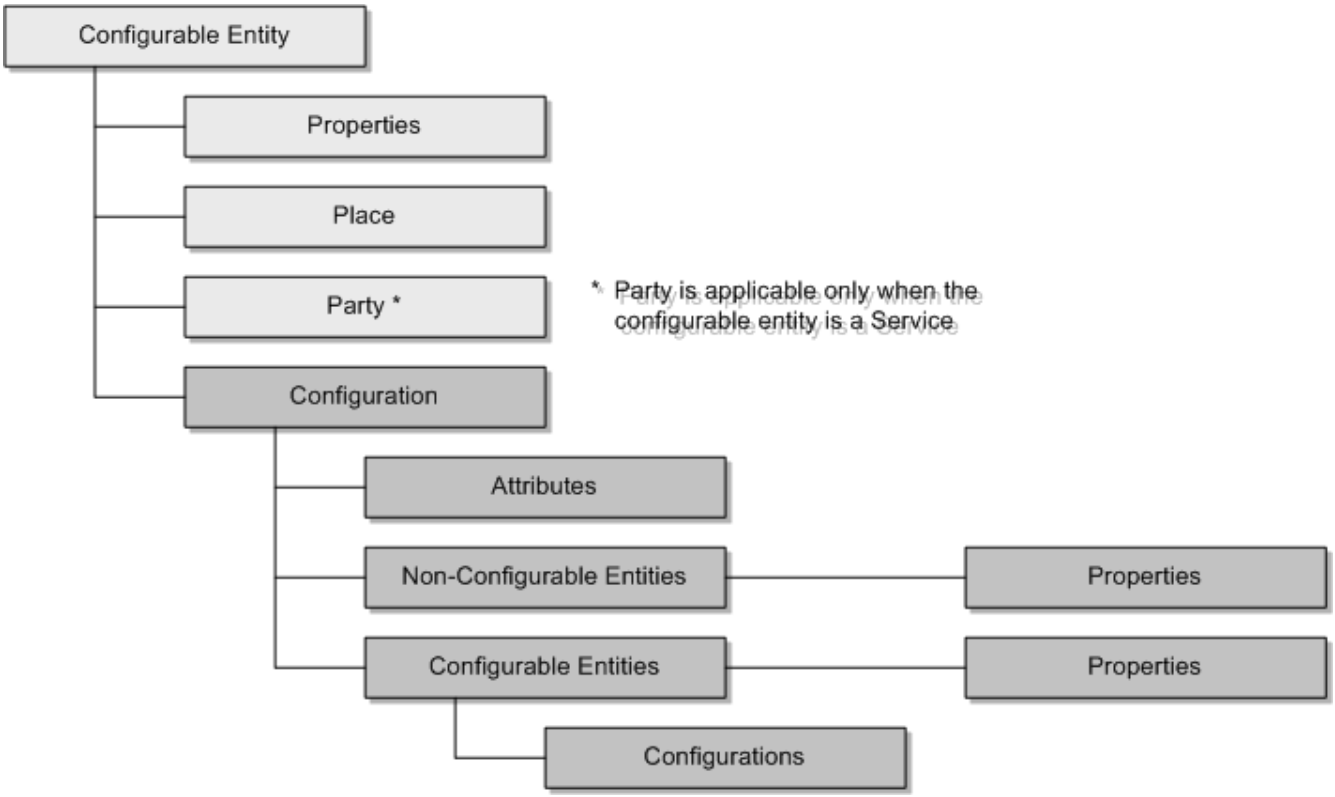
- Service Configuration
- Pipe Configuration (representing a pipe or channelized connectivity)
- Logical Device Configuration
- Logical Device Account Configuration
- Network Configuration
- Place Configuration

A successful response returns the following for the specified configuration:

- Configuration
- Configuration properties (attributes and characteristics)
- Configuration items (including any assigned or referenced resources)
- Child entities and their child configurations

For example, Figure 2-8 shows the data that can be retrieved, in dark gray, for a requested configuration. GetConfiguration does not retrieve information about relationships to other entities. However, you can customize GetConfiguration to return additional information. See "Customizing GetConfiguration" for more information.

Figure 2-8 Example Service Configuration Retrieval



Copyright © 2014, 2026, Oracle and/or its affiliates.

> **Note**
>
> A configurable place is actually a GeographicSite specialization of the abstract Place entity; GeographicSite is the only specialization of the Place entity that is configurable. See *Oracle Communications Information Model Reference* for more information.

getConfigurationRequest

This section describes getConfigurationRequest, in which you specify a search option that tells GetConfiguration which type of configuration to return. In the request, you can also specify additional request options that tell GetConfiguration what data to include in the response, or to omit from the response.

Request Search Options

In getConfigurationRequest, you specify a search option that indicates the type of configuration to retrieve. The search options, which are defined in the **ConfigurationMessages.xsd** schema file, are listed and described in Table 2-2.

> **Note**
>
> For each search option listed in Table 2-2, *Entity* represents:
>
> - DeviceInterface
> - Connectivity
> - LogicalDevice
> - LogicalDeviceAccount
> - Network
> - Place
> - Service

Table 2-2 GetConfiguration Search Options

Search Option	Description
*Entity*ConfigurationSearchByConfigId	GetConfiguration retrieves the configuration based on the specified configuration ID.
*Entity*ConfigurationSearchByEntityId	GetConfiguration retrieves the latest active configuration (any state other than CANCELLED) based on the specified entity ID. If there is only one configuration, GetConfiguration retrieves it.
*Entity*ConfigurationSearchByVersionNumber	GetConfiguration retrieves the configuration based on the specified entity ID and version number.
*Entity*ConfigurationSearchByConfigStatus	GetConfiguration retrieves the latest configuration based on the entity ID and configuration state. States can be IN_PROGRESS, DESIGNED, ISSUED, COMPLETED, PENDING_CANCEL, or CANCELLED.
*Entity*ConfigurationSearchByEffectiveDate	GetConfiguration retrieves the configuration based on the specified entity ID and configuration effective date.

Copyright © 2014, 2026, Oracle and/or its affiliates.

Table 2-2 (Cont.) GetConfiguration Search Options

Search Option	Description
ConnectivityConfigurationSearchByConnectivityIdentifier	GetConfiguration retrieves the latest active pipe configuration (any state other than CANCELLED) based on the specified connectivity identifier. If there is only one pipe configuration, GetConfiguration retrieves it. This search option is applicable only when getting pipe configurations.

Request Search Option Examples

Example 2-5 shows getConfigurationRequest with a search option of ServiceConfigurationSearchByConfigId in bold. The element below the search option shows the configuration ID to search for.

Example 2-5 getConfigurationRequest

```
<con:getConfigurationRequest>
    <responseLevel>ENTITY_CONFIGURATION_EXPANDED</responseLevel>
    <con:searchOptions xsi:type="con:GetServiceConfigurationType"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <con:configSearchOption
        xsi:type="con:ServiceConfigurationSearchByConfigId">
            <con:configurationId>123456</con:configurationId>
        </con:configSearchOption>
    </con:searchOptions>
</con:getConfigurationRequest>
```

Example 2-6 shows getConfigurationRequest with a search option of ConnectivityConfigurationSearchByVersionNumber in bold. The elements below the search option show the entity ID and configuration version number to search for.

Example 2-6 getConfigurationRequest

```
<con:getConfigurationRequest>
    <responseLevel>ENTITY_CONFIGURATION_EXPANDED</responseLevel>
    <con:searchOptions xsi:type="con:GetConnectivityConfigurationType"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <con:configSearchOption
        xsi:type="con:ConnectivityConfigurationSearchByVersionNumber">
            <con:entityId>1</con:entityId>
            <con:versionNumber>1</con:versionNumber>
        </con:configSearchOption>
    </con:searchOptions>
</con:getConfigurationRequest>
```

Additional Request Options

In getConfigurationRequest, you can also specify the following options. If an option is not specified in the request, the operation uses the default value.

- includeTags
  - When set to **true**, the response includes tags on configuration items, with the tag name and description populated.
  - When set to **false** (the default), the response does not include tags on configuration items.

  For information about tags, see "*SCD Design Studio Modeling Inventory*".

Copyright © 2014, 2026, Oracle and/or its affiliates.

- includeTagsOtherInfo
  - When set to **true**, the tag name, description, and other information is populated.
  - When set to **false** (the default), the tag name and description are populated; other information is not populated.

> **Note**
>
> The includeTagsOtherInfo option is only applicable when includeTags is **true**.

For information about tags, see "*SCD Design Studio Modeling Inventory*".

- includeNetworkTargets
  - When set to **true**, the response includes network targets.
  - When set to **false** (the default), the response does not include network targets.

For information about network targets, see "About Unified Inventory Management" in *UIM Concepts*.

Additional Request Options Example

Example 2-7 shows getConfigurationRequest with the additional request options.

Example 2-7 getConfigurationRequest

```
<con:getConfigurationRequest>
    <com:header></com:header>
    <con:searchOptions
    .
    .
    .
    </con:searchOptions>
    <con:includeTags>true</con:includeTags>
    <con:includeTagsOtherInfo>true</con:includeTagsOtherInfo>
</con:getConfigurationRequest>
```

ResponseLevel Element

getConfigurationRequest and updateConfigurationRequest define the <responseLevel> element. This element specifies an enumeration value, as defined by the **ConfigurationResponseLevelEnum** enumeration in the **ConfigurationMessages.xsd** schema file. (This element does not apply to getConfigurationDifferencesRequest.)

Depending on the enumeration value specified in the request, the level of information returned by the response can vary:

- ENTITY_CONFIGURATION (default option)

  Returns the entity and configuration information.

- ENTITY_CONFIGURATION_EXPANDED

  Returns the entity, configuration, and any child configurations.

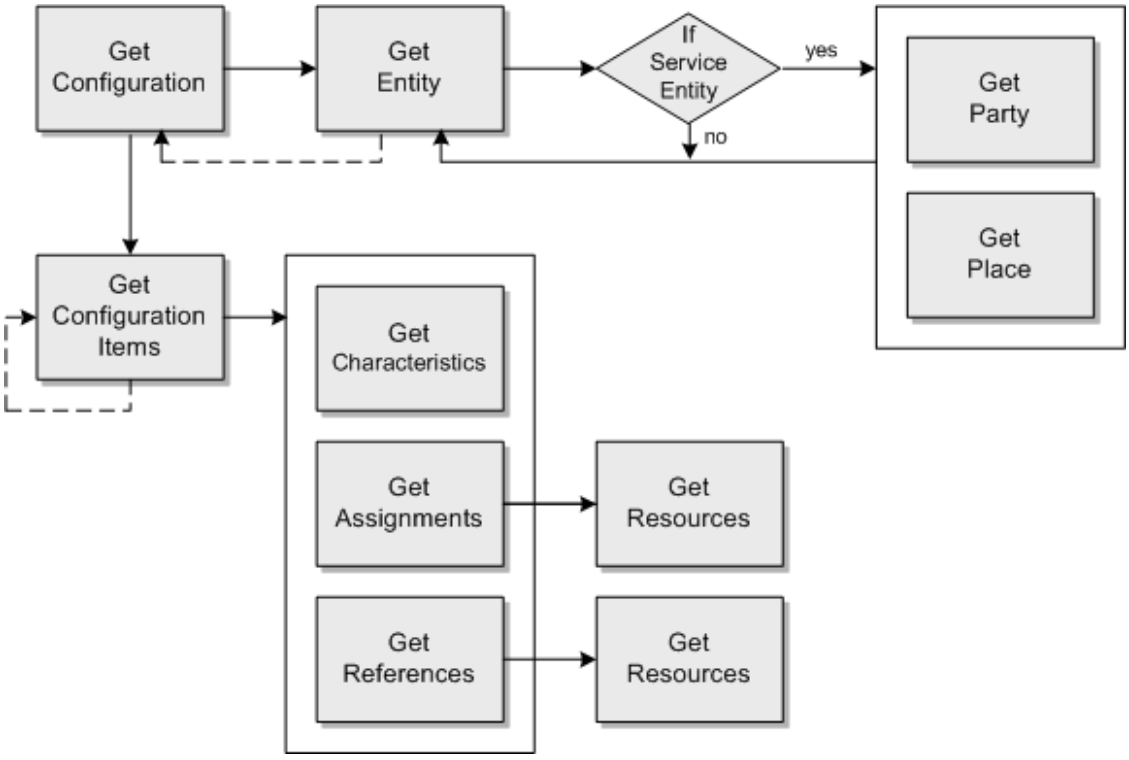
GetConfiguration Logic Flow

Figure 2-9 shows what occurs when the GetConfiguration operation is called.

Copyright © 2014, 2026, Oracle and/or its affiliates.

Depending on which search option is specified, the logic flow may start with the **Get Configuration** box, or it may start with the **Get Entity** box. For example, if the search option is *Entity*ConfigurationSearchByConfigId, the entry point to the logic flow is the **Get Configuration** box. If the search option is any other option, which are all based on an entity ID, the entry point to the logic flow is the **Get Entity** box.

Figure 2-9 GetConfiguration Logic Flow



getConfigurationResponse

getConfigurationResponse returns a varying level of information based on the <responseLevel> value the request specifies. See "ResponseLevel Element" for more information.

GetConfiguration returns an error when:

- The request specifies a search option other than the valid search options listed in Table 2-2.
- The request does not specify the data that the search option needs to perform the search.
- The operation cannot find the configuration ID, entity ID, or connectivity identifier specified in the request.

Copyright © 2014, 2026, Oracle and/or its affiliates.

Customizing GetConfiguration

> **Note**
>
> This section describes the use of rulesets and extension points to customize GetConfiguration. See "Overview" in *UIM Developer's Guide* for detailed information about rulesets and extension points.

For your implementation, you may need more information than GetConfiguration returns. For example, GetConfiguration does not retrieve information about relationships to other entities. If you have an assigned entity that is a physical device, you may want GetConfiguration to return the mapped logical device or some of its characteristics. Or, for a device interface, you may want GetConfiguration to return the mapped physical port. For such scenarios, you can customize GetConfiguration to return additional information.

Extension Points

The *UIM_SDK_Home***/cartridges/base/ora_uim_baseextpts** cartridge provides the following specification-based extension points for customizing GetConfiguration:

- BaseConfigurationManager_populateCustomProperties.rstp, which defines the following method signature:

```
public Map<String, String>
populateCustomProperties(ConsumableResource resource, InventoryConfigurationItem
item,
InventoryConfigurationVersion inventoryConfigurationVersion)
```

- BaseConfigurationManager_populateCustomProperties2.rstp, which defines the following method signature:

```
public Map<String, String>
populateCustomProperties(ConfigurationReferenceEnabled entity,
InventoryConfigurationItem item,
InventoryConfigurationVersion inventoryConfigurationVersion)
```

GetConfiguration always calls the populateCustomProperties() methods, which are empty methods that exist for customizing GetConfiguration. Within a configuration, each configuration item represents a resource, which may be assigned or referenced. The populateCustomProperties() methods define different inputs; a consumable resource entity versus a reference-enabled entity. The former method is called during the process of retrieving assigned resources, and the latter method is called during the process of retrieving referenced resources.

Customization Steps

To customize GetConfiguration:

1. Create a ruleset to retrieve any additional assigned resource data that your implementation requires. The ruleset must return a Map containing a name/value pair of the retrieved data name and corresponding data value.
2. Create a ruleset extension point to configure your ruleset to run **after** the populateCustomProperites() method, using the BaseConfigurationManager_populateCustomProperties.rstp extension point.

Copyright © 2014, 2026, Oracle and/or its affiliates.

3. Create a ruleset to retrieve any additional referenced resource data that your implementation requires. The ruleset must return a Map containing a name/value pair of the retrieved data name and corresponding data value.
4. Create a ruleset extension point to configure your ruleset to run **after** the populateCustomProperites() method, using the BaseConfigurationManager_populateCustomProperties2.rstp extension point.
5. Configure any applicable specifications with the appropriate ruleset extension point. (The base extension points are specification-based, not global.)
6. Deploy the cartridge or cartridges containing the ruleset, ruleset extension points, and specifications.
7. Call GetConfiguration.

   GetConfiguration calls the populateCustomProperties() methods, and your rulesets run afterward, populating the customProperty element in the response. See "Customized Response" for more information.

Customized Response

Example 2-8 shows an excerpt from the **Configuration.xsd** file, which defines the customProperty element.

Example 2-8 customProperty

```
<xs:element name="customProperty" type="invprop:PropertyType" nillable="true"
minOccurs="0" maxOccurs="unbounded">
    <xs:annotation>
        <xs:documentation>
            Custom Properties added for the entity Assignment/Reference.
        </xs:documentation>
    </xs:annotation>
</xs:element>
```

Example 2-9 shows an excerpt from the **Property.xsd** file, which defines the PropertyType structure. (The customProperty element references PropertyType in its definition.)

Example 2-9 PropertyType

```
<xs:complexType name="PropertyType">
    <xs:annotation>
        <xs:documentation>PropertyType holds a single dynamic property as a
        name-value pair.</xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:element name="name" type="xs:string" />
        <xs:element name="value" type="xs:string" nillable="true" />
        <xs:element name="action" type="invent:EntityActionEnum" minOccurs="0">
            <xs:annotation>
                <xs:documentation>
                    Action holds the property Action which indicates whether the
                    property needs to be added/deleted/updated.
                    Valid values for this element are defined by EntityActionEnum.
                </xs:documentation>
            </xs:annotation>
        </xs:element>
    </xs:sequence>
</xs:complexType>
```

Example 2-10 shows the structure that ends up in the response when customizations are in place.

Copyright © 2014, 2026, Oracle and/or its affiliates.

Example 2-10 Response

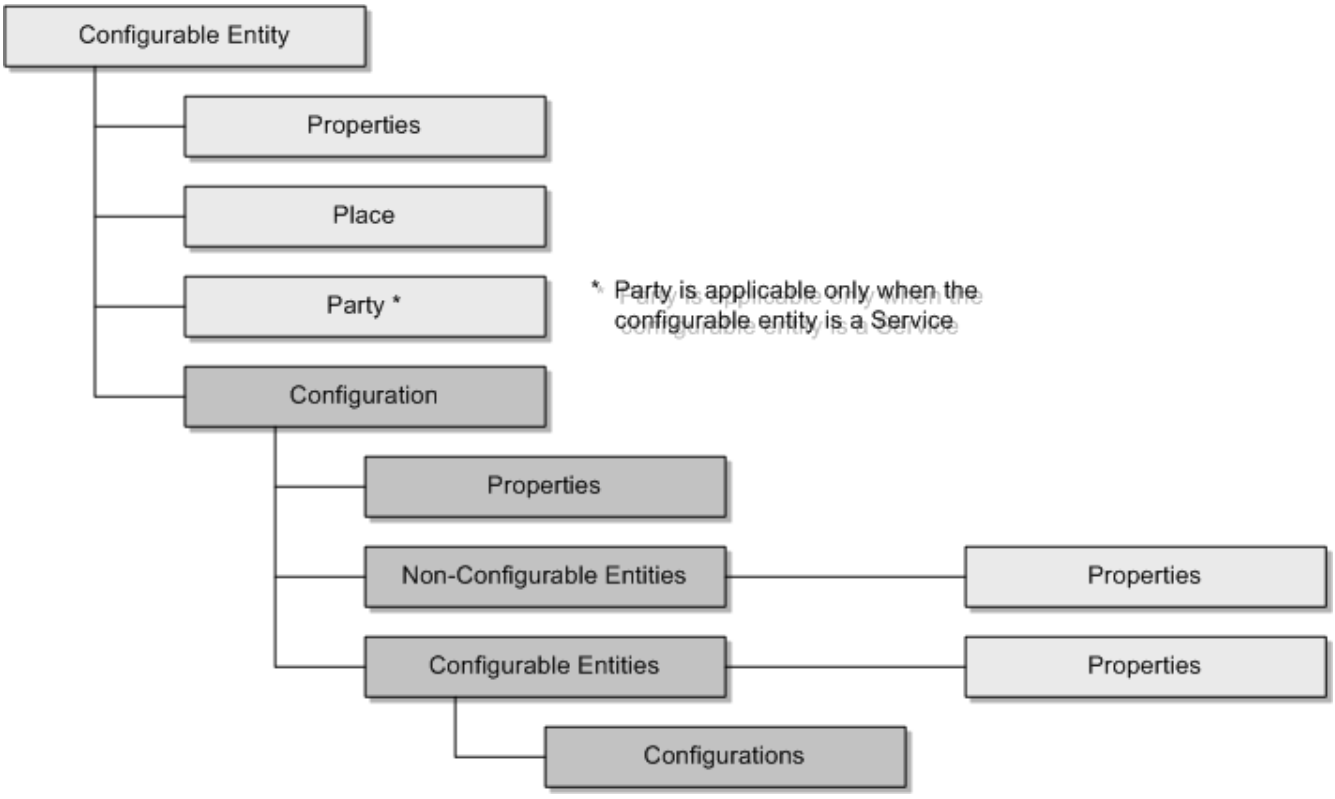
```
<con:customProperty>
    <invprop:name>customDataName1</invprop:name>
    <invprop:value>customDataValue1</invprop:value>
    <invprop:action><invprop:action>
</con:customProperty>
<con:customProperty>
    <invprop:name>customDataName2</invprop:name>
    <invprop:value>customDataValue2</invprop:value>
    <invprop:action><action>
</con:customProperty>
<con:customProperty>
    <invprop:name>customDataName3</invprop:name>
    <invprop:value>customDataValue3</invprop:value>
    <invprop:action><invprop:action>
</con:customProperty>
```

GetConfigurationDifferences

The GetConfigurationDifferences operation compares two versions of a service, pipe (representing a pipe or channelized connectivity), logical device, logical device account, network, or place configuration and returns the differences. The type of configuration compared is based on a search option specified in the request. A successful response returns the differences between properties (attributes and characteristics); differences between configuration items, including any assigned or referenced resources; and the differences between any child entities and their child configurations. For example, Figure 2-10 shows the data that is retrieved and compared, in dark gray, for a requested configuration (service, logical device, logical device account, network, pipe (representing a pipe or channelized connectivity), or place) comparison. You can customize GetConfigurationDifferences to return additional information. See "Customizing GetConfigurationDifferences" for more information.

Copyright © 2014, 2026, Oracle and/or its affiliates.

Figure 2-10 Example Service Configuration Differences



> **Note**
>
> GetConfigurationDifferences calls GetConfiguration, which returns the properties (attributes and characteristics) for all resources, but GetConfigurationDifferences does not compare the returned properties for the resources.

getConfigurationDifferencesRequest

This section describes getConfigurationDifferencesRequest, in which you specify a search option that tells GetConfigurationDifferences which type of configuration versions to compare and return. In the request, you can also specify additional request options that tell GetConfigurationDifferences what data to include in the response, or to omit from the response.

Request Search Options

In getConfigurationDifferencesRequest, you specify a search option that indicates the type of configuration versions. The search options, which are defined in the **ConfigurationMessages.xsd** schema file, are listed and described in Table 2-3.

Copyright © 2014, 2026, Oracle and/or its affiliates.

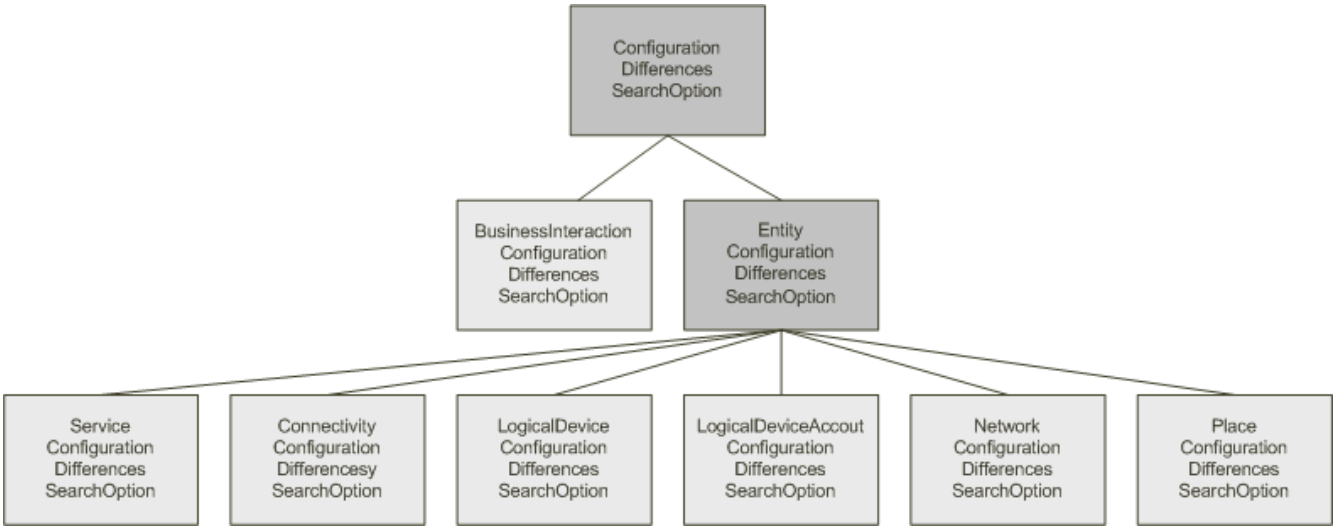
Table 2-3 GetConfigurationDifferences Search Options

Search Option	Description
BusinessInteractionConfigurationDifferencesSearchOption	GetConfigurationDifferences finds the differences between two configuration versions associated with this business interaction, based on the specified business interaction ID or external ID representing a business interaction. This search option also finds the differences between two configuration versions associated with any child business interactions.
ServiceConfigurationDifferencesSearchOption	GetConfigurationDifferences finds the differences between two configuration versions associated with this service, based on the specified service ID or external ID representing a service. This search option also finds the differences between two configuration versions associated with any child services.
ConnectivityConfigurationDifferencesSearchOption	GetConfigurationDifferences finds the differences between two configuration versions associated with this pipe, based on the specified pipe ID (representing a pipe or channelized connectivity). This search option does not find the configuration differences for child pipes.
LogicalDeviceConfigurationDifferencesSearchOption	GetConfigurationDifferences finds the differences between two configuration versions associated with this logical device, based on the specified logical device ID. This search option does not find the configuration differences for child logical devices because logical devices cannot have a parent/child relationship.
LogicalDeviceAccountConfigurationDifferencesSearchOption	GetConfigurationDifferences finds the differences between two configuration versions associated with this logical device account, based on the specified logical device account ID. This search option does not find the configuration differences for child logical device accounts because logical device accounts cannot have a parent/child relationship.
NetworkConfigurationDifferencesSearchOption	GetConfigurationDifferences finds the differences between two configuration versions associated with this network, based on the specified network ID. This search option does not find the configuration differences for child networks because networks cannot have a parent/child relationship.
PlaceConfigurationDifferencesSearchOption	GetConfigurationDifferences finds the differences between two configuration versions associated with this place, based on the specified place ID. This search option also finds the differences between two configuration versions associated with any child places.

All search options inherit from the abstract ConfigurationDifferencesSearchOption, and all entity-specific search options inherit from the abstract EntityConfigurationDifferencesSearchOption, as shown in Figure 2-11.

Copyright © 2014, 2026, Oracle and/or its affiliates.

Figure 2-11 Search Options



The request must specify one of the following:

- Business interaction ID
- Entity ID
- External ID for a business interaction
- External ID for a service entity

In the above list, the entity ID can be for a service, pipe (representing a pipe or channelized connectivity), logical device, logical device account, network, or place entity. However, the external ID for an entity can only be for a service entity.

Target and Source Configuration Versions

ConfigurationDifferencesEntitySearchOption, from which all search options inherit, defines the following:

- Target configuration version
- Source configuration version

The target and source configuration versions indicate the configuration versions to compare. The target configuration is the root of the comparison. Depending on what is specified in the request, the operation does the following:

- If the request specifies both the target and source configuration versions, the operation compares the two configuration versions.
- If the request specifies only a target configuration version, the operation compares the specified target configuration version to a defaulted source configuration version. In this scenario, the source configuration version defaults to the latest non-cancelled configuration version that precedes the specified target configuration version.
- If the request specifies neither, the operation compares a defaulted target configuration version to a defaulted source configuration version. In this scenario, the target configuration version defaults to the latest non-cancelled configuration version, and the source configuration version defaults to the latest non-cancelled configuration version that precedes the defaulted target configuration version.

Copyright © 2014, 2026, Oracle and/or its affiliates.

- If the request specifies only a source configuration version (which is not recommended), the operation compares a defaulted target configuration version to the specified source configuration version. In this scenario, the target configuration defaults to the latest non-cancelled configuration version that follows the specified source configuration version, if it exists. If the operation is unable to default the target configuration version, the response returns an error.

Request Search Option Examples

Example 2-11 shows getConfigurationDifferencesRequest with a search option of BusinessInteractionConfigurationDifferencesSearchOption in bold. The elements below the search option show the specified business interaction ID to search for.

This request example also shows how you can specify an external ID for a business interaction to search for. In the example, these elements are commented out because you can only specify one or the other when using this search option.

Example 2-11 getConfigurationDifferencesRequest

```
<con:getConfigurationDifferencesRequest>
    <com:header/>
    <con:searchOptions
    xsi:type="con:BusinessInteractionConfigurationDifferencesSearchOption"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <con:interaction xsi:type="bus:BusinessInteractionType">
            <bus:header>
                <bus:id>123456</bus:id>
                <!--
                <bus:externalIdentity xsi:type="invent:ExternalIdentityType">
                    <invent:externalObjectId>9876543</invent:externalObjectId>
                </bus:externalIdentity>
                -->
            </bus:header>
        </con:interaction>
    </con:searchOptions>
</con:getConfigurationDifferencesRequest>
```

Example 2-12 shows getConfigurationDifferencesRequest with a search option of ServiceConfigurationDifferencesSearchOption in bold. The elements below the search option show the specified service ID to search for. In this example, where the search option inherits from the EntityConfigurationDifferencesSearchOption, the request also specifies a source configuration version and a target configuration version.

Example 2-12 getConfigurationDifferencesRequest

```
<con:getConfigurationDifferencesRequest>
    <com:header></com:header>
    <con:searchOptions
    xsi:type="con:ServiceConfigurationDifferencesSearchOption"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <con:service xsi:type = "ser:ServiceCriteriaType">
            <ser:serviceId>2468</ser:serviceId>
        </con:service>
        <con:targetConfigurationCriteriaType
        xsi:type="conf:ConfigurationCriteriaType">
            <conf:configurationVersionNumber>2</conf:configurationVersion
            Number>
        </con:targetConfigurationCriteriaType>
        <con:sourceConfigurationCriteriaType
        xsi:type="conf:ConfigurationCriteriaType">
            <conf:configurationVersionNumber>1</conf:configurationVersion
```


Copyright © 2014, 2026, Oracle and/or its affiliates.

```
            Number>
        </con:sourceConfigurationCriteriaType>
    </con:searchOptions>
</con:getConfigurationDifferencesRequest>
```

Additional Request Options

In getConfigurationDifferencesRequest, you can also specify the following options. If an option is not specified in the request, the operation uses the default value.

- includeConfigItemDifferences
  - When set to **true**, the response includes changes on the configuration item itself. For example, adding or removing configuration items.
  - When set to **false** (the default), the response includes only changes on the resources.
- includeChildConfigDifferences
  - When set to **true**, the response includes differences for child configurations referenced or assigned to a configuration item on a parent configuration. For example, when a resource-facing service (RFS) is assigned on a customer-facing service (CFS).
  - When set to **false** (the default), the response does not include these differences.
- includeActionPerformedInTargetVersion
  - When set to **true**, the response includes the actionPerformedInTargetVersion element within the target configuration item differences. The actionPerformedInTargetVersion element is a Boolean value in the response; when **true** is returned, it indicates the action taken was performed in the target version; when **false** is returned, it indicates the action taken was performed in the source version.

    For example, when comparing version 1 (source) and version 4 (target), where versions 2 and 3 are not cancelled, and resource A is assigned in version 1: If resource A is modified in version 3, actionPerformedInTargetVersion is set to false; but if resource A is modified in version 4 (the version to compare), actionPerformedInTargetVersion is set to true.
  - When set to **false** (the default), the response does not include the actionPerformedInTargetVersion element within the target configuration item differences.
- includeTarget
  - When set to **true** (the default), the response includes the target configuration.
  - When set to **false**, the response does not include the target configuration.

    > **Note**
    >
    > The includeTarget option takes precedence over the returnTargetWhenNoChange option. For example, if returnTarget is **true** and returnTargetWhenNoChange is **false**, and the configurations versions being compared are the same, the response includes the target configuration.
- includeSource
  - When set to **true** (the default), the response includes the source configuration.
  - When set to **false**, the response does not include the source configuration.
- returnTargetWhenNoChange

Copyright © 2014, 2026, Oracle and/or its affiliates.

  - When set to **true**, and the versions being compared are the same, the response returns an action of **Unchanged** and populates the target configuration; the source configuration is not populated.
  - When set to **false** (the default), and the versions being compared are the same, the response returns an action of **Unchanged** and populates neither configuration (target or source).

- includeTags
  - When set to **true**, the response includes tags on configuration items, with the tag name and description populated.
  - When set to **false** (the default), the response does not include tags on configuration items.

  For information about tags, see the Design Studio Help.

- includeTagsOtherInfo
  - When set to **true**, tag name, description, and other information are populated.
  - When set to **false** (the default), tag name and description are populated; other information is not populated.

> **Note**
>
> The includeTagsOtherInfo option is only applicable when includeTags is **true**.

  For information about tags, see the Design Studio Help.

- includeNetworkTargets
  - When set to **true**, the response includes network targets.
  - When set to **false** (the default), the response does not include network targets.

  For information about network targets, see *UIM Concepts*.

Additional Request Options Example

Example 2-13 shows getConfigurationRequest with the additional request options.

Example 2-13 getConfigurationDifferencesRequest

```
<con:getConfigurationDifferencesRequest>
    <com:header></com:header>
    <con:searchOptions>
    .
    .
    .
    </con:searchOptions>
    <con:includeConfigItemDifferences>true</con:includeConfigItemDifferences>
    <con:includeChildConfigDifferences>true</con:includeChildConfigDifferences>
    <con:includeActionPerformedInTargetVersion>true</
con:includeActionPerformedInTargetVersion>
    <con:includeTarget>true</con:includeTarget>
    <con:includeSource>true</con:includeSource>
    <con:returnTargetWhenNoChange>true</con:returnTargetWhenNoChange>
    <con:includeTags>true</con:includeTags>
    <con:includeTagsOtherInfo>true</con:includeTagsOtherInfo>
</con:getConfigurationDifferencesRequest>
```

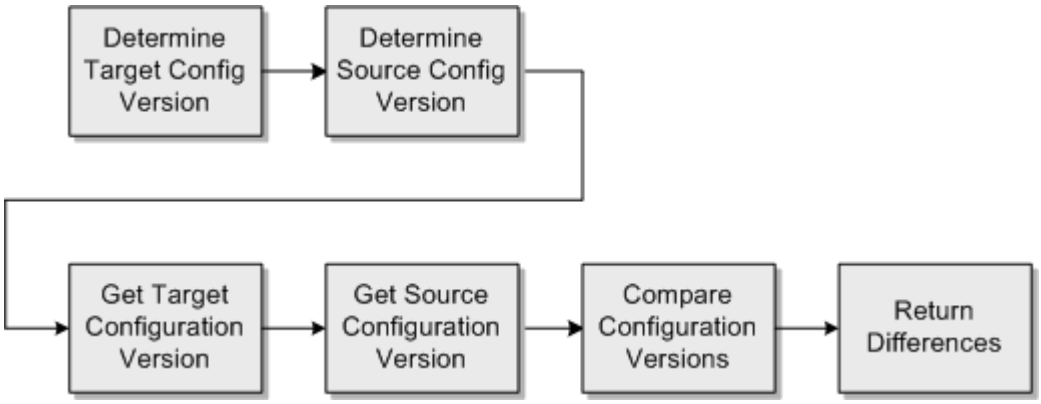

Copyright © 2014, 2026, Oracle and/or its affiliates.

