Oracle® Communications Order and Service Management

Cartridge Guide for Oracle Application Integration Architecture

Release 2.1.2

E79217-01

March 2018

Oracle Communications Order and Service Management Cartridge Guide for Oracle Application Integration Architecture, Release 2.1.2

E79217-01

Contents

Preface

Oracle Communications Order and Service Management (OSM) delivers pre-built cartridges supporting the Order-to-Activate business process to be used with the Oracle Communications Order to Cash Integration Pack for Oracle Communications Order and Service Management. This guide provides information about the OSM Order-to-Activate cartridges for the Oracle Communications Order to Cash Integration Pack for Oracle Communications Order and Service Management. It explains how to install and deploy the cartridges and provides detailed information and best practices on how to extend them for your own implementation.

Note: The Oracle Application Integration Architecture Order-to-Activate Process Integration Pack is renamed to Oracle Communications Order to Cash Integration Pack for Oracle Communications Order and Service Management. The OSM cartridges are referred to as Order-to-Activate cartridges in all OSM documentation because they support the Order-to-Activate business sub-process within the overall Order to Cash business process for service providers. See "Order-to-Activate Business Process Overview" for a description of the Order-to-Activate business sub-process.

The term Oracle Communications Order to Cash Integration Pack for Oracle Communications Order and Service Management and the term Order to Cash Integration Pack for OSM are used synonymously in OSM documentation.

For more information about the Oracle Communications Order to Cash Integration Pack for Oracle Communications Order and Service Management, see *Oracle Application Integration Architecture Oracle Communications Order to Cash Integration Pack Implementation Guide for Siebel CRM, Oracle Communications Order and Service Management, and Oracle Communications Billing and Revenue Management* in the Oracle Application Integration Architecture documentation.

Audience

This document is intended for programmers who have a working knowledge of:

- System interfaces
- Java development
- Java Messaging Service (JMS)
- XML Technologies (including XQuery and XPath)

This document assumes that you have read *OSM Concepts*, and have a conceptual understanding of:

- Cartridges
- Topologies
- Orders
- Order states
- Tasks
- Task states

Accessing Oracle Communications Documentation

OSM documentation and additional Oracle documentation (such as database and WebLogic Server documentation) is available from the Oracle Help Center website:

`http://docs.oracle.com`

Additional Oracle Communications documentation is available from the Oracle software delivery website:

`https://edelivery.oracle.com`

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.

Document Revision History

The following table lists the revision history for this guide.

Version	Date	Description
E79217-01	March 2018	Initial release

1 Overview of the Order-to-Activate Cartridges

This chapter describes how to use the Oracle Communications Order and Service Management (OSM) Order-to-Activate cartridges for the Oracle Communications Order to Cash Integration Pack for Oracle Communications Order and Service Management (Order to Cash Integration Pack for OSM).

About the Application Integration Architecture Order-to-Activate Cartridges

The Order-to-Activate cartridges are pre-built OSM cartridges that support the Oracle Order-to-Activate business process to be used with the Order to Cash Integration Pack for OSM. See "Order-to-Activate Business Process Overview" for a discussion of the Order-to-Activate business process.

Oracle Application Integration Architecture (Oracle AIA) integrates Oracle applications, such as OSM, Siebel Customer Relationship Management (Siebel CRM), Oracle Configure, Price, and Quote Cloud (Oracle CPQ Cloud), and Oracle Communications Billing and Revenue Management (BRM). External systems, such as workforce management applications, can also be included in the solution.

In the Order-to-Activate cartridges:

- OSM performs central order management by orchestrating the fulfillment of customer orders coming from Oracle AIA.
- OSM performs service order management by orchestrating service orders sent to fulfillment systems.

See *OSM Concepts* for more details.

Order-to-Activate Business Process Overview

The Oracle Order-to-Activate business process is at the core of business and operational support systems for any Communications Service Provider (CSP). The process extends from the time a quote or order is created to the time when the goods and services are delivered and properly billed. The Order-to-Activate cartridges can be used in an architecture that has Siebel CRM, Oracle CPQ Cloud, or both together.

The following are the steps for the functional flow of the Order-to-Activate business process for orders coming from Siebel CRM:

1. A customer order is captured in Siebel CRM. For some orders, the order may require technical qualification, such as validating that the network has enough capacity to offer the purchased products. After an order capture is complete and

the order is validated in Siebel CRM, the system submits it to OSM in the central order management role for delivery.

2. Customer orders (both Qualify and Deliver request types) received in OSM in the central order management role are first recognized (as Oracle AIA customer orders), mapped to fulfillment patterns, and enriched with fulfillment metadata.
3. OSM in the central order management role decomposes (and transforms, if the calculate service order solution option is used) the customer order, dividing it into suborders, called order components, which have cross-order components, cross-order lines, and cross-order dependencies that reflect the specific demands of the CSP.
4. The outcome is an order orchestration plan that is uniquely generated to match the fulfillment needs of that order. The fulfillment flow that is produced orchestrates fulfillment requests to different fulfillment providers (such as fulfillment system instances or stacks) using preconfigured fulfillment functions, like sync customer, initiate and fulfill billing, and provision order. OSM Order-to-Activate cartridges provide out of the box ready-to-use automatic integration to Oracle AIA web services. When the BRM pre-built integration option is in use, it takes the billing related requests (Sync Customer, Initiate and Fulfill Billing) made by OSM in the central order management role to Oracle AIA, from Oracle AIA to BRM. The Sync Customer Oracle AIA process integration also uses the Siebel CRM pre-built integration option to get customer account details.
5. OSM in the central order management role manages Order Lifecycle Management (OLM) events. For cancel and revision requests, OSM generates and executes compensation plans to efficiently match a change. OLM manages order data and status updates, and order fallout.
6. Throughout the fulfillment process, OSM in the central order management role maps fulfillment function responses to common statuses, which are then aggregated into order line statuses and order header status values. The status management capability updates Siebel CRM with relevant customer status and milestone values. OSM updates Siebel CRM when order lines reach their point-of-no-return (PoNR) to prevent the submission of new revisions. It also updates Siebel CRM with any enrichment to order lines that may have occurred during fulfillment. Errors may occur for many reasons. Oracle AIA reports such errors to OSM for fallout management. Additionally, validation logic in OSM may raise fallout incidents.
7. OSM detects, reports, and resolves order fulfillment fallout incidents such as system, validation, and fulfillment errors. The Oracle approach creates trouble tickets in Siebel CRM to take advantage of the rich notification, reporting, and management capabilities of Siebel CRM.

The following are the steps for the functional flow of the Order-to-Activate business process for orders coming from Oracle CPQ Cloud:

1. A customer order is captured in Oracle CPQ Cloud. After an order capture is complete and the order is validated in Oracle CPQ Cloud, the system submits it to OSM in the central order management role for delivery.
2. Customer orders received in OSM in the central order management role are first recognized (as Oracle AIA customer orders), mapped to fulfillment patterns, and enriched with fulfillment metadata.
3. OSM in the central order management role decomposes (and transforms, if the calculate service order solution option is used) the customer order, dividing it into suborders, called order components, which have cross-order components,

cross-order lines, and cross-order dependencies that reflect the specific demands of the CSP.

4. The outcome is an order orchestration plan that is uniquely generated to match the fulfillment needs of that order. The fulfillment flow that is produced orchestrates fulfillment requests to different fulfillment providers (such as fulfillment system instances or stacks) using preconfigured fulfillment functions, like sync customer, initiate and fulfill billing, and provision order. OSM Order-to-Activate cartridges provide out of the box ready-to-use automatic integration to Oracle AIA web services. When the BRM pre-built integration option is in use, it takes the billing related requests (Sync Customer, Initiate and Fulfill Billing) made by OSM in the central order management role to Oracle AIA, from Oracle AIA to BRM. The Sync Customer Oracle AIA process integration also uses the OSM Account Manager pre-built integration option to get customer account details.
5. OSM in the central order management role manages Order Lifecycle Management (OLM) events. For cancel and revision requests, OSM generates and executes compensation plans to efficiently match a change. OLM manages order data and status updates, and order fallout.
6. Throughout the fulfillment process, OSM in the central order management role maps fulfillment function responses to common statuses, which are then aggregated into order line statuses and order header status values. The status management capability updates Oracle CPQ Cloud with relevant customer status and milestone values. OSM updates Oracle CPQ Cloud when order lines reach their point-of-no-return (PoNR) to prevent the submission of new revisions. It also updates Oracle CPQ Cloud with any enrichment to order lines that may have occurred during fulfillment. Errors may occur for many reasons. Oracle AIA reports such errors to OSM for fallout management. Additionally, validation logic in OSM may raise fallout incidents.

The Order-to-Activate business process is a sub-process within the Order to Cash business process. The Order to Cash Integration Pack for OSM pre-built integration provides CSPs deployment and integration accelerators that build on forward-looking industry methodology and best practices. The Order to Cash Integration Pack for OSM automates Business Support Systems (BSS) Concept to Launch and BSS Order-to-Activate processes across Siebel CRM, Oracle CPQ Cloud, OSM, BRM, and Oracle Product Hub for Communications.

For more information about the Order to Cash business process see *Oracle Application Integration Architecture Oracle Communications Order to Cash Integration Pack Implementation Guide for Siebel CRM, Oracle Communications Order and Service Management, and Oracle Communications Billing and Revenue Management* in the Oracle Application Integration Architecture documentation.

Overview of the Order-to-Activate Cartridges

The Order to Cash Integration Pack for OSM solution integrates several Oracle applications that play particular roles in order processing:

- Siebel CRM for order capture and trouble ticketing
- Oracle CPQ Cloud for cloud-based order capture
- OSM for order processing and service fulfillment
- Oracle Communications Design Studio for product specification definition including fulfillment metadata and order line to fulfillment pattern mapping
- BRM for rating, billing, and revenue management

- Oracle AIA Error handling Framework for Fallout management

The order is captured by Siebel CRM or Oracle CPQ Cloud and is sent to OSM (in its central order management role) for processing. Using the recognition rules and other entities provided by the OSM cartridges in the Order to Cash Integration Pack for OSM solution, OSM decomposes the order and dynamically generates an orchestration plan that is used to manage the fulfillment of the customer's order across other enterprise systems.

To manage service fulfillment, OSM in the central order management role creates service orders that it sends to OSM in the service order management role. Depending on the order, recognition rules can be used again to process the order. Each service order is decomposed into processes and tasks that handle the order fulfillment.

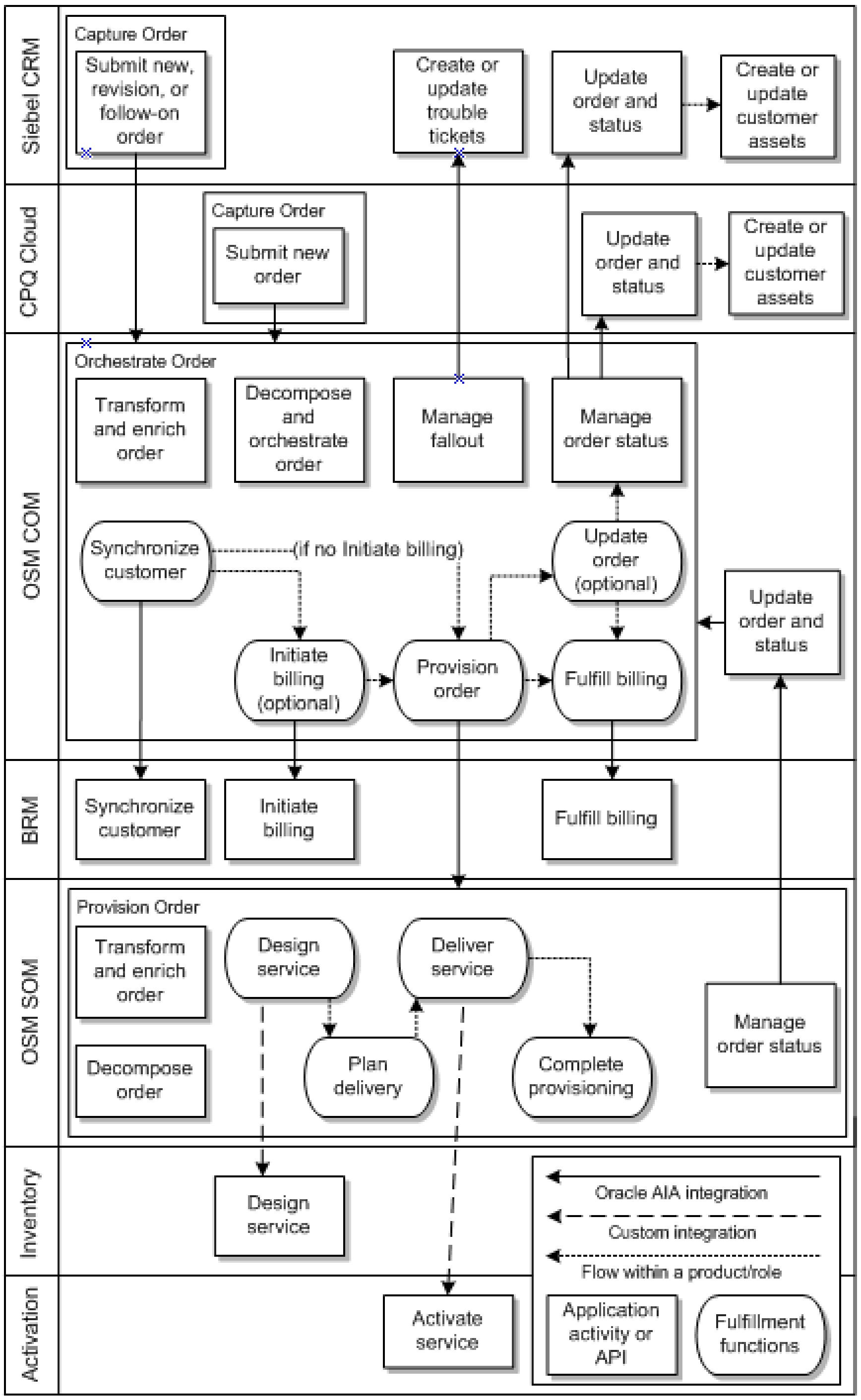
In the Oracle AIA solution, OSM does not directly interact with billing, CRM, or Provisioning systems. It interacts with Oracle AIA which in turn uses BRM Application Business Connector Service (ABCS) for billing and CRM ABCS for Siebel CRM.