GetConfigurationDifferences Logic Flow

Figure 2-12 shows what occurs when the GetConfigurationDifferences operation is called.

The first two boxes in Figure 2-12 represent functionality that is only performed when applicable, depending on what is specified in the request. For example, if the request specifies a target configuration version, the logic flow would start with **Determine Source Configuration Version**, and if the request specifies a target configuration version and a source configuration version, the logic flow would start with **Get Source Configuration Version**.

Figure 2-12 GetConfigurationDifferences Logic Flow



If the request specifies a business interaction ID or an external ID for a business interaction, the operation retrieves and compares any associated service configurations with their previous configuration version.

Child Configurations

If the configuration has a child configuration, and includeChildConfigDifferences is set to **true** in the request, the operation also compares two versions of the child configuration and returns those differences as well. See the description of includeChildConfigDifferences in "Additional Request Options" for more information.

All configurations have a start date and an end date. The operation determines which child configuration versions to compare based on the start and end dates of the child configurations and the start and end dates of the parent configurations.

Specifically, when comparing parent configuration versions P.3 and P.2, the operation determines which child configurations through the following process:

1. Find the first child configuration starting on or after the start date of parent configuration version P.3 and before the start date (if any) of the next configuration version (P.4). If not found, find the first child configuration version starting before the start date of parent configuration version P.3.
2. Do the same for parent configuration P.2:

   Find the first child configuration starting on or after the start date of parent configuration version P.2 and before the start date (if any) of the next configuration version (P.3). If not found, find the first child configuration version starting before the start date of parent configuration version P.2.

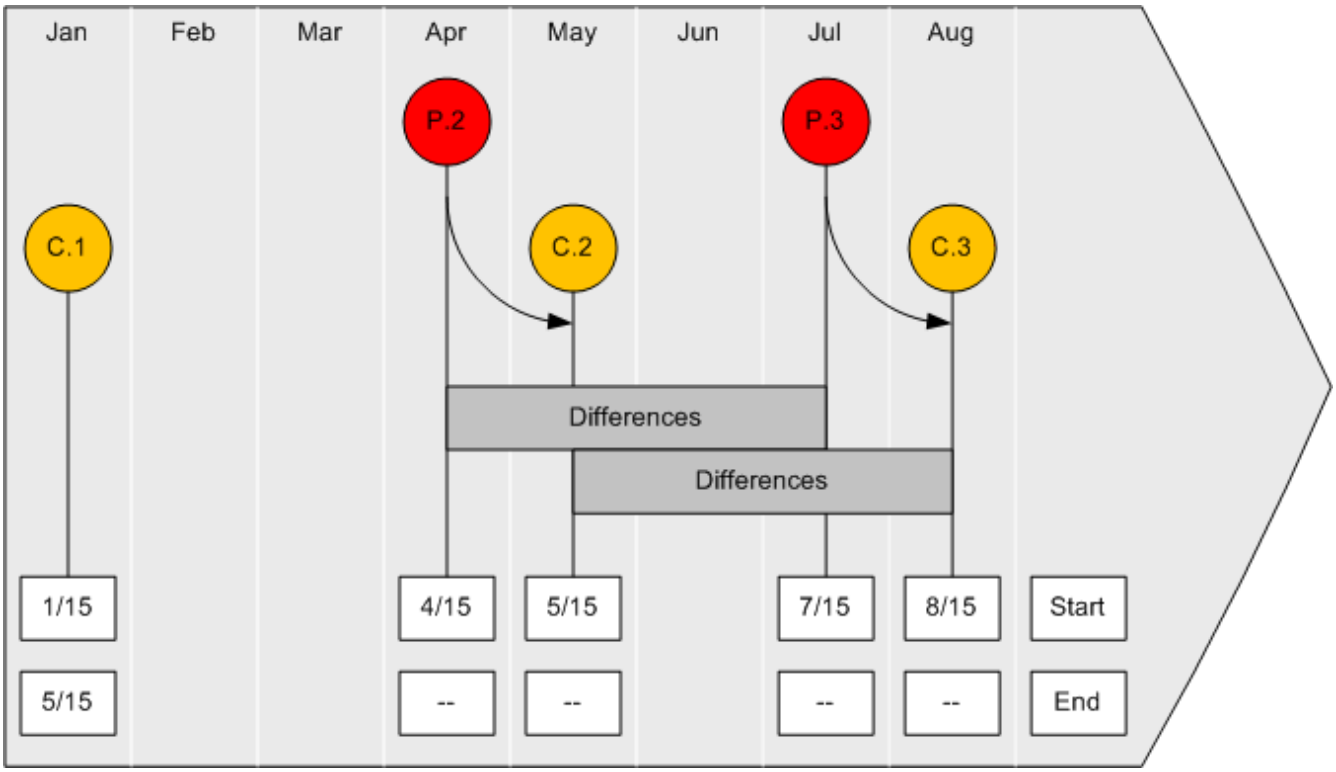
Copyright © 2014, 2026, Oracle and/or its affiliates.

The following examples show how the operation determines which child configuration versions to compare. In the examples, the parent configuration is represented as P, and the child configuration is represented as C. Versions of the parent configuration are represented as P.1, P.2, and P.3, and versions of the child configuration are represented as C.1, C.2, and C.3.

Example 1

Figure 2-13 shows an example where the start date and end date of the child configuration are on or after the parent configuration start date.

Figure 2-13 Child Start Date Is After Parent Start Date



In this example, the operation is comparing configuration versions P.2 and P.3. Configuration P is a parent to child configuration C, so the operation must determine which versions of child configuration C to compare.

The process:

1. Starts with parent configuration version P.3, which has a start date of 7/15.
2. Looks for a child configuration version with a start date that is on or after 7/15.
3. Finds child configuration version C.3, which has a start date of 8/15.
4. Proceeds to parent configuration version P.2, which has a start date of 4/15.
5. Looks for a child configuration version with a start date that is on or after 4/15.
6. Finds child configuration version C.2, which has a start date of 5/15.

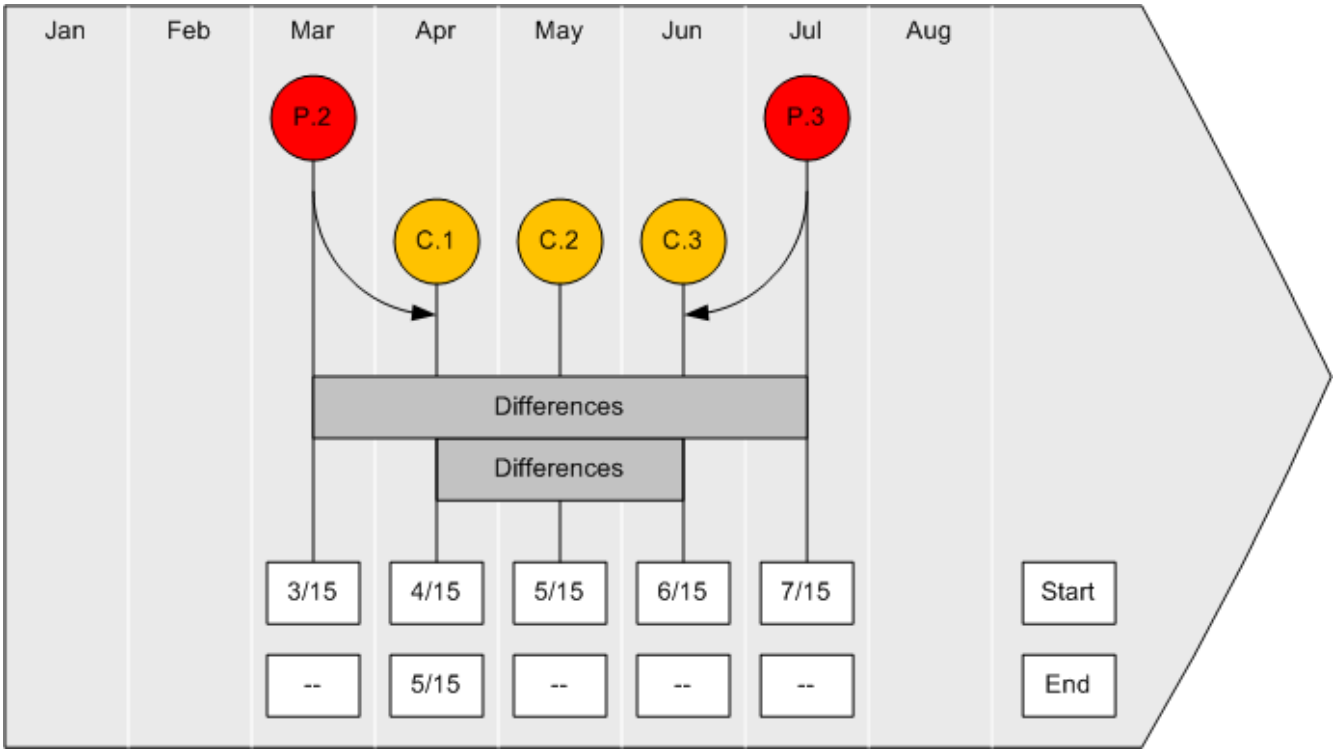
The operation determines that child configuration version C.3 is compared with child configuration version C.2.

Copyright © 2014, 2026, Oracle and/or its affiliates.

Example 2

In this example, the start dates and end dates of the child configurations are both before and after the parent's configuration start dates, as shown in Figure 2-14:

Figure 2-14 Child Start Date Before and After Parent Start Date



In this example, the operation is comparing configuration versions P.2 and P.3. Configuration P is a parent to child configuration C, so the operation must determine which versions of child configuration C to compare.

The process:

1. Starts with parent configuration version P.3, which has a start date of 7/15.
2. Looks for a child configuration version with a start date that is on or after 7/15, but does not find one.
3. Looks for a child configuration version with a start date that is before 7/15.
4. Finds child configuration version C.3, which has a start date of 6/15.
5. Proceeds to parent configuration version P.2, which has a start date of 3/15.
6. Looks for a child configuration version with a start date that is on or after 3/15.
7. Finds child configuration version C.1, which has a start date of 4/15.

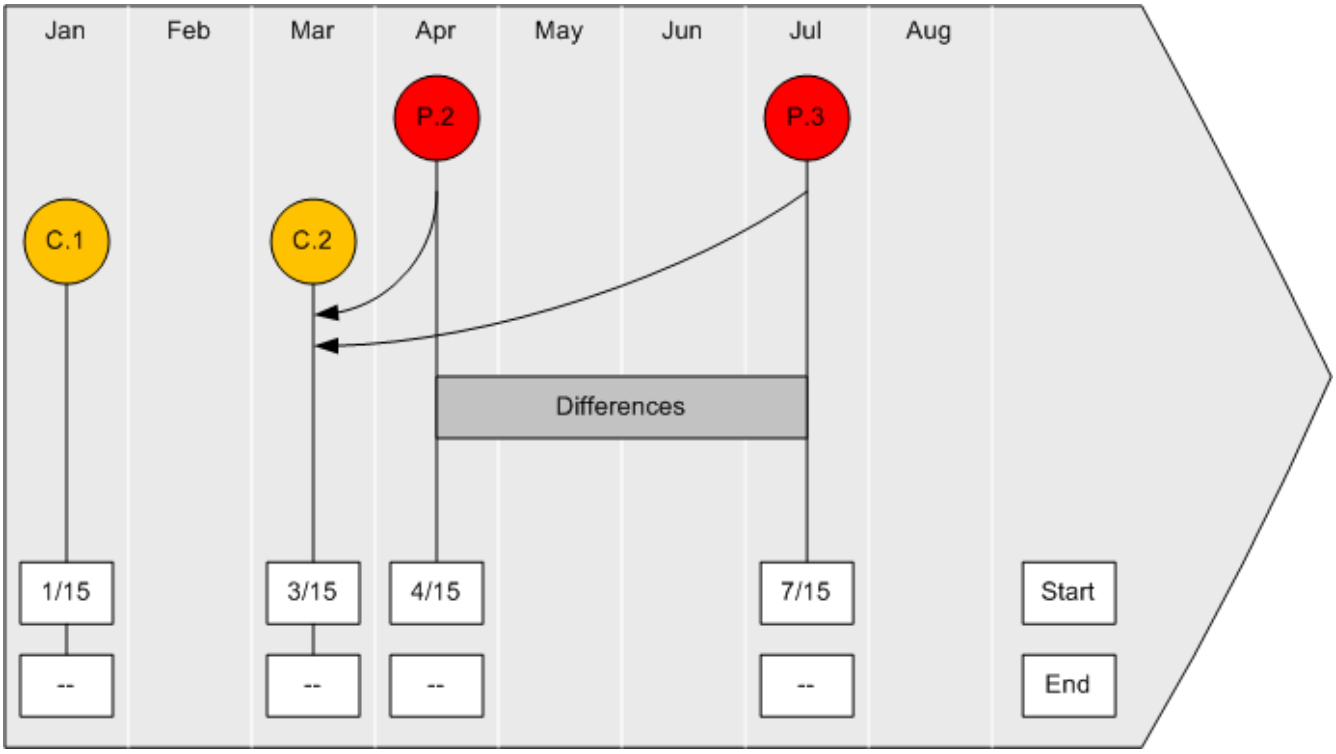
The operation determines that child configuration version C.3 is compared with child configuration version C.1.

Copyright © 2014, 2026, Oracle and/or its affiliates.

Example 3

In this example, the same child configuration version is applicable two versions of the parent configuration, as shown in Figure 2-15:

Figure 2-15 Same Child Configuration



In this example, the operation is comparing configuration versions P.2 and P.3. Configuration P is a parent to child configuration C, so the operation must determine which versions of child configuration C to compare.

The process:

1. Starts with parent configuration version P.3, which has a start date of 7/15.
2. Looks for a child configuration version with a start date that is on or after 7/15, but does not find one.
3. Looks for a child configuration version with a start date that is before 7/15.
4. Finds child configuration version C.2, which has a start date of 3/15.
5. Proceeds to parent configuration version P.2, which has a start date of 4/15.
6. Looks for a child configuration version with a start date that is on or after 4/15, but does not find one.
7. Looks for a child configuration version with a start date that is before 4/15.
8. Finds child configuration version C.2, which has a start date of 3/15.

The operation determines that child configuration version C.2 is compared with the same child configuration version. In this scenario, the response returns an action of **Unchanged** and may populate the target configuration, depending on what is specified in the request for the

Copyright © 2014, 2026, Oracle and/or its affiliates.

returnTargetWhenNoChange option. See the description of returnTargetWhenNoChange in "Additional Request Options" for more information.

Overriding the Process Logic that Determines Child Configurations

The process logic that determines the child configuration versions to compare resides in the BaseConfigurationManager.getEffectiveChildConfiguration() method, and Oracle provides a base extension point that defines this method.

You can override the logic by writing a custom ruleset that contains custom code that retrieves the child configuration versions based on your business requirements. You can then configure the custom ruleset to run at the provided base extension point, resulting in the custom ruleset running instead of the BaseConfigurationManager.getEffectiveChildConfiguration() method. See "Overview" in *UIM Developer's Guide* for more information about rulesets and extension points.

You can find the BaseConfigurationManager_getEffectiveChildConfiguration base extension point in the *UIM_SDK_Home***/cartridges/base/ora_uim_baseextpts.jar** file.

getConfigurationDifferencesResponse

getConfigurationDifferencesResponse returns a varying level of information based on the options specified in the request, as described in "Additional Request Options".

At a high-level, the response returns the following:

- Configuration differences []
  - Configuration item difference []
  - Target configuration
    - Configuration specification
    - Configuration item []
  - Source configuration
    - Configuration specification
    - Configuration item []

GetConfigurationDifferences returns an error when:

- The request specifies a search option other than the valid search options listed in Table 2-3.
- The request does not specify a business interaction ID, entity ID, external ID for a business interaction, or external ID for an entity that the search option needs to perform the search.
- The operation cannot find the business interaction ID, entity ID, external ID for the business interaction, or external ID for the service entity specified in the request.
- The request specifies a target configuration version and a source configuration version, and the source configuration version number is greater than the target configuration version number.
- The request specifies a source configuration version and the operation is unable to determine a target configuration version.
- The request specifies a target configuration version that is cancelled.
- The request specifies a source configuration version that is cancelled.

Copyright © 2014, 2026, Oracle and/or its affiliates.

GetConfigurationDifferences request is for configuration versions in the **Designed** or **Issued** state. For instance, the operation returns messages in the following scenarios:

- If the source or target configuration version requested is invalid, the following error message is given: “Invalid source and target versions. Source and target must be greater than 0, Source Version number cannot be greater than Target Version number. Source is 3 and Target is 0."
- If the configuration version requested is in the **In Progress**, **Cancelled, Completed**, or **Pending Cancel** state, for example, the following warning message is given and the operation continues processing: “Inventory Configuration 123 is in Completed state. This operation has been requested on a configuration version that is in the state that is not designed for this Web Service. Results may be inaccurate."
- The operation is intended for the requested configuration version to be compared to the previous **Completed** version. If no previous **Completed** configuration version exists, or no previous configuration version exists at all, the following message is given and the operation continues processing: “A previous configuration version does not exist."
- The operation is intended for the requested configuration version to be compared to the previous **Completed** version. If no previous **Completed** configuration version exists, or no previous configuration version exists at all, the following informational message is given and the operation continues processing: “A previous configuration version does not exist."

Customizing GetConfigurationDifferences

For your implementation, you may need more information than GetConfigurationDifferences returns. In such scenarios, you can customize GetConfigurationDifferences to return additional information.

Customizing GetConfigurationDifferences is similar to customizing GetConfiguration. See "Customizing GetConfiguration" for more information.

After you customize GetConfigurationDifferences to return any additional data you may need, your code that calls GetConfigurationDifferences needs to be customized to compare the versions that are returned in the response to determine the differences.

UpdateConfiguration

The UpdateConfiguration operation transitions a service or service configuration through its respective life-cycle states.

To transition a service, the request must specify the service action and service ID.

The valid service actions are:

- COMPLETE
- CANCEL
- DISCONNECT
- SUSPEND
- RESUME

To transition a service configuration, the request must specify the service configuration action and one of the following:

- Service ID
- Service configuration ID

Copyright © 2014, 2026, Oracle and/or its affiliates.

- Service ID and service configuration version number

If the first option is specified (service ID), the operation transitions the latest active service configuration.

The valid service configuration actions are:

- APPROVE
- ISSUE
- CANCEL
- COMPLETE

updateConfigurationResponse

updateConfigurationResponse includes a success or failure message regarding the update to transition the service or service configuration. The response returns a varying level of information based on the <responseLevel> value the request specifies. See "ResponseLevel Element" for more information.

UpdateConfiguration returns an error when:

- The request specifies an invalid service action or service configuration action.
- The request specifies invalid data for service ID, service configuration ID, or service configuration version number.

Customizing the Web Service Operations

You must customize the ProcessInteraction operation, and you can optionally customize the GetConfiguration and GetConfigurationDifferences operations. See the following sections for more information:

- Customizing ProcessInteraction
- Customizing GetConfiguration
- Customizing GetConfigurationDifferences

Extending Web Service Requests and Responses

You can extend web service requests and responses by extending **GenericHandler.class**, which supports the use of SOAP handlers and which is used by the UIM Service Fulfillment Web Service.

To extend a web service request or response:

1. In Design Studio, create a custom Inventory project.
2. Within your custom Inventory project, create a custom Java class that does the following:
   - Imports **javax.xml.rpc.handler.GenericHandler.class** (include the jaxrpc.jar if necessary)
   - Extends **GenericHandler**
   - Overrides the handleRequest() or handleResponse() methods, or both, per your specific business requirements
3. Build your custom Inventory project.

Copyright © 2014, 2026, Oracle and/or its affiliates.

A successful build of your custom Inventory project creates a deployable custom cartridge, which is a JAR file with the same name as your Inventory project.

4. Deploy your custom cartridge into your UIM traditional environment. If you use UIM cloud native, see "Deploying Cartridges" in *UIM Cloud Native Deployment Guide*.
5. Update the deployment plan:
   - For the UIM cloud native deployment, see "Customizing Images" in *UIM Cloud Native Deployment Guide*.
   - For the traditional deployment, update the *UIM_Home***/app/plan/Plan.xml** file to include the following:
     – Add the following <variable> elements under the <variable-definition> tag to define the variables of **HandlerName** and **HandlerClassName**, and to define their respective values, which is your custom Java class name and fully qualified custom Java class name:

```
<variable>
        <name>HandlerName</name>
        <value>MyCustomHandler</value>
</variable>
<variable>
        <name>HandlerClassName</name>
        <value>oracle.communications.webservice.ws.MyCustomHandler</value>
</variable>
```

     – Add the following <variable-assignment> elements under the <module-descriptor> element, as shown here:

```
<module-override>
    <module-name>InventoryWS.war</module-name>
    <module-type>war</module-type>
    ...
    ...
    <module-descriptor external="true">
        <root-element>webservices</root-element>
        <uri>WEB-INF/webservices.xml</uri>
<!-- ====================  START OF NEW CONTENT  ==================== -->
        <variable-assignment>
            <name>HandlerName</name>
            <xpath>
/webservices/webservice-description/
[webservice-description-name=
"oracle.communications.inventory.webservice.ws.InventoryWSPortImpl"]
/port-component/
[port-component-name="InventoryWSHTTPPort"]/handler/handler-name
            </xpath>
        </variable-assignment>
        <variable-assignment>
        <name>HandlerClassName</name>
            <xpath>
/webservices/webservice-description/
[webservice-description-name=
"oracle.communications.inventory.webservice.ws.InventoryWSPortImpl"]
/port-component/
[port-component-name="InventoryWSHTTPPort"]/handler/[handler-
name="MyCustomHandler"]/handler-class
            </xpath>
        </variable-assignment>
<!-- ====================  END OF NEW CONTENT  ==================== -->
    </module-descriptor>
    ...
```


Copyright © 2014, 2026, Oracle and/or its affiliates.

```
        ...
    </module-override>
```

- These additions to the **Plan.xml** file results in the following being added to the **webservice.xml** file at run-time:

```
<handler>
    <handler-name>MyCustomHandler</handler-name>
        <handler-class>oracle.communications.webservice.ws.MyCustomHandler
    </handler-class>
</handler>
```

6. In the UIM traditional deployment, redeploy the **inventory.ear** file.

   This action redeploys the UIM Service Fulfillment Web Service with the updated **Plan.xml** file.

   In the UIM cloud native deployment, rebuild the image with the modifications performed in the previous step. For details, see "Customizing Images" in *UIM Cloud Native Deployment Guide*. Also, create the instance with a generated image.

Additional Information

For more information about SOAP handlers, see "Creating and Using SOAP Message Handlers" in *Fusion Middleware Developing JAX-RPC Web Services for Oracle WebLogic Server*.

Deploying, Testing, and Securing the Web Service

Information about deploying, testing, and securing the web service is described in "Deploying, Testing, and Securing UIM Web Services".

Additional Request Options

In any SFWS request, you can optionally include the `decryptAttributes` parameter. This parameter controls whether encrypted characteristic values are returned as plain text (decrypted) or remain as cipher text (encrypted) in the response. If this parameter is not provided, all encrypted values will be returned as-is (still encrypted).

The supported values are:

- `ALL`: Use this option to decrypt all encrypted characteristics in the response. For example:

  ```
  <con:decryptAttributes>ALL</con:decryptAttributes>
  ```

- Comma-separated characteristic names: When you specify a comma-separated list of characteristic names in `decryptAttributes`, only the specified encrypted characteristics will be decrypted in the response. All other encrypted characteristics will remain as cipher text. For example:

  ```
  <con:decryptAttributes>char1, char2</con:decryptAttributes>
  ```

Sample Request:

```
<soapenv:Envelope
    xmlns:soapenv=" http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:con=" http://xmlns.<Oracle domain>/communications/inventory/
```


Copyright © 2014, 2026, Oracle and/or its affiliates.

```
webservice/configuration"
    xmlns:com=" http://xmlns.<Oracle domain>/communications/inventory/
webservice/common"
    xmlns:bus=" http://xmlns.<Oracle domain>/communications/inventory/
businessinteraction"
    xmlns:xsi=" http://www.w3.org/2001/XMLSchema-instance"
    >
    <soapenv:Body>
        <con:getConfigurationDifferencesRequest>
            <com:header/>

            <con:searchOptions
xsi:type="con:BusinessInteractionConfigurationDifferencesSearchOption">
                <con:interaction xsi:type="bus:BusinessInteractionType">
                    <bus:header>
                        <bus:id>225008</bus:id>
                    </bus:header>
                </con:interaction>
            </con:searchOptions>

            <con:includeConfigItemDifferences>true</
con:includeConfigItemDifferences>
            <con:includeChildConfigDifferences>true</
con:includeChildConfigDifferences>
            <con:includeTagsOtherInfo>true</con:includeTagsOtherInfo>
            <con:includeTags>true</con:includeTags>
            <con:decryptAttributes>ALL</con:decryptAttributes>
            </con:getConfigurationDifferencesRequest>
    </soapenv:Body>
</soapenv:Envelope>
```

Details of the newly added parameter `decyptAttributes` are as follows:

- When set to `ALL`, all encrypted characteristic values in the response are decrypted and returned.
- Alternatively, you can provide a comma-separated list of characteristic names. In this case, only the specified characteristics are decrypted and returned in the response.

Sample Response:

> **Note**
>
> Only a part of the response that returns characteristics data is provided below:

`decryptAttributes` is not provided or is `Empty`:

```
<ser:property>
  <prop:name xmlns:prop="http://xmlns.<Oracle domain>/communications/
inventory/property">Serviceencryptedchar2</prop:name>
  <prop:value xmlns:prop="http://xmlns.<Oracle domain>/communications/
inventory/property">AES256wMtox/bYDiKOYNxlB8lXBfQNKtH+v6MRUIUumNUTna4=</
prop:value>
</ser:property>
```


Copyright © 2014, 2026, Oracle and/or its affiliates.

```
<ser:property>
  <prop:name xmlns:prop="http://xmlns.<Oracle domain>/communications/
inventory/property">Serviceencryptedchar1</prop:name>
  <prop:value xmlns:prop="http://xmlns.<Oracle domain>/communications/
inventory/property">AES256sDi3UhMBr9Z1ekMw534gBwtbKoeX/foB8nOQnuHCZ8ckXJSboA</
prop:value>
</ser:property>
```

`<con:decryptAttributes>ALL</con:decryptAttributes>`:

```
<ser:property>
  <prop:name xmlns:prop="http://xmlns.<Oracle domain>/communications/
inventory/property">Serviceencryptedchar2</prop:name>
  <prop:value xmlns:prop="http://xmlns.<Oracle domain>/communications/
inventory/property">ServChar2</prop:value>
</ser:property>
<ser:property>
  <prop:name xmlns:prop="http://xmlns.<Oracle domain>/communications/
inventory/property">Serviceencryptedchar1</prop:name>
  <prop:value xmlns:prop="http://xmlns.<Oracle domain>/communications/
inventory/property">ServChar1</prop:value>
</ser:property>
```


Copyright © 2014, 2026, Oracle and/or its affiliates.

3

Working with the Network Resource Management Web Service

This chapter provides information about the Oracle Communications Unified Inventory Management (UIM) Network Resource Management (NRM) Web Service.

> **Note**
>
> If you use UIM cloud native deployment for updating configuration files, refer to "Customizing UIM Configuration Properties" in *UIM Cloud Native Deployment Guide*.

About the NRM Web Service

> **Note**
>
> Before reading about the NRM Web Service, read *UIM Concepts* to have an understanding of UIM.

The NRM Web Service enables an external system to manage entities in UIM by supporting operations that enable you to:

- Create, find, update, and delete entities in UIM
- Reserve and unreserve resource entities in UIM
  - Find or create a reservation in UIM when reserving resource entities
  - Find or delete a reservation in UIM when unreserving resource entities
- Update reservations in UIM
- Associate and disassociate entities in UIM
- Import and export entities into and from UIM

About the Web Service Packaging

The NRM Web Service is packaged in the **inventory.ear** file, within the **InventoryWS.war** file. When the installer deploys the **inventory.ear** file, the NRM Web Service is automatically deployed and ready to use.

Copyright © 2014, 2026, Oracle and/or its affiliates.

> **Note**
>
> The **InventoryWS.war** file includes all of the Service Fulfillment Web Service operations as well. See "Working with the Service Fulfillment Web Service" for information about these operations.

About the WSDL and Schema Files

The NRM Web Service is defined by the **InventoryWS.wsdl** file and is supported by several schema files. The WSDL file and supporting schema files are located in the *UIM_SDK_Home*/**webservices/schema_inventory_webservice.zip** file.

About the WSDL File

Within ZIP file, the WSDL file is located in the **ora_uim_webservices/wsdl** directory. The WSDL file defines several operations. Each web service operation defines a request, a response, and the possible faults that can be thrown. For example, the WSDL file defines the following for the CreateEntity operation:

- createEntityRequest
- createEntityResponse
- createEntityFault
- inventoryFault
- validationFault

The request, response, and faults each define an XML structure that is defined in the supporting schema files. The following excerpts show how an XML structure defined in the WSDL correlates to the supporting schema files.

For example, the WSDL file defines and references the invnsrm namespace (bolded):

```
xmlns:invnsrm="http://xmlns.oracle.com/communications/inventory/webservice/nsrm"
.
.
.
targetNamespace
.
.
.
<xsd:import
namespace="http://xmlns.oracle.com/communications/inventory/webservice/nsrm"
schemaLocation="./schemas/NSRMMessages.xsd"/>
.
.
.
<wsdl:message name="CreateEntityRequest">
    <wsdl:part name="CreateEntityRequest" element="invnsrm:createEntityRequest">
    </wsdl:part>
</wsdl:message>
```

This tells you that the createEntityRequest XML structure is defined in the schema file that defines the specified namespace as its target namespace. A search for the specified namespace reveals that **NSRMMessages.xsd** defines the referenced namespace as its target namespace.

Copyright © 2014, 2026, Oracle and/or its affiliates.

After you determine which schema file defines the XML structure that the WSDL file references, you can navigate through the schema files to determine child XML structures and elements.

About the Schema Files

Several schema files support the NRM Web Service. These schemas are categorized as reference schemas, web service schemas, and business schemas.

Reference Schemas

Within the ZIP file, the reference schemas are located in the **ora_uim_webservices/wsdl/referenceSchemas** directory. The reference schemas define common elements used by more than one operation. So, the elements are defined in one place and then referenced.

The reference schemas are:

- InventoryCommon.xsd
- InventoryFaults.xsd
- FaultRoot.xsd

Web Service Schemas

Within the ZIP file, the web service schemas are located in the **ora_uim_webservices/wsdl/schemas** directory. The web service schemas define elements specific to the web service, such as the request structures, the response structures, and any fault structures.

The web service schema is defined in the **NRMMessages.xsd** file.

> **Note**
>
> The web service schema uses the **type-mapping.xsdconfig** file to map XML namespaces to Java packages.

Business Schemas

Within the ZIP file, the business schemas are located in the **ora_uim_business/schemas** directory. Each web service operation wraps a call (or multiple calls) to the UIM business layer, which is exposed through APIs. The wrapped APIs are the same APIs that the UIM UI calls in response to user input. The business layer APIs are based on functional area, as are the business schemas.

The business schemas are:

- Activity.xsd
- BusinessInteraction.xsd
- Configuration.xsd
- Connectivity.xsd
- CustomNetworkAddress.xsd
- CustomObject.xsd

Copyright © 2014, 2026, Oracle and/or its affiliates.

- Entity.xsd
- InventoryGroup.xsd
- IPAddress.xsd
- LogicalDevice.xsd
- MediaStream.xsd
- Network.xsd
- NetworkAddress.xsd
- Number.xsd
- Party.xsd
- PhysicalDevice.xsd
- Place.xsd
- Property.xsd
- PropertyLocation.xsd
- Role.xsd
- Service.xsd
- Specification.xsd
- Structure.xsd
- TNBlockModelType.xsd

> **Note**
>
> The API schemas use the **xmlbeans-mapping.xsdconfig** file to map XML namespaces to Java packages.

CreateEntity

The CreateEntity operation enables external systems to send a request to UIM to create certain entities in UIM.

createEntityRequest

You must specify the type of entity to create based on the entity types defined in the schema files. Each entity type defines different elements that pertain specifically to the entity type, which you use to define the entity you are creating. Table 3-1 lists the valid entity types and the schema files in which they are defined.

Table 3-1 Entity Types for CreateEntity

Entity Type	Schema File
ActivityType	Activity.xsd
CustomNetworkAddressType	CustomNetworkAddress.xsd
CustomObjectType	CustomObject.xsd

Copyright © 2014, 2026, Oracle and/or its affiliates.

Table 3-1 (Cont.) Entity Types for CreateEntity

Entity Type	Schema File
EquipmentType	PhysicalDevice.xsd
FlowIdentifierType	LogicalDevice.xsd
InventoryGroupType	InventoryGroup.xsd
IPv4AddressType	IPAddress.xsd
IPv6AddressType	IPAddress.xsd
IPSubnetType	IPAddress.xsd
LogicalDeviceType	LogicalDevice.xsd
LogicalDeviceAccountType	LogicalDevice.xsd
PhysicalDeviceType	PhysicalDevice.xsd
PlaceType	Place.xsd
TelephoneNumberType	Number.xsd

> **Note**
>
> PlaceType represents a GeographicLocation, which is a specialization of the abstract Place entity. See *Oracle Communications Information Model Reference* for more information.

Multiple Entities

You can create multiple entities per request by specifying one or more <entity> elements; however, all <entity> elements must specify the same entity type per request. For example, you can create multiple logical devices with a single request, and you can create multiple logical device accounts with a single request, but you cannot create multiple logical devices and multiple logical device accounts with a single request.

Optional Elements

You can specify an existing inventory group with which to associate the created entities. If you specify an inventory group that does not exist in UIM, an error is thrown.

You can specify parameters that define name/value pairs, which you can use with custom code to extend the operation. CreateEntity does not process specified parameters unless customized to do so. See "Customizing the Web Service Operations" for more information.

Example

Example 3-1 shows a request that specifies an entity type of TelephoneNumberType, which defines telephone number-specific elements such as <tn:rangeFrom> and <tn:rangeTo>.

This particular request:

- Creates a range of telephone numbers based on the usTelephoneNumber specification
- Adds the characteristics of tnCountryCode, winback, responsibleProvider, and tnType to each of the telephone numbers created, as specified by the property name element

Copyright © 2014, 2026, Oracle and/or its affiliates.

- Sets the characteristic values, as specified by the property value element
- Associates the created telephone numbers with the MobileServingArea inventory group

Example 3-1 createEntityRequest

```
<nsrm:createEntityRequest>
    <nsrm:entity xsi:type="tn:TelephoneNumberType">
        <tn:specification>
            <spec:name>usTelephoneNumber</spec:name>
        </tn:specification>
        <tn:rangeFrom>9729630001</tn:rangeFrom>
        <tn:rangeTo>9729630020</tn:rangeTo>
        <tn:description>Owned Number</tn:description>
        <tn:property>
            <prop:name>tnCountryCode</prop:name>
            <prop:value>1</prop:value>
        </tn:property>
        <tn:property>
            <prop:name>winback</prop:name>
            <prop:value>false</prop:value>
        </tn:property>
        <tn:property>
           <prop:name>responsibleProvider</prop:name>
           <prop:value>AT&T</prop:value>
        </tn:property>
        <tn:property>
            <prop:name>tnType</prop:name>
            <prop:value>OWNED</prop:value>
        </tn:property>
    </nsrm:entity>
    <nsrm:inventoryGroup>
        <ig:specification>
            <spec:name>MobileServingArea</spec:name>
        </ig:specification>
        <ig:name>North Dallas</ig:name>
    </nsrm:inventoryGroup>
    <nsrm:parameter>
        <bi:name></bi:name>
        <bi:value></bi:value>
    </nsrm:parameter>
</nsrm:createEntityRequest>
```

createEntityResponse

createEntityResponse returns information about the created entities. The information returned in the response is dependent upon the entity types that were created, as specified in the request.

createEntityResponse returns an error message when:

- The request specifies a specification that does not exist in UIM
- The request specifies an inventory group that does not exist in UIM
- The call to the UIM API fails

FindEntity

The FindEntity operation enables external systems to send a request to UIM to find and return certain entities in UIM, based on specified search criteria.

Copyright © 2014, 2026, Oracle and/or its affiliates.

findEntityRequest

You must specify search criteria to find the entities to retrieve. You have the choice of specifying search criteria in one of two ways. With either choice, you must specify the type of entity to find based on the entity types defined in the schema files. Each entity type defines different elements that pertain specifically to the entity type, which you use as search criteria to find entities. Table 3-2 lists the valid entity types and the schema files in which they are defined.

Table 3-2 Entity Types for FindEntity

Entity Type	Schema File
ActivityType	Activity.xsd
CustomNetworkAddressType	CustomNetworkAddress.xsd
CustomObjectType	CustomObject.xsd
EquipmentType	PhysicalDevice.xsd
FlowIdentifierType	LogicalDevice.xsd
InventoryGroupType	InventoryGroup.xsd
IPv4AddressType	IPAddress.xsd
IPv6AddressType	IPAddress.xsd
IPSubnetType	IPAddress.xsd
LogicalDeviceType	LogicalDevice.xsd
LogicalDeviceAccountType	LogicalDevice.xsd
PhysicalDeviceType	PhysicalDevice.xsd
PlaceType	Place.xsd
TelephoneNumberType	Number.xsd

The choices are:

- <entity>

  In this search option, you specify the entity type and use the entity-specific elements to specify search criteria.

  For each entity type, the <entity> structure varies. For example, TelephoneNumberType defines <rangeFrom> and <rangeTo>, but none of the other entity types define these elements.

  Even though the <entity> structure varies per entity type, the following elements are common across most entity types:

  – specification

    The search returns entities created from the specified specification.

  – id

    The search returns the entity with the specified id. (InventoryGroupType is only entity type with no id; the inventory group name is the id.)

  – name

    The search returns entities with the specified name.

Copyright © 2014, 2026, Oracle and/or its affiliates.

- description

  The search returns entities with the specified description.

- property

  The search returns entities with the data specified by property, which is an unbounded structure that provides the ability to specify the following:

  * Name of a characteristic
  * Value of specified characteristic

  > **Note**
  >
  > Within each *Entity*Type structure, the property element defines name, value, and action. However, action is not used; rather, the NRM Web Service operations always assume an operand of EQUALS.

- <criteria>

  In this search option, you specify the entity type and use the following search criteria:

  - specification

    The search returns entities created from the specified specification.

  - adminState

    The search returns entities in the specified administrative state, which is defined by the following enumeration values:

    ```
    <xs:enumeration value="END_OF_LIFE"/>
    <xs:enumeration value="INSTALLED"/>
    <xs:enumeration value="PENDING_INSTALL"/>
    <xs:enumeration value="PENDING_REMOVE"/>
    <xs:enumeration value="PENDING_UNAVAILABLE"/>
    <xs:enumeration value="PENDING_AVAILABLE"/>
    <xs:enumeration value="PLANNED"/>
    <xs:enumeration value="UNAVAILABLE"/>
    <xs:enumeration value="PENDING_DISCONNECT"/>
    <xs:enumeration value="DISCONNECTED"/>
    ```

  - assignmentState

    The search returns entities in the specified assignment state, which is defined by the following enumeration values:

    ```
    <xs:enumeration value="PENDING_ASSIGN"/>
    <xs:enumeration value="ASSIGNED"/>
    <xs:enumeration value="PENDING_UNASSIGN"/>
    <xs:enumeration value="UNASSIGNED"/>
    <xs:enumeration value="DISCONNECTED"/>
    <xs:enumeration value="PENDING_AVAILABLE"/>
    <xs:enumeration value="PENDING_UNAVAILABLE"/>
    <xs:enumeration value="PORTED"/>
    <xs:enumeration value="UNAVAILABLE"/>
    <xs:enumeration value="TRANSITIONAL"/>
    ```

  - inventoryGroup

    The search returns entities associated with the specified inventory group.

    If searching for telephone number entities, you can specify the inventoryGroup geographicLocation and the search returns telephone number entities associated with

Copyright © 2014, 2026, Oracle and/or its affiliates.

inventory groups that are associated with the specified place. If searching for entities other than telephone numbers, the inventoryGroup geographicLocation is not used.

- geographicLocation

  The search returns inventory group entities associated with the specified place. If searching for entities other than inventory groups, geographicLocation is not used.

- quantity

  The search returns the specified quantity of entities. For example, if the search finds 1,000 entities and the criteria specifies a quantity of 50, the first 50 entities found are returned. If the quantity is not given, the default retrieved is 10. If you want more than 10 entities returned, then a quantity must be given. Also, this quantity must be less than the maximum query range provided in the **system-config.properties** file for the uim.ws.search.query.range property setting. This setting can be set for all entities or for specific entities. Refer to *UIM System Administrator's Guide* for more information on this property setting.

- reservation

  If you specify reservation information, FindEntity also reserves any found entities. See "ReserveEntity" for more information.

- lock

  Row locking is used to optimize concurrent resource allocation for consumable entities.

  If you specify row-locking information, FindEntity does not release locked entities; you must manually release locked entities by calling the RowLockManager.releaseLock() method, or wait for the timer to release locked entities.

  If you specify row-locking information for entities that are not consumable (Geographic Location and Inventory Group), an error is thrown.

  See "*Optimizing Concurrent Resource Allocation in UIM*" in *UIM Developer's Guide* for more information about row locking, and see the Javadoc for information about the RowLockManager.releaseLock() method.

- criteriaItem

  The search returns entities based on data specified by criteriaItem, which is an unbounded structure that provides the ability to specify the following:

  * Name of a criteria item as defined by the *Entity*SearchCriteria class, where *Entity* is the name of a specific entity such as TelephoneNumber, LogicalDevice, and so forth (see "Determining Criteria Item Names")
  * Value of specified criteria item
  * Enumerated operand with which to evaluate the specified criteria item and corresponding specified value:

    ```
    <xs:enumeration value="EQUALS" />
    <xs:enumeration value="NOT_EQUALS" />
    <xs:enumeration value="BEGINS_WITH" />
    <xs:enumeration value="ENDS_WITH" />
    <xs:enumeration value="CONTAINS" />
    ```

- property

  The search returns entities with the data specified by property, which is an unbounded structure that provides the ability to specify the following:

  * Name of a characteristic

Copyright © 2014, 2026, Oracle and/or its affiliates.

* Value of specified characteristic
* Enumerated operand with which to evaluate the specified characteristic and corresponding specified value:

```
<xs:enumeration value="EQUALS" />
<xs:enumeration value="NOT_EQUALS" />
<xs:enumeration value="BEGINS_WITH" />
<xs:enumeration value="ENDS_WITH" />
<xs:enumeration value="CONTAINS" />
```

Multiple Entities

You specify one entity type to search for per request. For example, you can search for logical devices with a single request, and you can search for logical device accounts with a single request, but you cannot search for logical devices and logical device accounts with a single request.

Examples

Example 3-2 shows a request that specifies an entity type of TelephoneNumberType, and uses the telephone number-specific element of <tn:id> to search for the 9729630012 telephone number created from the usTelephoneNumber specification.

> **Note**
>
> Example 3-2 shows all of the telephone-number specific elements, even though they are not being used. Your requests do not need to include optional, unused elements; they are shown here as an example of entity-specific elements.

Example 3-2 findEntityRequest

```
<nsrm: findEntityRequest >
    <ent:entity xsi:type="tn:TelephoneNumberType">
        <tn:specification>
            <spec:name>usTelephoneNumber</spec:name>
        </tn:specification>
        <tn:id>9729630012</tn:id>
        <tn:name/>
        <tn:rangeFrom/>
        <tn:rangeTo/>
        <tn:description/>
        <tn:state/>
        <tn:startDate/>
        <tn:endDate/>
        <tn:property/>
            <prop:name/>
            <prop:value/>
        </tn:property>
    </ent:entity>
</nsrm: findEntityRequest >
```

Example 3-3 shows a request that specifies an entity type of TelephoneNumberType, but the telephone number-specific elements are not used to specify the search criteria (nor are they shown). Rather, this example specifies search criteria to find telephone numbers created from the usTelephoneNumber specification.

Copyright © 2014, 2026, Oracle and/or its affiliates.

This request specifies row-locking information that indicates to lock and return four telephone numbers.

> **Note**
>
> Example 3-3 shows all of the optional search criteria elements, even though they are not all being used. Your requests do not need to include optional, unused elements; they are shown here as an example of the possible search criteria elements.

Example 3-3 findEntityRequest

```
<nsrm:findEntityRequest>
    <nsrm:criteria>
        <ent:entityType xsi:type="tn:TelephoneNumberType">
            <!-- tn-specific elements are not shown -->
        </ent:entityType>
        <ent:specification>
            <spec:name>usTelephoneNumber</spec:name>
        </ent:specification>
        <ent:adminState/>
        <ent:assignmentState/>
        <ent:inventoryGroup/>
            <ig:geographicPlace/>
        </ent:inventoryGroup>
        <ent:geographicPlace/>
        <ent:quantity/>
            <ent:reservation/>
        <ent:lock>
            <ent:lockAllOrNone>true</ent:lockAllOrNone>
            <ent:quantity>4</ent:quantity>
        </ent:lock>
        <ent:criteriaItem/>
        <ent:property/>
    </nsrm:criteria>
</nsrm:findEntityRequest>
```

findEntityResponse

findEntityResponse returns the found entities, based on the search criteria specified in the request. The information returned in the response is dependent upon the entity types that were searched for and subsequently found.

If you specified reservation information in the request, reservation information is also returned.

findEntityResponse returns an error message when:

- The request specifies a lock policy for the non-consumable entities of Geographic Location or Group Inventory
- The request specifies a lock policy that specifies the number of resources to lock, and there are not enough resources available to lock
- The call to the UIM API fails

Copyright © 2014, 2026, Oracle and/or its affiliates.

FindTNBlock

The FindTNBlock operation enables external systems to send a request to UIM to find and return telephone number blocks in UIM, based on specified search criteria.

findTNBlockRequest

You must specify search criteria to find the telephone number blocks to retrieve. In the search criteria, you must specify the entity type of TelephoneNumberType, which is defined in the **Number.xsd** schema file.

In the <criteria> search option, you specify the entity type (TelephoneNumberType) and use the following search criteria:

- specification

  The search returns entities created from the specified Telephone Number specification.

- adminState

  The search returns entities in the specified administrative state, which is defined by the following enumeration values:

  ```
  <xs:enumeration value="END_OF_LIFE"/>
  <xs:enumeration value="INSTALLED"/>
  <xs:enumeration value="PENDING_INSTALL"/>
  <xs:enumeration value="PENDING_REMOVE"/>
  <xs:enumeration value="PENDING_UNAVAILABLE"/>
  <xs:enumeration value="PENDING_AVAILABLE"/>
  <xs:enumeration value="PLANNED"/>
  <xs:enumeration value="UNAVAILABLE"/>
  <xs:enumeration value="PENDING_DISCONNECT"/>
  <xs:enumeration value="DISCONNECTED"/>
  ```

- assignmentState

  The search returns entities in the specified assignment state, which is defined by the following enumeration values:

  ```
  <xs:enumeration value="PENDING_ASSIGN"/>
  <xs:enumeration value="ASSIGNED"/>
  <xs:enumeration value="PENDING_UNASSIGN"/>
  <xs:enumeration value="UNASSIGNED"/>
  <xs:enumeration value="DISCONNECTED"/>
  <xs:enumeration value="TRANSITIONAL"/>
  <xs:enumeration value="PORTED"/>
  ```

- inventoryGroup

  The search returns entities associated with the specified inventory group.

  If searching for telephone number entities, you can specify the inventoryGroup and the search returns telephone number entities associated with inventory groups.

- (optional) quantity

  The search returns the specified quantity of telephone number block. For example, if the search finds 300 telephone number blocks and the criteria specifies a quantity of 50, the first 50 telephone number blocks found are returned.

- criteriaItem

Copyright © 2014, 2026, Oracle and/or its affiliates.

The search returns entities based on data specified by criteriaItem, which is an unbounded structure that provides the ability to specify the following:

- Name of a criteria item as defined by the *Entity*SearchCriteria class, where *Entity* is the name of a specific entity such as TelephoneNumber, LogicalDevice, and so forth (see "Determining Criteria Item Names")
- Value of specified criteria item
- Enumerated operand with which to evaluate the specified criteria item and corresponding specified value:

```
<xs:enumeration value="EQUALS" />
<xs:enumeration value="NOT_EQUALS" />
<xs:enumeration value="BEGINS_WITH" />
<xs:enumeration value="ENDS_WITH" />
<xs:enumeration value="CONTAINS" />
```

Example

Example 3-4 shows a request that specifies an entity type of TelephoneNumberType to search for a telephone number block of size 10 for the BATTNSpec specification.

Example 3-4 findTNBlockRequest

```
<nsrm:findTNBlockRequest>
         <nsrm:criteria>
            <ent:entityType xsi:type="tn:TelephoneNumberType"></ent:entityType>
            <ent:specification>
               <spec:name>BATTNSpec</spec:name>
            </ent:specification>
           <ent:assignmentState>UNASSIGNED</ent:assignmentState>
        <ent:adminState>INSTALLED</ent:adminState>
        <ent:inventoryGroup>
        <gro:name>NORTH_AMERICA</gro:name>
        </ent:inventoryGroup>
        <ent:criteriaItem>
        <ent:name>serviceSpec</ent:name>
        <ent:value xsi:type="xs:string">BATServiceSpec</ent:value>
        <ent:operator>EQUALS</ent:operator>
        </ent:criteriaItem>
        <ent:serviceSpec>BATServiceSpec</ent:serviceSpec>
        <ent:criteriaItem>
        <ent:name>CONDITION_TYPE</ent:name>
        <ent:value xsi:type="xs:string">INFORMATIONAL</ent:value>
        <ent:operator>EQUALS</ent:operator>
        </ent:criteriaItem>
            <ent:criteriaItem>
               <ent:name>rangeFrom</ent:name>
               <ent:value xsi:type="xs:string">1</ent:value>
               <ent:operator>GREATER_THAN_EQUAL</ent:operator>
            </ent:criteriaItem>
                <ent:criteriaItem>
               <ent:name>rangeTo</ent:name>
               <ent:value xsi:type="xs:string">100000000</ent:value>
               <ent:operator>LESS_THAN_EQUAL</ent:operator>
            </ent:criteriaItem>
                <ent:criteriaItem>
               <ent:name>blockSize</ent:name>
               <ent:value xsi:type="xs:string">10</ent:value>
               <ent:operator>EQUALS</ent:operator>
        </ent:criteriaItem>
            <ent:quantity>50</ent:quantity>
```


Copyright © 2014, 2026, Oracle and/or its affiliates.

```
            </nsrm:criteria>
</nsrm:findTNBlockRequest>
```

findTNBlockResponse

findTNBlockResponse returns the found telephone number blocks, based on the search criteria specified in the request.

findTNBlockResponse returns an error message when:

- The request does not find the specified Inventory Group. In this case, the following error message is displayed:

  ```
  No InventoryGroups found with criteria.
  ```

- The call to the UIM API fails

UpdateEntity

The UpdateEntity operation enables external systems to send a request to UIM to update certain entities in UIM.

updateEntityRequest

You must specify the type of entity to update based on the entity types defined in the schema files. Each entity type defines different elements that pertain specifically to the entity type, which you use to specify what to update. Table 3-3 lists the valid entity types and the schema files in which they are defined.

Table 3-3 Entity Types for UpdateEntity

Entity Type	Schema File
ActivityType	Activity.xsd
CustomNetworkAddressType	CustomNetworkAddress.xsd
CustomObjectType	CustomObject.xsd
FlowIdentifierType	LogicalDevice.xsd
InventoryGroupType	InventoryGroup.xsd
IPv4AddressType	IPAddress.xsd
IPv6AddressType	IPAddress.xsd
IPSubnetType	IPAddress.xsd
LogicalDeviceAccountType	LogicalDevice.xsd
LogicalDeviceType	LogicalDevice.xsd
PhysicalDeviceType	PhysicalDevice.xsd
PlaceType	Place.xsd
TelephoneNumberType	Number.xsd

You must specify the same entity type twice: First, within the <entityDetails> element to specify the data to update; and second, within the <entitySearchCriteria> element to find the entities to update.

<entitySearchCriteria> defines the following search criteria:

Copyright © 2014, 2026, Oracle and/or its affiliates.

- specification

  The search returns entities created from the specified specification.

- adminState

  The search returns entities in the specified administrative state, which is defined by the following enumeration values:

  ```
  <xs:enumeration value="END_OF_LIFE"/>
  <xs:enumeration value="INSTALLED"/>
  <xs:enumeration value="PENDING_INSTALL"/>
  <xs:enumeration value="PENDING_REMOVE"/>
  <xs:enumeration value="PENDING_UNAVAILABLE"/>
  <xs:enumeration value="PENDING_AVAILABLE"/>
  <xs:enumeration value="PLANNED"/>
  <xs:enumeration value="UNAVAILABLE"/>
  <xs:enumeration value="PENDING_DISCONNECT"/>
  <xs:enumeration value="DISCONNECTED"/>
  ```

- assignmentState

  The search returns entities in the specified assignment state, which is defined by the following enumeration values:

  ```
  <xs:enumeration value="PENDING_ASSIGN"/>
  <xs:enumeration value="ASSIGNED"/>
  <xs:enumeration value="PENDING_UNASSIGN"/>
  <xs:enumeration value="UNASSIGNED"/>
  <xs:enumeration value="DISCONNECTED"/>
  <xs:enumeration value="PENDING_AVAILABLE"/>
  <xs:enumeration value="PENDING_UNAVAILABLE"/>
  <xs:enumeration value="PORTED"/>
  <xs:enumeration value="UNAVAILABLE"/>
  <xs:enumeration value="TRANSITIONAL"/>
  ```

- inventoryGroup

  The search returns entities associated with the specified inventory group.

  If searching for telephone number entities, you can specify the inventoryGroup geographicLocation and the search returns telephone number entities associated with inventory groups that are associated with the specified place. If searching for entities other than telephone numbers, the inventoryGroup geographicLocation is not used.

- geographicLocation

  The search returns inventory group entities associated with the specified place. If searching for entities other than inventory groups, geographicLocation is not used.

- quantity

  The search returns the specified quantity of entities. For example, if the search finds 1,000 entities and the criteria specifies a quantity of 50, the first 50 entities found are returned.

- reservation

  If you specify reservation information, UpdateEntity ignores it; FindEntity is the only operation that uses the reservation element. See "FindEntity" for more information.

- lock

  Row locking is used to optimize concurrent resource allocation for consumable entities.

  If you specify row-locking information, UpdateEntity releases locked entities; you do not need to manually release locked entities by calling the RowLockManager.releaseLock() method, or wait for the timer to release locked entities.

Copyright © 2014, 2026, Oracle and/or its affiliates.

If you specify row-locking information for entities that are not consumable (Geographic Location and Inventory Group), an error is thrown.

See "Optimizing Concurrent Resource Allocation in UIM" in *UIM Developer's Guide* for more information about row locking.

- criteriaItem

  The search returns entities based on data specified by criteriaItem, which is an unbounded structure that provides the ability to specify the following:

  – Name of a criteria item as defined by the *Entity*SearchCriteria class, where *Entity* is the name of a specific entity such as TelephoneNumber, LogicalDevice, and so forth (see "Determining Criteria Item Names")

  – Value of specified criteria item

  – Enumerated operand with which to evaluate the specified criteria item and corresponding specified value:

```
<xs:enumeration value="EQUALS" />
<xs:enumeration value="NOT_EQUALS" />
<xs:enumeration value="BEGINS_WITH" />
<xs:enumeration value="ENDS_WITH" />
<xs:enumeration value="CONTAINS" />
```

- property

  The search returns entities with the data specified by property, which is an unbounded structure that provides the ability to specify the following:

  – Name of characteristic

  – Value of specified characteristic

  – Enumerated operand with which to evaluate the specified characteristic and corresponding specified value:

```
<xs:enumeration value="EQUALS" />
<xs:enumeration value="NOT_EQUALS" />
<xs:enumeration value="BEGINS_WITH" />
<xs:enumeration value="ENDS_WITH" />
<xs:enumeration value="CONTAINS" />
```

Multiple Entities

You specify one <entityDetails> element per request, and one <entitySeachCriteria> element per request. For example, if the search criteria that <entitySearchCriteria> specifies returns 50 records, all 50 records are updated with the same data that <entityDetails> specifies.

Optional Elements

You can specify parameters that define name/value pairs, which you can use with custom code to extend the operation. UpdateEntity does not process specified parameters unless customized to do so. See "Customizing the Web Service Operations" for more information.

Examples

Example 3-5 shows a request that specifies an entity type of TelephoneNumberType and that specifies to update the description to **Update for Testing**. The request then specifies an entity type of TelephoneNumberType a second time to specify the search criteria to find the entities to update. In this example, the search criteria is to find telephone numbers created from the usTelephoneNumber specification.

Copyright © 2014, 2026, Oracle and/or its affiliates.

Example 3-5 updateEntityRequest and TelephoneNumberType

```
<nsrm:updateEntityRequest>
    <nsrm:entityDetails xsi:type="tn:TelephoneNumberType">
        <tn:description>Update for Testing</tn:description>
    </nsrm:entityDetails>
    <ent:entitySearchCriteria>
        <ent:entityType xsi:type="tn:TelephoneNumberType" />
        <ent:specification>
            <spec:name>usTelephoneNumber</spec:name>
        </ent:specification>
    </ent:entitySearchCriteria>
    <nsrm:parameter>
        <bi:name></bi:name>
        <bi:value></bi:value>
    </nsrm:parameter>
</nsrm:updateEntityRequest>
```

Example 3-6 shows a request that specifies an entity type of ActivityType and that specifies the activity as **complete**. The request then specifies an entity type of ActivityType a second time to specify the search criteria to find the entities to update. In this example, the search criteria finds an activity by its name.

Example 3-6 updateEntityRequest and ActivityType

```
<nsrm:updateEntityRequest>
    <nsrm:entityDetails xsi:type="act:ActivityType">
        <act:action>complete</act:action>
    </nsrm:entityDetails>
    <nsrm:entitySearchCriteria xsi:type="act:ActivitySearchCriteriaType">
        <ent:entityType xsi:type="act:ActivityType" >
            <act:name>Acquire Property Location</act:name>
            <act:businessInteractionId>1</act:businessInteractionId>
        </ent:entityType>
    </nsrm:entitySearchCriteria>
    <nsrm:parameter>
        <bus:name></bus:name>
        <bus:value></bus:value>
    </nsrm:parameter>
</nsrm:updateEntityRequest>
```

updateEntityResponse

updateEntityResponse returns information about the updated entities. The information returned in the response is dependent upon the entity types that were updated, as specified in the request.

updateEntityResponse returns an error message when:

- The request specifies two different entity types in the request entity type and the criteria entity type
- The request specifies a lock policy that specifies the number of resources to lock, and there are not enough resources available to lock
- The call to the UIM API fails

DeleteEntity

The DeleteEntity operation enables external systems to send a request to UIM to delete certain entities in UIM.

Copyright © 2014, 2026, Oracle and/or its affiliates.

deleteEntityRequest

You must specify search criteria to search for entities to delete. You have the choice of specifying search criteria in one of two ways. With either choice, you must specify the type of entity to find based on the entity types defined in the schema files. Each entity type defines different elements that pertain specifically to the entity type, which you use as search criteria to find entities to delete. Table 3-4 lists the valid entity types and the schema files in which they are defined.

Table 3-4 Entity Types for DeleteEntity

Entity Type	Schema File
ActivityType	Activity.xsd
CustomNetworkAddressType	CustomNetworkAddress.xsd
CustomObjectType	CustomObject.xsd
EquipmentType	PhysicalDevice.xsd
FlowIdentifierType	LogicalDevice.xsd
InventoryGroupType	InventoryGroup.xsd
IPv4AddressType	IPAddress.xsd
IPv6AddressType	IPAddress.xsd
IPSubnetType	IPAddress.xsd
LogicalDeviceType	LogicalDevice.xsd
LogicalDeviceAccountType	LogicalDevice.xsd
PhysicalDeviceType	PhysicalDevice.xsd
PlaceType	Place.xsd
TelephoneNumberType	Number.xsd

The choices are:

- <entity>

  In this search option, you specify the entity type and use the entity-specific elements to specify search criteria.

  For each entity type, the <entity> structure varies. For example, TelephoneNumberType defines <rangeFrom> and <rangeTo>, but none of the other entity types define these elements.

  Even though the <entity> structure varies per entity type, the following elements are common across most entity types:

  – specification

    The search returns entities created from the specified specification.

  – id

    The search returns the entity with the specified id. (InventoryGroupType is only entity type with no id; the inventory group name is the id.)

  – name

    The search returns entities with the specified name.

Copyright © 2014, 2026, Oracle and/or its affiliates.

- description

  The search returns entities with the specified description.

- property

  The search returns entities with the data specified by property, which is an unbounded structure that provides the ability to specify the following:

  * Name of a characteristic
  * Value of specified characteristic

  > **Note**
  >
  > Within each *Entity*Type structure, the property element defines name, value, and action. However, action is not used; rather, the NRM Web Service operations always assume an operand of EQUALS.

- <criteria>

  In this search option, you specify the entity type and use the following search criteria:

  - specification

    The search returns entities created from the specified specification.

  - adminState

    The search returns entities in the specified administrative state, which is defined by the following enumeration values:

    ```
    <xs:enumeration value="END_OF_LIFE"/>
    <xs:enumeration value="INSTALLED"/>
    <xs:enumeration value="PENDING_INSTALL"/>
    <xs:enumeration value="PENDING_REMOVE"/>
    <xs:enumeration value="PENDING_UNAVAILABLE"/>
    <xs:enumeration value="PENDING_AVAILABLE"/>
    <xs:enumeration value="PLANNED"/>
    <xs:enumeration value="UNAVAILABLE"/>
    <xs:enumeration value="PENDING_DISCONNECT"/>
    <xs:enumeration value="DISCONNECTED"/>
    ```

  - assignmentState

    The search returns entities in the specified assignment state, which is defined by the following enumeration values:

    ```
    <xs:enumeration value="PENDING_ASSIGN"/>
    <xs:enumeration value="ASSIGNED"/>
    <xs:enumeration value="PENDING_UNASSIGN"/>
    <xs:enumeration value="UNASSIGNED"/>
    <xs:enumeration value="DISCONNECTED"/>
    <xs:enumeration value="PENDING_AVAILABLE"/>
    <xs:enumeration value="PENDING_UNAVAILABLE"/>
    <xs:enumeration value="PORTED"/>
    <xs:enumeration value="UNAVAILABLE"/>
    <xs:enumeration value="TRANSITIONAL"/>
    ```

  - inventoryGroup

    The search returns entities associated with the specified inventory group.

    If searching for telephone number entities, you can specify the inventoryGroup geographicLocation and the search returns telephone number entities associated with

Copyright © 2014, 2026, Oracle and/or its affiliates.

inventory groups that are associated with the specified place. If searching for entities other than telephone numbers, the inventoryGroup geographicLocation is not used.

- geographicLocation

  The search returns inventory group entities associated with the specified place. If searching for entities other than inventory groups, geographicLocation is not used.

- quantity

  The search returns the specified quantity of entities. For example, if the search finds 1,000 entities and the criteria specifies a quantity of 50, the first 50 entities found are returned.

- reservation

  If you specify reservation information, DeleteEntity ignores it; FindEntity is the only operation that uses the reservation element. See "FindEntity" for more information.

- lock

  Row locking is used to optimize concurrent resource allocation for consumable entities; however, DeleteEntity does not use row locking.

  If you specify row-locking information, DeleteEntity ignores it.

- criteriaItem

  The search returns entities based on data specified by criteriaItem, which is an unbounded structure that provides the ability to specify the following:

  * Name of a criteria item as defined by the *Entity*SearchCriteria class, where *Entity* is the name of a specific entity such as TelephoneNumber, LogicalDevice, and so forth (see "Determining Criteria Item Names")
  * Value of specified criteria item
  * Enumerated operand with which to evaluate the specified criteria item and corresponding specified value:

    ```
    <xs:enumeration value="EQUALS" />
    <xs:enumeration value="NOT_EQUALS" />
    <xs:enumeration value="BEGINS_WITH" />
    <xs:enumeration value="ENDS_WITH" />
    <xs:enumeration value="CONTAINS" />
    ```

- property

  The search returns entities with the data specified by property, which is an unbounded structure that provides the ability to specify the following:

  * Name of a characteristic
  * Value of specified characteristic
  * Enumerated operand with which to evaluate the specified characteristic and corresponding specified value:

    ```
    <xs:enumeration value="EQUALS" />
    <xs:enumeration value="NOT_EQUALS" />
    <xs:enumeration value="BEGINS_WITH" />
    <xs:enumeration value="ENDS_WITH" />
    <xs:enumeration value="CONTAINS" />
    ```


Copyright © 2014, 2026, Oracle and/or its affiliates.

Multiple Entities

You specify one entity type to delete per request, and one entity search criteria to find the entities to delete per request. For example, you can search for logical devices to delete with a single request, and you can search for logical device accounts to delete with a single request, but you cannot search for logical devices and logical device accounts to delete with a single request.

Optional Elements

You can specify parameters that define name/value pairs, which you can use with custom code to extend the operation. DeleteEntity does not process specified parameters unless customized to do so. See "Customizing the Web Service Operations" for more information.

Examples

Example 3-7 shows a request that specifies an entity type of TelephoneNumberType, and uses entity-specific elements to find a particular telephone number to delete, 9729630014, created from the usTelephoneNumber specification.

Example 3-7 deleteEntityRequest

```
<nsrm:deleteEntityRequest>
    <ent:entity xsi:type="tn:TelephoneNumberType">
        <tn:specification>
            <spec:name>usTelephoneNumber</spec:name>
        </tn:specification>
        <tn:id>9729630014</tn:id>
    </ent:entity>
    <nsrm:parameter>
        <bi:name></bi:name>
        <bi:value></bi:value>
    </nsrm:parameter>
</nsrm:deleteEntityRequest>
```

Example 3-8 shows a request that specifies an entity type of TelephoneNumberType, and specifies search criteria to find telephone numbers to delete created from the usTelephoneNumber specification that are installed and unassigned.

Example 3-8 deleteEntityRequest

```
<nsrm:deleteEntityRequest>
    <ent:criteria>
        <ent:entityType xsi:type="tn:TelephoneNumberType"/>
        <ent:specification>
            <spec:name>usTelephoneNumber</spec:name>
        </ent:specification>
        <ent:adminState>INSTALLED</ent:adminState>
        <ent:assignmentState>UNASSIGNED</ent:assignmentState>
    </ent:criteria>
    <nsrm:parameter>
        <bi:name></bi:name>
        <bi:value></bi:value>
    </nsrm:parameter>
</nsrm:deleteEntityRequest>
```


Copyright © 2014, 2026, Oracle and/or its affiliates.

deleteEntityResponse

deleteEntityResponse returns information about the deleted entities. The information returned in the response is dependent upon the entity types that were deleted, as specified in the request.

deleteEntityResponse returns an error message when:

- The call to the UIM API fails

ReserveEntity

The ReserveEntity operation enables external systems to send a request to UIM to find an existing reservation or newly create a reservation, and to add certain resource entities to the existing or newly created reservation.

reserveEntityRequest

The request structure defines the ResourceReservationType entity type; you do not specify the entity type for the reservation.

Within the reservation-specific elements, you specify one or both of the following:

- An existing reservation number
- Reservation information with which to create a reservation, which at a minimum must include:
  - reservedFor
  - reservedForType
  - reservationType

If you specify an existing reservation number, the operation attempts to find the reservation based on the specified reservation number. If the reservation is found, additional resources are added to it. If the reservation is not found, an error is thrown.

If you specify reservation information with which to create a reservation, the operation generates a reservation number and creates a reservation using the specified information.

If you specify both a reservation number and reservation information with which to create a reservation, the operation attempts to find the reservation based on the specified reservation number. If the reservation is found, additional resources are added to it, but the reservation is not updated with the specified reservation information. If the reservation is not found, the operation creates a reservation using the specified reservation number and reservation information.

If you specify neither a reservation number nor reservation information, an error is thrown.

Resource Entity Search Criteria

You must specify search criteria to find existing resource entities to add to the reservation. You have the choice of specifying search criteria in one of two ways. With either choice, you must specify the type of resource entity to find based on the entity types defined in the schema files. Each entity type defines different elements that pertain specifically to the entity type, which you use as search criteria to find entities. Table 3-5 lists the valid entity types and the schema files in which they are defined.

Copyright © 2014, 2026, Oracle and/or its affiliates.

Table 3-5 Entity Types for ReserveEntity

Entity Type	Schema File
CustomNetworkAddressType	CustomNetworkAddress.xsd
CustomObjectType	CustomObject.xsd
FlowIdentifierType	LogicalDevice.xsd
IPv4AddressType	IPAddress.xsd
IPv6AddressType	IPAddress.xsd
IPSubnetType	IPAddress.xsd
LogicalDeviceType	LogicalDevice.xsd
LogicalDeviceAccountType	LogicalDevice.xsd
PhysicalDeviceType	PhysicalDevice.xsd
TelephoneNumberType	Number.xsd

The choices are:

- <resourceEntities>

  In this search option, you specify the entity type and use the entity-specific elements to specify search criteria.