For more details on Oracle AIA, Siebel CRM, Oracle CPQ Cloud and Oracle AIA interactions, see *OSM Concepts* and *Oracle Application Integration Architecture Oracle Communications Order to Cash Integration Pack Implementation Guide for Siebel CRM, Oracle Communications Order and Service Management, and Oracle Communications Billing and Revenue Management*.

Figure 1–1 illustrates the integration between the systems. The integration includes the following:

- Customer order submission from Siebel CRM and Oracle CPQ Cloud to OSM and updates from OSM to Siebel CRM and Oracle CPQ Cloud
- Siebel CRM or Oracle CPQ Cloud creates or updates customer assets internally in response to status messages from OSM.
- Customer data synchronization and order billing from OSM to BRM
- Service provisioning from OSM central order management to OSM service order management
- Trouble ticket logging for fallout from OSM to Siebel CRM

Figure 1–1 Order-to-Activate Cartridges System Interactions Flow



Order-to-Activate Cartridge Solution Options

The Order-to-Activate cartridge solution has two options.

- Calculate Service Order: This solution option includes conceptual model entities, the order transformation manager, and Calculate Service Order. It provides access to the latest improvements in OSM and enhances the functionality of the solution.

For more information about the order transformation manager and Calculate Service Order, see *OSM Concepts*.

- Non-Calculate Service Order: This solution option does not include the order transformation manager or Calculate Service Order. It is like the pre-2.1.0 versions of the Order-to-Activate cartridges, but it also includes the conceptual model entities that were not available before.

For more information about the conceptual model, see *Design Studio Concepts*.

The two options cannot be used together, so you must use the central order management cartridges and service order management cartridges from the same option.

OSM Cartridge Types Supporting the Order to Cash Integration Pack for OSM Solution

There are two categories of cartridges that support the Order to Cash Integration Pack for OSM solution in OSM: productized cartridges and demonstration cartridges.

Productized cartridges are customized cartridges supplied by Oracle. They support integration with other applications.

Demonstration cartridges demonstrate the capabilities of OSM and are preconfigured with fulfillment patterns either in Simple or Typical topologies. See "About Fulfillment Topologies" for more details on topologies.

Demonstration cartridges complement productized cartridges to provide a working end-to-end sample set of product specifications and fulfillment patterns. See "Extending the Cartridges" for more details.

OSM central order management orchestrates the fulfillment of customer orders by mapping them to the product specifications of the demonstration cartridges.

OSM entities (recognition rules, tasks, roles, decomposition sequences, and others) play a vital role in the Order to Cash Integration Pack for OSM solution. For more information on an entity in a cartridge, open the entity in Design Studio and click the Information icon.

The cartridges are subclassified into central order management cartridges, or service order management cartridges depending on the fulfillment functions they perform.

Extending the Cartridges

Using Design Studio, OSM enables you to extend the functionality of a productized cartridge to have required functionality.

You can consider a demonstration cartridge as a starting point to understand the capabilities it can offer and then plan to extend the productized cartridge according to your requirements. You can extend a productized cartridge by adding product specifications, inheriting from existing specifications, fulfillment patterns, decomposition sequences, and modifying other entities.

See "Extending Order-to-Activate Cartridges" for more information on extending cartridges.

Example

If you have productized and demonstration cartridges in different namespaces with corresponding product specification type entities, you can create a customized product specification by modifying the product specification type entity in the demonstration cartridge and mapping it to the appropriate product specification type entity in the productized cartridge.

To facilitate this customization, OSM derives the **ProductSpec** name from the property **ProductSpecMappingProperty** of the **OrderItemSpecifcation** entity and lets you map it to the appropriate entity in the productized cartridge under the same namespace.

Note: A namespace is a unique qualifier that logically binds related entities, cartridges, and specifications. To view the namespace and other details about an entity, open the entity in Design Studio and click the Information icon.

Time Zones in Order-to-Activate Cartridges

OSM supports orders and users in multiple time zones. The time zone used by OSM is configured on the server at system installation, and is used to time-stamp incoming and outgoing orders and to schedule work for groups. See *OSM Installation Guide* for more details.

When OSM uses Order-to-Activate cartridges, the OSM server accepts and processes only those orders that have time stamps in the Coordinated Universal Time (UTC) in the GMT time zone (also called the Z convention). For example, 2010-03-12 08:23Z.

The Order-to-Activate cartridges support only Z convention-based fields, except for the **RequestedDeliveryDateTime** field.

The **RequestedDeliveryDateTime** field on the ProcessSalesOrderFulfillmentEBM (incoming customer order) is mapped to the web service API's **date time** field for initial order creation. This field allows the use of the **+/-hh:mm** convention along with the Z convention.

The other Oracle AIA-relevant date and time fields that follow the Z convention are:

- ActualDeliveryDateTime
- ExpectedDeliveryDateTime
- EarliestDeliveryDateTime
- StartDateTime
- EndDateTime
- ServiceUsageStartDateTime
- PurchaseDate
- CycleStartDateTime

Order Creation in the Order-to-Activate Cartridges

In the course of processing a customer order that has been received and created by the Order-to-Activate cartridges, the following additional orders are created automatically, in this order:

1. CloseCreationFailedTroubleTicketOrder: An order of this type is created when the customer order is created. It determines whether there are any open trouble tickets for previous revisions of this order. If it finds any open trouble tickets for the order, it closes them. This order is then closed automatically.

2. ResumePendingInBoundMessage: An order of this type is created when the customer order transitions to the In Progress state. This order checks to see whether any messages are waiting for the order that might have been received when the order was temporarily in a Suspended state. If any such messages are found, they are rerouted to the normal message queue for the provisioning order. This order is then closed automatically.
3. CloseCreationFailedTroubleTicketOrder: Another order of this type is created when a customer order is completed successfully. It determines whether there are any open trouble tickets for the customer order (for example, if someone has recovered from fallout manually and not closed the trouble ticket manually). If it finds any open trouble tickets for the order, it closes them. This order is then closed automatically.

Order-to-Activate Emulators

Emulators are included with the Order-to-Activate cartridges. These emulators enable you to perform testing before all of the solution components are connected. An Ant build file is used to build and deploy emulators, which are enterprise applications built and deployed into WebLogic for central order management and service order management. There are different sets of emulators available with the Order-to-Activate cartridges:

- The Oracle AIA emulators emulate responses from Oracle AIA when a central order management cartridge is used in a standalone (without integration with other applications) environment.
- The Inventory emulators emulate responses from the Unified Order Management (UIM) software. These responses are used when service order management is used without a connection to a live UIM system for inventory requests.
- The technical order management emulators emulate responses from a technical order management system, such as responses to activation commands. These responses are used when service order management is used without a connection to a live technical order management system.

About Fulfillment Topologies

A fulfillment topology defines the arrangement of various network elements, processes, systems, software, that are used to perform a complete service. The Order to Cash Integration Pack for OSM solution comes with three sample fulfillment topology definitions:

- **Simple fulfillment topology**: This topology, available for both versions of the Order-to-Activate solution, supports a single instance of each fulfillment system.
- **Typical fulfillment topology**: This topology supports one Siebel CRM or Oracle CPQ Cloud system, three BRM system instances, and three provisioning system instances. See "Order-to-Activate Cartridge Solution Options" for more information about solution types.
- **Complex fulfillment topology**: This topology supports multiple instances of all fulfillment systems. See "Order-to-Activate Cartridge Solution Options" for more information about solution types.

You can use the sample fulfillment topologies as examples to for configuring your own topologies for providing order fulfillment services. You can build your own topologies depending on the systems and instances required. Generally your fulfillment topology includes all of the systems that participate in the order capture and order fulfillment.

OSM uses fulfillment patterns named Danube and Nile (code names for sample fulfillment patterns), for Simple and Typical topologies, respectively. These fulfillment patterns:

- Match the number of fulfillment system types used in each of the fulfillment topology scenarios
- Stay agnostic to the number and domain of fulfillment providers, that is, the fulfillment patterns is independent of the number of system instances participating (For instance, even if there are three billing system instances, the fulfillment pattern for each product specification remains the same)
- Provide fulfillment pattern variations that collectively provide significant coverage of requirements

See *OSM Concepts* for more details on topologies.

Simple Fulfillment Topology

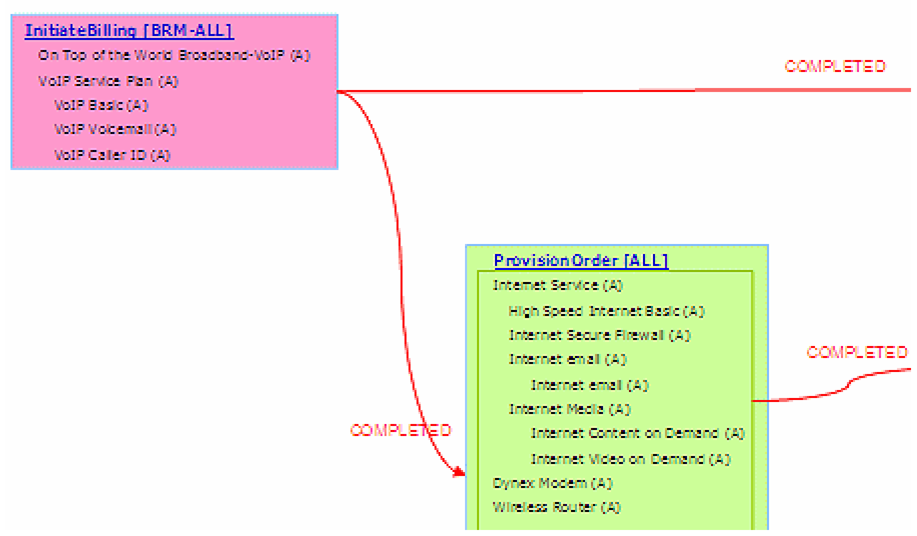
The Simple fulfillment topology uses one Siebel CRM or Oracle CPQ Cloud system, one BRM system, and one provisioning system in the process of fulfilling an order.

The sample demonstration cartridge adopts the Simple fulfillment topology and Danube fulfillment pattern in fulfilling an order. That is, in Simple fulfillment topology, the relationship between Siebel CRM or Oracle CPQ Cloud, BRM, and central order management is set to support communication using the Danube fulfillment pattern in fulfilling an order.

The Danube Fulfillment Pattern

The Danube fulfillment pattern is used with the Simple fulfillment topology in the OSM fulfillment process. Figure 1–2 illustrates a smaller portion of a sample Danube fulfillment pattern.

Figure 1–2 Danube Fulfillment Pattern



- The main fulfillment functions in Figure 1–2 are represented in a box and are indicated by the bold item underlined. The activity name is followed by the target fulfillment system instance in square brackets. For example, InitiateBilling[BRM-ALL].

- The arrows between the fulfillment functions and the fulfillment pattern represent the dependency for starting the activity at the arrowhead end on the indicated milestone. For example, COMPLETED.