  For each entity type, the <entity> structure varies. For example, TelephoneNumberType defines <rangeFrom> and <rangeTo>, but none of the other entity types define these elements.

  Even though the <entity> structure varies per entity type, the following elements are common across most entity types:

  – specification

    The search returns entities created from the specified specification.

  – id

    The search returns the entity with the specified id. (InventoryGroupType is only entity type with no id; the inventory group name is the id.)

  – name

    The search returns entities with the specified name.

  – description

    The search returns entities with the specified description.

  – property

    The search returns entities with the data specified by property, which is an unbounded structure that provides the ability to specify the following:

    * Name of a characteristic
    * Value of specified characteristic

Copyright © 2014, 2026, Oracle and/or its affiliates.

> **Note**
>
> Within each *Entity*Type structure, the property element defines name, value, and action. However, action is not used; rather, the NRM Web Service operations always assume an operand of EQUALS.

- <resourceCriteria>

  In this search option, you specify the entity type and use the following search criteria:

  – specification

    The search returns entities created from the specified specification.

  – adminState

    The search returns entities in the specified administrative state, which is defined by the following enumeration values:

```
<xs:enumeration value="END_OF_LIFE"/>
<xs:enumeration value="INSTALLED"/>
<xs:enumeration value="PENDING_INSTALL"/>
<xs:enumeration value="PENDING_REMOVE"/>
<xs:enumeration value="PENDING_UNAVAILABLE"/>
<xs:enumeration value="PENDING_AVAILABLE"/>
<xs:enumeration value="PLANNED"/>
<xs:enumeration value="UNAVAILABLE"/>
<xs:enumeration value="PENDING_DISCONNECT"/>
<xs:enumeration value="DISCONNECTED"/>
```

  – assignmentState

    The search returns entities in the specified assignment state, which is defined by the following enumeration values:

```
<xs:enumeration value="PENDING_ASSIGN"/>
<xs:enumeration value="ASSIGNED"/>
<xs:enumeration value="PENDING_UNASSIGN"/>
<xs:enumeration value="UNASSIGNED"/>
<xs:enumeration value="DISCONNECTED"/>
<xs:enumeration value="PENDING_AVAILABLE"/>
<xs:enumeration value="PENDING_UNAVAILABLE"/>
<xs:enumeration value="PORTED"/>
<xs:enumeration value="UNAVAILABLE"/>
<xs:enumeration value="TRANSITIONAL"/>
```

  – inventoryGroup

    The search returns entities associated with the specified inventory group.

    If searching for telephone number entities, you can specify the inventoryGroup geographicLocation and the search returns telephone number entities associated with inventory groups that are associated with the specified place. If searching for entities other than telephone numbers, the inventoryGroup geographicLocation is not used.

  – geographicLocation

    The search returns inventory group entities associated with the specified place. If searching for entities other than inventory groups, geographicLocation is not used. (You cannot reserve inventory groups, so geographicLocation is not used for ReserveEntity.)

  – quantity

Copyright © 2014, 2026, Oracle and/or its affiliates.

The search returns the specified quantity of entities. For example, if the search finds 1,000 entities and the criteria specifies a quantity of 50, the first 50 entities found are returned.

- reservation

  If you specify reservation information, ReserveEntity ignores it; FindEntity is the only operation that uses the reservation element. See "FindEntity" for more information.

- lock

  Row locking is used to optimize concurrent resource allocation for consumable entities.

  ReserveEntity always uses row locking, regardless of whether or not you specify the number of rows to lock. ReserveEntity releases locked entities; you do not need to manually release locked entities by calling the RowLockManager.releaseLock() method, or wait for the timer to release locked entities.

  See "*Optimizing Concurrent Resource Allocation in UIM*" in *UIM Developer's Guide* for more information about row locking.

- criteriaItem

  The search returns entities based on data specified by criteriaItem, which is an unbounded structure that provides the ability to specify the following:

  * Name of a criteria item as defined by the *Entity*SearchCriteria class, where *Entity* is the name of a specific entity such as TelephoneNumber, LogicalDevice, and so forth (see "Determining Criteria Item Names")
  * Value of specified criteria item
  * Enumerated operand with which to evaluate the specified criteria item and corresponding specified value:

    ```
    <xs:enumeration value="EQUALS" />
    <xs:enumeration value="NOT_EQUALS" />
    <xs:enumeration value="BEGINS_WITH" />
    <xs:enumeration value="ENDS_WITH" />
    <xs:enumeration value="CONTAINS" />
    ```

- property

  The search returns entities with the data specified by property, which is an unbounded structure that provides the ability to specify the following:

  * Name of a characteristic
  * Value of specified characteristic
  * Enumerated operand with which to evaluate the specified characteristic and corresponding specified value:

    ```
    <xs:enumeration value="EQUALS" />
    <xs:enumeration value="NOT_EQUALS" />
    <xs:enumeration value="BEGINS_WITH" />
    <xs:enumeration value="ENDS_WITH" />
    <xs:enumeration value="CONTAINS" />
    ```

Multiple Entities

You specify one reservation per request, and one search criteria per request. For example, if the search criteria returns 50 resources, all 50 resources are added to the specified reservation.

Copyright © 2014, 2026, Oracle and/or its affiliates.

Optional Elements

You can specify parameters that define name/value pairs, which you can use with custom code to extend the operation. ReserveEntity does not process specified parameters unless customized to do so. See "Customizing the Web Service Operations" for more information.

Example

Example 3-9 shows a request that specifies reservation information with which to create a reservation, and specifies search criteria to find two logical device resources based on their specification name and ID. Based on this request, a new reservation is created, and the two logical device resources are added to the reservation.

Example 3-9 reserveEntityRequest

```
<nsrm:reserveEntityRequest>
    <nsrm:reservation>
        <config:state/>
        <config:reservationNumber/>
        <config:reservationType>SHORTTERM</config:reservationType>
        <config:expiry/>
        <config:reservedForType>ORDER</config:reservedForType>
        <config:reservedFor>Customer XYZ</config:reservedFor>
        <config:reason/>
    </nsrm:reservation>
    <nsrm:entityType xsi:type="ld:LogicalDeviceType"/>
    <nsrm:resources>
        <ent:entity xsi:type="ld:LogicalDeviceType">
            <ld:specification>
                <spec:name>SIMCard</spec:name>
            </ld:specification>
            <ld:id>3101500000000009901</ld:id>
        </ent:entity>
        <ent:entity xsi:type="ld:LogicalDeviceType">
            <ld:specification>
                <spec:name>SIMCard</spec:name>
            </ld:specification>
            <ld:id>3101500000000009902</ld:id>
        </ent:entity>
    </nsrm:resources>
    <nsrm:parameter>
        <bi:name></bi:name>
        <bi:value></bi:value>
    </nsrm:parameter>
</nsrm:reserveEntityRequest>
```

reserveEntityResponse

reserveEntityResponse returns information about the reservation and the reserved resource entities. The resource entity information returned in the response is dependent upon the resource entity types that were reserved, as specified in the request.

reserveEntityResponse returns an error message when:

- The request specifies a reservation that does not exist
- The request specifies no reservation number and no reservation information with which to create a reservation

Copyright © 2014, 2026, Oracle and/or its affiliates.

- The request specifies a resource that does not exist
- The call to the UIM API fails

ReserveTNBlock

The ReserveTNBlock operation enables external systems to send a request to UIM to find an existing reservation or create a new reservation, and to add telephone number blocks to the existing or newly created reservation.

reserveTNBlockRequest

The request structure defines the ResourceReservationType entity type; you do not specify the entity type for the reservation.

Within the reservation-specific elements, you specify one or both of the following:

- An existing reservation number
- Reservation information with which to create a reservation, which at a minimum must include:
  - reservedFor
  - reservedForType
  - reservationType

If you specify an existing reservation number, the operation attempts to find the reservation based on the specified reservation number. If the reservation is found, the telephone number block is added to it. If the reservation is not found, an error is thrown.

If you specify reservation information with which to create a reservation, the operation generates a reservation number and creates a reservation using the specified information.

If you specify both a reservation number and reservation information with which to create a reservation, the operation attempts to find the reservation based on the specified reservation number. If the reservation is found, the telephone number block is added to it, but the reservation is not updated with the specified reservation information. If the reservation is not found, the operation creates a reservation using the specified reservation number and reservation information.

If you specify neither a reservation number nor reservation information, an error is thrown.

Telephone Number Block Search Criteria

You must specify search criteria to find an existing telephone number block to add to an existing reservation or to a new reservation. The reserveTNBlockRequest XML structure is defined in the **TNBlockModelType.xsd** schema file.

You can specify the following in the request criteria:

- <reservation>

  If this request option, specify the reservation information by using the following search criteria:

  - reservationNumber

    The reservation number after the reservation has been created.

  - reservationType

Copyright © 2014, 2026, Oracle and/or its affiliates.

The type of reservation; for example, Shortterm or Longterm.

- startDate

  The date on which the reservation becomes effective.

- (Optional) expiryDate

  The date on which the reservation expires.

- reservedForType

  The type of entity or process for which the reservation is made.

- reservedFor

  Identifies who is making the reservation.

- reason

  The reason for the reservation.

- <tnBlockCriteria>

  In this request option, you can specify the following criteria:

  - startNumber

    The starting number of a range of numbers that you want to reserve.

  - endNumber

    The ending number of a range of numbers that you want to reserve.

  - blockSize

    The telephone number block size that you want to reserve.

Example

Example 3-10 shows a request that specifies an existing reservation number of 1125010, and specifies the criteria to reserve a telephone number block size of 10 within a telephone number range between 295 and 395.

Example 3-10 reserveTNBlockRequest

```
<nsrm:reserveTNBlockRequest>
 <nsrm:reservation>
 <con:reservationNumber>1125010</con:reservationNumber>
      <con:reservationType>SHORTTERM</con:reservationType>
      <con:startDate>2018-07-19T20:47:12.380+05:30</con:startDate>
      <con:expiryDate>2018-08-31T21:47:12.380+05:30</con:expiryDate>
      <con:reservedForType>CUSTOMER</con:reservedForType>
      <con:reservedFor>CUSTOMER_NAME</con:reservedFor>
      <con:reason>REASON_FOR_RESERVATION</con:reason>
   </nsrm:reservation>
   <nsrm:tnBlockCriteria>
      <tnb:startNumber>295</tnb:startNumber>
      <tnb:endNumber>395</tnb:endNumber>
      <tnb:blockSize>10</tnb:blockSize>
   </nsrm:tnBlockCriteria>
</nsrm:reserveTNBlockRequest>
```

reserveTNBlockResponse

reserveTNBlockResponse returns information about the reservation and the reserved telephone number block.

Copyright © 2014, 2026, Oracle and/or its affiliates.

reserveTNBlockResponse returns an error message when:

- The request specifies a reservation that does not exist
- The request specifies no reservation number and no reservation information with which to create a reservation
- The request specifies a telephone number block that does not exist
- The call to the UIM API fails

UnreserveEntity

The UnreserveEntity operation enables external systems to send a request to UIM to unreserve certain resource entities from an existing reservation in UIM. If no resources remain for the reservation after the specified resource entities are unreserved, the reservation is deleted.

unreserveEntityRequest

The request structure defines the ResourceReservationType entity type; you do not specify the entity type.

Within the reservation-specific elements, you specify an existing reservation number. If the reservation is not found, an error is thrown.

Resource Entity Search Criteria

You must specify search criteria to find existing resource entities to unreserve. You have the choice of specifying search criteria in one of two ways. With either choice, you must specify the type of resource entity to find based on the entity types defined in the schema files. Each entity type defines different elements that pertain specifically to the entity type, which you use as search criteria to find entities. Table 3-6 lists the valid entity types and the schema files in which they are defined.

Table 3-6 Entity Types for UnreserveEntity

Entity Type	Schema File
CustomNetworkAddressType	CustomNetworkAddress.xsd
CustomObjectType	CustomObject.xsd
FlowIdentifierType	LogicalDevice.xsd
IPv4AddressType	IPAddress.xsd
IPv6AddressType	IPAddress.xsd
IPSubnetType	IPAddress.xsd
LogicalDeviceType	LogicalDevice.xsd
LogicalDeviceAccountType	LogicalDevice.xsd
PhysicalDeviceType	PhysicalDevice.xsd
TelephoneNumberType	Number.xsd

The choices are:

- <resourceEntities>

Copyright © 2014, 2026, Oracle and/or its affiliates.

In this search option, you specify the entity type and use the entity-specific elements to specify search criteria.

For each entity type, the <entity> structure varies. For example, TelephoneNumberType defines <rangeFrom> and <rangeTo>, but none of the other entity types define these elements.

Even though the <entity> structure varies per entity type, the following elements are common across most entity types:

- specification

  The search returns entities created from the specified specification.

- id

  The search returns the entity with the specified id. (InventoryGroupType is only entity type with no id; the inventory group name is the id.)

- name

  The search returns entities with the specified name.

- description

  The search returns entities with the specified description.

- property

  The search returns entities with the data specified by property, which is an unbounded structure that provides the ability to specify the following:

  * Name of a characteristic
  * Value of specified characteristic

  > **Note**
  >
  > Within each *Entity*Type structure, the property element defines name, value, and action. However, action is not used; rather, the NRM Web Service operations always assume an operand of EQUALS.

- <resourceCriteria>

  In this search option, you specify the entity type and use the following search criteria:

  - specification

    The search returns entities created from the specified specification.

  - adminState

    The search returns entities in the specified administrative state, which is defined by the following enumeration values:

```
<xs:enumeration value="END_OF_LIFE"/>
<xs:enumeration value="INSTALLED"/>
<xs:enumeration value="PENDING_INSTALL"/>
<xs:enumeration value="PENDING_REMOVE"/>
<xs:enumeration value="PENDING_UNAVAILABLE"/>
<xs:enumeration value="PENDING_AVAILABLE"/>
<xs:enumeration value="PLANNED"/>
<xs:enumeration value="UNAVAILABLE"/>
<xs:enumeration value="PENDING_DISCONNECT"/>
<xs:enumeration value="DISCONNECTED"/>
```


Copyright © 2014, 2026, Oracle and/or its affiliates.

- assignmentState

  The search returns entities in the specified assignment state, which is defined by the following enumeration values:

  ```
  <xs:enumeration value="PENDING_ASSIGN"/>
  <xs:enumeration value="ASSIGNED"/>
  <xs:enumeration value="PENDING_UNASSIGN"/>
  <xs:enumeration value="UNASSIGNED"/>
  <xs:enumeration value="DISCONNECTED"/>
  <xs:enumeration value="PENDING_AVAILABLE"/>
  <xs:enumeration value="PENDING_UNAVAILABLE"/>
  <xs:enumeration value="PORTED"/>
  <xs:enumeration value="UNAVAILABLE"/>
  <xs:enumeration value="TRANSITIONAL"/>
  ```

- inventoryGroup

  The search returns entities associated with the specified inventory group.

  If searching for telephone number entities, you can specify the inventoryGroup geographicLocation and the search returns telephone number entities associated with inventory groups that are associated with the specified place. If searching for entities other than telephone numbers, the inventoryGroup geographicLocation is not used.

- geographicLocation

  The search returns inventory group entities associated with the specified place. If searching for entities other than inventory groups, geographicLocation is not used. (You cannot unreserve inventory groups, so geographicLocation is not used for UnreserveEntity.)

- quantity

  The search returns the specified quantity of entities. For example, if the search finds 1,000 entities and the criteria specifies a quantity of 50, the first 50 entities found are returned.

- reservation

  If you specify reservation information, UnreserveEntity ignores it; FindEntity is the only operation that uses the reservation element. See "FindEntity" for more information.

- lock

  Row locking is used to optimize concurrent resource allocation for consumable entities; however, UnreserveEntity does not use row locking.

  If you specify row-locking information, UnreserveEntity ignores it.

- criteriaItem

  The search returns entities based on data specified by criteriaItem, which is an unbounded structure that provides the ability to specify the following:

  * Name of a criteria item as defined by the *Entity*SearchCriteria class, where *Entity* is the name of a specific entity such as TelephoneNumber, LogicalDevice, and so forth (see "Determining Criteria Item Names")
  * Value of specified criteria item
  * Enumerated operand with which to evaluate the specified criteria item and corresponding specified value:

    ```
    <xs:enumeration value="EQUALS" />
    <xs:enumeration value="NOT_EQUALS" />
    <xs:enumeration value="BEGINS_WITH" />
    ```


Copyright © 2014, 2026, Oracle and/or its affiliates.

```
<xs:enumeration value="ENDS_WITH" />
<xs:enumeration value="CONTAINS" />
```

- property

  The search returns entities with the data specified by property, which is an unbounded structure that provides the ability to specify the following:

  * Name of a characteristic
  * Value of specified characteristic
  * Enumerated operand with which to evaluate the specified characteristic and corresponding specified value:

    ```
    <xs:enumeration value="EQUALS" />
    <xs:enumeration value="NOT_EQUALS" />
    <xs:enumeration value="BEGINS_WITH" />
    <xs:enumeration value="ENDS_WITH" />
    <xs:enumeration value="CONTAINS" />
    ```

Multiple Entities

You specify one reservation per request, and one search criteria per request. For example, if the search criteria returns 50 resources, all 50 resources are unreserved for the specified reservation. If no resource entities remain on the reservation, the reservation is deleted.

Optional Elements

You can specify parameters that define name/value pairs, which you can use with custom code to extend the operation. UnreserveEntity does not process specified parameters unless customized to do so. See "Customizing the Web Service Operations" for more information.

Examples

Example 3-11 shows a request that specifies an existing reservation number of 12345678, and specifies search criteria to find a particular telephone number range based on its specification name. Based on this request, the telephone numbers 8588880081 through 8588880083 are unreserved for reservation 12345678.

Example 3-11 unreserveEntityRequest with a Telephone Number Range

```
<nsrm:unreserveEntityRequest>
    <nsrm:reservation>
        <config:reservationNumber>12345678</config:reservationNumber>
    </nsrm:reservation>
    <nsrm:resourceCriteria>
        <ent:entityType xsi:type="tn:TelephoneNumberType"/>
        <ent:specification>
            <spec:name>BATTNSpec</spec:name>
        </ent:specification>
        <ent:criteriaItem>
            <ent:name>rangeFrom</ent:name>
            <ent:value xsi:type="xs:string">8588880081</ent:value>
            <ent:operator>EQUALS</ent:operator>
        </ent:criteriaItem>
        <ent:criteriaItem>
            <ent:name>rangeTo</ent:name>
            <ent:value xsi:type="xs:string">8588880083</ent:value>
            <ent:operator>EQUALS</ent:operator>
        </ent:criteriaItem>
```


Copyright © 2014, 2026, Oracle and/or its affiliates.

```
    </nsrm:resourceCriteria>
</nsrm:unreserveEntityRequest>
```

Example 3-12 shows a request that specifies an existing reservation number of 123456789, and specifies search criteria to find a particular telephone number resource based on its specification name and resource ID. Based on this request, the 9729630012 telephone number resource is unreserved. The 9729630012 telephone number is the only resource on the reservation, and the reservation is deleted.

> **Note**
>
> If this type of request is used for a reservation of a range of numbers the entire range is unreserved. Therefore, this type of request is only valid to unreserve a single reserved telephone number.

Example 3-12 unreserveEntityRequest for a Single Telephone Number

```
<nsrm:unreserveEntityRequest>
     <nsrm:reservation>
        <config:reservationNumber>123456789</config:reservationNumber>
    </nsrm:reservation>
    <nsrm:entityType xsi:type="tn:TelephoneNumberType"/>
    <nsrm:resources>
        <ent:entity xsi:type="tn:TelephoneNumberType">
            <tn:specification>
                <spec:name>usTelephoneNumber</spec:name>
            </tn:specification>
            <tn:id>9729630012</tn:id>
        </ent:entity>
    </nsrm:resources>
    <nsrm:parameter>
        <bi:name></bi:name>
        <bi:value></bi:value>
    </nsrm:parameter>
</nsrm:unreserveEntityRequest>
```

unreserveEntityResponse

unreserveEntityResponse returns information about the reservation and the unreserved resources (entities). The information returned in the response is dependent upon the resource entity types that were unreserved.

unreserveEntityResponse returns an error message when:

- The specified reservation number is not found
- The request specifies search criteria that retrieves resources not related to the specified reservation number
- The call to the UIM API fails

UpdateReservation

The UpdateReservation operation enables external systems to send a request to UIM to update a reservation in UIM. This operation updates only reservation information; it does not update resources on the reservation, and it does not reserve or unreserve resources on the

Copyright © 2014, 2026, Oracle and/or its affiliates.

reservation. See "ReserveEntity" for information about reserving resources for a reservation, and see "UnreserveEntity" for information about unreserving resources for a reservation.

updateReservationRequest

The request structure defines the ResourceReservationType entity type; you do not specify the entity type.

Within the reservation-specific elements, you specify an existing reservation number and for whom the reservation is reserved, as well as any reservation information to update. If the reservation is not found, an error is thrown.

Multiple Reservations

You can update only one reservation per request.

Optional Elements

You can specify parameters that define name/value pairs, which you can use with custom code to extend the operation. UpdateReservation does not process specified parameters unless customized to do so. See "Customizing the Web Service Operations" for more information.

Example

Example 3-13 shows a request that specifies an existing reservation number of 12345678 for Clark Kent, and specifies reservation information with which to update the reservation. <reservationNumber> and <reservedFor> are required elements used to retrieve the reservation. The remaining elements are optional and are used to specify the data with which to update the reservation.

Example 3-13 updateReservationRequest

```
<nsrm:updateReservationRequest>
    <nsrm:reservation>
        <config:reservationNumber>12345678</config:reservationNumber>
        <config:reservationType>LONGTERM</config:reservationType>
        <config:expiry>2018-12-31T00:00:00.000-06:00</config:expiry>
        <config:reservedForType>CSR</config:reservedForType>
        <config:reservedFor>Clark Kent</config:reservedFor>
        <config:reason>Testing</config:reason>
    </nsrm:reservation>
    <nsrm:parameter>
        <bi:name></bi:name>
        <bi:value></bi:value>
    </nsrm:parameter>
</nsrm:updateReservationRequest>
```

updateReservationResponse

updateReservationResponse returns information about the updated reservation.

updateReservationResponse returns an error message when:

- The specified reservation number is not found
- The call to the UIM API fails

Copyright © 2014, 2026, Oracle and/or its affiliates.

AssociateEntity

The AssociateEntity operation enables external systems to send a request to UIM to associate certain entities in UIM.

associateEntityRequest

You must specify an association type of ASSOCIATE or PAIR, which are enumeration values defined in the **entity.xsd** file.

An association of type ASSOCIATE indicates a one-to-many association between a single specified source entity and multiple specified target entities. An association type of PAIR indicates a one-to-one association between a source entity and a target entity; in this type of association, multiple source entities and multiple target entities can be specified, but the number of each specified must be the same.

You must specify search criteria to find existing source entities to associate; and you must specify search criteria to find existing target entities to associate. You have the choice of specifying search criteria in one of two ways. With either choice, you must specify the type of source/target entity to find based on the entity types defined in the schema files. Each entity type defines different elements that pertain specifically to the entity type, which you use as search criteria to find entities. Table 3-7 lists the valid entity types and the schema files in which they are defined.

Table 3-7 Entity Types for AssociateEntity

Entity Type	Schema File
ActivityType	Activity.xsd
CustomNetworkAddressType	CustomNetworkAddress.xsd
CustomObjectType	CustomObject.xsd
EquipmentType	PhysicalDevice.xsd
InventoryGroupType	InventoryGroup.xsd
LogicalDeviceType	LogicalDevice.xsd
LogicalDeviceAccountType	LogicalDevice.xsd
PhysicalDeviceType	PhysicalDevice.xsd
PlaceType	Place.xsd
TelephoneNumberType	Number.xsd

The choices are:

- <entity>

  In this search option, you specify the entity type and use the entity-specific elements to specify search criteria.

  For each entity type, the <entity> structure varies. For example, TelephoneNumberType defines <rangeFrom> and <rangeTo>, but none of the other entity types define these elements.

  Even though the <entity> structure varies per entity type, the following elements are common across most entity types:

  – specification

Copyright © 2014, 2026, Oracle and/or its affiliates.

The search returns entities created from the specified specification.

- id

  The search returns the entity with the specified id. (InventoryGroupType is only entity type with no id; the inventory group name is the id.)

- name

  The search returns entities with the specified name.

- description

  The search returns entities with the specified description.

- property

  The search returns entities with the data specified by property, which is an unbounded structure that provides the ability to specify the following:

  * Name of a characteristic
  * Value of specified characteristic

> **Note**
>
> Within each *Entity*Type structure, the property element defines name, value, and action. However, action is not used; rather, the NRM Web Service operations always assume an operand of EQUALS.

- <criteria>

  In this search option, you specify the entity type and use the following search criteria:

  - specification

    The search returns entities created from the specified specification.

  - adminState

    The search returns entities in the specified administrative state, which is defined by the following enumeration values:

    ```
    <xs:enumeration value="END_OF_LIFE"/>
    <xs:enumeration value="INSTALLED"/>
    <xs:enumeration value="PENDING_INSTALL"/>
    <xs:enumeration value="PENDING_REMOVE"/>
    <xs:enumeration value="PENDING_UNAVAILABLE"/>
    <xs:enumeration value="PENDING_AVAILABLE"/>
    <xs:enumeration value="PLANNED"/>
    <xs:enumeration value="UNAVAILABLE"/>
    <xs:enumeration value="PENDING_DISCONNECT"/>
    <xs:enumeration value="DISCONNECTED"/>
    ```

  - assignmentState

    The search returns entities in the specified assignment state, which is defined by the following enumeration values:

    ```
    <xs:enumeration value="PENDING_ASSIGN"/>
    <xs:enumeration value="ASSIGNED"/>
    <xs:enumeration value="PENDING_UNASSIGN"/>
    <xs:enumeration value="UNASSIGNED"/>
    <xs:enumeration value="DISCONNECTED"/>
    <xs:enumeration value="PENDING_AVAILABLE"/>
    <xs:enumeration value="PENDING_UNAVAILABLE"/>
    ```


Copyright © 2014, 2026, Oracle and/or its affiliates.

```
<xs:enumeration value="PORTED"/>
<xs:enumeration value="UNAVAILABLE"/>
<xs:enumeration value="TRANSITIONAL"/>
```

- inventoryGroup

  The search returns entities associated with the specified inventory group.

  If searching for telephone number entities, you can specify the inventoryGroup geographicLocation and the search returns telephone number entities associated with inventory groups that are associated with the specified place. If searching for entities other than telephone numbers, the inventoryGroup geographicLocation is not used.

- geographicLocation

  The search returns inventory group entities associated with the specified place. If searching for entities other than inventory groups, geographicLocation is not used.

- quantity

  The search returns the specified quantity of entities. For example, if the search finds 1,000 entities and the criteria specifies a quantity of 50, the first 50 entities found are returned.

- reservation

  If you specify reservation information, AssociateEntity ignores it; FindEntity is the only operation that uses the reservation element. See "FindEntity" for more information.

- lock

  Row locking is used to optimize concurrent resource allocation for consumable entities.

  If you specify row-locking information, AssociateEntity releases locked entities; you do not need to manually release locked entities by calling the RowLockManager.releaseLock() method, or wait for the timer to release locked entities.

  If you specify row-locking information for entities that are not consumable (Geographic Location and Inventory Group), an error is thrown.

  See "*Optimizing Concurrent Resource Allocation in UIM*" in *UIM Developer's Guide* for more information about row locking.

- criteriaItem

  The search returns entities based on data specified by criteriaItem, which is an unbounded structure that provides the ability to specify the following:

  * Name of a criteria item as defined by the *Entity*SearchCriteria class, where *Entity* is the name of a specific entity such as TelephoneNumber, LogicalDevice, and so forth (see "Determining Criteria Item Names")
  * Value of specified criteria item
  * Enumerated operand with which to evaluate the specified criteria item and corresponding specified value:

    ```
    <xs:enumeration value="EQUALS" />
    <xs:enumeration value="NOT_EQUALS" />
    <xs:enumeration value="BEGINS_WITH" />
    <xs:enumeration value="ENDS_WITH" />
    <xs:enumeration value="CONTAINS" />
    ```

- property

Copyright © 2014, 2026, Oracle and/or its affiliates.

The search returns entities with the data specified by property, which is an unbounded structure that provides the ability to specify the following:

* Name of a characteristic
* Value of specified characteristic
* Enumerated operand with which to evaluate the specified characteristic and corresponding specified value:

```
<xs:enumeration value="EQUALS" />
<xs:enumeration value="NOT_EQUALS" />
<xs:enumeration value="BEGINS_WITH" />
<xs:enumeration value="ENDS_WITH" />
<xs:enumeration value="CONTAINS" />
```

Multiple Entities

You specify one association type per request, and two sets of search criteria per request; one to find the source entities to associate, and one to find the target entities to associate.

Example

Example 3-14 shows a request that associates the specified source and target entities with an association type of ASSOCIATE. The source entity (only one source entity is specified in this example) is the MobileServingArea inventory group. The target entities are all logical devices created from the SIMCard specification that are installed and unassigned.

Example 3-14 associateEntityRequest

```
<nsrm:associateEntityRequest>
    <nsrm:associationType>ASSOCIATE</nsrm:associationType>
    <nsrm:sourceEntities>
        <nsrm:entityType xsi:type="ig:InventoryGroupType" />
        <ent:criteria>
            <ent:entityType xsi:type="ig:InventoryGroupType" />
            <ent:specification>
                <spec:name>MobileServingArea</spec:name>
            </ent:specification>
        </ent:criteria>
    </nsrm:sourceEntities>
    <nsrm:targetEntities>
        <nsrm:entityType xsi:type="ld:LogicalDeviceType" />
        <ent:criteria>
            <ent:entityType xsi:type="ld:LogicalDeviceType" />
            <ent:specification>
                <spec:name>SIMCard</spec:name>
            </ent:specification>
            <ent:adminState>INSTALLED</ent:adminState>
            <ent:assignmentState>UNASSIGNED</ent:assignmentState>
        </ent:criteria>
    </nsrm:targetEntities>
</nsrm:associateEntityRequest>
```

associateEntityResponse

associateEntityResponse returns information about the associated entities. The information returned in the response is dependent upon the entity types that were associated, as specified in the request.

associateEntityResponse returns an error message when:

Copyright © 2014, 2026, Oracle and/or its affiliates.

- The request specifies search criteria that results in no entities found to associate
- The request specifies an association type of PAIR and the number of sources and targets found is the not the same
- The call to the UIM API fails

DisassociateEntity

The DisassociateEntity operation enables external systems to send a request to UIM to disassociate certain existing associated entities in UIM.

disassociateEntityRequest

You must specify an association type of ASSOCIATE or PAIR, which are enumeration values defined in the **entity.xsd** file.

An association of type ASSOCIATE indicates a one-to-many association between a single specified source entity and multiple specified target entities. An association type of PAIR indicates a one-to-one association between a source entity and a target entity.

You must specify search criteria to find existing source entities to disassociate; and you must specify search criteria to find existing target entities to disassociate. You have the choice of specifying search criteria in one of two ways. With either choice, you must specify the type of source/target entity to find based on the entity types defined in the schema files. Each entity type defines different elements that pertain specifically to the entity type, which you use as search criteria to find entities. Table 3-8 lists the valid entity types and the schema files in which they are defined.

Table 3-8 Entity Types for DisassociateEntity

Entity Type	Schema File
ActivityType	Activity.xsd
CustomNetworkAddressType	CustomNetworkAddress.xsd
CustomObjectType	CustomObject.xsd
EquipmentType	PhysicalDevice.xsd
InventoryGroupType	InventoryGroup.xsd
LogicalDeviceType	LogicalDevice.xsd
LogicalDeviceAccountType	LogicalDevice.xsd
PhysicalDeviceType	PhysicalDevice.xsd
PlaceType	Place.xsd
TelephoneNumberType	Number.xsd

The choices are:

- <entity>

  In this search option, you specify the entity type and use the entity-specific elements to specify search criteria.

  For each entity type, the <entity> structure varies. For example, TelephoneNumberType defines <rangeFrom> and <rangeTo>, but none of the other entity types define these elements.

Copyright © 2014, 2026, Oracle and/or its affiliates.

Even though the <entity> structure varies per entity type, the following elements are common across most entity types:

- specification

  The search returns entities created from the specified specification.

- id

  The search returns the entity with the specified id. (InventoryGroupType is only entity type with no id; the inventory group name is the id.)

- name

  The search returns entities with the specified name.

- description

  The search returns entities with the specified description.

- property

  The search returns entities with the data specified by property, which is an unbounded structure that provides the ability to specify the following:

  * Name of a characteristic
  * Value of specified characteristic

  > **Note**
  >
  > Within each *Entity*Type structure, the property element defines name, value, and action. However, action is not used; rather, the NRM Web Service operations always assume an operand of EQUALS.

- <criteria>

  In this search option, you specify the entity type and use the following search criteria:

  - specification

    The search returns entities created from the specified specification.

  - adminState

    The search returns entities in the specified administrative state, which is defined by the following enumeration values:

    ```
    <xs:enumeration value="END_OF_LIFE"/>
    <xs:enumeration value="INSTALLED"/>
    <xs:enumeration value="PENDING_INSTALL"/>
    <xs:enumeration value="PENDING_REMOVE"/>
    <xs:enumeration value="PENDING_UNAVAILABLE"/>
    <xs:enumeration value="PENDING_AVAILABLE"/>
    <xs:enumeration value="PLANNED"/>
    <xs:enumeration value="UNAVAILABLE"/>
    <xs:enumeration value="PENDING_DISCONNECT"/>
    <xs:enumeration value="DISCONNECTED"/>
    ```

  - assignmentState

    The search returns entities in the specified assignment state, which is defined by the following enumeration values:

    ```
    <xs:enumeration value="PENDING_ASSIGN"/>
    <xs:enumeration value="ASSIGNED"/>
    ```


Copyright © 2014, 2026, Oracle and/or its affiliates.

```
<xs:enumeration value="PENDING_UNASSIGN"/>
<xs:enumeration value="UNASSIGNED"/>
<xs:enumeration value="DISCONNECTED"/>
<xs:enumeration value="PENDING_AVAILABLE"/>
<xs:enumeration value="PENDING_UNAVAILABLE"/>
<xs:enumeration value="PORTED"/>
<xs:enumeration value="UNAVAILABLE"/>
<xs:enumeration value="TRANSITIONAL"/>
```

– inventoryGroup

  The search returns entities associated with the specified inventory group.

  If searching for telephone number entities, you can specify the inventoryGroup geographicLocation and the search returns telephone number entities associated with inventory groups that are associated with the specified place. If searching for entities other than telephone numbers, the inventoryGroup geographicLocation is not used.

– geographicLocation

  The search returns inventory group entities associated with the specified place. If searching for entities other than inventory groups, geographicLocation is not used.

– quantity

  The search returns the specified quantity of entities. For example, if the search finds 1,000 entities and the criteria specifies a quantity of 50, the first 50 entities found are returned.

– reservation

  If you specify reservation information, DisassociateEntity ignores it; FindEntity is the only operation that uses the reservation element. See "FindEntity" for more information.