> **Note:** Milestones track the progress of the order fulfillment process. You can configure the milestones for each fulfillment pattern in various topologies. OSM sends the status updates to Siebel CRM or Oracle CPQ Cloud that include the details of the last reached milestone for each order line item.

- Dependencies are established at the order line level. For readability purposes, Figure 1–2 combines all dependencies between two order components into a single arrow.
- SyncCustomer is sensitive to only the Add(A), Update(U), and Move-Add (MA) fulfillment functions.

> **Note:** Each customer order line in the incoming customer order has an action code. Some fulfillment functions process order lines only with specific action codes.
>
> For example, SyncCustomer processes UPDATE order lines only when there are significant updates (certain fields have updated values). The following are some of the action codes:
>
> - Add: Adds a new instance
> - Update: Updates the current instance with the revised details
> - Move-Add: Adds a new instance after moving existing customer details to a new location. For example, you can add a new service to an existing customer after moving its details.
> - Delete: Deletes the current instance
> - Resume: Resumes the current instance
> - Suspend: Suspends the current instance
> - Move-Delete: Deletes an instance as part of moving the existing customer details.

- The relevant order line item actions indicated in Figure 1–2 are a property of the fulfillment function and not the fulfillment pattern.

Typical and Complex Fulfillment Topologies

The Typical fulfillment topology uses one Siebel CRM or Oracle CPQ Cloud system, three BRM system instances, and three provisioning system instances in the process of fulfilling an order.

The Complex fulfillment topology allows for multiple instances of any external system.

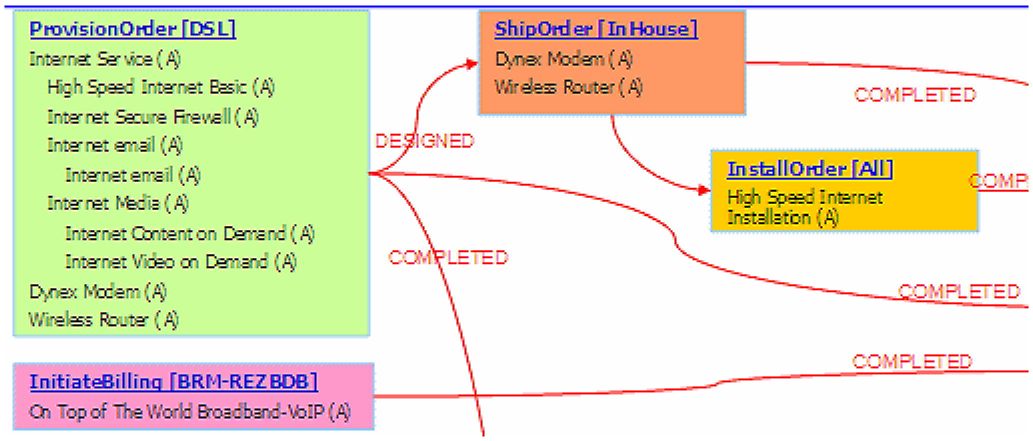
To fulfill an order in a Typical fulfillment topology, the Nile fulfillment pattern is used.

The Complex topology is similar to the Typical topology, and has one or more instance of each type of external system, depending on options selected while installing the Order-to-Activate cartridges.

The Nile Fulfillment Pattern

The Nile fulfillment pattern is used with the Typical and Complex fulfillment topologies in the OSM fulfillment process. The exact systems included depend on the options selected when installing the Order-to-Activate cartridges. Figure 1–3 depicts a smaller portion of the actual fulfillment pattern for the Typical topology.

Figure 1–3 Nile Fulfillment Pattern



- The main fulfillment functions in Figure 1–3 are represented in a box and are indicated by the bold item underlined. The activity name is followed by the target fulfillment system instance in brackets. For example, SyncCustomer[BRM-REZBDB].
- The arrows between the fulfillment functions and the fulfillment pattern represent the dependency for starting the activity at the arrowhead end on the indicated milestone. For example, COMPLETED.
- Dependencies are established at the order line item level. For readability purposes, Figure 1–3 combines all dependencies between two order components into a single arrow.
- Service Bundle (FulfillBilling processing granularity) is set to WholeItem FulfillBilling and OSM produces a single invocation in this case.
- SyncCustomer accepts all line items, and ProvisionOrder accepts all line items except billing-only line items.
- SyncCustomer is sensitive to only the Add(A), Update(U), and Move-Add(MA) fulfillment functions.
- The relevant order line actions indicated in Figure 1–3 are a property of the fulfillment function and not the fulfillment pattern.
- For Initiate - Fulfill billing fulfillment patterns, OSM fulfillment patterns are required to compute the new and prior values for the Start Cycle Date, Start Usage Date, and Purchase Date.

2 Performing an Interactive Installation of the Order-to-Activate Components

This chapter contains information about installing the Oracle Communications Order and Service Management (OSM) Order-to-Activate cartridges in an OSM environment using the interactive installer. It also provides information about uninstalling the cartridges.

Cartridge Installation Overview

The Order-to-Activate cartridges are installed into Oracle Communications Design Studio and deployed from there onto the OSM server. For the cartridges to work properly, various entities must be created in Oracle WebLogic Server in the server that contains OSM. An Ant script is provided to create these entities.

The general process for installing the OSM Order-to-Activate cartridges is:

- Ensure that the system requirements are met. See "System Requirements."
- Perform the pre-installation tasks, which set up the Design Studio workspace in Eclipse for the Order-to-Activate cartridges. See "Order-to-Activate Cartridge Pre-Installation Tasks."
- Install the Order-to-Activate cartridges. See "Installing the Order-to-Activate Cartridges." This activity includes:
  - Importing the installation cartridge and using it to import the other Order-to-Activate cartridges
  - Configuring the WebLogic server resources, which includes adding users and setting up communications for OSM
- If you intend to have two versions of the Order-to-Activate cartridges deployed to the same instance, you must ensure that the appropriate version handles any new orders. See "Post-Installation Tasks for Multiple Simultaneous Versions."
- Build the cartridges and deploy them to the OSM servers. See "Building and Deploying the Order-to-Activate Cartridges."
- (Optional) Run one or more test orders to validate that the installation was successful. See "Testing the Order-to-Activate Cartridges."

System Requirements

To install the Order-to-Activate cartridges successfully, ensure that you have the following software installed on your local Windows system:

- The supported version of WebLogic Server and Application Development Framework (ADF). (See *OSM Installation Guide* for more information.)
- OSM Software Development Kit (SDK) components.
- Java JDK: Use the version of Java that matches the one being used by the OSM server. See the discussion of software requirements in *OSM Installation Guide*.
- Eclipse with Design Studio plug-ins: See *Design Studio Installation Guide* for information about installing Design Studio plug-ins.

You must also have the following installed, either on your local Windows system or on a remote system:

- OSM server installed into the supported version of a WebLogic Server domain.

Order-to-Activate Cartridge Compatibility

To install or upgrade the Order-to-Activate cartridges, you must ensure compatibility between the following:

- The OSM software version and the Order-to-Activate cartridge version

  OSM is compatible with all cartridges developed in the same release or a previous release, including Order-to-Activate cartridges. For information about updating Order-to-Activate cartridges from a previous release, see "Updating Prior Versions of the Cartridges to Work with Newer Versions of OSM."
- The OSM Order-to-Activate cartridge version and the Oracle Application Integration Architecture (Oracle AIA) Order to Cash Integration Pack for OSM version

For Order-to-Activate cartridge compatibility information see *Order-to-Activate Cartridge Product Compatibility Matrix* (in the **OSM Cartridges for Oracle Application Integration Architecture** section of the OSM documentation) on the Oracle Help Center website:

`http://docs.oracle.com/en/industries/communications/order-service-management/index.html`

Order-to-Activate Cartridge Pre-Installation Tasks

Before you install the Order-to-Activate cartridges, you must set Design Studio preferences. The preferences settings ensure proper installation of the cartridges and the correct mapping of applications such as WebLogic Server, Java SDK, and OSM.

> **Note:** Be careful to set the Design Studio preferences to the correct values. If they are set to the incorrect values, you will have to fix the values and then perform many of the installation steps again.

To set Design Studio preferences:

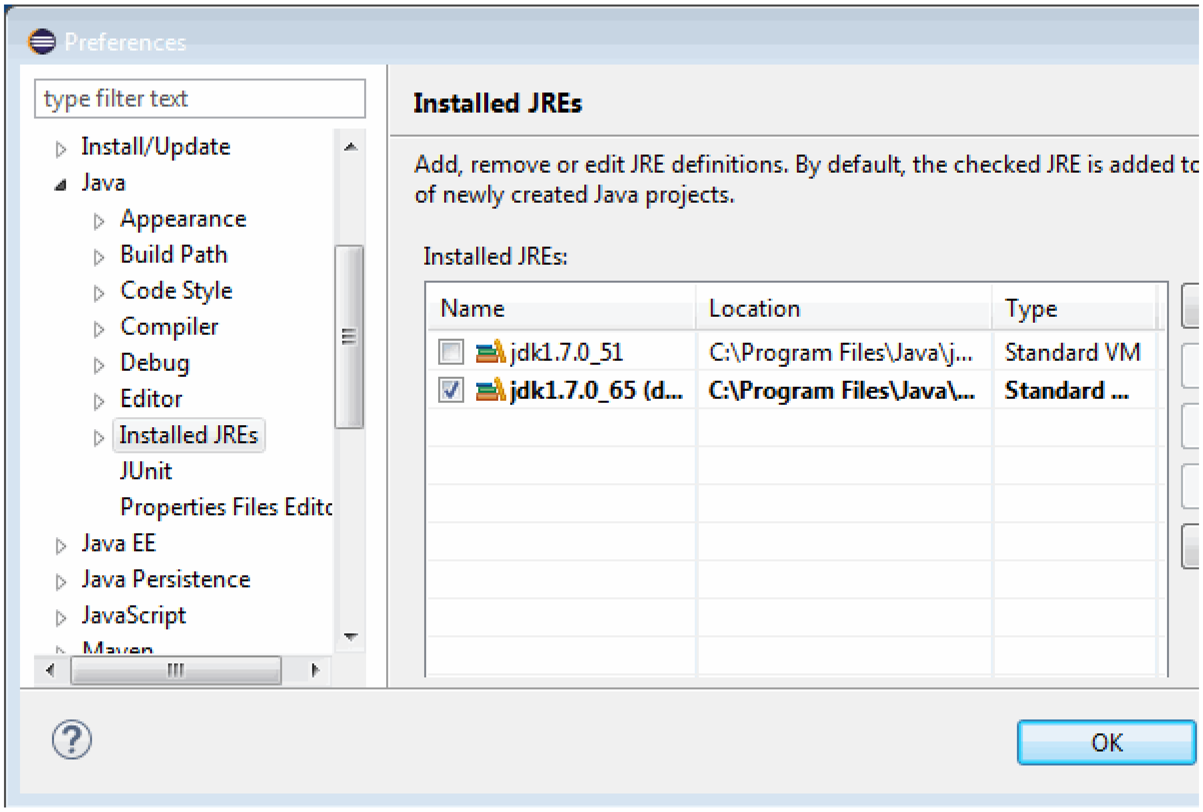
1. Start Design Studio.
2. From the **Window** menu, select **Preferences**.

   The Preferences dialog box is displayed.
3. In the Preferences navigation tree, expand **Oracle Design Studio**.
4. Select **Order and Service Management Preferences.**

The Order and Service Management Preferences page includes the Deploy Properties section in which you can provide home directories for various tools.

5. In the **WebLogic Home** field, enter or browse to the directory in which WebLogic Server is installed, for example **C:\Oracle Middleware\wlserver**.
6. In the **Java SDK Home** field, enter or browse to the directory in which you have installed the JDK for the version of Java that matches the version of Java on your OSM server, for example, **C:\Oracle Middleware\Java\jdk180_66**.
7. In the **OSM SDK Home** field, enter or browse to the directory in which you have installed the OSM SDK, for example, **C:\Oracle Communications\OSM7\SDK**.
8. Select **Inherit significance from order contributors** and **Inherit keys from order contributors**.
9. Expand **Order and Service Management Preferences** and select **Application Integration Architecture (AIA) Preferences.**
10. In the **Oracle Middleware Home** field, enter the directory in which you have installed Oracle Middleware products, for example, **C:\Oracle Middleware**.
11. In the Preferences navigation tree, expand **Java** and select **Compiler**. Ensure that **Compiler compliance level** is set to the appropriate value for your version of Java. For example, if you are using Java 8, set this value to **1.8**.
12. Under **Java**, select **Installed JREs**.
13. If the Java directory that you entered for **Java SDK Home** in step 6 is not displayed, add it and ensure that it is selected, as shown in Figure 2–1.

Figure 2–1 Installed JREs Page (Partial)



Installed JREs

Add, remove or edit JRE definitions. By default, the checked JRE is added tc of newly created Java projects.

Installed JREs:

	Name	Location	Type
☐	jdk1.7.0_51	C:\Program Files\Java\j...	Standard VM
☑	**jdk1.7.0_65 (d...**	**C:\Program Files\Java\...**	**Standard ...**

14. Click **OK**.
15. From the **Project** menu, deselect **Build Automatically**.
16. If you are using WebLogic in Production mode and have a WebLogic cluster, you must perform this step. In the domain directory on *each* computer containing managed servers for your domain, create the following directories for each managed server located on that computer:

    - O2A_SAF_*managedServerName*
    - O2A_UIM_SAF_*managedServerName*
    - O2A_TOM_SAF_*managedServerName*

    For example, if the current computer contains the first two managed servers, which are named osm_ms01 and osm_ms02, you would add the following directories in that domain directory:

    - O2A_SAF_osm_ms01
    - O2A_UIM_SAF_osm_ms01
    - O2A_TOM_SAF_osm_ms01
    - O2A_SAF_osm_ms02
    - O2A_UIM_SAF_osm_ms02
    - O2A_TOM_SAF_osm_ms02

    Then, if managed servers osm_ms03 and osm_ms04 were located on another computer, you would go to the domain directory on that computer and add directories for osm_ms03 and osm_ms04 there.

Installing the Order-to-Activate Cartridges

This section describes how to install the Order-to-Activate cartridges.

Before installing the Order-to-Activate cartridges, read the following sections:

- Cartridge Installation Overview
- System Requirements
- Order-to-Activate Cartridge Pre-Installation Tasks

Getting the Installation Package

To get the Order-to-Activate installation package:

1. Go to the Oracle software delivery website:

   `https://edelivery.oracle.com/`

2. In the **Product** field, select **Oracle Communications Order and Service Management Cartridge for Provisioning Fulfillment** and select your platform.
3. Download the installer file for the **Oracle Communications Order and Service Management Cartridges for Application Integration Architecture**.
4. Unzip the downloaded file into a directory on your Windows system.

   The **OracleComms_OSM_O2A_CartridgesInstaller_b***yyyymmdd***.zip** file is created.

5. Unzip **OracleComms_OSM_O2A_CartridgesInstaller_b***yyyymmdd***.zip**.

The **OSM.PIP** directory containing the **OracleComms_OSM_O2A_Install.zip** file is created.

6. Continue with the "Importing the Installation Cartridge and Configuring the Installation Build File" procedure.

Importing the Installation Cartridge and Configuring the Installation Build File

To import the installation cartridge and configure the installation build file:

1. Start Design Studio.
2. From the **Studio** menu, select **Show Design Perspective**.
3. From the **Window** menu, select **Show View**, and then select **Other**.

   The Show View window is displayed.
4. Expand **Ant** and select **Ant** from below it. Click **OK**.

   The Ant view opens.
5. From the **File** menu, select **Import Studio Project**.

   The Import Projects dialog box is displayed.
6. Select **Select archive file** and click **Browse**.
7. Browse to the **OSM.PIP** directory and select **OracleComms_OSM_O2A_Install.zip**.
8. Click **Open**.

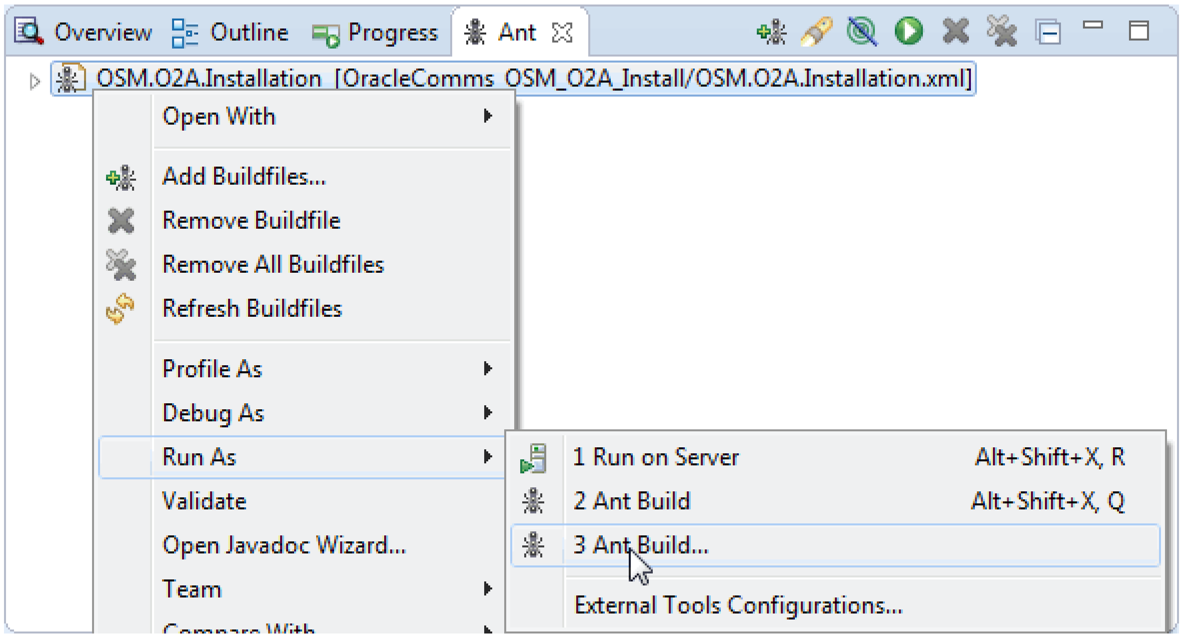
   The **OracleComms_OSM_O2A_Install** project is displayed and selected in the **Projects** field.
9. Click **Finish**.

   The **OracleComms_OSM_O2A_Install** project is imported.
10. Open the Ant view.
11. Right-click in the Ant view and select **Add Buildfiles**.

    The Buildfile Selection dialog box is displayed.
12. Expand **OracleComms_OSM_O2A_Install** and select **OSM.O2A.Installation.xml**.
13. Click **OK**.

    The **OSM.O2A.Installation** item is displayed in the Ant view.
14. Right-click **OSM.O2A.Installation** and select **Run As**.
15. Select **Ant Build...** (not **Ant Build**), as shown in Figure 2–2.

Figure 2–2 Run As Menu



The Edit Configuration dialog box is displayed.

16. Click the **Build** tab and deselect **Build before launch**.
17. Click the **Properties** tab and deselect **Use global properties as specified in the Ant runtime preferences**.
18. Click the **JRE** tab and select **Run in the same JRE as the Workspace**.
19. Click **Close** and click **Yes**.
20. Continue with the appropriate procedure to import the cartridges. The Order-to-Activate cartridge solution has two options: one includes the order transformation manager and Calculate Service Order. The other option does not include the order transformation manager or Calculate Service Order. You must decide which version you want to use before importing any cartridges. See "Order-to-Activate Cartridge Solution Options" for more information about the solution types.
    - To use the version of the Order-to-Activate cartridges that include the order transformation manager and Calculate Service Order, continue with the "Importing the OSM Order-to-Activate Cartridges for the Calculate Service Order Solution Option" procedure.
    - To use the version of the Order-to-Activate cartridges that does not include the order transformation manager and Calculate Service Order, continue with the "Importing the OSM Order-to-Activate Cartridges for the Solution Option Without Calculate Service Order" procedure.

Importing the OSM Order-to-Activate Cartridges for the Calculate Service Order Solution Option

To import the OSM Order-to-Activate cartridges for the Calculate Service Order solution option:

1. Ensure that Design Studio is running.

2. In the Ant view, expand **OSM.O2A.Installation** and double-click **import_ Solution**.
3. In the first Ant Input Request window, enter **y** and click **OK**.
4. In the second Ant Input Request window, do one of the following:
   - To import the central order management cartridges to the current workspace, enter **c** and click **OK**.
   - To import the service order management cartridges to the current workspace, enter **s** and click **OK**.
   - To import both the central order management and service order management cartridges to the current workspace, enter **a** and click **OK**.

   The cartridges appropriate for the settings you selected are imported into the workspace. This may take a few minutes.

   Note: If you configure central order management and service order management on different OSM instances, make sure you configure AIA to use both endpoints appropriately.

5. If you see a message indicating that the Python interpreter is not configured, do one of the following:
   - If you do not need to use Python for your custom configuration, click **Don't ask again**. You do not need to configure this for the Order-to-Activate configuration.
   - If you need to use Python for your custom configuration, click **Manual config** and configure Python according to your needs and environment.
6. Continue with the "Configuring WebLogic Server Resources" procedure.

Importing the OSM Order-to-Activate Cartridges for the Solution Option Without Calculate Service Order

To import the OSM Order-to-Activate cartridges for the solution option without Calculate Service Order:

1. Ensure that Design Studio is running.
2. In the Ant view, expand **OSM.O2A.Installation** and double-click **import_ Solution**.
3. In the first Ant Input Request window, enter **n** and click **OK**.
4. In the second Ant Input Request window, do one of the following:
   - To import central order management cartridges to the current workspace, enter **c** and click **OK**.
   - To import service order management cartridges to the current workspace, enter **s** and click **OK**.
   - To import both central order management and service order management cartridges to the current workspace, enter **a** and click **OK**.

Note: If you configure central order management and service order management on different OSM instances, make sure you configure AIA to use both endpoints appropriately.

---

5. In the third Ant Input Request window, do one of the following:
   - To import cartridges for the Complex topology, enter **c** and click **OK**.
   - To import cartridges for the Typical topology, enter **t** and click **OK**.
   - To import cartridges for the Simple topology, enter **s** and click **OK**.

   The cartridges appropriate for the settings you selected are imported into the workspace. This may take a few minutes.
6. Continue with the "Configuring WebLogic Server Resources" procedure.

Configuring WebLogic Server Resources

This section describes how to configure the WebLogic Server resources, which also configures the metadata for the composite cartridge.

Note: This procedure must be performed on each workspace you are using for the solution. Because of this, if you have central order management and service order management in separate workspaces, you must perform this procedure twice: once for each of these workspaces.

To configure the WebLogic Server resources:

1. Ensure that Design Studio is running.
2. Open the Ant view.
3. Find the one row in Table 2–1 that matches whether you want to use calculate service order; whether your current workspace is for central order management (COM), service order management (SOM), or both; and your desired topology. For the cartridge listed in the corresponding "Cartridge containing SolutionConfig.xml" column of the table:
   a. Right-click in the Ant view and select **Add Buildfiles**.

      The Buildfile Selection dialog box is displayed.
   b. Expand the appropriate cartridge from Table 2–1 and click on the **SolutionConfig.xml** file.
   c. Click **OK**.

Table 2–1 Solution Configurations and Corresponding Cartridge Containing SolutionConfig.xml

Using Calculate Service Order Option?	Current Workspace Is for:	Topology	Cartridge Containing SolutionConfig.xml
Yes	COM only	All	OracleComms_OSM_O2A_COM_CSO_Solution
Yes	SOM only	All	OracleComms_OSM_O2A_SOM_CSO_Solution
Yes	COM and SOM	All	OracleComms_OSM_O2A_COMSOM_CSO_Solution

Table 2–1 (Cont.) Solution Configurations and Corresponding Cartridge Containing SolutionConfig.xml

Using Calculate Service Order Option?	Current Workspace Is for:	Topology	Cartridge Containing SolutionConfig.xml
No	COM only	Simple	OracleComms_OSM_O2A_COM_Simple_NP_Soln
No	COM only	Typical or Complex	OracleComms_OSM_O2A_COM_Typical_NP_Soln
No	SOM only	All	OracleComms_OSM_O2A_SOM_NP_Soln
No	COM and SOM	Simple	OracleComms_OSM_O2A_COMSOM_Simple_NP_Soln
No	COM and SOM	Typical or Complex	OracleComms_OSM_O2A_COMSOM_Typical_NP_Soln

The XML file is displayed in the Ant view. The **SolutionConfig.xml** file is listed as the name of the cartridge it was added from. For example, if you added the **SolutionConfig.xml** file from **OracleComms_OSM_O2A_COMSOM_SimpleSolution**, it is listed in the Ant view as **OracleComms_OSM_O2A_COMSOM_SimpleSolution**.

4. Configure the build file for the **SolutionConfig.xml** file you have added:

   a. In the Ant view, right-click the name of the cartridge for the **SolutionConfig.xml** file and select **Run As**.

   b. Select **Ant Build...** (not **Ant Build**).