– lock

  Row locking is used to optimize concurrent resource allocation for consumable entities; however, DisassociateEntity does not use row locking.

  If you specify row-locking information, DisassociateEntity ignores it.

– criteriaItem

  The search returns entities based on data specified by criteriaItem, which is an unbounded structure that provides the ability to specify the following:

  * Name of a criteria item as defined by the *Entity*SearchCriteria class, where *Entity* is the name of a specific entity such as TelephoneNumber, LogicalDevice, and so forth (see "Determining Criteria Item Names")
  * Value of specified criteria item
  * Enumerated operand with which to evaluate the specified criteria item and corresponding specified value:

    ```
    <xs:enumeration value="EQUALS" />
    <xs:enumeration value="NOT_EQUALS" />
    <xs:enumeration value="BEGINS_WITH" />
    <xs:enumeration value="ENDS_WITH" />
    <xs:enumeration value="CONTAINS" />
    ```

– property

  The search returns entities with the data specified by property, which is an unbounded structure that provides the ability to specify the following:

  * Name of a characteristic

Copyright © 2014, 2026, Oracle and/or its affiliates.

* Value of specified characteristic
* Enumerated operand with which to evaluate the specified characteristic and corresponding specified value:

```
<xs:enumeration value="EQUALS" />
<xs:enumeration value="NOT_EQUALS" />
<xs:enumeration value="BEGINS_WITH" />
<xs:enumeration value="ENDS_WITH" />
<xs:enumeration value="CONTAINS" />
```

Multiple Entities

You specify one association type per request, and two sets of search criteria per request; one to find the source entities to associate, and one to find the target entities to associate.

Example

Example 3-15 shows a request that disassociates the specified source and target entities. The source entity (only one source entity is specified in this example) is the MobileServingArea inventory group. The target entities are all logical devices created from the SIMCard specification that are installed and unassigned.

Example 3-15 disassociateEntityRequest

```
<nsrm:disassociateEntityRequest>
    <nsrm:sourceEntities>
        <nsrm:entityType xsi:type="ig:InventoryGroupType" />
        <ent:criteria>
            <ent:entityType xsi:type="ig:InventoryGroupType" />
            <ent:specification>
                <spec:name>MobileServingArea</spec:name>
            </ent:specification>
        </ent:criteria>
    </nsrm:sourceEntities>
    <nsrm:targetEntities>
        <nsrm:entityType xsi:type="ld:LogicalDeviceType" />
        <ent:criteria>
            <ent:entityType xsi:type="ld:LogicalDeviceType" />
            <ent:specification>
                <spec:name>SIMCard</spec:name>
            </ent:specification>
            <ent:adminState>INSTALLED</ent:adminState>
            <ent:assignmentState>UNASSIGNED</ent:assignmentState>
        </ent:criteria>
    </nsrm:targetEntities>
</nsrm:disassociateEntityRequest>
```

disassociateEntityResponse

disassociateEntityResponse returns information about the disassociated entities. The information returned in the response is dependent upon the entity types that were disassociated, as specified in the request.

disassociateEntityResponse returns an error message when:

- The call to the UIM API fails

Copyright © 2014, 2026, Oracle and/or its affiliates.

ImportEntity

The ImportEntity operation enables external systems to send a request to import certain entities into UIM.

importEntityRequest

You specify a SOAP attachment that is a spreadsheet containing the entities for import. The spreadsheet must be a specific format: see "Spreadsheet Format".

For information about SOAP attachments, see the SoapUI website at:

http://www.soapui.org/SOAP-and-WSDL/adding-headers-and-attachments.html#2

Multiple Entities

You can import multiple entities of varying entity types per request. For example, a spreadsheet may define fifteen rows that result in the import of five telephone number entities, five logical device entities, and five physical device entities.

Example

Example 3-16 shows importEntityRequest, which uses of a SOAP attachment.

Example 3-16 importEntityRequest

```
<soapEnv:Envelope xmls:soapenv="http://schemas.xmlsoap.org/soap/envelope/".....>
    <soapEnv:Header/>
    <soapEnv:Body>
        <nsrm:importEntityRequest>
            <octet/>
        </nsrm:importEntityRequest>
    <soapEnv:Body/>
</soapEnv:Envelope/>
```

Spreadsheet Format

Example 3-17 shows an excerpt from the *UIM_CONFIG_PATH***/config/importExport.properties** file, which defines the column names for the ImportEntity spreadsheet. You can change the name of existing column names defined in this file, but you cannot add new columns to the spreadsheet by defining additional column names in this file.

Example 3-17 importExport.properties: Column Names

```
#Use import properties to customize column names for excel of ImportEntity
import.rowNumber=rowNumber
import.action=action
import.entityType=entityType
import.id=id
import.rangeFromID=rangeFromID
import.rangeToID=rangeToID
import.quantity=quantity
import.name=name
import.rangeFromName=rangeFromName
import.rangeToName=rangeToName
import.specification=specification
import.description=description
```


Copyright © 2014, 2026, Oracle and/or its affiliates.

```
import.attribute=attribute
import.characteristic=characteristic
import.relatedRow=relatedRow
```

Table 3-9 describes the columns defined in the **importExport.properties** file.

Table 3-9 Spreadsheet Column Names

Name	Description
rowNumber	rowNumber is required and must be unique; it is a numeric value that you assign to each row in the spreadsheet. UIM uses rowNumber for referencing other rows in the spreadsheet for creating relationships.
action	action is required and must be one of the following values: • **create**: Creates one or more entities and associates them with other entities • **associate**: Associates one or more entities with other entities • **information**: Provides search criteria information for the specified entityType
entityType	entityType is required and must be one of the following values: • InventoryGroup • IPv4Address • IPv6Address • IPSubnet • LogicalDevice • LogicalDeviceAccount • PhysicalDevice • TelephoneNumber
id	When the action is **create** and the specification requires a user-provided ID, id is required. When the action is **create** and the specification automatically generates an ID, do not specify id. When the action is **associate** or **information**, id can optionally specify search criteria. The value can be numeric or alphanumeric.
rangeFromID	rangeFromID populates the same database table and column as id, but it is used when creating entities in bulk. When the action is **create**, and you are creating LogicalDevice, LogicalDeviceAccount, or PhysicalDevice entities in bulk, and the specification requires a user-provided ID, rangeFromID is required and specifies the starting ID. When rangeFromID is specified, you must also specify quantity. See "quantity". When the action is **create**, and you are creating TelephoneNumber entities, rangeFromID is not valid. For TelephoneNumber entities, you specify rangeFromName and rangeToName. See "rangeFromName" and "rangeToName". When the action is **associate** or **information**, rangeFromID, along with rangeToID, can optionally specify search criteria. The value must be numeric.
rangeToID	When the action is **associate** or **information**, rangeToID, along with rangeFromID, can optionally specify search criteria. The value must be numeric.
quantity	When the action is **create**, and you are creating LogicalDevice, LogicalDeviceAccount, or PhysicalDevice entities in bulk, quantity specifies the number of entities to create. When the action is **create**, and you are creating TelephoneNumber entities, quantity is not valid. For TelephoneNumber entities, you specify rangeFromName and rangeToName. See "rangeFromName" and "rangeToName". The value must be numeric.

Copyright © 2014, 2026, Oracle and/or its affiliates.

Table 3-9 (Cont.) Spreadsheet Column Names

Name	Description
name	When the action is **create**, name specifies the name of the entity or entities you are creating. When creating entities in bulk, all entities are created with the same name. When the action is **associate** or **information**, name can optionally specify search criteria. The value can be alphanumeric or numeric.
rangeFromName	(rangeFromName populates the same database table and column as name.) When the action is **create**, and you are creating TelephoneNumber entities, rangeFromID is specified along with rangeToName. See "rangeToName". When the action is **create**, and you are creating LogicalDevice, LogicalDeviceAccount, or PhysicalDevice entities in bulk, rangeFromName specifies the starting name of the entities to create. When rangeFromName is specified in this scenario, you must also specify quantity. See "quantity". When the action is **associate** or **information**, rangeFromName, along with rangeToName, can optionally specify search criteria. The value must be numeric.
rangeToName	When the action is **create**, and you are creating TelephoneNumber entities, rangeToID is specified along with rangeFromName. See "rangeFromName". When the action is **associate** or **information**, rangeToName, along with rangeFromName, can optionally specify search criteria. The value must be numeric.
specification	When the action is **create**, specification is required to specify the specification used to create the entity. When the action is **associate** or **information**, specification can optionally specify search criteria. The value must represent an existing specification in UIM.
description	When the action is **create**, description specifies the name of the entity or entities you are creating. When creating entities in bulk, all entities are created with the same name. Specifying a description is optional. The value must be alphanumeric.
attribute	When the action is **create**, attribute specifies the attribute name/value pair per cell for the entity or entities you are creating. The name/value pair is specified as *attributeName=attributeValue*. For example, or myAttribute=123. The number of attribute columns depends on the number of attributes defined for the entity you are creating. For example, when creating a PhysicalDevice entity, your spreadsheet may specify four attribute columns containing the name/value pair for the following attributes: • networkLocation • physicalAddress • serialNumber • physicalLocation When creating entities in bulk, specifying *attributeName=attributeValue* creates all entities with same attribute and same attribute value. For example, if you specify myAttr=123, all entities are created with the myAttr attribute and all myAttr attribute values are m123. When the action is **associate** or **information**, attribute can specify search criteria. For example, *attributeName=attributeValue* searches for entities with the specified attribute and attribute value. For example, if you specify myAttr=123, the search looks for entities with the myAttr attribute that has an attribute value of 123. The value for any given attribute must match the data type defined for the attribute.

Copyright © 2014, 2026, Oracle and/or its affiliates.

Table 3-9 (Cont.) Spreadsheet Column Names

Name	Description
characteristic	When the action is **create**, characteristic specifies the characteristic name/value pair per cell for the entity or entities you are creating. The name/value pair is specified as *characteristicName=characteristicValue*. For example, myChar=123. The number of characteristic columns depends on the number of characteristics defined for the entity you are creating. The characteristic column works similarly to the attribute column. See "attribute" for an example. When creating entities in bulk, you can specify: • *characteristicName=characteristicValue* to create all entities with same characteristic and same characteristic value. For example, if you specify myChar=123, all entities are created with the myChar characteristic and all myChar characteristic values are 123. • *characteristicName*.rangeFrom=*characteristicStartValue* to create all entities with the same characteristic and a range of characteristic values. In this scenario, *characteristicStartValue* must be numeric. For example, if you specify myChar.rangeFrom=123, all entities are created with the myChar characteristic and characteristic values are 123, 124, 125, and so forth. When the action is **associate** or **information**, characteristic can specify search criteria as follows: • *characteristicName=characteristicValue* searches for entities with the specified characteristic and characteristic value. For example, if you specify myChar=123, the search looks for entities with the myChar characteristic that has a characteristic value of 123. • *characteristicName*.range=*characteristicStartValue, characteristicEndValue* searches for entities with the specified characteristic and range of characteristic values. In this scenario, *characteristicStartValue* and *characteristicEndValue* must be numeric. For example, if you specify myChar.range=123,200, the search looks for entities with the myChar characteristic that has a characteristic value ranging from 123 through 200. The value for any given characteristic must match the data type defined for the characteristic.

Copyright © 2014, 2026, Oracle and/or its affiliates.

Table 3-9 (Cont.) Spreadsheet Column Names

Name	Description
relatedRow	relatedRow specifies rowNumber from the row with which the current row entity will get paired (associated with a type of PAIR). When the action is **create** for the source row and target row, both rows must point to each other (the source row's pair value must reflect the target row's rowNumber, and the target row's pair value must reflect the source row's rowNumber). The value must be numeric. The following list shows all valid relations. (In the list, LD is LogicalDevice, LDA is LogicalDeviceAccount, PD is PhysicalDevice, TN is TelephoneNumber, and IG is InventoryGroup): • LD:LDA(1:*n*) When LD and LDA are source-target or target-source, *n* LDAs are associated with specified LD. • LD:PD(1:*n*) When LD and PD are source-target or target-source, *n* PDs are associated with specified LD. • LD:TN (*n*:*n*) When LD and TN are source-target or target-source, a preconfigured custom involvement is created with each pair of LD and TN. For example, when five LDs and five TNs are specified, the first LD and first TN are paired, the second LD and second TN are paired, and so forth. • LD:IG(*m*:*n*) When LD and IG are source-target or target-source, all LDs are associated to all IGs. For example, *m* x *n* associations are created. • PD:TN (*n*:*n*) When PD and TN are source-target or target-source, a preconfigured custom involvement is created with each pair of PD and TN. For example, when five PDs and five TNs are specified, the first PD and first TN are paired, the second PD and second TN are paired, and so forth. • PD:IG (*m*:*n*) When PD and IG are source-target or target-source, all PDs are associated to all IGs. • LDA:IG (*m*:*n*) When LDA and IG are source-target or target-source, all LDAs are associated to all IGs. • TN:IG (*m*:*n*) When TN and IG are source-target or target-source, all TNs are associated to all IGs.

Note

The following columns from Table 3-9 do not support the IPAM-specific entities of IPv4Address, IPv6Address or IPSubnet:

- rangeFromID, rangeToID
- quantity
- rangeFromName, rangeToName
- relatedRow

Table 3-10 shows an example input spreadsheet for the request. The table shows various values for the id, rangeFromID, rangeToID, quantity, name, rangeFromName, and

Copyright © 2014, 2026, Oracle and/or its affiliates.

rangeToName columns when using the **ImportEntity** operation. In this example, all rows specify the **create** action and the **LogicalDevice** entityType and a dash represents no data for a cell.

> **Note**
>
> The following columns were omitted from the spreadsheet example for clarity:
>
> - action (required column)
> - entityType (required column)
> - description
> - attribute
> - characteristic
> - relatedRow

Table 3-11 shows the results of processing each row in Table 3-10.

Table 3-10 Example Spreadsheet

row Number	id	range FromID	range ToID	quantity	name	range FromName	range ToName	specification
1	-	-	-	-	testLD1	-	-	LDSpec
2	-	-	-	-	testLD2	-	-	LDSpec
3	-	-	-	2	testLD3	-	-	LDSpec
4	-	-	-	2	-	11001	-	LDSpec
5	-	-	-	2	testLD5	-	-	LDSpec
6	-	-	-	2	-	12001	-	LDSpec
7	1000	-	-	-	testLD7	-	-	LDSpec_ManualID
8	1001	-	-	-	testLD8	-	-	LDSpec_ManualID
9	-	1002	-	2	testLD9	-	-	LDSpec_ManualID
10	-	1004	-	2	-	13001	-	LDSpec_ManualID
11	-	1006	-	2	testLD11	-	-	LDSpec_ManualID
12	-	1008	-	2	-	14001	-	LDSpec_ManualID

Table 3-11 Example Spreadsheet Results

row Number	Result
1	One LogicalDevice entity is created from the LDSpec specification. Then entity is named testLD1. The entity id is generated.
2	One LogicalDevice entity is created from the LDSpec specification. Then entity is named testLD2. The entity id is generated.

Copyright © 2014, 2026, Oracle and/or its affiliates.

Table 3-11 (Cont.) Example Spreadsheet Results

row Number	Result
3	Two LogicalDevice entities are created from the LDSpec specification. Both entities are named testLD3. The entity ids are generated.
4	Two LogicalDevice entities are created from the LDSpec specification. The first entity is named 11001, and the second entity is named 11002. The entity ids are generated.
5	Two LogicalDevice entities are created from the LDSpec specification. Both entities are named testLD5. The entity ids are generated.
6	Two LogicalDevice entities are created from the LDSpec specification. The first entity is named 12001, and the second entity is named 12002. The entity ids are generated.
7	One LogicalDevice entity are created from the LDSpec_ManualID specification. The entity is named testLD7. The entity id is 1000.
8	One LogicalDevice entity are created from the LDSpec_ManualID specification. The entity is named testLD8. The entity id is 1001.
9	Two LogicalDevice entities are created from the LDSpec_ManualID specification. Both entities are named testLD9. The first entity id is 1002 and the second entity id is 1003.
10	Two LogicalDevice entities are created from the LDSpec_ManualID specification. The first entity is named 13001, and the second entity is named 13002. The first entity id is 1004 and the second entity id is 1005.
11	Two LogicalDevice entities are created from the LDSpec_ManualID specification. Both entities are named testLD11. The first entity id is 1006 and the second entity id is 1007.
12	Two LogicalDevice entities are created from the LDSpec_ManualID specification. The first entity is named 14001, and the second entity is named 14002. The first entity id is 1008 and the second entity id is 1009.

Spreadsheet Row Order

The rows in the spreadsheet must be provided to ImportEntity in a specific order. The order is based on a combination of the action, relatedRow, and entityType column values.

The spreadsheet row order must be:

1. **create**-**create** paired rows

   First, provide the rows of entities to create that are to be paired with another entity; that is, both rows specify the **create** action and both rows specify a relatedRow value that indicates each other.

Copyright © 2014, 2026, Oracle and/or its affiliates.

If you are creating entities of varying entity type, specify your **create**-**create** paired rows in the following entityType order:

- TelephoneNumber
- LogicalDevice
- LogicalDeviceAccount
- PhysicalDevice
- InventoryGroup

2. **create** rows

   Next, provide the rows of entities to create that are not to be paired with another entity; that is, rows that specify the **create** action and specify no relatedRow value.

   If you are creating entities of varying entity type, specify your **create** rows in the following entityType order:

   - TelephoneNumber
   - LogicalDevice
   - LogicalDeviceAccount
   - PhysicalDevice
   - InventoryGroup

3. **create-information** paired rows

   Next, provide the rows of entities to create for which additional information is provided in a corresponding row; that is, one row specifies the **create** action, one row specifies the **information** action, and both rows specify a relatedRow value that indicates each other.

   If you are creating entities of varying entity type, specify your **create**-**information** paired rows in the following entityType order:

   - TelephoneNumber
   - LogicalDevice
   - LogicalDeviceAccount
   - PhysicalDevice
   - InventoryGroup

4. **associate-information** paired rows

   Last, provide the rows of entities to associate for which additional information is provided in a corresponding information row; that is, one row specifies the **associate** action, one row specifies the **information** action, and both rows specify a relatedRow value that indicates each other.

   If you are associating entities of varying entity type, specify your **associate**-**information** paired rows in the following entityType order:

   - TelephoneNumber
   - LogicalDevice
   - LogicalDeviceAccount
   - PhysicalDevice
   - InventoryGroup

Copyright © 2014, 2026, Oracle and/or its affiliates.

importEntityResponse

importEntityResponse returns information about the imported entities. The information returned in the response is dependent upon the entity types that were imported, as specified in the request.

importEntityResponse returns an error message when the request specifies the following. (In the list, LD is LogicalDevice, LDA is LogicalDeviceAccount, PD is PhysicalDevice, and TN is TelephoneNumber):

- A rowNumber less than zero, or duplicate row numbers
- More than 10,000 rows
- A column name that is not defined in the properties file
- No action
- The **create** action with no specification
- The **create** action with a specification that does not exist
- A quantity of zero or less than zero
- The **create** action for an LD, LDA, or PD, and specifies a rangeToID or rangeToName (which is used only for TN)
- No id or rangeToID for Manual ID specifications of LD, LDA, or PD
- No quantity with rangeFromID for Manual ID specifications of LD, LDA, or PD
- A rangeFromID that is not numeric for LD, LDA, or PD
- Both id and quantity LD, LDA, or PD
- Both id and rangeFromID LD, LDA, or PD
- Both id and rangeFromID for auto-generated ID specifications of LD, LDA, or PD
- Both Name and rangeFromName for LD, LDA, PD, or TN
- No name and no rangeFromName for LD, LDA, PD, or TN
- A rangeFromName that is not numeric LD, LDA, PD, or TN
- An incorrect attribute for any entity
- No rangeFromName and no rangeToName for TN
- id, rangeFromID, rangeToID, quantity, rangeFromName, or rangeToName for InventoryGroup
- More than one LD for LD-LDA association
- More than one LD for LD-PD association
- Pairing information for association when action is create in both rows does not match
- Invalid association types (For example, LD-LD, PD-LDA, and so forth)
- An unequal number of entities for LD-TN or PD-TN
- Incorrect relatedRow information (For example, relatedRow does not specify the **information** action)
- An InventoryGroup that does not exist
- The call to the UIM API fails

Copyright © 2014, 2026, Oracle and/or its affiliates.

ExportEntity

The ExportEntity operation enables external systems to send a request to export certain existing entities from UIM. The entities found for export are returned in a spreadsheet through a SOAP attachment in the response.

exportEntityRequest

You must specify search criteria to find existing entities to export. Search criteria is specified using the <criteria> element, which includes entity type.

You specify the type of entity to export based on the entity types defined in the schema files. Each entity type defines different elements that pertain specifically to the entity type, which you use to specify what to update. Table 3-12 lists the valid entity types and the schema files in which they are defined.

Table 3-12 Entity Types for ExportEntity

Entity Type	Schema File
LogicalDeviceType	LogicalDevice.xsd
LogicalDeviceAccountType	LogicalDevice.xsd
PhysicalDeviceType	PhysicalDevice.xsd
TelephoneNumberType	Number.xsd

<criteria> defines the following search criteria:

- specification

  The search returns entities created from the specified specification.

- adminState

  The search returns entities in the specified administrative state, which is defined by the following enumeration values:

```
<xs:enumeration value="END_OF_LIFE"/>
<xs:enumeration value="INSTALLED"/>
<xs:enumeration value="PENDING_INSTALL"/>
<xs:enumeration value="PENDING_REMOVE"/>
<xs:enumeration value="PENDING_UNAVAILABLE"/>
<xs:enumeration value="PENDING_AVAILABLE"/>
<xs:enumeration value="PLANNED"/>
<xs:enumeration value="UNAVAILABLE"/>
<xs:enumeration value="PENDING_DISCONNECT"/>
<xs:enumeration value="DISCONNECTED"/>
```

- assignmentState

  The search returns entities in the specified assignment state, which is defined by the following enumeration values:

```
<xs:enumeration value="PENDING_ASSIGN"/>
<xs:enumeration value="ASSIGNED"/>
<xs:enumeration value="PENDING_UNASSIGN"/>
<xs:enumeration value="UNASSIGNED"/>
<xs:enumeration value="DISCONNECTED"/>
<xs:enumeration value="PENDING_AVAILABLE"/>
```


Copyright © 2014, 2026, Oracle and/or its affiliates.

```
<xs:enumeration value="PENDING_UNAVAILABLE"/>
<xs:enumeration value="PORTED"/>
<xs:enumeration value="UNAVAILABLE"/>
<xs:enumeration value="TRANSITIONAL"/>
```

- inventoryGroup

  The search returns entities associated with the specified inventory group.

- quantity

  The search returns the specified quantity of entities. For example, if the search finds 1,000 entities and the criteria specifies a quantity of 50, the first 50 entities found are returned.

- reservation

  If you specify reservation information, ExportEntity ignores it; FindEntity is the only operation that uses the reservation element. See "FindEntity" for more information.

- lock

  Row locking is used to optimize concurrent resource allocation for consumable entities; however, ExportEntity does not use row locking.

  If you specify row-locking information, ExportEntity ignores it.

- criteriaItem

  The search returns entities based on data specified by criteriaItem, which is an unbounded structure that provides the ability to specify the following:

  – Name of a criteria item as defined by the *Entity*SearchCriteria class, where *Entity* is the name of a specific entity such as TelephoneNumber, LogicalDevice, and so forth (see "Determining Criteria Item Names")

  – Value of specified criteria item

  – Enumerated operand with which to evaluate the specified criteria item and corresponding specified value:

  ```
  <xs:enumeration value="EQUALS" />
  <xs:enumeration value="NOT_EQUALS" />
  <xs:enumeration value="BEGINS_WITH" />
  <xs:enumeration value="ENDS_WITH" />
  <xs:enumeration value="CONTAINS" />
  ```

- property

  The search returns entities with the data specified by property, which is an unbounded structure that provides the ability to specify the following:

  – Name of characteristic

  – Value of specified characteristic

  – Enumerated operand with which to evaluate the specified characteristic and corresponding specified value:

  ```
  <xs:enumeration value="EQUALS" />
  <xs:enumeration value="NOT_EQUALS" />
  <xs:enumeration value="BEGINS_WITH" />
  <xs:enumeration value="ENDS_WITH" />
  <xs:enumeration value="CONTAINS" />
  ```


Copyright © 2014, 2026, Oracle and/or its affiliates.

Multiple Entities

You can export multiple entities of varying entity types per request. For example, the request search criteria may return telephone number entities, logical device entities, and physical device entities, and the varying types of entities are exported.

Example

Example 3-18 shows search criteria to find LogicalDevice entities created from the SIMCard specification that are installed and unassigned.

Example 3-18 exportEntityRequest

```
<nsrm:exportEntityRequest>
    <ent:criteria>
        <ent:entityType xsi:type="ld:LogicalDeviceType" />
        <ent:specification>
            <spec:name>SIMCard</spec:name>
        </ent:specification>
        <ent:adminState>INSTALLED</ent:adminState>
        <ent:assignmentState>UNASSIGNED</ent:assignmentState>
    </ent:criteria>
</nsrm:exportEntityRequest>
```

exportEntityResponse

exportEntityResponse returns a spreadsheet containing the exported entities as a SOAP attachment in the response. The entities returned in the response are dependent upon the entity types that were exported, as specified in the request.

Example 3-19 shows an excerpt from the *UIM_CONFIG_PATH***/config/importExport.properties** file, which defines the column names for the ExportEntity spreadsheet. These column names are common across all ExportEntity-supported entity types. You can change the name of existing column names defined in this file, but you cannot add new columns to the spreadsheet by defining additional column names in this file.

Example 3-19 importExport.properties: Column Names

```
#Use export properties to customize column names for excel of ExportEntity
export.slNo=serialNumber
export.entityType=entityType
export.id=ID
export.name=name
export.specification=specification
export.description=description
export.attribute=attribute
export.characteristic=characteristic
```

Exported entities are grouped by entity type per sheet. Example 3-20 shows an excerpt from the *UIM_CONFIG_PATH***/config/importExport.properties** file, which defines the sheet names. You can change the name of existing sheets, but you cannot add new sheets to the spreadsheet by defining additional sheets in this file.

Example 3-20 importExport.properties: Sheet Names

```
#Use export properties to customize name sheets for different entities
export.sheet.telephoneNumber=TelephoneNumber
export.sheet.logicalDevice=LogicalDevice
```


Copyright © 2014, 2026, Oracle and/or its affiliates.

```
export.sheet.logicalDeviceAccount=LogicalDeviceAccount
export.sheet.physicalDevice=PhysicalDevice
```

The sheets are created when entities are found for export. So, all sheets are not always returned. For example, if the specified search criteria finds only logical devices to export, only the LogicalDevice sheet is returned.

If no entities are found for export, no sheets are created, and no attachment is returned in the response.

TelephoneNumber Sheet

If telephone number entities are found for export based on the specified search criteria, the TelephoneNumber sheet is created. This sheet contains the column names that are common across all ExportEntity-supported entity types (shown in Example 3-19), as well as column names that represent telephone number attributes.

Example 3-21 shows an excerpt from the *UIM_CONFIG_PATH***/config/** **importExport.properties** file, which defines the available telephone number attribute column names you can add to the sheet by setting to **true**, or omit from the sheet by setting to **false**.

You can change the name of existing column names defined in this file, but you cannot add new columns to the TelephoneNumber sheet by defining additional column names in this file.

Example 3-21 importExport.properties: TN-Specific Column Names

```
#Use TelephoneNumber export properties for exporting attributes
tn.export.partition=true
tn.export.owner=true
tn.export.permissions=true
```

LogicalDevice Sheet

If logical device entities are found for export based on the specified search criteria, the LogicalDevice sheet is created. This sheet contains the column names that are common across all ExportEntity-supported entity types (shown in Example 3-19), as well as column names that represent logical device attributes.

Example 3-22 shows an excerpt from the *UIM_CONFIG_PATH***/config/** **importExport.properties** file, which defines the available logical device attribute column names you can add to the sheet by setting to **true**, or omit from the sheet by setting to **false**.

You can change the name of existing column names defined in this file, but you cannot add new columns to the LogicalDevice sheet by defining additional column names in this file.

Example 3-22 importExport.properties: LD-Specific Column Names

```
#Use LogicalDevice export properties for exporting attributes
ld.export.partition=true
ld.export.owner=true
ld.export.permissions=true
ld.export.networkLocationEntityCode=true
ld.export.deviceIdentitfier=true
```

LogicalDeviceAccount Sheet

If logical device account entities are found for export based on the specified search criteria, the LogicalDeviceAccount sheet is created. This sheet contains the column names that are common across all ExportEntity-supported entity types (shown in Example 3-19), as well as column names that represent logical device account attributes.

Copyright © 2014, 2026, Oracle and/or its affiliates.

Example 3-23 shows an excerpt from the *UIM_CONFIG_PATH***/config/** **importExport.properties** file, which defines the available logical device account attribute column names you can add to the sheet by setting to **true**, or omit from the sheet by setting to **false**.

You can change the name of existing column names defined in this file, but you cannot add new columns to the LogicalDeviceAccount sheet by defining additional column names in this file.

Example 3-23 importExport.properties: LDA-Specific Column Names

```
#Use LogicalDeviceAccount export properties for exporting attributes
lda.export.partition=true
lda.export.owner=true
lda.export.permissions=true
```

PhysicalDevice Sheet

If physical device entities are found for export based on the specified search criteria, the PhysicalDevice sheet is created. This sheet contains the column names that are common across all ExportEntity-supported entity types (shown in Example 3-19), as well as column names that represent physical device attributes.

Example 3-24 shows an excerpt from the *UIM_CONFIG_PATH***/config/** **importExport.properties** file, which defines the available physical device attribute column names you can add to the sheet by setting to **true**, or omit from the sheet by setting to **false**.

You can change the name of existing column names defined in this file, but you cannot add new columns to the PhysicalDevice sheet by defining additional column names in this file.

Example 3-24 importExport.properties: PD-Specific Column Names

```
#Use PhysicalDevice export properties for exporting attributes
pd.export.partition=true
pd.export.owner=true
pd.export.permissions=true
pd.export.networkLocation=true
pd.export.physicalLocation=true
pd.export.physicalAddress=true
pd.export.serialNumber=true
```

exportEntityResponse Faults

exportEntityResponse returns an error message when:

- The call to the UIM API fails

Determining Criteria Item Names

This section provides detailed information regarding determining criteria item names, as referenced from "FindEntity", "FindTNBlock", "UpdateEntity", "DeleteEntity", "ReserveEntity", "UnreserveEntity", "AssociateEntity", "DisassociateEntity", and "ExportEntity".

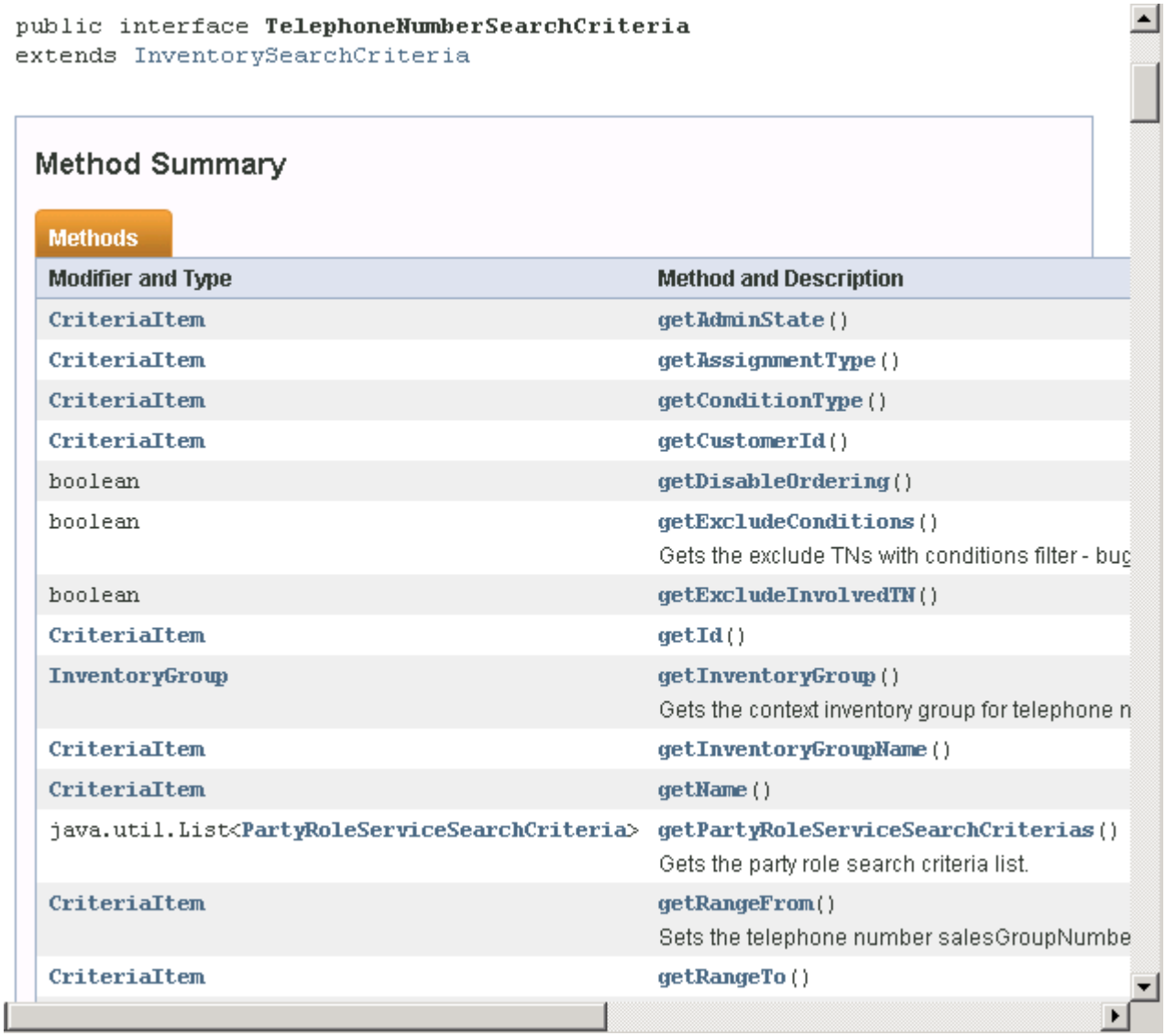
When using the criteriaItem structure, the search returns entities based on specified criteria item name/value pairs.

To determine the valid criteria item names you can specify, you must look in the Javadoc for the *Entity*SearchCriteria class, where *Entity* is the name of a specific entity such as TelephoneNumber, LogicalDevice, and so forth. For each *Entity*SearchCriteria class, you can

Copyright © 2014, 2026, Oracle and/or its affiliates.

only specify criteria items that are native to the class. For example, Figure 3-1 shows an excerpt of the TelephoneNumberSearchCriteria Javadoc.

Figure 3-1 Javadoc Example



In this example, the following are valid criteria item names you can specify:

- adminState
- assignmentState
- conditionType
- customerId
- id
- inventoryGroupName
- name

Copyright © 2014, 2026, Oracle and/or its affiliates.