      The Edit Configuration dialog box is displayed.

   c. Click the **Build** tab and deselect **Build before launch**.

   d. Click the **Properties** tab and deselect **Use global properties as specified in the Ant runtime preferences**.

   e. Click the **JRE** tab and select **Run in the same JRE as the Workspace**.

   f. Click **Close** and click **Yes**.

   g. Right-click the name of the cartridge for the **SolutionConfig.xml** file and select **Run As** again.

   h. Select **Ant Build...** (not **Ant Build**).

      The Edit Configuration dialog box is displayed.

      Note: It is necessary to close and reopen the Edit Configuration dialog box because after you have deselected the **Use global properties...** check box, Eclipse prevents you from changing the properties until you close and re-open the Edit Configuration dialog box.

   i. Click the **Properties** tab and set the property values according to Table 2–2.

      Note: Do not change any properties that are not listed in Table 2–2.

Table 2–2 Configuration Properties in the Properties Tab

Property Name	Description	Notes
aia.emulator.serverName	Name of the cluster or server within WebLogic Server to which you want to deploy the emulators. Set this to one of the following: ■ If OSM is installed to a cluster, set this value to the name of the cluster. ■ If OSM is installed to the administration server, set this to the name of the administration server. ■ If OSM is installed to a single managed server, set this value to the managed server name. If both central order management and service order management are in the same OSM server instance, set this to the name of the cluster or server for the single OSM instance. If central order management and service order management are in different OSM server instances, set this to the name of the cluster or server for central order management in the central order management build file or to the name of the cluster or server for service order management in the service order management build file.	Always set this property if you are installing the Oracle AIA emulators.
cf.adminServerListenAddress	Host name of the system where WebLogic Server for central order management is running. If you are in a clustered environment, set this to the server where the administration server is located.	Set this if the name of the cartridge associated with the build file contains **COM** or **COMSOM**.
cf.adminServerListenPort	Port on which WebLogic Server for central order management is listening. For a clustered environment, set this to the port on which the administration server is listening.	Set this if the name of the cartridge associated with the build file contains **COM** or **COMSOM**.
cf.clusterName	Name of the cluster for central order management, exactly as it is shown in the WebLogic Server Administration Console.	Set this if the name of the cartridge associated with the build file contains **COM** or **COMSOM** and you are in a clustered WebLogic environment.
cf.userName	Name of a user with administrative privileges on the WebLogic server for listening on the port specified by **cf.adminServerListenAddress** and **cf.adminServerListenPort**.	Set this if the name of the cartridge associated with the build file contains **COM** or **COMSOM**.
lf.adminServerListenAddress	Host name of the system where WebLogic Server for service order management is running. If you are in a clustered environment, set this to the server where the administration server is located.	Set this if the name of the cartridge associated with the build file contains **SOM**.

Table 2–2 (Cont.) Configuration Properties in the Properties Tab

Property Name	Description	Notes
lf.adminServerListenPort	Port on which WebLogic Server for service order management is listening. For a clustered environment, set this to the port on which the administration server is listening.	Set this if the name of the cartridge associated with the build file contains **SOM**.
lf.clusterName	Name of the cluster for service order management, exactly as it is shown in the WebLogic Server Administration Console.	Set this if the name of the cartridge associated with the build file contains **SOM** and you are in a clustered WebLogic environment.
lf.userName	Name of a user with administrative privileges on the WebLogic server listening on the port specified by **lf.adminServerListenAddress** and **lf.adminServerListenPort**.	Set this if the name of the cartridge associated with the build file contains **SOM**.

j. Click **Close** and click **Yes**.

5. In the Ant view, expand the name of the cartridge you are configuring and double-click the **config_All** target.
6. The first Ant Input Request window requests the WebLogic administrator user password. Enter the password for the user you entered in **cf.userName** or **lf.userName** (whichever value you configured for the build file you are running). Click **OK**.
7. In the second Ant Input Request window, enter **y** to use the same password for all of the users being created or enter **n** to use a different password for each user. Click **OK**.
8. Enter the passwords requested for the Order-to-Activate users by doing the following.

Note: Ensure that the passwords you enter meet the security requirements of your WebLogic Server domain. By default, the WebLogic server requires passwords of at least eight characters, with at least one numeric or special character. However, the requirements for your domain may be different.

- If you entered **y** in step 7 to use the same passwords for all users, enter the common password for the Order-to-Activate users and click **OK**.
- If you entered **n** in step 7 to use different passwords for each user, you are prompted for passwords for the following users: **COM user** (the username in the WebLogic Administration Console is osm), **COM Order Event user** (osmoe), **COM Data Change Event user** (osmde), **COM Fallout user** (osmfallout), **SOM user** (osmlf), **SOM Order Event user** (osmoelf), and **SOM Order Abort user** (osmlfaop). Enter the passwords and click **OK** after each entry.

After you have entered all passwords, the system creates the users in the WebLogic domain. This may take a few minutes.

Note: Although **config_All** has now created users in the WebLogic Server domain, it is still possible to cancel **config_All** at a later point and rerun it. If **config_All** finds the users are already present in the domain, it will skip adding them again and will continue with the rest of the configuration process.

---

9. In the next Ant Input Request window, enter **s** if you are using a standalone WebLogic Server environment for OSM or enter **c** if you are using a clustered environment for OSM. Click **OK**.
10. In the next Ant Input Request window, specify whether you intend to connect to Oracle AIA. Enter **d** (for development environment) if you do not intend to connect to Oracle AIA or enter **p** (for production environment) if you intend to connect to Oracle AIA. Click **OK**.

    Queues are created in the WebLogic server. This may take several minutes.
11. If you selected **d** in step 10, in the next Ant Input Request window, enter **d** to deploy emulators or enter **n** to skip deploying the emulators. Click **OK** and go to step 13.

    Internal changes are implemented. This may take a few minutes.
12. If you selected **p** in step 10, do the following:
    a. In the next Ant Input Request window, do one of the following:

       If Oracle AIA is deployed to the administration server, enter the host name and port of the administration server for Oracle AIA.

       If Oracle AIA is deployed to a single managed server, enter the host name and port of the managed server for Oracle AIA.

       If Oracle AIA is deployed to a cluster, enter the host names and ports of all of the managed servers in the Oracle AIA cluster in the following format:

       ```
       hostname:port,hostname:port,hostname:port
       ```

       For example:

       ```
       server1.host.com:7101,server1.host.com:7201,server2.host.com:7101
       ```

    b. Click **OK**.

    c. In the next Ant Input Request window, enter the user name that OSM (in the role you are currently configuring: central order management if the current build file contains **COM** or **COMSOM,** or service order management if the current build file contains **SOM**) uses to connect to Oracle AIA, and click **OK**. This user name is the one that is used to make the JMS connection to Oracle AIA.

    d. In the next Ant Input Request window, enter the password for the user name you entered in step c, and click **OK**.

Note: If you want to change this user name or password later, go into the WebLogic Server Administration console, access the JMS Module **oms_jms_module**, click **O2A_RemoteSAFContext**, and modify the user name and password there.

---

   e. In the next Ant Input Request window, do one of the following to specify the server to contain the SAF agent for Oracle AIA:

      If OSM (in the role you are currently configuring: central order management if the current build file contains **COM** or **COMSOM**, or service order management if the current build file contains **SOM**) is deployed to the administration server, enter the name of the administration server for OSM.

      If OSM is deployed to a single managed server, enter the name of the managed server for OSM.

      If OSM is deployed to a cluster, enter the name of the OSM cluster.

      Internal changes are implemented. This may take a few minutes.

13. If the name of the cartridge associated with the current build file contains **SOM** (but not **COMSOM)**, and you are using the option *without* calculate service order, go to step 19.
14. If the name of the cartridge associated with the current build file contains **COM** or **COMSOM**, you are using the option *without* calculate service order, and you are using Simple topology, go to step 17.
15. If the name of the cartridge associated with the current build file contains **COM** or **COMSOM**, you are using the option *without* calculate service order, and you are using either Typical or Complex topology, do the following:
   a. In the next Ant Input Request window, enter **t** to use Typical topology or enter **c** to use Complex topology, and click **OK**.
   b. If you entered **t** for Typical topology, go to step 17.
   c. If you entered **c** for Complex topology, do one of the following and click **OK**:

      Enter **s** to use a topology that has multiple billing and provisioning systems and one each of install and shipping systems.

      Enter **m** to use a topology that has multiple billing, provisioning, install, and shipping systems.

      Enter **n** to use a topology that has multiple billing, provisioning, and shipping systems and one install system.

      Enter **p** to use a topology that has multiple billing, provisioning, and install systems and one shipping system.
   d. Go to step 17.
16. If the name of the cartridge associated with the current build file contains **COM** or **COMSOM** and you are using the calculate service order option, do the following:
   a. In the next Ant Input Request window, enter **s** to use Simple topology, enter **t** to use Typical topology, or enter **c** to use Complex topology, and click **OK**.
   b. If you entered **s** for Simple topology or **t** for Typical topology, go to step 19.
   c. If you entered **c** for Complex topology, do one of the following in the next Ant Input Request window and click **OK**:

      Enter **s** to use a topology that has multiple billing and provisioning systems and one each of install and shipping systems.

      Enter **m** to use a topology that has multiple billing, provisioning, install, and shipping systems.

Enter **n** to use a topology that has multiple billing, provisioning, and shipping systems and one install system.

Enter **p** to use a topology that has multiple billing, provisioning, and install systems and one shipping system.

17. If the name of the cartridge associated with the current build file contains **COM** (but not **COMSOM)**, or if you are using the option *without* calculate service order, go to step 19.
18. If the name of the cartridge associated with the current build file contains **SOM** or **COMSOM**, and you are using the calculate service order option, do the following.

Note: In any of the steps below, "OSM" refers to the instance of OSM related to the buildfile you are configuring: service order management if the current build file contains **SOM**, or a combined central order management and service order management environment if the current build file contains **COMSOM**.

a. In the next Ant Input Request window, enter **s** if you are using a standalone WebLogic Server environment for OSM or enter **c** if you are using a clustered environment for OSM. The answer you give here should always be the same as the answer you gave in step 9, because you are referring to the same OSM environment. Click **OK**.
b. In the next Ant Input Request window, do one of the following, and click **OK**:

   Enter **l** (the lower-case letter L) if you want a development environment without installing the Oracle Communications Unified Inventory Management (UIM) (inventory) emulator.

   Enter **d** if you want a development environment and you want to deploy the UIM emulator.

   Enter **p** (for production environment) if you intend to connect to UIM.

   Enter **x** if the connection to UIM is going to be installed by the Oracle Communications Rapid Service Design and Order Delivery (RSDOD) installer.
c. In the next Ant Input Request window, enter the name of the UIM application user. If you intend to connect to a live instance of UIM, use a user name that is (or will be) also configured as a user for the UIM product (see the UIM documentation for more information). Click **OK**.
d. In the next Ant Input Request window, enter the password for the UIM application user, and click **OK**.

Note: If you want to change this user name or password later, use the available WebLogic Server tools to change the appropriate entry in the credential store.

e. If you selected **l**, **d**, or **x** in step b, go to step j.
f. If you selected **p** in step b, in the next Ant Input Request window, do one of the following to specify the server to contain the SAF agent for UIM/inventory:

   If OSM is deployed to the administration server, enter the name of the administration server for OSM.

If OSM is deployed to a single managed server, enter the name of the managed server for OSM.

If OSM is deployed to a cluster, enter the name of the OSM cluster.

g. If you selected **p** in step b, in the next Ant Input Request window, do one of the following:

If UIM is deployed to the administration server, enter the host name and port of the administration server for UIM.

If UIM is deployed to a single managed server, enter the host name and port of the managed server for UIM.

If UIM is deployed to a cluster, enter the host names and ports of all of the managed servers in the UIM cluster in the following format:

```
hostname:port,hostname:port,hostname:port
```

For example:

```
server1.host.com:7101,server1.host.com:7201,server2.host.com:7101
```

h. If you selected **p** in step b, in the next Ant Input Request window, enter the user name that OSM uses to connect to UIM, and click **OK**. This user name is the one that is used to make the JMS connection to UIM.

i. If you selected **p** in step b, in the next Ant Input Request window, enter the password for the user name you entered in step h, and click **OK**.

Note: If you want to change this user name or password later, go into the WebLogic Server Administration console, access the JMS Module **oms_jms_module**, click **O2A_UIM_RemoteSAFContext**, and modify the user name and password there.

j. In the next Ant Input Request window, enter **s** if you are using a standalone WebLogic Server environment for OSM or enter **c** if you are using a clustered environment for OSM. The answer you give here should always be the same as the answer you gave in step 9, because you are referring to the same OSM environment. Click **OK**.

k. In the next Ant Input Request window, do one of the following, and click **OK**:

Enter **l** (the lower-case letter L) if you want a development environment without installing the technical order management (activation) emulator.