- rangeFrom
- rangeTo
- blockSize

Criteria items that are native to the class are listed as type **CriteriaItem** in the Javadoc method summary **Modifier and Type** column. You cannot specify criteria items that are type boolean, java.util.List, or another UIM entity class, such as InventoryGroup as shown in the example.

> **Note**
>
> Be mindful that getter and setter method names alter the criteria item name.
>
> For example, the getAdminState() method spells AdminState with an uppercase “A", but the criteria item name is actually adminState with a lowercase “a".

For information about accessing the UIM Javadoc, see "Overview" in *UIM Developer's Guide*.

Customizing the Web Service Operations

You can customize any of the web service operations by creating a custom Java class that extends an existing UIM class. In the custom Java class, you can define methods that override and modify the methods defined in the parent class you are extending.

To customize web service operations:

1. Open the *UIM_CONFIG_PATH***/config/nsrm-ws.properties** file.

   The file lists the delegate web service classes and the delegate API classes. All of the delegate classes listed in the **nsrm-ws.properties** file are described in the Javadoc.

2. Use the Javadoc to determine which delegate class you want to customize.

   To access the Javadoc, enter the following in your Web browser:

   http://*server*:*port*/**ora_uim_javadoc**

   where *server* is the specific server on which the application is deployed and *port* is the port on which the application listens.

   For detailed instructions on accessing the Javadoc, see "Overview" in *UIM Developer's Guide*.

3. Create a custom Java class that extends a delegate class.

4. In the custom Java class, you can customize any of the methods defined in the parent class by defining the same methods in the child class, and modifying the methods as needed for your business requirements.

   Your custom code can also run a ruleset. For example, you may want to utilize existing functionality provided in the base rulesets. See "Overview" in *UIM Developer's Guide* for more information about rulesets, including how to run a ruleset from within custom code.

5. In the **nsrm-ws.properties** file:

   a. Copy and paste the property that defines the delegate class you extended.

   b. Comment out the original property that defines the delegate class you extended.

   c. Update the copied property to reflect the name of your custom class.

Copyright © 2014, 2026, Oracle and/or its affiliates.

For example:

```
#ws.delegate.TelephoneNumberType=oracle.communications.inventory.
#webservice.delegate.TelephoneNumberDelegate
ws.delegate.TelephoneNumberType=oracle.communications.inventory.
webservice.delegate.MyCustomTNDelegate
```

6. Deploy the custom code.

   For traditional UIM:

   - If your custom code resides within an Inventory cartridge project, you deploy the cartridge through Design Studio. See "Overview" in *UIM Developer's Guide* for more information.
   - If your custom code resides within a Web Archive (WAR) file in the **custom.ear** file, you deploy the **custom.ear** file through the WebLogic Administration Console. See "Developing Custom SOAP Web Services" for more information.

   For cloud native UIM:

   - Deploy the custom code into UIM cloud native by rebuilding the customized image and creating the instance with generated image. For more information, see "Customizing Images" in *UIM Cloud Native Deployment Guide*.

Extending Web Service Requests and Responses

You can extend web service requests and responses by extending **GenericHandler.class**, which supports the use of SOAP handlers and which is used by the UIM Network Resource Management Web Service.

Extending Network Resource Management Web Service requests and responses is done the same way as extending Service Fulfillment Web Service requests and responses. Both web services are packaged together in the **InventoryWS.war** file, so all of the steps are the same. See "Extending Web Service Requests and Responses" in the Service Fulfillment Web Service chapter for detailed instructions about extending web service requests and responses.

Deploying, Testing, and Securing the Web Service

Information about deploying, testing, and securing the web service is described in "Deploying, Testing, and Securing UIM Web Services".

Copyright © 2014, 2026, Oracle and/or its affiliates.

4

Developing Custom SOAP Web Services

This chapter provides information about integrating Oracle Communications Unified Inventory Management (UIM) with external systems by developing custom SOAP web services. It describes the approach to developing SOAP web services and the guidelines you should follow.

About the UIM Reference Web Service

The chapter uses the UIM Reference Web Service as an example that you can extend.

> **Note**
>
> Previous Reference Web Service operations were deprecated in earlier releases. The deprecated operations have now been removed from the **reference_webservice.zip** file.

The UIM Reference Web Service is part of the UIM Software Developer's Kit (SDK). The UIM SDK provides the resources required to build an Inventory cartridge in Design Studio. For more information about the UIM SDK, see "Overview" in *UIM Developer's Guide*.

This chapter assumes you are using Design Studio to develop custom web services. If you use an integrated development environment (IDE) other than Design Studio, you can ignore the **.classpath** and **.project** files in the **reference_webservice.zip** file.

You can view the contents of **reference_webservice.zip** file in Oracle Communications Service Catalog and Design - Design Studio by importing the archive ZIP file into Design Studio. The ZIP file contains several types of files including the following:

- WSDL File

  The **ReferenceUim.wsdl** file defines the CreateLogicalDevice web service operation that creates a logical device. CreateLogicalDevice also defines an input, an output, and the possible faults that can be thrown.

  See "About the WSDL File" for more information about the **ReferenceUim.wsdl** file.

- Schema Files

  The schema files define XML structures for the inputs, outputs, faults and operation definitions of the Reference Web Service.

  See "About the Schema Files" for more information about the schema files.

- Java Source Files

  The Java source files provide the web service operation code. For example, these source files provide the following:

  – Input request and output response XML mapping

  – An API manager call to UIM core for the operation

Copyright © 2014, 2026, Oracle and/or its affiliates.

  - Transaction management for the operation with the commit or rollback result

  See "Developing the Web Service" for more information about the Java source files, including a listing and description of each type of class file, and information about which files need to be created or modified.

- Ant Build File

  The **build.xml** file defines Ant targets that you can run to build a custom web service. Ant targets are a set of executable tasks defined in the **build.xml** file. See "About the Ant Build File" for more information.

About the WSDL and Schema Files

The Reference Web Service operation is defined by the **ReferenceUim.wsdl** file, and is supported by several schema files. The WSDL file and supporting schema files are located in the *UIM_SDK_Home***/webservices/reference_webservice.zip** file, where *UIM_SDK_Home* is the local directory for the UIM SDK.

About the WSDL File

The WSDL file is located in the **wsdl** directory of the **reference_webservice.zip** file. The WSDL file defines the web service operation CreateLogicalDevice. This operation defines a request, a response, and the possible faults that can be thrown on error. For example, the WSDL file defines the following for the CreateLogicalDevice operation:

- createLogicalDeviceRequest
- createLogicalDeviceResponse

The request, response, and faults each define an XML structure that is defined in the supporting schema files. Example 4-1 shows the port definition, the operation, and the input request message.

Example 4-1 WSDL File Excerpt

```
<wsdl:portType name="ReferenceUimPort">
        <wsdl:operation name="CreateLogicalDevice">
       <wsdl:input message="invws:CreateLogicalDeviceRequest" />
       <wsdl:output message="invws:CreateLogicalDeviceResponse" />
       <wsdl:fault name="InventoryFault" message="invws:InventoryFault" />
       <wsdl:fault name="ValidationFault" message="invws:ValidationFault" />
    </wsdl:operation>
</wsdl:portType>
.
.
.
<wsdl:message name="CreateLogicalDeviceRequest">
    <wsdl:part name="createLogicalDeviceRequest"
               element="invldmsgs:createLogicalDeviceRequest"/>
</wsdl:message>
```

This WSDL excerpt shows the message CreateLogicalDeviceRequest is defined by the element createLogicalDeviceRequest. createLogicalDeviceRequest references the **invldmsgs** namespace which indicates where the XML structure is defined. See "About Namespaces" for more information.

Copyright © 2014, 2026, Oracle and/or its affiliates.

About the Schema Files

There are several schema files that support the Reference Web Service operation. These schemas are categorized as reference schemas and web service schemas.

Reference Schemas

The reference schemas define common elements used by all of the UIM web services, not just by the Reference Web Service. These elements are defined in the framework and then referenced in the various WSDL files.

The reference schemas are:

- InventoryCommon.xsd
- InventoryFaults.xsd
- FaultRoot.xsd

The reference schemas are contained in the **uim-webservices-framework.jar**. You can copy them into your workspace using the **get-framework-files** Ant target defined in the **build.xml** file. The **build.xml** file is contained in the **reference_webservice.zip** file. See "About the Ant Build File" for more information.

> **Note**
>
> The reference schemas use the **Inventory.xsdconfig** file to map XML namespaces to Java packages.

Web Service Schemas

Within the **reference_webservice.zip** file, the example schema file is located in the **wsdl/schemas** directory. The web service schema defines elements specific to the web service, such as the request structures, the response structures, and any fault structures.

The example web service schema file name is **LogicalDeviceMessages.xsd**.

> **Note**
>
> The web service schemas use the **type-mapping.xsdconfig** file to map XML namespaces to Java packages.

About Namespaces

The WSDL file defines a namespace to avoid naming conflicts. You use the namespace to determine the schema file location of the schema reference. Example 4-2 shows how a namespace defined in the WSDL file correlates to the supporting schema files.

In this example, the **ReferenceUim.wsdl** defines and references the **invldmsgs** namespace.

Example 4-2 Namespace Example

.
.

Copyright © 2014, 2026, Oracle and/or its affiliates.

```
.
xmlns:invldmsgs="http://xmlns.oracle.com/communications/inventory/webservice/
logicaldevice"
.
.
.
<xsd:import namespace=
    "http://xmlns.oracle.com/communications/inventory/webservice/logicaldevice"
     schemaLocation="./schemas/LogicalDeviceMessages.xsd"/>
     .
     .
     .
    <wsdl:message name="CreateLogicalDeviceRequest">
                <wsdl:part name="createLogicalDeviceRequest"
           element="invldmsgs:createLogicalDeviceRequest"/>
       </wsdl:message>
    .
    .
    .
```

The CreateLogicalDeviceRequest message declaration tells you that createLogicalDeviceRequest is defined in the schema file that supports the **invldmsgs** namespace. A search for the namespace and for the following string reveals that the **LogicalDeviceMessages.xsd** schema file defines the structures for the **invldmsgs** namespace:

```
xmlns.oracle.com/communications/inventory/webservice/logicaldevice
```

After you determine that the **LogicalDeviceMessages.xsd** schema file defines the XML structure, you can navigate through the schema files to determine child XML structures if applicable.

Refer to the following website for more information on namespaces:

https://www.w3.org/TR/REC-xml-names/

About the Ant Build File

The **build.xml** file defines Ant targets that you can run to build a custom web service. These Ant targets are a set of executable tasks that aid building a web service.

Table 4-1 describes the Ant targets defined in the **build.xml** file. See "Developing and Running Custom Web Services" for information about when to run these Ant targets. For information more information about running Ant targets within Design Studio, see "Overview" in *UIM Developer's Guide*.

Table 4-1 build.xml Ant Targets

Ant Target	Description
clean	Deletes the generated, temporary, and deliverable files and directories.
all	Initiates the complete build process for the web service supporting both HTTP and JMS. Identical to the **build.full** Ant target, it calls the following Ant targets in this order: **clean**,**generate-from-wsdl**, **build-service**.

Copyright © 2014, 2026, Oracle and/or its affiliates.

Table 4-1 (Cont.) build.xml Ant Targets

Ant Target	Description
copyResources	Copies the properties files that store localized error messages to the appropriate UIM deployment directory. These properties files are located in a ZIP file in the **config/resources/logging** directory and are copied to the *UIM_HOME*/**config/resources/logging** directory. **Note**: For cloud native deployments, if you want to add additional logging properties files, you add these files to solution cartridges or localization cartridges (ora_uim_localization_reference_cartproj.zip).
wspolicy	Updates the WAR file with the web service policy files, which describe the authentication and encryption mechanism for web service calls.
build.full	Initiates the complete build process for the web service supporting both HTTP and JMS. Calls the following Ant targets in this order: **clean**,**generate-from-wsdl**, **build-service**.
build.full.http	Initiates the complete build process for the web service WAR file supporting HTTP. Calls the following Ant targets in this order: **clean**,**generate-from-wsdl**, **build-service-http**.
build.full.jms	Initiates the complete build process for the web service WAR file supporting JMS. Calls the following Ant targets in this order: **clean**,**generate-from-wsdl**, **build-service-jms**.
build-service	Builds the web service WAR file for both HTTP and JMS, and stores it in the **webarchive** directory. The name of the WAR file is *wsdl_name***.war,** where *wsdl_name* is the name specified by the WSDL_NAME parameter in the **COMPUTERNAME.properties** file.
build-service-http	Builds the web service WAR file for HTTP and stores it in the **webarchive** directory. The name of the WAR file is *wsdl_name***HTTP.war**, where *wsdl_name* is the name specified by the WSDL_NAME parameter in the **COMPUTERNAME.properties** file.
build-service-jms	Builds the web service WAR file for JMS and stores it in the **webarchive** directory. The name of the WAR file is *wsdl_name***JMS.war**, where *wsdl_name* is the name specified by the WSDL_NAME parameter in the **COMPUTERNAME.properties** file.
build.deliverable	Builds the web service cartridge JAR file and stores it in the **deliverables** directory. Calls the **build.full** Ant target first to get a complete build for the WAR file.
generate-from-wsdl	Performs WSDL-to-Java conversions and generates object representations of the schemas. This includes business schema files such as **LogicalDevice.xsd**. Calls the **get-framework-files** Ant target.
get-framework-files	Extracts the framework schema files **InventoryCommon.xsd** and **InventoryFaults.xsd** from the **uim-webservices-framework.jar** file stored in the directory specified by APP_LIB parameter defined in the **COMPUTERNAME.properties** file. The framework schema XSD files are also located in the **schema_Inventory_webservice.zip** file in the UIM SDK.

Copyright © 2014, 2026, Oracle and/or its affiliates.

Table 4-1 (Cont.) build.xml Ant Targets

Ant Target	Description
extract.ear	Extracts the **application.xml** file from the EAR file specified by the EAR_PATH parameter defined in the **COMPUTERNAME.properties** file into the *reference_webservice_home*/**META-INF** directory, where *reference_webservice_home* is the location of the extracted **reference_webservice.zip** file. The **application.xml** file needs to be edited manually so that the EAR file can be updated for proper deployment of the web services. **Note**: The above description is not applicable for UIM cloud native deployments. Building customized image packages the WAR file.
update.ear	Updates the EAR file specified by the EAR_PATH parameter in the **COMPUTERNAME.properties** file by adding the generated web service WAR file and the edited **application.xml** file in the **webarchive** directory into the EAR file. The updated EAR file can be deployed to test the web services. **Note**: The above description is not applicable for UIM cloud native deployments. Building customized image packages the WAR file.

> **Note**
>
> The UIM Reference Web Service is an example web service to follow for developing custom web services. The Reference Web Service cannot be used for production deployments.

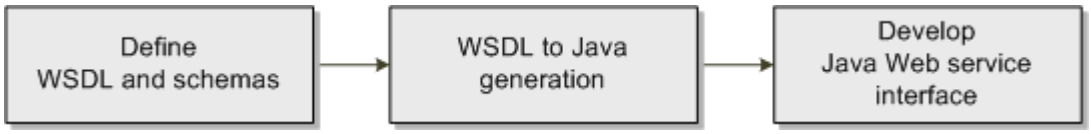
Guidelines for Developing Custom Web Services

This section describes the guidelines for developing a web service. It also contains class diagrams that represent the UIM Reference Web Service development classes.

Using the WSDL-First Approach to Developing Custom Web Services

The WSDL-first approach (also known as the top-down approach), is the recommended way to achieve interoperability, platform independence, and WSDL consistency across web services. Figure 4-1 shows the design and development sequence of the WSDL-first approach.

Figure 4-1 WSDL-First Design and Development Sequence



- Define WSDL and schemas

  Write the WSDL and the corresponding schemas (XSD files) to define the operations and data.

Copyright © 2014, 2026, Oracle and/or its affiliates.

- WSDL-to-Java generation

  Use the **build.xml** Ant targets provided by the Reference Web Service to generate Java source files based on the WSDL and schema definitions.

- Develop Java web service interface implementation

  Use the web service development environment and tools provided by the Reference Web Service to implement the web service interface by creating new Java source files and changing existing ones.

For example, the UIM Reference Web Service module was designed using the WSDL-first approach. This means that:

- The ReferenceUimPortImpl Java source file is generated based on the WSDL. This generation results in the WSDL operation being defined in the ReferenceUimPortImpl Java source file, but with no coding details.
- Within the ReferenceUimPortImpl Java source, an operation is manually modified to call its respective operation in the AdapterRouter class.
- The AdapterRouter class calls the respective operation in each individual Adapter class.
- The build generates the ReferenceUimPort interface based on the WSDL.

Class Diagrams

In the following class diagrams, *Action* represents a UIM business action such as Create, and *Entity* represents a UIM entity such as LogicalDevice. In the Reference Web Service, an example of *ActionEntity* is the CreateLogicalDevice operation. You should use the CreateLogicalDevice example as a template when creating custom web services. Consider the following recommendations:

- Follow the naming convention of *ActionEntity* for consistency on new operations.
- Follow the template code example for the user environment and transaction management functionality. See "Transaction Guidelines" for more information on transaction management.
- Make calls to UIM core functionality by invoking the API manager methods.

Some additional types of classes may be needed depending on the complexity of the web service operation that is developed. See "Creating Java Source Files" for more information about these additional types of classes.

Figure 4-2 through Figure 4-7 show the class designs provided by the Reference Web Service. These designs include request types, response types, fault types, adapters, and implementations.

You should follow the patterns illustrated in Figures 4-2 through 4-7 when you design interfaces and classes.

> **Note**
>
> Several of the XSD files in this section are not in the Reference Web Service even though they are referenced in the **ReferenceUim.wsdl** file. You pull these files into the wsdl directory by initiating the **get-framework-files** Ant target in the **build.xml** file. See "About the Ant Build File" for more information.

Copyright © 2014, 2026, Oracle and/or its affiliates.

Figure 4-2 shows the recommended class design for custom request types. **ReferenceUim.wsdl** specifies the element createLogicalDeviceRequest as type CreateLogicalDeviceRequestType, which is defined in **LogicalDeviceMessages.xsd**. CreateLogicalDeviceRequestType extends InventoryRequestType, which is defined in **InventoryCommon.xsd**.

Figure 4-2 Request Types

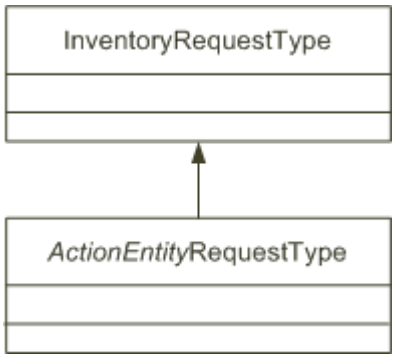


Figure 4-3 shows the recommended class design for custom response types. **ReferenceUim.wsdl** specifies the element createLogicalDeviceResponse as type CreateLogicalDeviceResponseType, which is defined in **LogicalDeviceMessages.xsd**. CreateLogicalDeviceResponseType extends InventoryResponseType, which is defined in **InventoryCommon.xsd**.

Figure 4-3 Response Types

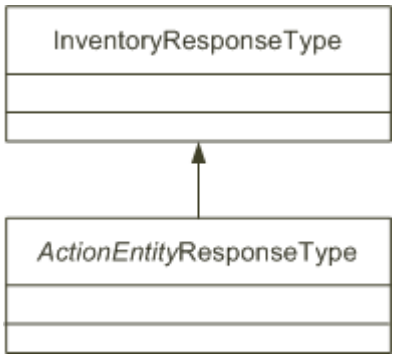


Figure 4-4 shows the recommended class design for custom fault types. **ReferenceUim.wsdl** uses the base fault types of InventoryFaultType and ValidationFaultType. These types are defined in **InventoryFault.xsd**. InventoryFaultType defines a sequence of faults, which are defined by ApplicationFaultType in **FaultRoot.xsd**.

Copyright © 2014, 2026, Oracle and/or its affiliates.

Figure 4-4 Fault Types

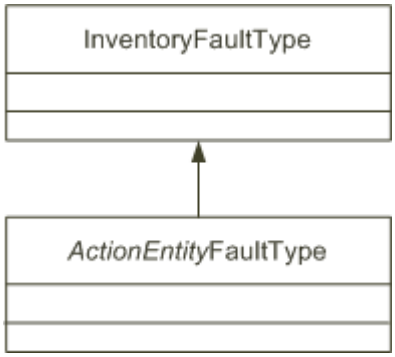


Figure 4-5 shows the recommended class design for custom adapters. The example adapter file is **LogicalDeviceAdapter.java,** which extends **InventoryAdapterRoot.java**. The UIM-owned **InventoryAdapterRoot.java** class extends the Platform-owned **AdaptorRoot.java** class.

Figure 4-5 Adapters

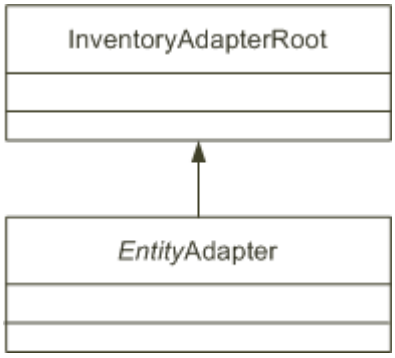


Figure 4-6 shows the recommended class design for the implementation class. The **ReferenceUim.wsdl** file is used to generate the **ReferenceUimPort.java** source file. The **ReferenceUimPortImpl.java** example file provides a skeleton class that implements the interface in the **ReferenceUimPort.java** source file.

> **Note**
>
> The sequence of the method signatures in the implementation class is important and must match the generated source. The generated source is based on the WSDL file definitions.

Copyright © 2014, 2026, Oracle and/or its affiliates.

Figure 4-6 Web Service Implementation

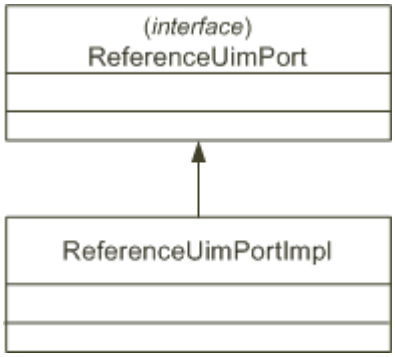
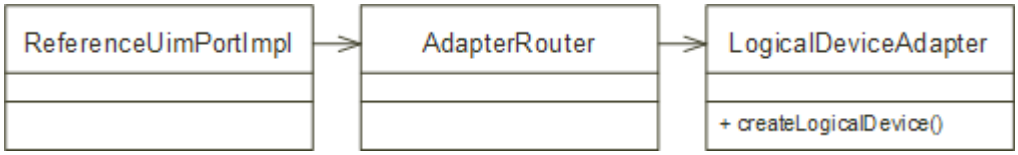


Figure 4-7 shows the class diagram with the ReferenceUimPortImpl, AdapterRouter and LogicalDeviceAdapter classes and their relationships. This design is recommended for building your own custom web services for other actions and entities similar to this example.

Figure 4-7 Implementation Pattern



WSDL Interface Guidelines

ReferenceUim.wsdl defines a single port type (a web service interface) that defines all of the exposed custom operations. When developing new web service operations, you create them within this single port.

The current recommended practice in creating UIM web service operations is to use a single port. Multiple ports are not defined. The only time you use multiple ports is when you have a port for HTTP and another for JMS. Multiple ports should not be used for categorically grouping operations.

Operation Signatures

Oracle recommends you follow naming patterns for the following:

- Operation names
- Request type names
- Response type names
- Fault type names

The naming patterns discussed in this section give consistency for the operations signatures.

Signature Components

A web service operation signature contains the following:

Copyright © 2014, 2026, Oracle and/or its affiliates.

- Operation name

  The pattern for defining an operation name is [*action*][*EntityName*] where action represents a verb action (such as create, update, delete) and the EntityName represents the entity acted upon (such as Equipment, LogicalDevice, TelephoneNumber). For example:

  - createEquipment
  - updateLogicalDevice
  - deleteTelephoneNumber

- Request type

  The pattern for defining a request type is *operationName*RequestType. A single request type is defined per operation. For example:

  - CreateEquipmentRequestType
  - UpdateLogicalDeviceRequestType
  - DeleteTelephoneNumberRequestType

- Response type

  The pattern for defining a response type is *operationName*ResponseType. A single response type is defined per operation. For example:

  - CreateEquipmentResponseType
  - UpdateLogicalDeviceResponseType
  - DeleteTelephoneNumberResponseType

- Fault types

  The pattern for defining a fault type is *businessFault*FaultType, where *businessFault* represents a specific business fault that might be thrown back to the user. Multiple business faults can be defined per operation. For example:

  - EquipmentNotUniqueFaultType
  - EquipmentNotFoundFaultType
  - NotAuthorizedFaultType

  Fault types contain the error codes and stack trace set by the business logic. One-to-one mapping between thrown business logic exceptions and the defined business faults is required to capture the different exceptions.

Signature Pattern and Examples

The signature pattern of an operation in the Reference Web Service is defined as follows:

```
public OperationNameResponseType operationName(
        OperationNameRequestType operationNameRequest) throws
            businessFault1FaultType,
            businessFault2FaultType,
            businessFaultNFaultType
```

For example, the createLogicalDevice method is defined in the **LogicalDeviceAdapter.java** file as the following:

```
public CreateLogicalDeviceResponseType createLogicalDevice(
                CreateLogicalDeviceRequestType createLogicalDeviceRequest)
throws
            InventoryFaultType,
            ValidationFaultType
```


Copyright © 2014, 2026, Oracle and/or its affiliates.

Table 4-2 shows the operation signature pattern on commonly used actions. In the table, *Entity* represents the name of the entity (such as Equipment, LogicalDevice, TelephoneNumber) acted upon by the operation.

Table 4-2 Operation Signature Examples

Action	Operation Signature
Create	`Create`*`Entity`*`ResponseType create`*`Entity`* `(Create`*`Entity`*`RequestType request) throws` *`businessFault1`*`FaultType,` *`businessFault2`*`FaultType`
Find	`Find`*`Entity`*`ResponseType find`*`Entity`* `(Find`*`Entity`*`RequestType request) throws` *`businessFault1`*`FaultType,` *`businessFault2`*`FaultType`
Update	`Update`*`Entity`*`ResponseType update`*`Entity`* `(Update`*`Entity`*`RequestType request) throws` *`businessFault1`*`FaultType,` *`businessFault2`*`FaultType`
Delete	`Delete`*`Entity`*`ResponseType delete`*`Entity`* `(Delete`*`Entity`*`RequestType request) throws` *`businessFault1`*`FaultType,` *`businessFault2`*`FaultType`
Calculate	`Calculate`*`Entity`*`ResponseType calculate`*`Entity`* `(Calculate`*`Entity`*`RequestType request) throws` *`businessFault1`*`FaultType,` *`businessFault2`*`FaultType`

Schema Guidelines

A custom web service schema is represented by multiple XSD files. The UIM API-level entity definitions closely follow the TM Forum (TMF) SID standard. Using XSD files that parallel the UIM APIs ensures SID standard compliance. For example, you build the XSD files parallel to business entities such as Service, Equipment, LogicalDevice, and so forth.

Keeping the XSD files separate from the WSDL makes the WSDL independent of web services and reusable across other software technologies. XSD files differ from WSDL files because they contain data structure definitions. The WSDL references these data structure definitions, but does not define them. Also, naming standards for the WSDL do not include **Type** in the name; naming standards for the schema do include **Type** in the name.

For example, the **ReferenceUim.wsdl** file defines createLogicalDeviceRequest as type CreateLogicalDeviceRequestType, which is defined in the **LogicalDeviceMessages.xsd** file. Similarly, the **ReferenceUim.wsdl** file defines createLogicalDeviceResponse as type CreateLogicalDeviceResponseType in the **LogicalDeviceMessages.xsd** file.

Copyright © 2014, 2026, Oracle and/or its affiliates.

Transaction Guidelines

The Reference Web Service performs transaction actions in a specific order when managing operation transactions.

> **Note**
>
> You must follow the steps in this order or transaction errors may occur, which can be hard to debug.

To correctly manage the transaction, you write code that performs the following steps:

1. Start the user environment.
2. Start the transaction.
3. Set the user environment on the transaction.
4. Set up the request, call the API method on the entity manager class, and manage the response.
5. Commit or rollback the transaction.
6. Ensure a rollback is completed if an error occurred.
7. Ensure the user environment is ended with a call to the **endUserEnvironment** method on success or failure.

Example 4-3 provides a code section of how to manage the user environment, transaction and API manager call. The code section contains the recommended steps described previously with the relevant code lines in bold.

This code section is taken from the LogicalDeviceAdapter class with some logging logic removed. The Reference Web Service contains the full class code in the **LogicalDeviceAdapter.java** file. You can use this code as a template for similar entity adapter classes when building custom web services.

Example 4-3 LogicalDeviceAdapter.java Code Section with a Transaction

```
UserEnvironment userEnvironment = null;
InventoryTransactionValue transValue = null;
    try {
        userEnvironment = startUserEnvironment();
        transValue = startTransaction();
        transValue.setUserEnvironment(userEnvironment);
        LogicalDeviceManager logicalDeviceManager =
                PersistenceHelper.makeLogicalDeviceManager();
        List<LogicalDevice> results = new ArrayList<LogicalDevice>(
            createLogicalDeviceRequest.getLogicalDevices().length);
        LogicalDeviceType[] ldTypes = createLogicalDeviceRequest
            .getLogicalDevices();
        List<oracle.communications.inventory.xmlbeans.LogicalDeviceType>
            ldTypesList = XMLBeansMappingUtils.fromEntityType(ldTypes);
        mapToLogicalDevice(logicalDeviceManager, ldTypesList, results);
        // call the API method
        results = logicalDeviceManager.createLogicalDevice(results);
        response.setLogicalDevices(mapToWebServiceResponseLDType(results));
        commitOrRollback(transValue);
    } catch (Throwable t) {
```


Copyright © 2014, 2026, Oracle and/or its affiliates.

```
    try {
rollback(transValue);
        } catch (Exception ignore) {
                log.error("", false, ignore, "Rollback failed");
        }
        log.error("", t, "LogicalDeviceAdapter.createLogicalDeviceFault");
        InventoryFaultType ift = FaultFactory.getFaultType(t);
        throw ift;
} finally {
         if (userEnvironment != null && userEnvironment.hasErrors()) {
                 response.setMessages(new String[] { FAILED });
         } else {
                 response.setMessages(new String[] { SUCCESS });
         }
         FeedbackUtils.copyFeedbacktoResponse(response);
         endUserEnvironment(userEnvironment, response);
}
```

Developing and Running Custom Web Services

You develop custom web services by working in Design Studio projects. In Design Studio you can generate the WAR files from the contents of the projects. You then import the WAR file into a deployable EAR file for deployment and testing. This section provides instructions to guide you through the WAR file creation and the deployment process.

> **Note**
>
> This chapter assumes you are using Design Studio to develop custom web services. You can alternatively build custom web services by using provided scripted builds with UIM installed on Linux. For information about using scripted builds, see "Overview" in *UIM Developer's Guide*.

This section assumes that you are working in Design Studio and therefore working in a Windows environment. Based on this assumption, the locations of all required UIM and Oracle WebLogic Server files are described using Windows paths.

> **Note**
>
> Oracle recommends that you perform the instructions to import, configure, and run the CreateLogicalDevice web service operation before introducing any custom code for a new web service. A successful test of CreateLogicalDevice ensures that your project is configured properly before the start of your custom web service development.

You perform the tasks described in the following development work sections to create a custom web service. The result of this work is the deployment of an EAR file that contains a new WAR file that defines the custom web service.

Pre-development work:

- Configuring Your Work Environment
- Importing the Reference Web Service Project
- Configuring the Imported Project

Copyright © 2014, 2026, Oracle and/or its affiliates.

Development work:

- Locating the API Method Signature in the Javadoc
- Developing the Web Service

Post-development work:

- Generating Java Source Based on the WSDL
- Creating the WAR File
- Packaging the WAR File in the EAR File
- Deploying the EAR File
- Deploying, Testing, and Securing the Web Service

Configuring Your Work Environment

Before you begin developing a custom web service, configure your work environment.

WebLogic Server

You must install Oracle WebLogic Server locally. This installation provides the correct version of the JDK. Depending on the references in your code, you should determine the specific WebLogic files that are required from the installation to build the web services project.

You can also run WebLogic Server locally; however, UIM is not supported on Windows. Therefore, UIM can run on Windows for development purposes only. You can optionally run WebLogic Server remotely.

UIM

To build your project and deploy in a traditional environment, you must have access to some of the UIM installation files. You can copy these files from a UIM installation on a UNIX machine to your machine, or you can install UIM locally. The following UIM files are needed:

- *UIM_SDK_Home***/webservices/reference_webservice.zip**
- *UIM_Home***/app/custom.ear** or **inventory.ear**

> **Note**
>
> This file is not required for UIM cloud native deployments.

- WebLogic Server patch files

> **Note**
>
> You can use WebLogic Server patch files if they are applicable. These files are located in the *UIM_SDK_Home*/**lib/*.jar** directory.

Design Studio

Install and configure Design Studio to work with the Reference Web Service, and to develop new custom web services. See "Overview" in *UIM Developer's Guide* for information about

Copyright © 2014, 2026, Oracle and/or its affiliates.

using Design Studio to extend UIM, including information about installing and configuring Design Studio.

> **Note**
>
> Configure Design Studio to use the correct version of JDK as specified by the WebLogic Server installation. See "Unified Inventory Management System Administration Overview" in *UIM System Administrator's Guide* for version information. If not configured to use the correct version of JDK, problems can be encountered that are difficult to trace, debug, and resolve.

You must also set the ANT_HOME system variable. See "Overview" in *UIM Developer's Guide* for more information.

Importing the Reference Web Service Project

Import the **reference_webservice.zip** file into Design Studio. For instructions on how to import projects into Design Studio using archive files, see "*SCD Design Studio Modeling Inventory*".

To see the ZIP file directories and files in Design Studio after the import, open the Java perspective with a Navigator view. Table 4-3 shows the directories and top-level files for the **reference_webservice.zip** file.

Table 4-3 Contents of reference_webservice.zip

Directory/File	Description
codegen	The **codegen** directory contains files that are generated from the WSDL and schema files. This directory is initially empty after the import of the **reference_webservice.zip** file.
config	The **config** directory contains a properties file that defines localized error messages used by the web services module.
etc	The **etc** directory contains the **COMPUTERNAME.properties** file. See "Configuring the COMPUTERNAME.properties File" for more information.
src	The **src** directory contains the Java source files that define the Reference Web Service.
test	The **test** directory contains input test XML files for testing the Reference Web Service.
webarchive	The **webarchive** directory contains the generated **ReferenceUim.war** file.
WEB-INF	The **WEB-INF** directory contains the **web.xml** file. The **web.xml** file is a web application deployment descriptor for the web service.

Copyright © 2014, 2026, Oracle and/or its affiliates.