Enter **d** if you want a development environment and you want to deploy the technical order management emulator.

Enter **p** (for production environment) if you intend to connect to the technical order management system.

Enter **x** if the connection to technical order management is going to be installed by the RSDOD installer.

l. In the next Ant Input Request window, enter the name of the technical order management application user. If you intend to connect to a live instance of a technical order management system, use a user name that is (or will be) also configured as a user for the technical order management system. Click **OK**.

m. In the next Ant Input Request window, enter the password for the technical order management application user, and click **OK**.

Note: If you want to change this user name or password later, use the available WebLogic Server tools to change the appropriate entry in the credential store.

---

n. If you selected **l**, **d**, or **x** in step k, go to step 19.

o. If you selected **p** in step k, in the next Ant Input Request window, do one of the following to specify the server to contain the SAF agent for technical order management:

   If OSM is deployed to the administration server, enter the name of the administration server for OSM.

   If OSM is deployed to a single managed server, enter the name of the managed server for OSM.

   If OSM is deployed to a cluster, enter the name of the OSM cluster.

p. If you selected **p** in step k, in the next Ant Input Request window, do one of the following:

   If technical order management is deployed to the administration server, enter the host name and port of the administration server for technical order management.

   If technical order management is deployed to a single managed server, enter the host name and port of the managed server for technical order management.

   If technical order management is deployed to a cluster, enter the host names and ports of all of the managed servers in the technical order management cluster in the following format:

   ```
   hostname:port,hostname:port,hostname:port
   ```

   For example:

   ```
   server1.host.com:7101,server1.host.com:7201,server2.host.com:7101
   ```

q. If you selected **p** in step k, in the next Ant Input Request window, enter the user name that OSM uses to connect to technical order management, and click **OK**. This user name is the one that is used to make the JMS connection to technical order management.

r. If you selected **p** in step k, in the next Ant Input Request window, enter the password for the user name you entered in step q, and click **OK**.

Note: If you want to change this user name or password later, go into the WebLogic Server Administration console, access the JMS Module **oms_jms_module**, click **O2A_TOM_RemoteSAFContext**, and modify the user name and password there.

---

19. Wait while the system configures the rest of the WebLogic resources. This may take a few minutes.

Note: There is no further configuration needed for the SAF communication to work. Some previous versions of the Order-to-Activate cartridges required further external configuration, but this is no longer required.

---

20. When the **config_All** process is finished, shut down any affected WebLogic domains and restart them.

Note: If you have made a mistake setting the Design Studio preferences and it causes this procedure to fail, the Console view in Design Studio will display "BUILD FAILED." First, correct the preferences using the instructions in "Order-to-Activate Cartridge Pre-Installation Tasks." Next, go to the **Properties** tab of the Edit Configuration dialog box, select **Use global properties as specified in the Ant runtime preferences** to update the values, and then deselect **Use global properties as specified in the Ant runtime preferences** again. Then, select **Clean** from the **Project** menu and clean and build the OracleComms_OSM_O2A_Install project. Exit and restart Design Studio, and then begin the procedure for configuring the WebLogic Server resources again.

Post-Installation Tasks for Multiple Simultaneous Versions

If you intend to have multiple versions of the Order-to-Activate cartridges deployed to the same OSM instance, you must change the relevancy of the recognition rules that you want to use to handle new orders.

The recognition rules that are provided with the Order-to-Activate cartridges depend on the type of installation you have chosen. Table 2–3 provides links to more information about the cartridges containing the provided recognition rules:

Table 2–3 Recognition Rule Cartridges

Cartridge	Deployment Configuration
OracleComms_OSM_O2A_COM_Recognition_Sample	Central-order-management-only OSM instance, with or without Calculate Service Order
OracleComms_OSM_O2A_COMSOM_CSO_Recognition	Central order management and service order management on the same OSM instance, with Calculate Service Order
OracleComms_OSM_O2A_COMSOM_Recognition_Sample	Central order management and service order management on the same OSM instance, without Calculate Service Order
OracleComms_OSM_O2A_SOM_CSO_Recognition	Service-order-management-only OSM instance, with Calculate Service Order
OracleComms_OSM_O2A_SOM_Recognition_Sample	Service-order-management-only OSM instance, without Calculate Service Order

You must increase the value of the **Relevancy** field in all of the recognition rules in the relevant cartridge for the version of the Order-to-Activate cartridges you wish to use to process new orders. For more information about the Relevancy field, see the discussion of the Order Recognition Rule editor in the Design Studio Modeling OSM Orchestration Help.

Building and Deploying the Order-to-Activate Cartridges

> **Note:** The WebLogic Server resources must be configured in the target environment before you can build and deploy the Order-to-Activate cartridges. See "Configuring WebLogic Server Resources."

To build and deploy the Order-to-Activate cartridges:

1. Start Design Studio.
2. In the **Project** menu, ensure that **Build Automatically** is deselected.
3. From the **Project** menu, select **Clean**.

   The Clean window is displayed.
4. Ensure that **Clean all projects**, **Start a build immediately**, and **Build the entire workspace** are all selected, and click **OK**.
5. Ensure that all of the cartridges have built successfully by looking for "BUILD SUCCESSFUL" in the Console view and ensuring that there are no error markers for the cartridges in the Problems view.
6. Create a new Studio environment project and a new Studio environment. See the Design Studio Help for details on creating a Studio Environment project and a Studio environment.

   > **Note:** If you are deploying central order management and service order management to different OSM server instances, you need two environment entities: one pointing to the central order management cluster or standalone server and the other pointing to the service order management cluster or standalone server.

   If you are connecting to a WebLogic Server cluster, use the proxy port in the connection address in the Studio Environment editor.
7. From the **Studio** menu, select **Show Environment Perspective**.
8. Open the Design Studio environment in which the cartridges are to be deployed.

   The Cartridge Management section is displayed and shows a list of available cartridges.
9. Find the one or more rows in Table 2–4 that match whether you want to use calculate service order; whether your current workspace is for central order management (COM), service order management (SOM), or both; and your desired topology. For the cartridge listed in the corresponding "Cartridge to Deploy from Workspace" column of the table, click **Deploy**.

   If you have central order management and service order management the same workspace, you should deploy the cartridges for both central order management and service order management to the same OSM instance. If you have central order management and service order management in different workspaces, you can deploy them to either separate OSM instances or the same OSM instance.

Table 2–4 Solution Configurations and Corresponding Cartridge to Deploy

Using Calculate Service Order Option?	Current Workspace Is for:	Topology	Cartridge to Deploy from Workspace
Yes	COM only	All	OracleComms_OSM_O2A_COM_CSO_Solution
Yes	SOM only	All	OracleComms_OSM_O2A_SOM_CSO_Solution
Yes	COM and SOM	All	OracleComms_OSM_O2A_COMSOM_CSO_Solution
No	COM only	Simple	OracleComms_OSM_O2A_COM_Simple_NP_Soln
No	COM only	Typical or Complex	OracleComms_OSM_O2A_COM_Typical_NP_Soln
No	SOM only	All	OracleComms_OSM_O2A_SOM_NP_Soln
No	COM and SOM	Simple	OracleComms_OSM_O2A_COMSOM_Simple_NP_Soln
No	COM and SOM	Typical or Complex	OracleComms_OSM_O2A_COMSOM_Typical_NP_Soln

When the cartridges have finished deploying, a confirmation dialog box is displayed.

See the Design Studio Help for more details on deploying cartridges.

Testing the Order-to-Activate Cartridges

To test the Order-to-Activate cartridges:

1. Open Design Studio in the Cartridge or Package Explorer view.
2. If you have already connected to the environment with a user other than **osm**, do the following:
   a. Right-click the cartridge containing the sales orders for your solution:

      For the calculate service order solution option: **OracleComms_OSM_O2A_COM_CSO_SalesOrders**

      For the service option without calculate service order: **OracleComms_OSM_O2A_SalesOrders_NP_Sample**

   b. Select **Submit Test**, then select the Studio environment for your central order management instance, and then select **Clear Environment Credentials**.
3. Then submit the test order by doing the following:
   a. Right-click the cartridge containing the sales orders for your solution:

      For the calculate service order solution option: **OracleComms_OSM_O2A_COM_CSO_SalesOrders**

      For the service option without calculate service order: **OracleComms_OSM_O2A_SalesOrders_NP_Sample**

   b. Select **Submit Test**, then select the Studio environment for your central order management instance, and then select the order that you would like to submit.

   The Test Environment Connection dialog box is displayed.

   For information about the test orders, see "OracleComms_OSM_O2A_COM_CSO_SalesOrders" or "OracleComms_OSM_O2A_SalesOrders_NP_Sample."
4. In the **User Name** field, enter **osm**.

5. In the **Password** field, enter the password for the osm user.

   If the test is successful, the order is submitted and the OSM Order Management web client is launched in Design Studio.

6. Log in to the Order Management web client with your OSM user credentials.

   The Orchestration plan, order summary, and other details are displayed.

Switching Between Live and Emulator Configurations

If you initially configured your system to connect to emulators and would like to change to using live configurations, or vice versa, remove the existing queues and create new queues using the following procedure.

To switch between live and emulator configurations:

1. Start Design Studio with the workspace from which you deployed the Order-to-Activate cartridges.
2. Open the Ant view.
3. In the Ant view, expand the **SolutionConfig.xml** build file for your workspace. The name of the build file will be the name of the cartridge it was added from, as indicated in Table 2–1.
4. If necessary, configure any build file properties that might have changed since the cartridges were deployed. For example, a password might have changed. See step 4 in "Configuring WebLogic Server Resources" for more information about configuring the build file.
5. Double-click the **unconfig_Resource** target for the build file.

   Note: The values you enter in this step should be the values that are currently configured for the solution, not any values you want to use in the future. The **unconfig_Resource** target is requesting information about what it should remove.

6. In the first Ant Input Request window, enter the password for the WebLogic administrator user name specified by the **cf.userName** or **lf.userName** (whichever value you configured for the build file you are running) property in the build file configuration. Click **OK**.
7. In the second Ant Input Request window, enter **s** if you are using a standalone WebLogic Server environment or enter **c** if you are using a clustered environment. Click **OK**.
8. In the next Ant Input Request window, specify whether you intend to connect to Oracle AIA. Enter **d** (for development environment) if you do not intend to connect to Oracle AIA or enter **p** (for production environment) if you intend to connect to Oracle AIA. Click **OK**.

   The system removes WebLogic queue configuration. This may take a few minutes.

9. Perform the steps in "Configuring WebLogic Server Resources." When you are prompted for information about whether you want to use emulators or connect to a live system, select the options you would like to use going forward. Existing WebLogic Server users will not be removed or re-created in this process.

Configuring a Workspace Without Configuring WebLogic Server

You can configure a workspace to contain the Order-to-Activate cartridges without configuring a WebLogic Server domain. You might do this if more than one person will be using Design Studio with the same instance of OSM. To do this, first follow the instructions located in these sections:

- Getting the Installation Package
- Importing the Installation Cartridge and Configuring the Installation Build File
- Importing the OSM Order-to-Activate Cartridges for the Calculate Service Order Solution Option

  or

  Importing the OSM Order-to-Activate Cartridges for the Solution Option Without Calculate Service Order

Then, do the following:

1. In the Ant view, expand the name of the cartridge you are configuring and double-click the **config_Metadata_And_ModelVariable** target.
   - If you are using the solution option without calculate service order, the target runs without any prompts and the process is complete.
   - If you are using the calculate service order solution option, continue with the rest of the steps in this procedure.
2. In the first Ant Input Request window, enter **s** to use Simple topology, enter **t** to use Typical topology, or enter **c** to use Complex topology, and click **OK**.
   - If you selected the Simple or Typical topology, the target runs without any further prompts and the process is complete.
   - If you selected Complex topology, continue with the rest of the steps in this procedure.
3. If you entered **c** for Complex topology, do one of the following in the next Ant Input Request window and click **OK**:
   - Enter **s** to use a topology that has multiple billing and provisioning systems and one each of install and shipping systems.
   - Enter **m** to use a topology that has multiple billing, provisioning, install, and shipping systems.
   - Enter **n** to use a topology that has multiple billing, provisioning, and shipping systems and one install system.
   - Enter **p** to use a topology that has multiple billing, provisioning, and install systems and one shipping system.

   The target runs without any further prompts and the process is complete.