Table 4-3 (Cont.) Contents of reference_webservice.zip

Directory/File	Description
wsdl	The **wsdl** directory contains the **ReferenceUim.wsdl** file that defines the Reference Web Service example operation. This directory also contains schema files that support the WSDL definition inputs, outputs, and faults in the **schemas** directory. Reference schemas **InventoryCommon.xsd**, **InventoryFault.xsd**, and **InventoryFaultRoot.xsd** reside in the **uim_webservices_framework.jar** file and **schema_Inventory_webservice.zip** file and are automatically copied to the **wsdl/referenceSchemas** directory when you run the provided **get-framework-files** Ant target later in the process. The **schema_Inventory_webservice.zip** file is located in the UIM SDK.
.classpath	The **.classpath** file is an Eclipse-specific file provided with the imported project. This file contains the directories for the class path entries for building.
.project	The **.project** file is an Eclipse-specific file provided with the imported project. This file defines the project library list, which lists JAR files that are required to build the project.
build.xml	The **build.xml** file defines several Ant targets that you can run to build a custom web service, as described in Table 4-1.
reference_webservice.inventoryCartridge	The **reference_webservice.inventoryCartridge** file is an internal Design Studio file. It maintains project information, such as the project type, the UIM software version, and Design Studio project dependency information.
.buildNumber	The **.buildNumber** file is in the project directory. It is an internal Design Studio file.
.studio	The **.studio** file is in the project directory. It is an internal Design Studio file.

> **Note**
>
> After importing the archive ZIP file into your workspace, unresolved errors appear in Design Studio until you configure the project. See "Configuring the Imported Project" for more information.

Configuring the Imported Project

You configure the project to build and deploy a web service. Configuring the imported project involves the following actions:

- Configuring the COMPUTERNAME.properties File
- Configuring the web.xml File
- Configuring the Project Library List

Copyright © 2014, 2026, Oracle and/or its affiliates.

Configuring the COMPUTERNAME.properties File

You set the variables in the **COMPUTERNAME.properties** file as the first step in configuring your project. This file contains the values that vary between projects, such as path names and the WSDL name. To configure the **reference_webservice/etc/COMPUTERNAME.properties** file:

1. Copy and rename the **COMPUTERNAME.properties** file to reflect the name of the computer on which you have Design Studio installed. You can determine your computer name by running the following DOS command:

   ```
   echo %COMPUTERNAME%
   ```

   An example of the resulting file name is **xlc123tx.properties**.

2. Update the parameter values defined in the file to reflect the information appropriate to the computer on which you are developing custom web services.

   Table 4-4 file defines the following parameters for the properties file:

Table 4-4 COMPUTERNAME.properties File Parameters

Parameter	Description

WSDL_NAME=ReferenceUim	The name of the WSDL file without the file extension. It is also used for deriving the context path and service URI for the generated web services WAR file. For example, in this case the web service context path and URI for the HTTP protocol is:
	`/ReferenceUim/ReferenceUimHTTP`
	and for JMS protocol is:
	`/ReferenceUim/ReferenceUimJMS`

QUEUE_NAME=inventoryCustomWSQueue	The name of the JMS Web Service Queue. It matches the name of the queue used in the WSDL for the SOAP <address> element for the service port. If you package your custom web service in an EAR file other than the provided custom.ear file, you must create your own message queue and configure your custom web service to use that queue by changing the QUEUE_NAME. See "Packaging the WAR File in the EAR File" for more information.
MODULE_NAME=reference_webservice	The name of the web service module. The name is used for creating the distributable web service cartridge. It is also the name of the directory where the generated web service WAR file is stored.
FMW_HOME=C:/Oracle/Middleware/Oracle_Home	The Fusion Middleware WebLogic Server installation directory name.
WL_HOME=${FMW_HOME}/wlserver_*Release*	The WebLogic Server installation path that incorporates the FMW_HOME parameter, where *Release* is the directory version name portion of the WebLogic Server library files if needed given the installation.
DOMAIN_HOME=C:/Oracle/Middleware/Oracle_Home/projects/domains	The directory path of the WebLogic Server domains.

Copyright © 2014, 2026, Oracle and/or its affiliates.

Table 4-4 (Cont.) COMPUTERNAME.properties File Parameters

Parameter	Description
DOMAIN_NAME=*uim_Release*	The domain name where *uim_Release* is the domain name with the UIM release number.
UIM_HOME=${DOMAIN_HOME}/${DOMAIN_NAME}/UIM	The UIM home path. The DOMAIN_HOME, DOMAIN_NAME and UIM_HOME parameters collectively specify the UIM installation path. **Note**: The UIM home path is not applicable for UIM cloud native deployments.
APP_LIB=UIM_SDK_Home/lib	The working directory to which dependent JAR files are extracted from the **inventory.ear** file. This working directory is automatically created for you based on the name provided here.
EAR_PATH=${UIM_HOME}/app/custom.ear	The directory where the **custom.ear** file is located. If you package your custom web service in a custom EAR file other than the provided **custom.ear** file, you must configure your custom web service to use your custom EAR file by changing the EAR_PATH value. See "Packaging the WAR File in the EAR File" for more information. **Note**: The EAR_PATH value is not applicable for UIM cloud native deployments.
POMS_ROOT=C:/uim/OracleCommunications/POMSClient/lib	The location of the POMS JAR file.
PLATFORM=C:/uim/OracleCommunications/commsplatform/ws	The location of the Platform web service JAR file.
PATCH_CLASSPATH=*pathFileDirectoryAndFile*	The path to any WebLogic patch files, if applicable. You must replicate this parameter for each WebLogic patch file to specify the path and specific patch file name.

Configuring the web.xml File

The **web.xml** file must be modified to contain the listener class reference. To configure the **reference_webservice/WEB-INF/web.xml** file, add the following:

```
<listener>
    <listener-class>
        oracle.communications.inventory.api.framework.listener.
        InventoryWebApplicationListener
    </listener-class>
</listener>
```

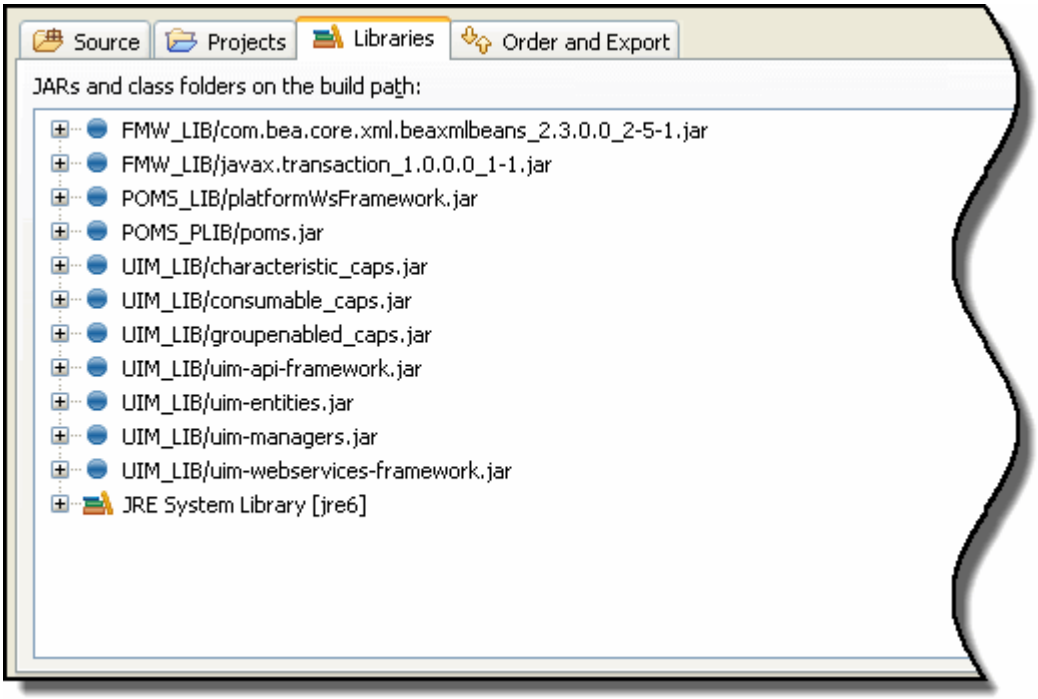
Configuring the Project Library List

The project library list of JAR files does not indicate the location of the files, so you must configure the project library list to point to the location of the JAR files.

Figure 4-8 shows the imported project library list, which includes the JAR files needed to compile the project.

Copyright © 2014, 2026, Oracle and/or its affiliates.

Figure 4-8 Project Library List Before Configuring



The required JAR files can be categorized into three groups:

- WebLogic files (FMW_LIB)
- Platform files (POMS_LIB and POMS_PLIB)
- UIM files (UIM_LIB)

You perform the following to configure the Design Studio project:

- Add new variables named FMW_LIB, POMS_LIB, POMS_PLIB, and UIM_LIB to your project.
- Define these variables to point to the directories listed in Table 4-5.

Table 4-5 Location of JAR Files

Variable Name	Directory Name
FMW_LIB	*FMW_Home*
POMS_LIB	*Oracle_Home*/commsplatform/ws
POMS_PLIB	*Oracle_Home*/POMSClient/lib
UIM_LIB	*UIM_SDK_Home*/lib

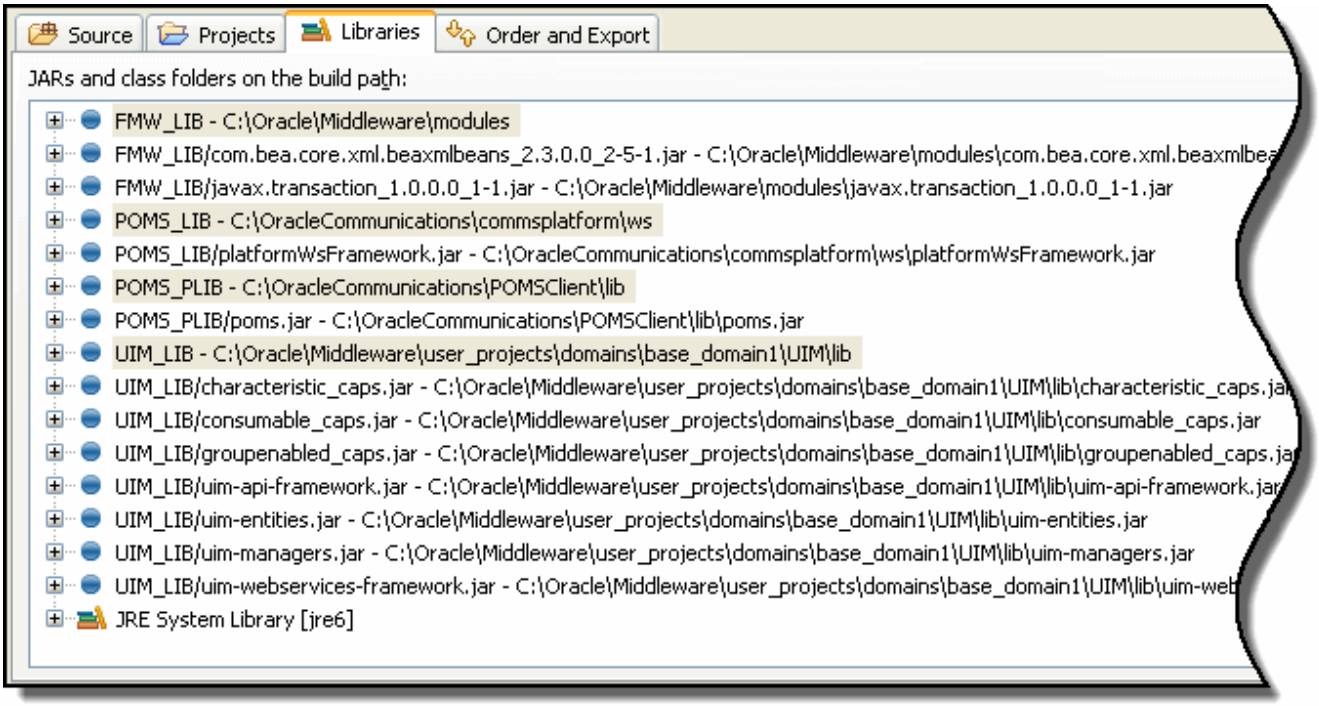
For detailed instructions on how to configure the project library list, see "*SCD Design Studio Modeling Inventory*".

Result of Configuring Project Library List

Figure 4-9 shows the project library list after the variables are added. Notice that the library list now includes the location of the JAR files, not just the JAR file names.

Copyright © 2014, 2026, Oracle and/or its affiliates.

Figure 4-9 Project Library List After Configuring



Adding the variables is one way to configure the library list; you can alternatively perform the following:

1. Write down the names of the required files.
2. Click **Add External JARS.**
3. Navigate to the directory location of JAR files.
4. Add it directly to the library list.

Either way, the result is the same. The library list has the location to the files needed to compile the project.

Locating the API Method Signature in the Javadoc

When creating a new web service, you wrap a call to an API manager method. For an overview of the primary API manager classes, see "Overview" in *UIM API Overview.*

To locate a particular API method:

1. Access the Javadoc.

   For instructions on how to access the Javadoc, see "Overview" in *UIM Developer's Guide*.
2. Perform a wildcard search for ***Manager** class.

   All manager class names end in **Manager**, such as TelephoneNumberManager.class, EquipmentManager.class, and so forth.
3. Open the appropriate manager class.

   All exposed methods are defined in manager classes; so, look for a manager class with a name similar to the functional area that may contain the method you plan to wrap.

Copyright © 2014, 2026, Oracle and/or its affiliates.

4. Locate the method you plan to wrap.

Information to Capture

You must capture a specific set of information to create a new web service. This information is available in the Javadoc after locating the method you plan to wrap. Capture the following information:

- Class name that defines the method to wrap
- Package in which the class resides
- Method signature information:
  - Method name
  - Input parameters
  - Return values
  - Exceptions thrown

For example, the CreateLogicalDevice web service operation wraps the createLogicalDevice() API method. The following information was used to define this web service in the **LogicalDeviceAdapter.java** file:

- LogicalDeviceManager is the UIM manager class that defines the createLogicalDevice() method.
- LogicalDeviceManager resides in the package oracle.communications.inventory.api.logicaldevice.
- The method signature information includes:
  - Method name: createLogicalDevice
  - Input parameters: Collection of LogicalDevice objects
  - Return values: List of LogicalDevice objects
  - Exceptions thrown: ValidationException

Developing the Web Service

Developing a new web service involves creating a new WSDL file, new schema files, and new Java source files. This section provides information about creating these files.

Creating the WSDL File

The imported project contains the **ReferenceUim.wsdl** file, which defines the example web service operation. Model your custom WSDL file after the **ReferenceUim.wsdl** file. For more information, see the W3C Web Services Description Language website at:

http://www.w3.org/TR/wsdl

Copyright © 2014, 2026, Oracle and/or its affiliates.

> **Note**
>
> The **ReferenceUim.wsdl** file is written to be independent of the application server. However, the **generate-from-wsdl** Ant target in the **build.xml** file is specific to generating the required source files for deployment into a WebLogic Server environment. This target is also needed to pull in other XSD files referenced in the example **ReferenceUim.wsdl** file.

WSDL Naming Conventions

The **ReferenceUim.wsdl** file uses WSDL_NAME variable in the **COMPUTERNAME.properties** file for naming its various SOAP elements. This naming convention allows the **build.xml** Ant targets to parse these elements consistently, and to generate the correct source files for the web service interfaces and implementation. Consider the following list of naming conventions for the WSDL file:

- ReferenceUim

  This is the name of the WSDL file without the file extension as set by the WSDL_NAME variable in the **COMPUTERNAME.properties** file. This name is also used to automatically set other important variables in the **build.xml** file, such as SERVICE_NAME and PORT_NAME. This name is assumed to be the name of the root definitions element in the WSDL file. This name identifies the name of the following files, which are generated later in the process: **ReferenceUimPort.java**, **ReferenceUimPortImpl.java**, **ReferenceUim.war**, **ReferenceUimHTTP.war,** and **ReferenceUimJMS.war**.

- ReferenceUimPort

  This is the name of the PortType that is generated for the implementation later in the process. It is used by the generated source **ReferenceUimPort.java** and **ReferenceUimPortImpl.java**.

- ReferenceUimHTTPSoapBinding

  This is the name of the SOAP binding for web service operations that are exposed through the HTTP transport protocol. The list of operations identified in this binding element can be a subset of the operations identified in the <PortType> element. The list of operations can be the same as or different from the JMS protocol operations.

- ReferenceUimJMSSoapBinding

  This is the name of the SOAP binding for web service operations that are exposed through the JMS transport protocol. The list of operations identified in this binding element can be a subset of the operations identified in the <PortType> element. The list of operations can be the same as or different from the HTTP protocol operations.

- ReferenceUimHTTPPort

  This is the name of the HTTP transport port used in the UIMReference service definition.

  It references the ReferenceUimHTTPSoapBinding binding element identified earlier. Also, the SOAP address location uses the following for the context path (HTTP):

  ```
  http://localhost:7001/ReferenceUim/ReferenceUimHTTP
  ```

- ReferenceUimJMSPort

  This is the name of the JMS transport port used in the UIMReference service definition.

  It references the ReferenceUimJMSSoapBinding binding element identified earlier. Also, the SOAP address location uses the following for the context path (JMS):

Copyright © 2014, 2026, Oracle and/or its affiliates.

```
jms://localhost:7001/ReferenceUim/ReferenceUimJMS?URI=inventoryCustomWSQueue
```

For example, if you create a new file named **MyInventoryWs.wsdl**, the naming conventions result in:

- MyInventoryWsPort
- MyInventoryWsHTTPSoapBinding
- MyInventoryWsJMSSoapBinding
- MyInventoryWsHTTPPort
- MyInventoryWsJMSPort

Creating Schema Files

The imported project provides supporting schemas for the Reference Web Service operation. The schemas define the inputs, outputs, and faults of the wrapped methods. The schemas are used to generate the Java representation of the incoming/outgoing XML, which can then be mapped to an internal Java entity class (see "*Entity*Mapper.java"). The Java representation is generated by the **generate-from-wsdl** Ant target.

For a new web service, new schemas must be written that reflect the inputs and outputs of the wrapped method.

> **Note**
>
> The Reference Web Service schema files are written to be independent of the application server. However, the **generate-from-wsdl** Ant target in the **build.xml** file is specific to generating the required source files for deployment into a WebLogic Server environment.

Modifying the Mapping File

The imported project provides the **type-mapping.xsdconfig** mapping file. This file maps XML namespaces to Java packages. For a new web service, you modify the mapping file to update the namespace-to-Java package mappings.

Creating Java Source Files

The imported project provides the supporting Java code for the Reference Web Service operation. The following list of Java files is a recommended set of classes to implement.

- **ReferenceUimPortImpl.java**
- **AdapterRouter.java**
- ***Entity*Adapter.java**
- ***Entity*Validator.java**
- ***Entity*Utils.java**
- ***Entity*Worker.java**
- ***Entity*Mapper.java**
- ***Entity*Exception.java**
- **FaultFactory.java**

Copyright © 2014, 2026, Oracle and/or its affiliates.

> **Note**
>
> The example Reference Web Service does not contain all of the following source files.

The following sections describe detailed information about the recommended Java source files.

ReferenceUimPortImpl.java

The ReferenceUimPortImpl class is the entry point into the web service logic. This class calls the AdapterRouter class.

ReferenceUimPortImpl.java is a generated source file with the content based on the **ReferenceUim.wsdl** file. This file is generated by the **generate-from-wsdl** Ant target and is placed in the following directory:

```
codegen/WebServiceImpl/oracle/communications/inventory/webservice/ws
```

Copy this file to the following destination directory:

```
src/oracle/communications/inventory/webservice/ws
```

You use this destination directory as a starting point for the implementation of the web service calling the respective adapter classes.

This class must be modified to call the AdapterRouter for each new web service operation. Because this is a generated file, the modifications are based on the WSDL file.

> **Note**
>
> When modifying this file with additional operations, do not change the order of the methods. The generated **ReferenceUimPortImpl.java** source file has a specific order of the web service methods, which is based on the order of how the corresponding operation names are defined in the WSDL. Even though changing the order of the methods is allowable by the Java language syntax, doing so may cause the web service to not be found at run time.

AdapterRouter.java

The AdapterRouter class routes the call to a specific adapter. If the input from the external source requires mapping, the corresponding mapper class is the input/output parameter for this AdapterRouter class.

This class must be modified for each new web service operation.

*Entity*Adapter.java

Adapter classes extend the InventoryAdapterRoot class, which extends the Platform-owned AdapterRoot class. The UIM Reference Web Service provides the LogicalDeviceAdapter class as an example. Adapters wrap the calls to the UIM API methods. Typically, one adapter class exists per manager class, such as **EquipmentAdapter.java** and **TelephoneNumberAdapter.java**. However, one adapter class can wrap multiple methods from different manager classes.

Oracle recommends that adapters be as thin as possible. They should simply contain a call to the Manager API or to other worker classes.

Copyright © 2014, 2026, Oracle and/or its affiliates.

An adapter calls *Entity*Validator and, if validations pass, calls the business layer API method.

An existing adapter class must be modified, or a new adapter class written, for each new web service operation.

*Entity*Validator.java

Validator classes define an input validation method per a web service operation (where applicable). The adapter classes call the corresponding input validation method before calling the wrapped API method. They throw an error if the validation is not passed.

For cases where input data is passed in, a new validator class is needed per entity. The UIM Reference Web Service example does not contain this type of class file.

*Entity*Utils.java

Utils classes define common utility methods used by the Reference Web Service operations.

The existing *Entity*Utils class can be extended or a new utils class written, as needed during the development of a new web service.

*Entity*Worker.java

Worker classes define methods used by the web service operations.

*Entity*Worker classes can be written as needed during the development of a new web service. The UIM Reference Web Service example does not contain this type of class file.

*Entity*Mapper.java

Mapper classes map the generated object representation of the schemas (external) to the Java entity class (internal) for input parameters. Mapper classes map the Java entity class (internal) to the generated object representation of the schemas (external) for output parameters. One mapper class maps a single entity. A mapper class can be shared across methods in an adapter class if the methods use the same entity.

For cases where the source code references the entity data, a new mapper class is needed per entity.

> **Note**
>
> In the UIM Reference Web Service example, the mapping code logic is contained in the adapter class.

*Entity*Exception.java

Exception classes define exceptions specific to a web service.

The existing *Entity*Exception classes can be extended or a new exception class written, as needed during the development of a new web service.

FaultFactory.java

The FaultFactory class maps Exception objects thrown by the API method to InventoryFaultType objects returned by the web service.

You may need to modify this class for a new web service; it depends on whether the API method introduces any new Exception objects that are not already mapped.

Copyright © 2014, 2026, Oracle and/or its affiliates.

Generating Java Source Based on the WSDL

The imported project contains the **build.xml** file, which defines the **generate-from-wsdl** Ant target. The **generate-from-wsdl** Ant target copies the latest framework schema files into the web services project and generates the Java source based on the input WSDL file and supporting schemas. You can run the **generate-from-wsdl** Ant target to automatically copy the framework files and generate the Java source.

The generated package structure and generated files include:

- **codegen/src/oracle/communications/inventory/webservice**

  This package contains the generated Java source files.

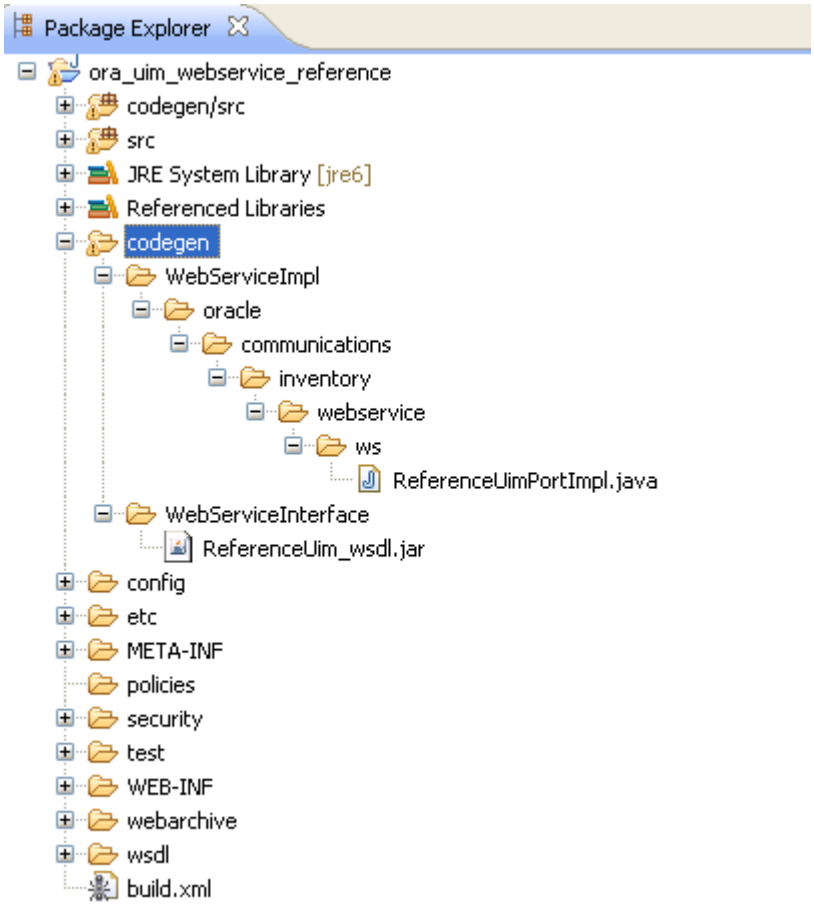
- **codegen/WebServiceImpl/oracle/communications/inventory/webservice/ws**

  This package contains the generated Java implementation source code file. Figure 4-10 shows the generated **ReferenceUimPortImpl.java** source file for the provided Reference Web Service.

- **codegen/WebServiceInterface**

  This package contains the generated JAR file. Figure 4-10 shows the generated **ReferenceUim_wsdl.jar** file for the provided Reference Web Service.

Figure 4-10 Package Explorer View Including the codegen Directory Generated Files



Copyright © 2014, 2026, Oracle and/or its affiliates.

After the source is generated, the project workspace has access to all the dependent files needed to compile the project. The compiled classes are stored in the **out** directory. Class files compiled from Java source files that are part of the original imported project are also placed in the **out** directory, such as the class files in the **out/oracle/communications/inventory/webservice/adapter** directory.

Creating the WAR File

The WAR file contains the compiled classes from the developed custom web service, plus the JAR file containing the UIM API method that the web service wraps.

The imported project contains the **build.xml** file, which defines the Ant targets to build the WAR file. The following Ant targets build the WAR file:

- **build-service** builds the WAR file for both HTTP and JMS
- **build-service-http** builds the WAR file for HTTP
- **build-service-jms** builds the WAR file for JMS

You can run any of these Ant targets to automatically build the WAR file.

Figure 4-11 shows the created **ReferenceUim.war** file which resides in the **webarchive/reference_webservice** directory. The created WAR file name is *wsdl_name*.**war**, where *wsdl_name* is the name specified by the WSDL_NAME parameter in the **COMPUTERNAME.properties** file. The WAR file resides in the **webarchive/***module_name* directory, where *module_name* is the name specified by the MODULE_NAME parameter in the **COMPUTERNAME.properties** file.

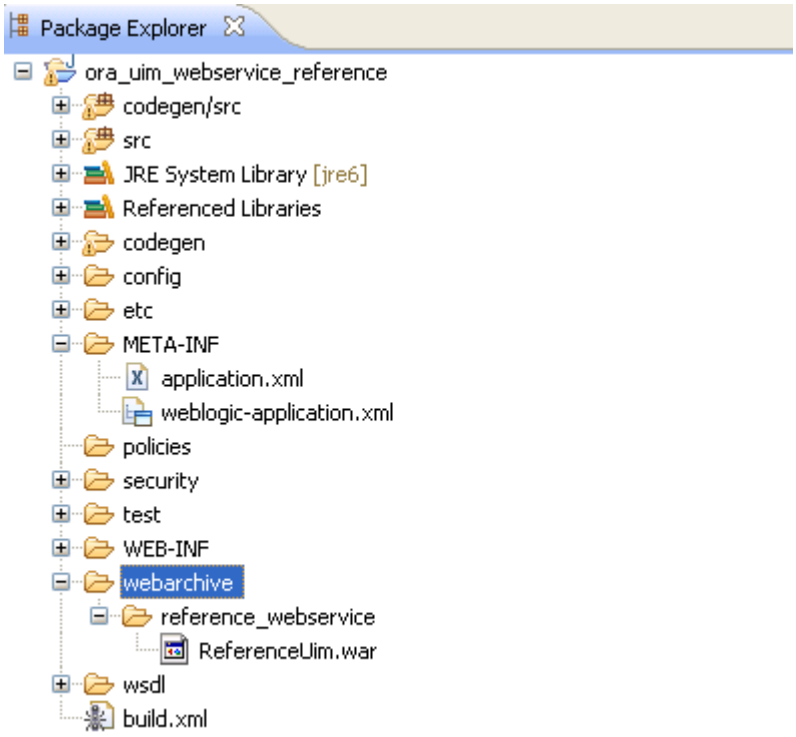
The WAR file contains the following:

- Compiled generated source files (WSDL and XML object representations)
- Compiled developed source files (contents of the **src** directory)
- JAR file that contains the classes that define the wrapped methods (UIM business logic)

Figure 4-11 shows the generated directory structure and the **ReferenceUim.war** file for the Reference Web Service.

Copyright © 2014, 2026, Oracle and/or its affiliates.

Figure 4-11 Package Explorer View Including the webarchive Directory



Packaging the WAR File in the EAR File

The Reference Web Service WAR file is not packaged in the **inventory.ear** file and is therefore not automatically deployed into UIM. Rather, you must manually import the provided **ReferenceUim.war** file into an EAR file to deploy.

In UIM traditional deployments, when developing custom web services, you have the option of packaging the custom web service WAR file into:

- The **custom.ear** file

  If you develop a single custom web service, Oracle recommends you use the provided **custom.ear** file. This approach saves you additional development work because you can use the provided inventoryCustomWSQueue and the corresponding listener class.

- Any custom EAR files

  If you develop multiple custom web services, Oracle recommends you use a separate custom EAR for each web service. This approach involves additional development work because you must create and configure your own message queue and corresponding listener class. This is the safest approach for multiple custom web services and provides the most efficient performance. See "Additional Custom Work" for more information.

In UIM cloud native deployments, the generated WAR file is packaged in the **inventory.ear** file using customized image generation. For more information, see "Creating UIM Cloud Native Images" in *UIM Cloud Native Deployment Guide*. Remaining steps in extracting the application file and deploying the **inventory.ear** file are managed differently in UIM cloud native deployments. For more information refer to "Overview of the UIM Cloud Native Deployment" in *UIM Cloud Native Deployment Guide*.

Copyright © 2014, 2026, Oracle and/or its affiliates.

Extracting and Updating the application.xml File

Every EAR file contains an **application.xml** file, which defines the WAR files that comprise the EAR file. Regardless of which packaging option you choose, the custom web service WAR file needs to be included in the EAR file, so the **application.xml** file must be updated to include the name of the custom web service WAR file.

Extracting the File

The provided **custom.ear** file contains an **application.xml** file that you can manually extract, use as a starting point, and modify as needed.

If you are using the provided **custom.ear** file to package your custom web service, you can use the **extract.ear** Ant target to automatically extract the **application.xml** file. The EAR file is specified by the EAR_PATH parameter in the **COMPUTERNAME.properties** file. The XML file is extracted into the *reference_webservice_home***/META-INF** directory, where *reference_webservice_home* is the location of the extracted **reference_webservice.zip** file. See "About the Ant Build File" for more information on the Ant targets.

> **Note**
>
> This **extract.ear** Ant target only works if EAR_PATH is set to **custom.ear**; it does not work if EAR_PATH is set to a custom EAR file name or set to **inventory.ear**. The **extract.ear** Ant target is provided in the **build.xml** file.

Updating the application.xml File

Example 4-4 shows the original **application.xml** file from the **custom.ear** file. For the custom web service, you add the following information to the <module> element to identify the following for the custom web service:

- The WAR file name, such as **ReferenceUim.war**
- The WSDL file prefix, such as **ReferenceUim**

You add the <web-uri> item for the WAR file name and the <context-root> item for the WSDL name, as shown in Example 4-5.

Example 4-4 Original application.xml

```
<?xml version = '1.0' encoding = 'windows-1252'?>
<application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/
application_5.xsd" version="5" xmlns="http://java.sun.com/xml/ns/javaee">
    <display-name>oracle.communications.inventory.customear</display-name>
    <module>
        <java></java>
    </module>
</application>
```

Example 4-5 Updated application.xml

```
<?xml version = '1.0' encoding = 'windows-1252'?>
<application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/
application_5.xsd" version="5" xmlns="http://java.sun.com/xml/ns/javaee">
    <display-name>oracle.communications.inventory.customear</display-name>
```


Copyright © 2014, 2026, Oracle and/or its affiliates.

```
    <!-- Custom Web Service WAR -->
    <module>
        <web>
            <web-uri>ReferenceUim.war</web-uri>
            <context-root>ReferenceUim</context-root>
        </web>
    </module>
</application>
```

Additional Custom Work

This section describes additional work you must perform when packaging your custom web services in your own custom EAR files.

> **Note**
>
> - This section is not applicable for UIM cloud native deployments.
> - This section is not applicable if you are packaging your custom web service in the provided **custom.ear** file.

Example 4-6 shows the **custom.ear** file content with a view of the notable directories and files. An EAR file format is similar to a ZIP file format.

Example 4-6 custom.ear File Content

```
01      META-INF
02          application.xml
03          weblogic-application.xml
04      InventoryCustomQueueMDB.jar
05          META-INF
06              ejb-jar.xml
07              weblogic-ejb-jar.xml
08          oracle
09              communications
10                  inventory
11                      ejb
12                          message
13                              custom
14                                  impl
15                                      InventoryCustomQueueListener.class
16      poms-ejbs.jar
```

Your custom EAR file content must use the **custom.ear** file content as a template to apply your modifications. Your modifications may include the following:

- Referencing corelib and customlib
- Creating a Message Queue
- Creating a Listener Class
- Configuring the Listener Class

Referencing corelib and customlib

Each custom EAR file must contain a reference to the following libraries:

```
oracle.communications.inventory.corelib
oracle.communications.inventory.customlib
```


Copyright © 2014, 2026, Oracle and/or its affiliates.

You reference these libraries within the **META-INF/weblogic-application.xml** file (line 03 in Example 4-6).

For an example to emulate, see the **weblogic-application.xml** file in the **custom.ear** file. Example 4-7 shows the contents of the **weblogic-application.xml** file.