Uninstalling Order-to-Activate Components

You can uninstall the Order-to-Activate components that are no longer needed by undeploying the cartridges and deleting the Oracle AIA emulators.

Undeploying Cartridges

You can undeploy an individual cartridge that is not required. See the Design Studio Modeling OSM Processes Help for information about how to undeploy individual cartridges.

To undeploy all of the cartridges, undeploy any composite cartridges that you have in your workspace. The composite cartridges provided as part of the Order-to-Activate cartridges are listed below.

- OracleComms_OSM_O2A_COM_CSO_Solution
- OracleComms_OSM_O2A_COM_SimpleSolution
- OracleComms_OSM_O2A_COM_TypicalSolution
- OracleComms_OSM_O2A_COMSOM_SimpleSolution
- OracleComms_OSM_O2A_COMSOM_TypicalSolution
- OracleComms_OSM_O2A_SOM_CSO_Solution
- OracleComms_OSM_O2A_SOM_Solution

Deleting the Oracle AIA Emulator

To delete an Oracle AIA emulator:

1. Ensure that the WebLogic server is running.
2. Open the WebLogic Server Administration Console.
3. Click the **Deployments** link.

   The Summary of Deployments page is displayed.
4. From the Deployments table, select the emulator to delete.
5. Click **Stop**.
6. Click **Delete**.

   A confirmation dialog box is displayed.
7. Click **OK**.

Removing the Inventory and Technical Order Management Emulators

To undeploy the inventory and technical order management emulators:

1. Ensure that Design Studio is running and that you are in the workspace that contains the service order management cartridges.
2. In the Ant view, expand the cartridge name and double-click **unconfig_inv_tom_MDBs_Emulators**.

   The inventory and technical order management queues will be removed and the inventory and technical order management emulators will be undeployed.

3

Performing a Silent Installation of the Order-to-Activate Cartridges

This chapter contains information about installing and deploying the Oracle Communications Order and Service Management (OSM) Order-to-Activate cartridges in an OSM environment without running the interactive installers.

Cartridge Installation Overview

The Order-to-Activate cartridges are installed into Oracle Communications Design Studio and deployed from there onto the OSM server. For the cartridges to work properly, various entities must be created in Oracle WebLogic Server in the server that contains OSM. An Ant script is provided to create these entities.

The general process for installing the OSM Order-to-Activate cartridges is:

- Ensure that the system requirements are met. See "System Requirements."
- Perform the pre-installation tasks, which set up the environment for running the silent installation. See "Setting Up the Installation Environment."
- Configure the **build.properties** file. See "Configuring the build.properties File."
- Build the cartridges and deploy them to the OSM servers. See "Performing the Silent Installation."
- (Optional) Run one or more test orders to validate that the installation was successful. See "Testing the Order-to-Activate Cartridges."

Directory Placeholders Used in This Chapter

Table 3–1 contains the placeholders that are used in this chapter.

Table 3–1 Placeholders Used in This Chapter

Placeholder	Directory Description
MW_home	The location where Oracle Fusion Middleware components are installed. This directory contains the base directory for WebLogic Server and the **oracle_common** directory, among other files and directories.
domain_home	The directory that contains the configuration for the domain into which OSM is installed. The default is *MW_home*/**user_projects/domains/***domain_name* (where *domain_name* is the name of the OSM domain), but it is frequently set to some other directory at installation.

Table 3–1 (Cont.) Placeholders Used in This Chapter

Placeholder	Directory Description
silent_install_dir	The **headlessBuild** directory under the directory into which you have extracted the Order-to-Activate software; for example **C:\O2A\install\headlessBuild**.

System Requirements

To install the Order-to-Activate cartridges successfully, ensure that you have the following software installed on your local Windows system:

- The supported version of WebLogic Server and Application Development Framework (ADF). (See *OSM Installation Guide* for more information.)
- OSM Software Development Kit (SDK) components.
- Java JDK: Use the version of Java that matches the one being used by the OSM server. See the discussion of software requirements in *OSM Installation Guide*.
- Apache Ant version 1.8.4 or later. Ensure that you have installed it properly according to the Ant documentation, including setting or updating the ANT_HOME, Path, and JAVA_HOME environment variables. The JAVA_HOME environment variable should point to a JDK, rather than a JRE.
- Eclipse with Design Studio plug-ins: See *Design Studio Installation Guide* for information about installing Design Studio plug-ins. You do not need to run the GUI for this application during the silent installation, but it must present on your system so that the silent installer can access the libraries.

You must also have the following installed, either on your local Windows system or on a remote system:

- OSM server installed into the supported version of a WebLogic Server domain.

Order-to-Activate Cartridge Compatibility

To install or upgrade the Order-to-Activate cartridges, you must ensure compatibility between the following:

- The OSM software version and the Order-to-Activate cartridge version

  OSM is compatible with all cartridges developed in the same release or a previous release, including Order-to-Activate cartridges. For information about updating Order-to-Activate cartridges from a previous release, see "Updating Prior Versions of the Cartridges to Work with Newer Versions of OSM."
- The OSM Order-to-Activate cartridge version and the Oracle Application Integration Architecture (Oracle AIA) Order to Cash Integration Pack for OSM version

For Order-to-Activate cartridge compatibility information see *Order-to-Activate Cartridge Product Compatibility Matrix* (in the **OSM Cartridges for Oracle Application Integration Architecture** section of the OSM documentation) on the Oracle Help Center website:

```
http://docs.oracle.com/en/industries/communications/order-service-management/index
.html
```

Setting Up the Installation Environment

Before you can perform a silent installation of the Order-to-Activate cartridges, you must set up the installation environment.

Getting the Installation Package

To get the Order-to-Activate installation package:

1. Go to the Oracle software delivery website:

   `https://edelivery.oracle.com/`

2. In the **Product** field, select **Oracle Communications Order and Service Management Cartridge for Provisioning Fulfillment** and select your platform.
3. Download the installer file for the **Oracle Communications Order and Service Management Cartridges for Application Integration Architecture**.
4. Unzip the downloaded file into a directory on your Windows system.

   The **OracleComms_OSM_O2A_CartridgesInstaller_b***yyyymmdd***.zip** file is created.

5. Unzip **OracleComms_OSM_O2A_CartridgesInstaller_b***yyyymmdd***.zip**.

   The **headlessBuild** directory containing files needed for the silent installation is created. This directory will be referred to in this chapter as *silent_install_dir*.

   The **OSM.PIP** directory containing the **OracleComms_OSM_O2A_Install.zip** file, which is also used by the silent installation, is also created.

Setting Up Files and Directories

To set up the files and directories:

1. Back up and edit the *silent_install_dir***/build.properties** file.
2. In the file, edit the value of the **osm.sdk.home** parameter so that it points to your local copy of the OSM SDK.

   > **Note:** Use (forward) slashes in the path, rather than the backward slash usually used in Windows paths.

   You do not need to edit any other properties in the file at this time.

3. Save and close the file.
4. Create the *silent_install_dir***/security** directory.
5. If you are using WebLogic in Production mode and have a WebLogic cluster, you must perform this step. In the domain directory on *each* computer containing managed servers for your domain, create the following directories for each managed server located on that computer:
   - O2A_SAF_*managedServerName*
   - O2A_UIM_SAF_*managedServerName*
   - O2A_TOM_SAF_*managedServerName*

For example, if the current computer contains the first two managed servers, which are named osm_ms01 and osm_ms02, you would add the following directories in that domain directory:

- O2A_SAF_osm_ms01
- O2A_UIM_SAF_osm_ms01
- O2A_TOM_SAF_osm_ms01
- O2A_SAF_osm_ms02
- O2A_UIM_SAF_osm_ms02
- O2A_TOM_SAF_osm_ms02

Then, if managed servers osm_ms03 and osm_ms04 were located on another computer, you would go to the domain directory on that computer and add directories for osm_ms03 and osm_ms04 there.

Encrypting the Passwords Used by the Silent Installer

There are several passwords you will need for the silent installer. For security reasons, these passwords must all be encrypted.

Encrypting the WebLogic Server Administrator Password for Connecting to WebLogic

This password is always needed when running the silent installer. Configure it by running WebLogic Scripting Tool (WLST).

To encrypt the WebLogic Server administrator password for connecting to WebLogic:

1. In a command shell on the Windows system, change to the *MW_home*/**oracle_common/common/bin** directory.
2. Run the **wlst.cmd** command.
3. Enter the following command:

   ```
   connect('username','password','protocol://hostname:port')
   ```

   where:

   - *username* is the name of the WebLogic Server user that belongs to the Administrators group
   - *password* is the clear-text password for that user
   - *protocol* is **t3s** if connecting to an SSL port on WebLogic Server, otherwise it is **t3**.
   - *hostname* is the IP address or name of the system on which the WebLogic Server domain for OSM is running
   - *port* is a port on which the administration server is listening

   for example:

   ```
   connect('weblogic','password1','t3://host1.example.com:7001')
   ```

4. Enter the following command:

   ```
   storeUserConfig('configFilePath/ConfigFile.secure',
   'keyFilePath/KeyFile.secure')
   ```

where:

- *configFilePath* is the path, formatted for WLST, to store the configuration file. This path can be any directory to which you have write access, but the files will eventually need to be located in the *silent_install_dir*, so you may want to enter that directory here now.

  Note: For WLST, use a (forward) slash in the path, rather than the backward slashes usually used in Windows paths.

- *keytFilePath* is the path, formatted for WLST, to store the key file. This path can be any directory to which you have write access, but the files will eventually need to be located in the *silent_install_dir*, so you may want to enter that directory here now.

for example:

```
storeUserConfig('C:/O2A/install/headlessBuild/ConfigFile.secure',
'C:/O2A/install/headlessBuild/KeyFile.secure')
```

5. Exit WLST using the following command:

```
exit()
```

6. If you saved the files to a different location than *silent_install_dir*, copy the files to the *silent_install_dir* directory now.

Encrypting the WebLogic Server Administrator Password for Use with XML Import/Export

This password is always needed when running the silent installer. Configure it by running Ant.

To encrypt the WebLogic Server administrator password for use with XML Import/Export:

1. In a command shell on the Windows system, change to the *silent_install_dir* directory.
2. Enter the following command:

```
ant create-osm-admin-server-xmlie-encrypted-password-properties-file
```

3. When prompted to enter the user name, enter the name of a WebLogic Server user that belongs to the Administrators group.
4. When prompted to enter the password, enter the plain-text password for the user.
5. When prompted to enter the password again, reenter the password for the user.

   The script will finish and create the following files in the *silent_install_dir*/**security** directory:

   - osm_admin_server.xmlie.password.salt.store
   - osm_admin_server.xmlie.properties

Encrypting Passwords for the Standard Order-to-Activate User Accounts

These passwords are always needed when running the silent installer. Configure them by running Ant.

To encrypt the passwords for the standard Order-to-Activate user accounts:

1. In a command shell on the Windows system, change to the *silent_install_dir* directory.
2. Enter the following command:

   ```
   ant create-comsom-encrypted-password-properties-file
   ```

3. When prompted, enter and reenter the passwords for the Order-to-Activate users.

   The script will finish and create the following files in the *silent_install_dir*/**security** directory:

   - osm.password.salt.store
   - osmde.password.salt.store
   - osmfallout.password.salt.store
   - osmlf.password.salt.store
   - osmlfaop.password.salt.store
   - osmoe.password.salt.store
   - osmoelf.password.salt.store
   - userConfig.properties

Encrypting the UIM Application User Password

The Unified Inventory Manager (UIM) password is needed when running the silent installer to configure Order-to-Activate for the service order management role. Configure it by running Ant.

To encrypt the UIM application user password:

1. In a command shell on the Windows system, change to the *silent_install_dir* directory.
2. Enter the following command:

   ```
   ant create-uim-encrypted-password-properties-file
   ```

3. When prompted to enter the user name, enter the name of a user defined to the UIM application. If you intend to use UIM emulators, you can enter any value that is validly formatted for a user name here.
4. When prompted to enter the password, enter the plain-text password for the user.
5. When prompted to enter the password again, reenter the password for the user.

   The script will finish and create the following files in the *silent_install_dir*/**security** directory:

   - uim.password.salt.store
   - uimUserConfig.properties

Encrypting the Technical Order Management Application User Password

This password is needed when running the silent installer to configure Order-to-Activate for the service order management role. Configure it by running Ant.

To encrypt the technical order management application user password:

1. In a command shell on the Windows system, change to the *silent_install_dir* directory.
2. Enter the following command:

   ```
   ant create-tom-encrypted-password-properties-file
   ```

3. When prompted to enter the user name, enter the name of a user defined to the technical order management application. If you intend to use technical order management emulators, you can enter any value that is validly formatted for a user name here.
4. When prompted to enter the password, enter the plain-text password for the user.
5. When prompted to enter the password again, reenter the password for the user.