Example 4-7 weblogic-application.xml

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<weblogic-application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.bea.com/ns/weblogic/weblogic-application http://
www.bea.com/ns/weblogic/weblogic-application/1.0/weblogic-application.xsd" xmlns="http://
www.bea.com/ns/weblogic/weblogic-application">
<!-- Use the common Oracle Platform Security Services -->
<!-- oracle.communications.inventory application policies -->
  <application-param>
    <param-name>jps.policystore.applicationid</param-name>
    <param-value>oracle.communications.inventory</param-value>
  </application-param>
  <library-ref>
      <library-name>oracle.communications.inventory.corelib</library-name>
      <specification-version>7.3</specification-version>
      <implementation-version>7.3.1.0.0</implementation-version>
      <exact-match>false</exact-match>
  </library-ref>
  <library-ref>
      <library-name>oracle.communications.inventory.customlib</library-name>
      <specification-version>7.2</specification-version>
      <implementation-version>7.2.0.0.0</implementation-version>
      <exact-match>false</exact-match>
  </library-ref>
</weblogic-application>
```

Creating a Message Queue

If you create multiple custom web services, Oracle recommends they reside in different EAR files. Web services that reside in different EAR files cannot listen to the same queue, so you must create a message queue for each web service. In addition, Oracle recommends that you provide a message-driven bean (MDB) to dispatch requests for multi-threaded processing to ensure optimum performance of your custom web service.

See the WebLogic Server Administration Console documentation for information about creating message queues. See the Java Platform, Java EE Tutorial website at:

https://docs.oracle.com/javaee/7/tutorial/ejb-intro003.htm

for more information on message-driven beans.

Creating a Listener Class

You must create one listener class for every message queue you create. Example 4-8 shows a custom listener class named MyCustomQueueListener. The listener classes implement MessageListener.

Your custom listener class must reside within your custom EAR file. Specifically, it must reside within the **oracle/communications/inventory/ejb/message/custom/impl** directory (lines 08-14 in Example 4-6), within a custom MDB JAR file (line 04 in Example 4-6).

Example 4-8 Listener Class

```
package oracle.communications.inventory.webservice.mdb;
```


Copyright © 2014, 2026, Oracle and/or its affiliates.

```
import java.util.HashMap;
import java.util.Set;

import javax.ejb.MessageDrivenContext;
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.persistence.PersistenceContext;

import oracle.communications.inventory.api.framework.logging.Log;
import oracle.communications.inventory.api.framework.logging.LogFactory;
import weblogic.wsee.server.jms.JmsWebservicesMessageDispatcher;

@PersistenceContext(name = "persistence/EntityManager", unitName = "default")
public class MyCustomQueueListener implements MessageListener
{
    private static Log log = LogFactory
        .getLog( MyCustomQueueListener.class );
    private static final String CONNECTION_FACTORY = "inventoryWSQueueCF";
    private HashMap<String, JmsWebservicesMessageDispatcher> listeners =
         new HashMap<String, JmsWebservicesMessageDispatcher>();
    private MessageDrivenContext context;

    public void setMessageDrivenContext(
        MessageDrivenContext messageDrivenContext )
    {
        this.context = messageDrivenContext;
    }

    public void onMessage( Message message )
    {
        try
        {
            String uri = message.getStringProperty( "URI" );
            JmsWebservicesMessageDispatcher listener = getListener( uri );
            if( log.isDebugEnabled() )
            {
                log.debug( "", "Thread " + Thread.currentThread().getId() + " "
                    + Thread.currentThread().hashCode()
                    + ": calling onMessage()..." );
            }
            if( listener != null )
            {
                listener.dispatchMessage( message );
            }
        }
        catch( Exception e )
        {
            log.error( "", "Failed to process JMS message: " + e.getMessage() );
            e.printStackTrace();
        }
    }

    public void ejbCreate()
    {
    }

    public void ejbRemove()
    {
        Set<String> keys = listeners.keySet();
        try
        {
            for( String key : keys )
```


Copyright © 2014, 2026, Oracle and/or its affiliates.

```
            {
                JmsWebservicesMessageDispatcher listener = listeners.get( key );
                listener.shutdown();
            }
        }
        catch( Exception e )
        {
            log.error( "", "Error closing the listener: " + e.toString() );
        }
        listeners.clear();
        listeners = null;
    }

    private JmsWebservicesMessageDispatcher getListener( String uri )
    {
        JmsWebservicesMessageDispatcher listener = null;
        Object obj = listeners.get( uri );
        try
        {
            if( obj == null )
            {
                listener = new JmsWebservicesMessageDispatcher( uri,
                    CONNECTION_FACTORY );
                listeners.put( uri, listener );
            }
            else
            {
                listener = (JmsWebservicesMessageDispatcher) obj;
            }
        }
        catch( Exception e )
        {
            e.printStackTrace();
        }
        return listener;
    }
}
```

Configuring the Listener Class

After you create one or more listener classes, you must configure them to listen to their respective queues. This is accomplished by modifying the **ejb-jar.xml** and **weblogic-ejb-jar.xml** files, as shown in Example 4-9 and Example 4-10.

The **ejb-jar.xml** and **weblogic-ejb-jar.xml** files must reside within your custom EAR file. Specifically, they must reside within the **META-INF** directory (line 05 in Example 4-6), within a custom MDB JAR file (line 04 in Example 4-6).

Example 4-9 ejb-jar.xml

```
<?xml version = '1.0' encoding = 'windows-1252'?>
<ejb-jar xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/j2ee/
ejb-jar_3_0.xsd"
version="3.0">
  <enterprise-beans>
    <message-driven>
      <ejb-name>MyCustomQueueListener</ejb-name>
      <ejb-class>
        oracle.communications.inventory.webservice.mdb.MyCustomQueueListener
      </ejb-class>
```


Copyright © 2014, 2026, Oracle and/or its affiliates.

```
      <transaction-type>Bean</transaction-type>
      <message-destination-type>javax.jms.Queue</message-destination-type>
    </message-driven>
  </enterprise-beans>
</ejb-jar>
```

Example 4-10 weblogic-ejb-jar.xml

```
<?xml version = '1.0' encoding = 'windows-1252'?>
<weblogic-ejb-jar xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.bea.com/ns/weblogic/weblogic-ejb-jar
http://www.bea.com/ns/weblogic/weblogic-ejb-jar/1.0/weblogic-ejb-jar.xsd"
xmlns="http://www.bea.com/ns/weblogic/weblogic-ejb-jar">
    <weblogic-enterprise-bean>
        <ejb-name>MyCustomQueueListener</ejb-name>
        <message-driven-descriptor>
            <destination-jndi-name>MyCustomQueue</destination-jndi-name>
        </message-driven-descriptor>
    </weblogic-enterprise-bean>
</weblogic-ejb-jar>
```

Importing the WAR File into the EAR File

After you determine which EAR file is to contain the custom web service WAR file, import the WAR file into the appropriate EAR file.

The imported project contains the **build.xml** file, which defines the **update.ear** Ant target. The **update.ear** Ant target updates the EAR file by adding the custom web service WAR file and the edited **application.xml** file. The **update.ear** Ant target determines the location of the EAR file to be updated by using the path you specified in the **COMPUTERNAME.properties** EAR_PATH parameter. Run the **update.ear** Ant target to automatically perform these updates to the EAR file.

See "About the Ant Build File" for more information on Ant targets.

Deploying the EAR File

The imported project contains the **build.xml** file, which defines the **copyResources** Ant target. The **copyResources** Ant target copies the **referenceWS.properties** file from the imported project to the *UIM_Home*/**config/resources/logging** directory. Before deploying the updated EAR file for the first time, run the **copyResources** Ant target. Unless you change the **referenceWS.properties** file, you only need to run this Ant target one time.

> **Note**
>
> The **copyResources** Ant target is not applicable for UIM cloud native deployments. If you want to add and deploy any logging properties files, you should add them to solution cartridges or to the **ora_uim_localization_reference** cartridge. For more information on deploying cartridges, see "Deploying Cartridges" in *UIM Cloud Native Deployment Guide*.

If your UIM environment resides on another machine, you must copy the updated EAR file to that machine before deploying.

For instructions on how to deploy an EAR file, see "Unified Inventory Management System Administration Overview" in *UIM System Administrator's Guide*.

Copyright © 2014, 2026, Oracle and/or its affiliates.

> **Note**
>
> After you have gone through all the steps in this chapter once, you only need to run the **clean**, **all**, and **update.ear** Ant targets to rebuild the EAR file before deploying it.

Verifying the Deployment

After you have deployed the updated EAR file, verify that the deployment includes the custom web service by viewing the web services in the WebLogic Server Administration Console. See "Verifying Deployments" for more information.

Specifying a Deployment Plan

If you placed your custom web service in the **custom.ear** file, or in any custom EAR file, you must specify a deployment plan for the updated EAR file.

Specifying a deployment plan enables the EAR file to retrieve property values from the *UIM_Home***/app/AppFileOverrides/platform/runtime-poms.properties** file, which defines property values that are used by the persistence framework for cache coordination.

To specify a deployment plan:

1. Log in to the WebLogic Server Administration Console.
2. In the left panel, under Domain Structure, click the **Deployments** link.

   The Summary of Deployments page appears.
3. In the left panel, under Change Center, click **Lock & Edit**.
4. Select the check box next to the updated EAR file that contains your custom web service.
5. Click **Update**.

   The Update Application Assistant page appears.
6. Click **Change Path**.
7. Change the path to *UIM_Home***/app/plan**.
8. Choose **Plan.xml**, and click **Next**.
9. Choose **Redeploy this application using the following deployment files**, and click **Finish**.

Deploying, Testing, and Securing the Web Service

Information about deploying, testing, and securing the web service is described in "Deploying, Testing, and Securing UIM Web Services".

Copyright © 2014, 2026, Oracle and/or its affiliates.

5

Developing Custom REST Web Services

This chapter provides information about integrating Oracle Communications Unified Inventory Management (UIM) with external systems by developing custom REST web services. It describes the approach to developing web services and the guidelines you should follow.

About the UIM REST Reference Web Services

This chapter uses the UIM Reference Web Service as an example that you can extend.

The UIM Reference Web Service is part of the UIM Software Developer's Kit (SDK). The UIM SDK provides the resources required to build an Inventory cartridge in Design Studio. For more information about the UIM SDK, see "Overview" in *UIM Developer's Guide*.

This chapter assumes you are using Design Studio to develop custom rest web services. If you use an integrated development environment (IDE) other than Design Studio, you can ignore the .classpath and .project files in the reference_rest_webservice.zip file.

You can view the contents of **reference_rest _webservice.zip** file in Oracle Communications Service Catalog and Design – Design Studio by importing the archive ZIP file into Design Studio. The ZIP file contains several types of files including the following:

- **YAML Files**:

  The **UIMSample1_0.yaml** file defines a sample web service operation. The YAML file also defines the paths that defines individual API endpoints and HTTP methods (GET, POST, PATCH, DELETE), components, parameters and payload of operation. See "About the YAML File" for more information about the UIMSample1_0.yaml file.

- **Java Source Files** :

  The Java source files provide the web service operation code. For example, these source files provide the following:

  – Model files generated out of components mentioned in yaml

  – An API manager to call UIM core for the operation

  – Transaction management for the operation with the commit or rollback results

  See "Developing the REST Web Service" for more information about the Java source files that includes the list and descriptions for each type of the class files, and information about the files that need to be created or modified.

- **Gradle Build File**:

  The build.gradle file defines Gradle targets that you can run to build a custom REST web service. Gradle targets are a set of executable tasks defined in the **build.gradle** file. See "About the Gradle Build File" for more information.

Prerequisites for Customizing REST Web Services

You require the following prerequisites for customizing REST web services:

- Install Gradle. See “*Installing Gradle”* for more information.

Copyright © 2014, 2026, Oracle and/or its affiliates.

- Set up a proxy. See "*Setting Up Proxy*" for more information.
- Update the properties file from ***reference_rest_webservice/etc/<COMPUTERNAME>.properties***

Installing Gradle

To install Gradle:

1. Open the command prompt.
2. Run the **gradlew** command.
   The system installs Gradle from wrapper properties.

Setting Up Proxy

After you install Gradle, the corresponding jar files are pulled from the Maven repository. You can set up proxy by updating the **reference_rest_webservice/gradle.properties** file.

About the YAML File

The Reference REST web service operation is defined by the **UIMSample1_0.yaml** file. The YAML file is located at **UIM_SDK_Home/webservices/reference_rest_webservice.zip\yaml**, where **UIM_SDK_Home** is the local directory for UIM SDK.

The YAML file defines the REST web service operation. The operation defines a **requestBody**, parameters, and all possible server responses. For example, the YAML file defines the following for the **createInventoryGroup** operation with HTTP method POST:

The request contains all possible responses where each response defines a JSON structure that is defined in the supporting schemas.

The following example shows the path definition, operation, and the input request message within a Sample YAML file:

```
paths:
  /inventoryGroup:
    post:
      operationId: createInventoryGroup
      summary: Create Ig
      description: |
        Creates a inventory group with the given details
      tags:
        - Sample Inventory Group
      requestBody:
        description: The ig to create.
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/InventoryGroup'
      responses:
        '201':
          description: The ig were created successfully.
          content:
            application/json:
              schema:
```


Copyright © 2014, 2026, Oracle and/or its affiliates.

```
                type: array
                items:
                  $ref: '#/components/schemas/InventoryGroup'
          '400':
            description: The request isn't valid.
            content:
              application/json:
                schema:
                  $ref: '#/components/schemas/Error'
          '401':
            description: You aren't authorized to make this request.
            content:
              application/json:
                schema:
                  $ref: '#/components/schemas/Error'
          '403':
            description: The request is forbidden.
            content:
              application/json:
                schema:
                  $ref: '#/components/schemas/Error'
          '500':
            description: An internal server error occurred.
            content:
              application/json:
                schema:
                  $ref: '#/components/schemas/Error'
.
.
.
components:
  schemas:
    InventoryGroup:
      type: object
      description: A inventoryGroup to associate with the resource.
      properties:
        id:
          type: string
          description: The ID of the ig.
          readOnly: true
        href:
          type: string
          format: uri
          description: The URI for the ig.
        name:
          type: string
          description: The name of the resource.
        description:
          type: string
          description: A free-text description for the resource.
        igSpecification:
          $ref: 'Combined.yaml#/components/schemas/Specification'
        startDate:
          type: string
          description: The date and time when the time period starts.
          format: date-time
```


Copyright © 2014, 2026, Oracle and/or its affiliates.

```
endDate:
  type: string
  description: The date and time when the time period ends.
  format: date-time
place:
  type: array
  items:
    $ref: 'Combined.yaml#/components/schemas/PlaceRef'
  description: The list of associated geographic places.
parentGroupRef:
  type: array
  items:
    $ref: '#/components/schemas/GroupRef'
inventoryGroupItems:
  type: array
  items:
    $ref: '#/components/schemas/GroupItemRef'
  description: The list of associated inventory group items.
```

The above example shows a model schema for **InventoryGroup** which is the used as the request body with **Content-Type : application/json**. For the response, with SUCCESS (20x), the same schema of **InventoryGroup** appears. The error model mentioned in the example is for the error codes.

For more details on OpenApi structure, see https://swagger.io/docs/specification/about/

About the Gradle Build File

The **build.gradle** file defines Gradle targets that you can run to build a custom web service. These build targets are a set of executable tasks that help in building a web service.

Table 5-1 describes the Gradle targets defined in the **build.gradle** file. See "Developing Custom REST Web Services" for information about when to run these Gradle targets, the Gradle commands that you should run, and the project to be imported to Design Studio.

Table 5-1 build.gradle and gradleTargets

Target	Description
Clean	Deletes the generated, temporary, and deliverable files and directories.
Build	Builds the entire source code using Swagger Code Generator in **build/libs**.
copyResources	Copies the properties files that store localized error messages to the appropriate UIM deployment directory. These properties files are located in a ZIP file in the **config/resources/logging** directory and are copied to the **UIM_Home/config/resources/logging** directory.
War	Generates a war file with the name mentioned in **WAR_NAME** in **COMPUTERNAME.properties** file.

Copyright © 2014, 2026, Oracle and/or its affiliates.

Table 5-1 (Cont.) build.gradle and gradleTargets

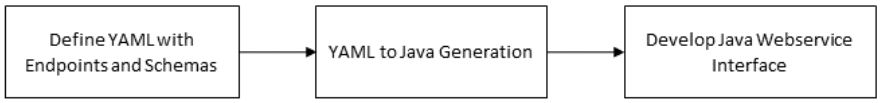
Target	Description
extractCustomEar	Extracts the **application.xml** file from the EAR file specified by the **EAR_PATH** parameter defined in the **COMPUTERNAME.properties** file into the **reference_rest_webservice _home/META-INF** directory, where **reference_rest_webservice_home** is the location of the extracted reference_rest_webservice.zip file. You must edit the **application.xml** file manually so that the EAR file can be updated for proper deployment of the web services.
copyLibs	Copies all the Gradle-pulled jars into **reference_rest_webservice\lib**, which can be used as CART\lib for Design Studio **.classpath** reference. **Note**: Run this target after the build.
copyModelToSource, copyApiToSource, copyApiImplToSource	These targets are not for use, if you run these will replace the model, api, impl with the generate source losing the manual changes.

For more information on Gradle, see https://docs.gradle.org/current/userguide/userguide.html

Guidelines for Developing Custom REST Web Services

This section describes the guidelines for developing a REST web service. It explains class diagrams that represent the UIM Reference REST Web Service development classes.

You use the Design-First approach to develop custom REST web services.



The Design-first approach is as follows:

- Define YAML with paths and components. Write the YAML to define the operations and data.
- Yaml-to-Java generation: Use the **build.gradle** Gradle targets provided by the Reference REST web service to generate Java source files, based on the YAML.
- Develop a Java web service interface implementation: Use the web service development environment and tools provided by the Reference REST web service to implement the web service interface by creating new Java source files and changing the existing files.

For example, the UIM Reference Rest Web Service module is designed using the Design-first approach. This means that:

Copyright © 2014, 2026, Oracle and/or its affiliates.

- The InventoryGroupApiServiceImpl, InventoryGroupApiService, and InventoryGroupType Java source files are generated based on the YAML (Swagger Code generator). This generation results in the YAML operation being defined in the Java source file, but with no coding details as a sample template.
- The build generates model, api, impl Java sources.

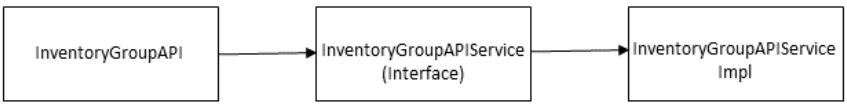
About Class Diagrams

In the Reference REST web service, a sample POST method createInventoryGroup operation is available. You use the **createInventoryGroup** sample as a template while creating the custom REST web services.

Consider the following recommendations:

- Follow the naming convention of <HTTP method definition><EntityName appended with sample> for consistency on new operations.
  - <HTTP method definition>: POST means create, GET means retrieve, DELETE means delete, PATCH means update.
  - < EntityName appended with sample>: Do not use duplicate schema names with product REST, similar to LogicalDevice in custom webservice where you use LogicalDeviceSample.
- Follow the sample template code for the user environment and transaction management. See "Transaction Guidelines for Rest" for more information on transaction management.
- Run UIM core functionality by invoking the API manager methods.

The following figure shows the recommended class design for the implementation class. The UIMSample1_0 .yaml file is used to generate the InventoryGroupApiServiceImpl, InventoryGroupApiService, and InventoryGroupType source files. The InventoryGroupApiServiceImpl.java example file provides a template class that implements the interface in the InventoryGroupApiService.java and the InventoryRootService .java source file within UIM.



The UIMSample1_0.yaml file has a sample structure of OpenApiSpecification.

The Open API Object contains:

- openapi: The version of OAS
- info: Contains general information about API like title, description, version, and so on
- tags: Used to grouping the API resources
- paths: Defines the endpoints of API
- components: Used to define data model (schemas)

Copyright © 2014, 2026, Oracle and/or its affiliates.

A web service operation signature contains the following:

- Paths: Defines the URL endpoint (/inventoryGroup) with baseuri from RestCustomUtils.java to form @Path("/customInventoryManagement/v3/inventoryGroup").
  - Http method: The endpoint with Http method post, get, and so on that creates **@POST** when the source file is generated.
    - OperationId: The operation to be performed for the endpoint with the corresponding HTTP method is defined here.
    - Parameters: The required set of parameters to be passed during runtime to the endpoint. For example: '/inventoryGroup/{id}' where parameter **id** is required.
    - requestBody: The schema (model) that is provided in the code generating @Consumes({ "application/json" }), which customizes the request to have content-type. If no content-type is mentioned, an HTTP “415 Unsupported Media Type” error occurs.
    - Responses: All possible server responses along with content-type to customize the response @Produces({ "application/json" }) are generated.
- Components: The data models that describe your API inputs and outputs in schema section
- Schemas: Defines all models used in paths. For example: InventoryGroup which will be generated as InventoryGroupType
  - Type: The type of model. For example: string, object.

    > **Note**
    >
    > The Object types further contain properties to define the model variables.

  - Properties: A variety of properties that can be defined with along with its type and description. You can reuse the models using **$ref**.

The following operations are run as per the **UIMSample1_0.yaml** file:

- POST (create): createInventoryGroup
- DELETE (delete): deleteSampleIG
- GET (retrieve): retrieveSampleIg
- PATCH(update): updateInventoryGroup

Transaction Guidelines for the REST Web Services

The Reference web service performs transaction actions in a specific order while managing operation transactions.

> **Note**
>
> You must follow the steps in the following order. Otherwise, transaction errors may occur.

To manage the transaction, you write code that performs the following steps:

Copyright © 2014, 2026, Oracle and/or its affiliates.

1. Start the user environment.
2. Start the transaction.
3. Set the user environment on the transaction.
4. Set up the request, call the API method on the entity manager class, and manage the response.
5. Commit or rollback the transaction.
6. (Optional) Perform a rollback when an error occurs.
7. Ensure the user environment is ended with a call to the endUserEnvironment method on success or failure.

The sample from InventoryGroupApiServiceImpl.java is as follows:

```
UserEnvironment userEnvironment = null;
          InventoryTransactionValue transValue = null;
          InventoryGroupManager lgManager =
PersistenceHelper.makeInventoryGroupManager();
          InventoryGroup result = null;
          InventoryGroupType resource = null;
          try {
              userEnvironment =
startUserEnvironment(restUtils.getHttpRequest());
              transValue = startTransaction();
              transValue.setUserEnvironment(userEnvironment);
            ……….
             result = lgManager.createInventoryGroup(ig);
           ……..……
finally {
            commitOrRollback(transValue);
```

Developing the REST Web Services

Developing a new web service involves generating new model file, new API file, new API impl file, and creating new Java source files. After you create these files, you should copy model file, API file, and API impl file to the corresponding paths within the **reference_rest_webservice** directory.

This section provides information about creating and copying these files.

Generating and Copying Model, API, and API impl Files

To generate and copy model, API, and API impl files:

1. Navigate to the **reference_rest_webservice** directory and run the following command:

   ```
   gradlew build
   ```

   The above command generates Model, API, and API impl files within the **reference_rest_webservice** directory.
2. Open the generated Model file from the **reference_rest_webservice/build/swagger-code-resourcemodel/src/main/java** folder.

Copyright © 2014, 2026, Oracle and/or its affiliates.

3. Remove the following lines from the Model file:

```
import org.springframework.validation.annotation.Validated;
@Validated
```

4. Save the Model file and copy the updated file to the **reference_rest_webservice\src\oracle\communications\inventory\rest\model** folder.
5. Copy the API file from the **reference_rest_webservice\build\swagger-code-resourceapi\src\gen\java\oracle\communications\inventory\rest\api** folder to the **reference_rest_webservice\src\oracle\communications\inventory\rest\model\api** folder.
6. Copy the API impl file from the **reference_rest_webservice\build\swagger-code-resourceapi\src\main\java\oracle\communications\inventory\rest\api\impl** folder to the **reference_rest_webservice\src\oracle\communications\inventory\rest\api\impl** folder.

Creating Java Source Files

Update the content in <Entity Name appended with sample>Impl.java as per the requirements.

You can refer to the InventoryGroupApiServiceImpl.java , IpSubnetSampleApiServiceImpl.java, LogicalDeviceSampleApiServiceImpl.java files.

Use the adapter information from uim-webservices-rest-adapter.jar that delivered as a part of UIM_LIB (<DOMAIN_HOME>/UIM/lib) for associating any existing entities from the product.

> **Note**
>
> A sample REST SDK testing payload is available at reference_rest _webservice.zip/doc/SamplePayload.txt.

Generating Java Source Based on the YAML File

To generate Java source from YAML using the Gradle build file:

1. Clean the **reference_rest_webservice** directory as follows:

```
gradlew clean
```

2. If the Model, API, and API imple files are not generated yet, run the following command:

```
gradlew build
```

3. Copy any properties files that store localized error messages to the corresponding UIM deployment directory **UIM_Home/config/resources/logging** as follows:

```
gradlew copyResources
```

> **Note**
>
> The properties files are located within a ZIP file in the **config/resources/logging** directory.

Copyright © 2014, 2026, Oracle and/or its affiliates.

Creating a WAR File

A WAR file contains the compiled classes from the developed custom web service.

Before creating a WAR file, copy all JAR files that are pulled by Gradle into the **reference_rest_webservice\lib** folder, which can be used as CART\lib, as follows:

```
gradlew copyLibs
```

To create a WAR file with Model, API, and API impl, run the following command:

```
gradlew war
```

You can customize the name for the WAR file by updating the **COMPUTERNAME.properties** file. The generated WAR file resides in the **reference_rest_webservice/build/libs** directory.

Packaging the WAR File in EAR File

The Reference REST Web Service WAR file is not packaged in the **inventory.ear** file and is therefore not automatically deployed into UIM. Rather, you must manually import the provided WAR file into an EAR file to deploy.

In UIM traditional deployments, when developing custom web services, you have the option of packaging the custom web service WAR file into:

- The **custom.ear** file: If you develop a single custom web service, Oracle recommends you use the provided **custom.ear** file.
- Any custom EAR files: If you develop multiple custom web services, Oracle recommends you use a separate custom EAR for each web service. This approach involves additional development work as you must create and configure your message queue and corresponding listener class. It is suitable for multiple custom web services and provides a better performance.

Extracting and Updating the EAR File

To include the corresponding custom REST web service WAR file name, extract and update the corresponding EAR file. The provided custom EAR file contains an application.xml file that you can manually extract, use as a starting point, and modify as needed.

To extract the EAR file, run the following command:

```
gradlew extractCustomEar
```

Copying application.xml and the WAR File into the EAR Folder

To copy the **application.xml** and WAR files into the corresponding EAR folders: **reference_rest_webservice\META-INF\application.xml** and **reference_rest_webservice\build\libs\ReferenceRestUim.war**

Copyright © 2014, 2026, Oracle and/or its affiliates.

Redeploying custom.ear

If you have placed your custom REST web service in the **custom.ear** file, or in any custom EAR file, you must specify a deployment plan for the updated EAR file.

To specify a deployment plan:

1. Log in to the WebLogic Remote Console.
2. Click **Monitoring Tree**.
3. In the left panel, under Domain Structure, navigate to **Deployments**, and then click **Application Management**.
   The Summary of Application Management page appears.
4. Select the check box next to the updated EAR file that contains your custom web service.
5. Click **Update/Redeploy**.
   The Update Application Assistant page appears.
6. Click **Change Path** and then click **Next**.
   The deployment plan is specified and **custom.ear** is redeployed.

Copyright © 2014, 2026, Oracle and/or its affiliates.

6

Deploying, Testing, and Securing UIM Web Services

This chapter provides information about deploying, testing, and securing Oracle Communications Unified Inventory Management (UIM) Web Services and any custom web services you may have created.

Deploying Web Services

> **Note**
>
> In UIM cloud native deployments, you must build customized images for deploying web services. For more information, see "Customizing Images" in *UIM Cloud Native Deployment Guide*.

Each web service is packaged in a WAR file, which is packaged in an EAR file. When you deploy the EAR file, you also deploy any web services that are packaged within the EAR file.

For example, the Service Fulfillment Web Service is packaged in the **inventory.ear** file, within the **InventoryWS.war** file. So, when you deploy the **inventory.ear** file, you also deploy the Service Fulfillment Web Service.

For instructions on how to deploy the **inventory.ear** file, see "Unified Inventory Management System Administration Overview" in *UIM System Administrator's Guide*.

For custom web services, you have the option of placing the custom WAR file within the **custom.ear** file, within any custom EAR file, or within the **inventory.ear** file. So, when you deploy the **custom.ear** file, or the specified custom EAR file, or the **inventory.ear** file, you also deploy the custom web service. See Developing Custom SOAP Web Services for more information.

Copyright © 2014, 2026, Oracle and/or its affiliates.

> **Note**
>
> If custom web services are packaged as part of an EAR file, the EAR must have a deployment plan defined with the following:
>
> 1. The plan directory is *UIM_Home*/app/plan. This is to ensure that the EclipseLink cache coordination configuration is available in the **classpath** for the custom EAR file.
> 2. For the deployment plan (the XML configuration file) ensure the following conditions are met:
>
> - If the EAR file has a deployment plan, it must be created under the plan directory *UIM_Home*/app/plan.
> - If the EAR file does not have any deployment plan, associate the existing plan (located under the plan directory *UIM_Home*/app/plan as **plan.xml**), to the custom EAR deployment. For a clustered environment, the plan file name is **ClusterPlan.xml**.

Verifying Deployments

You can verify that any UIM web service is deployed by viewing it in the WebLogic Server Administration Console.

To verify that a UIM web service is deployed:

1. Log in to the WebLogic Remote Console.
2. Click **Monitoring Tree**.
3. In the left panel, under Domain Structure, navigate to **Environment**, **Servers**, **AdminServer**, **Deployments**, and then **Application Runtimers**.

   The Summary of Application Runtimers page appears.
4. Expand **oracle.communications.inventory**.
5. Under **oracle.communications.inventory**, expand **Component Runtimers**.
6. Click the link that represents the name of the web service.

   The Settings page for the selected web service appears.
7. Click the **Testing** tab.
8. Expand the name of the web service.
9. Under the expanded web service, edit the **Context Root URL** to the required **WSDL URL** of the web service endpoint.

   The WSDL file appears. Here, you can view the web service operations that are deployed.

Testing Web Services

After you successfully deploy the web service, you can test the web service.

Web services can be tested by using any software designed to test web services, such as:

- LISA for testing SOAP XML through HTTP or JMS
- SoapUI for testing SOAP XML through HTTP

Copyright © 2014, 2026, Oracle and/or its affiliates.

> **Note**
>
> If you want to test the UIM cloud native instance, you should update proxy settings in SoapUI Preferences to exclude the domain name. The default domain name in the UIM cloud native toolkit is **uim.org**.

- HermesJMS for testing SOAP XML through JMS

Test Input XML

The UIM installation provides the GSM 3GPP cartridge pack and the Cable TV cartridge pack, and both cartridge packs use the Service Fulfillment Web Service. The cartridge packs provide test input XML that you can use to test the Service Fulfillment Web Service operations. For additional information about these cartridge packs, see *UIM GSM 3GPP Cartridge Pack Guide* and *UIM Cable TV Cartridge Pack Guide*.

You can also generate your own test input XML by using any software that generates XML based on schema, such as XML Spy, LISA, SoapUI, and so forth.

Pre-configuration for Testing

Before running the Service Fulfillment Web Service operations, you must have the UIM base cartridges deployed into your UIM environment. The base cartridges are located in the *UIM_SDK_Home***/cartridges/base** directory. For additional information about the base cartridges, see "Overview" in *UIM Cartridge Guide*.

Be aware of any pre-configurations that must be in place before testing any custom web services.

Securing Web Services

> **Note**
>
> In UIM cloud native deployments, changes such as **Adding Policy, Updating Policy, Removing Policy, or Updating Deployment Plan** that you perform using Oracle WebLogic Console do not persist once the domain restarts. Therefore, before making the changes, you should package the updates in a customized image. Refer to *UIM System Administration Guide* for more information on securing web services. Refer to the following sections to understand policies.

The Service Fulfillment Web Service has security enabled upon installation. Specifically, the HTTP and JMS web service ports are associated to the default WebLogic security policy file, **Auth.xml**. As a result, a user name and password must be sent in clear text over a secure tunnel (HTTPS/t3s).

> **Note**
>
> The user name and password, and the payload, are not encrypted to avoid significant performance impacts.

Copyright © 2014, 2026, Oracle and/or its affiliates.

When you create a new web service, it is up to you to secure the web service. See "Securing Custom Web Services" for more information.

About Policy Files

A policy file can be associated to a port, or to a specific operation defined for the port. When a policy file is associated to a port, it automatically secures all operations defined for the web service. When a policy file is not associated to a port, a policy file can be associated to one or more operations. If necessary, each operation can specify a different policy file. If no policy file is associated to the port, or to any operations, the web service is unsecured and no security validations are performed.

Upon installation of UIM, the WebLogic default policy file, **Auth.xml**, is associated to UIMInventoryHTTPPort and UIMInventoryJMSPort. So, all operations are automatically secured, and all operations under each port require a user name and password in the SOAP message header. Example 6-1 shows a SOAP message header with a user name and password specified.

Example 6-1 SOAP Message Header

```
<soapenv:Envelope xmlns:com="http://xmlns.oracle.com/communications/inventory/webservice/
common" xmlns:ser="http://xmlns.oracle.com/communications/inventory/webservice/service"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soapenv:Header>
        <wsse:Security soapenv:mustUnderstand="1"
        xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
        wss-wssecurity-secext-1.0.xsd">
            <wsse:UsernameToken wsu:Id="UsernameToken-1"
            xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
            wss-wssecurity-utility-1.0.xsd">
            <wsse:Username>uimuser1</wsse:Username>
                <wsse:Password
                    Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
                    wss-username-token-profile-1.0#PasswordText">Welcome@123
                </wsse:Password>
            </wsse:UsernameToken>
        </wsse:Security>
    </soapenv:Header>
    <soapenv:Body>
        <ser:captureInteractionRequest>
            .
            .
            .
        </ser:captureInteractionRequest>
    </soapenv:Body>
</soapenv:Envelope>
```

Modifying Web Service Security

You can modify the default security settings through the WebLogic Server Administration Console.

To modify the default web service security settings, see the following:

- Accessing Security
- Associating a Policy File
- Disassociating a Policy File
- Modifying the Deployment Plan

Copyright © 2014, 2026, Oracle and/or its affiliates.

Accessing Security

You can access the security policy currently applied to the web service at:

- WSDL
- Deployment plan
- WEB-INF\weblogic-webservices-policy.xml

For information on accessing the security policies in WSDL, see "Verifying Deployments".

For information on accessing the security policies in deployment plan and WEB-INF\weblogic-webservices-policy.xml, see "Authenticating Web Services in UIM" in *UIM System Administrator's Guide*.

Associating a Policy File

You can associate a policy file to a port, or to a specific operation defined for the port. Use this capability to configure security for the web service.

For more information on accessing security, see "Accessing Security" and for information on adding authentication to the web services, see "Authenticating Web Services" in *UIM System Administrator's Guide*.

Disassociating a Policy File

Remove the policy file from the location or path where you initially added it during association.

For more information on accessing security, see "Accessing Security" and for information on associating a policy file, see "Associating a Policy File".

Securing Custom Web Services

When you create a new web service, it is up to you to secure it. How you secure the web service depends upon how you created the web service. For example, if your custom web service is deployed with the **custom.ear** file, create a separate deployment plan. If your custom web service is deployed with the **inventory.ear** file, modify the existing deployment plan associated with **inventory.ear** that is provided with the UIM installation (**UIM_Home/app/plan/Plan.xml**).

To secure a custom web service, configure security for the web service.

For more information on accessing security, see "Accessing Security" and for information on associating a policy file, see "Associating a Policy File".

Copyright © 2014, 2026, Oracle and/or its affiliates.