   The script will finish and create the following files in the *silent_install_dir*/**security** directory:

   - tom.password.salt.store
   - tomUserConfig.properties

Encrypting the Password for Deploying the Cartridges

This password is always needed when running the silent installer. Configure it by running Ant.

To encrypt the password for deploying the cartridges:

1. In a command shell on the Windows system, change to the *silent_install_dir* directory.
2. Enter the following command:

   ```
   ant create-cmt-encrypted-password-properties-file
   ```

3. When prompted to enter the user name, enter the name of a WebLogic Server user that is a member of the Cartridge_Management_WebService group.
4. When prompted to enter the password, enter the plain-text password for the user.
5. When prompted to enter the password again, reenter the password for the user.

   The script will finish and create the following files in the *silent_install_dir*/**security** directory:

   - cmt.admin.password.salt.store
   - cmtUserConfig.properties

Encrypting the Oracle AIA JMS Connection Password

This password is needed when running the silent installer if you are connecting to a live instance of Oracle AIA. Configure it using the WebLogic Server utility weblogic.security.Encrypt.

To encrypt the Oracle AIA JMS connection password:

1. On the system where the OSM server is installed (as opposed to the Windows system on which you are performing the silent installation, if they are different systems), open a command shell and change to the *domain_home*/**bin** directory.
2. Do one of the following:

   - On UNIX or Linux, source the **setDomainEnv.sh** file, for example:

```
. setDomainEnv.sh
```

- On Windows, run the **setDomainEnv.cmd** file.

3. Enter the following command:

   ```
   java weblogic.security.Encrypt password
   ```

   where *password* is the plain-text password for the Oracle AIA JMS connection.

4. Copy the resulting value and save it for later to paste it as the value for the **solution.saf.password** parameter in the **build.properties** file.

Encrypting the UIM JMS Connection Password

This password is needed when running the silent installer to configure Order-to-Activate for the service order management role if you are using a live connection to UIM. Configure it using the WebLogic Server utility weblogic.security.Encrypt.

To encrypt the UIM JMS connection password:

1. On the system where the OSM server is installed (as opposed to the Windows system on which you are performing the silent installation, if they are different systems), open a command shell and change to the *domain_home*/**bin** directory.
2. Do one of the following:
   - On UNIX or Linux, source the **setDomainEnv.sh** file, for example:

     ```
     . setDomainEnv.sh
     ```

   - On Windows, run the **setDomainEnv.cmd** file.
3. Enter the following command:

   ```
   java weblogic.security.Encrypt password
   ```

   where *password* is the plain-text password for the UIM JMS connection.

4. Copy the resulting value and save it for later to paste it as the value for the **solution.uim.saf.password** parameter in the **build.properties** file.

Encrypting the Technical Order Management JMS Connection Password

This password is needed when running the silent installer to configure Order-to-Activate for the service order management role if you are using a live connection to technical order management. Configure it using the WebLogic Server utility weblogic.security.Encrypt.

To encrypt the technical order management JMS connection password:

1. On the system where the OSM server is installed (as opposed to the Windows system on which you are performing the silent installation, if they are different systems), open a command shell and change to the *domain_home*/**bin** directory.
2. Do one of the following:
   - On UNIX or Linux, source the **setDomainEnv.sh** file, for example:

     ```
     . setDomainEnv.sh
     ```

   - On Windows, run the **setDomainEnv.cmd** file.
3. Enter the following command:

```
java weblogic.security.Encrypt password
```

where *password* is the plain-text password for the technical order management JMS connection.

4. Copy the resulting value and save it for later to paste it as the value for the **solution.tom.saf.password** parameter in the **build.properties** file.

Configuring the build.properties File

This section describes how to configure the **build.properties** file for the Order-to-Activate cartridge silent installation.

Note: Every parameter in this file must have a value. In some situations, you will not need to set a specific value for a particular parameter, but in that case, leave the sample value provided in the file. Do not remove the value and leave it blank.

The **build.properties** file is located in the *silent_install_dir* directory. Edit the file using any standard text editor.

Configuring Software Path Settings

The software path settings, listed in Table 3–2, must be configured in all situations when performing a silent installation.

Note: You may want to ensure that the products below are installed into paths that do not have spaces in them. However, if your path does contain a space, enclose the whole path in double quotation marks.

Table 3–2 Software Path Settings in the build.properties File

Parameter Name	Sample Value	Description
osm.sdk.home	C:/OSM/SDK	Set this value to the directory on the local system that contains the OSM SDK. **Note:** Use (forward) slashes in the path, rather than the backward slashes usually used in Windows paths.
jdk.home	"C:/Program Files/Java/jdk1.7.0_80"	Set this value to the directory on the local system containing the same release of the JDK that the OSM server is using. **Note:** Use (forward) slashes in the path, rather than the backward slashes usually used in Windows paths.
weblogic.home	C:/Oracle/Middleware/wlserver	Set this value to *MW_home*/**wlserver** on the local system, the base directory for the WebLogic Server core files. **Note:** Use (forward) slashes in the path, rather than the backward slashes usually used in Windows paths.

Table 3–2 (Cont.) Software Path Settings in the build.properties File

Parameter Name	Sample Value	Description
oracle.middleware.home	C:/Oracle/Middleware	Set this value to *MW_home* on the local system, the location where Oracle Fusion Middleware components are installed **Note:** Use (forward) slashes in the path, rather than the backward slashes usually used in Windows paths.
studio.home	C:\\Eclipse\\Luna	Set this value to the location of an instance on the local system of Eclipse with Design Studio installed into it. **Note:** For this path, use double backward slashes rather than the single backward slashes usually used in Windows paths.
ant.home	C:/Ant/apache-ant	Set this value to the local standalone installation of Ant. **Note:** Use (forward) slashes in the path, rather than the backward slashes usually used in Windows paths.

Configuring Solution Import Settings

The solution import settings, listed in Table 3–3, must be configured in all situations when performing a silent installation.

Table 3–3 Solution Import Settings in the build.properties File

Parameter Name	Sample Value	Description
o2a.cso.type	y	Set this to one of the following values: ■ **y** to import the calculate service order solution option ■ **n** to import the solution option without calculate service order

Table 3–3 (Cont.) Solution Import Settings in the build.properties File

Parameter Name	Sample Value	Description
o2a.solution.type	y.t.a	Used for the calculate service order solution only. If you are using the service option *without* calculate service order, the value of this parameter does not matter, so leave it with the original value set in the file or set it to any of the values below. Set this to one of the following values: ■ **y.t.a** to import both central order management and service order management ■ **y.t.c** to import central order management only ■ **y.t.s** to import service order management only **Note:** If you configure central order management and service order management on different OSM instances, make sure you configure AIA to use both endpoints appropriately.
o2a.topology.deployment.type	c.c	Do not change the value of this parameter.
o2a.topology.deployment.product.type	a.t.n	Used for the option *without* calculate service order only. If you are using the calculate service order solution option, the value of this parameter does not matter, so leave it with the original value set in the file or set it to any of the values below. Set this to one of the following values: ■ **a.s.n** to import both central order management and service order management with simple topology ■ **a.t.n** to import both central order management and service order management with typical or complex topology ■ **c.s.n** to import central order management only with simple topology ■ **c.t.n** to import central order management only with typical or complex topology ■ **s.s.n** to import service order management only **Note:** If you configure central order management and service order management on different OSM instances, make sure you configure AIA to use both endpoints appropriately.

Configuring WebLogic Server Settings

The WebLogic Server settings, listed in Table 3–4, must be configured in all situations when performing a silent installation.

Table 3–4 WebLogic Server Settings in the build.properties File

Parameter Name	Sample Value	Description
weblogic.admin.user.name	ConfigFile.secure	This value refers to a file that is created in the "Encrypting the WebLogic Server Administrator Password for Connecting to WebLogic" section. Set this value to the name of the first file you created in the **storeUserConfig** command. This file must be located in *silent_install_dir*.
weblogic.admin.user.password	KeyFile.secure	This value refers to a file that is created in the "Encrypting the WebLogic Server Administrator Password for Connecting to WebLogic" section. Set this value to the name of the second file you created in the storeUserConfig command. This file must be located in *silent_install_dir*.
weblogic.admin.server.host	host1.example.com	Set this value to the host name or IP address of the WebLogic Server on which you want to create WebLogic Server resources. If OSM is deployed to a cluster, use the server on which the administration server is located.
weblogic.admin.server.port	7001	Set this value to the port of the WebLogic Server on which you want to create WebLogic Server resources. If OSM is deployed to a cluster, use the port on which the administration server is listening.
osm.server.name	cluster1	Do one of the following: ■ If OSM is deployed to a single managed server, set this to the name of the managed server. ■ If OSM is deployed to a cluster, set this to the name of the cluster. ■ If OSM is deployed to a single administration server, set this to the name of the administration server.

Configuring Solution Configuration Settings

The solution configuration settings, listed in Table 3–5, must be configured in all situations when performing a silent installation.

Table 3–5 Solution Configuration Settings in the build.properties File

Parameter Name	Sample Value	Description
o2a.release.version	2.1.0	Do not change the value of this parameter.
o2a.architecture.bridge.type	s.n	Set this to one of the following values: ■ **s.n** if OSM is in a standalone environment and you would like to set up local queues for Oracle AIA (for example, if you are using the Oracle AIA emulators) ■ **c.n** if OSM is in a clustered environment and you would like to set up local queues for Oracle AIA (for example, if you are using the Oracle AIA emulators) ■ **s.s** if OSM is in a standalone environment and you would like to set up a Store-and-forward (SAF) agent to connect to a live Oracle AIA instance ■ **c.s** if OSM is in a clustered environment and you would like to set up a SAF agent to connect to a live Oracle AIA instance
o2a.deploy.emulators.mode	d	Set this to one of the following values: ■ **d** to deploy the Oracle AIA emulators ■ **n** not to deploy the Oracle AIA emulators
topology.type	c	If you are using the service option *without* calculate service order, the value of this parameter does not matter, so leave it with the original value set in the file or set it to any of the values below. If you are using the calculate service order solution option, set this to one of the following values: ■ **s** to configure the solution to use the simple topology ■ **t** to configure the solution to use the typical topology ■ **c** to configure the solution to use the complex topology
complex.topology.options	x	Set this to one of the following values: ■ x if you are using the service option without calculate service order. Also set the value to **x** if you are using the calculate service order solution option and you have entered **s** or **t** for the **topology.type** parameter. ■ **s** to use a topology that has multiple billing and provisioning systems and one each of install and shipping systems ■ **m** to use a topology that has multiple billing, provisioning, install, and shipping systems ■ **n** to use a topology that has multiple billing, provisioning, and shipping systems and one install system ■ **p** to use a topology that has multiple billing, provisioning, and install systems and one shipping system

Table 3–5 (Cont.) Solution Configuration Settings in the build.properties File

Parameter Name	Sample Value	Description
osm.deployment.server.host	host1.example.com	Set this value to the host name or IP address of the WebLogic Server to which you will deploy the cartridges. If OSM is deployed to a cluster, use the server on which the proxy server is located.
osm.deployment.server.port	7001	Set this value to the port of the WebLogic Server to which you will deploy the cartridges. If OSM is deployed to a cluster, use the server on which the proxy server is located.
solution.com.saf.serverURLs	N/A	Do not change the value of this parameter.
solution.som.saf.serverURLs	N/A	Do not change the value of this parameter.
bea.aia.user	N/A	Do not change the value of this parameter.
bea.aia.password	N/A	Do not change the value of this parameter.
bea.aia.host	N/A	Do not change the value of this parameter.
bea.aia.port	N/A	Do not change the value of this parameter.
aia.server.name	N/A	Do not change the value of this parameter.
solution.com.saf.password	N/A	Do not change the value of this parameter.
solution.som.saf.password	N/A	Do not change the value of this parameter.
solution.com.deploymentTarget	cluster1	If you are configuring an environment for service order management only, do not change the value of this parameter. If you are configuring an environment for both central order management and service order management or for central order management only, do one of the following: ■ If OSM is deployed to a single managed server, set this to the name of the managed server. ■ If OSM is deployed to a cluster, set this to the name of the cluster. ■ If OSM is deployed to a single administration server, set this to the name of the administration server.
solution.som.deploymentTarget	cluster2	If you are configuring an environment for both central order management and service order management or for central order management only, do not change the value of this parameter. If you are configuring an environment for service order management only, do one of the following: ■ If OSM is deployed to a single managed server, set this to the name of the managed server. ■ If OSM is deployed to a cluster, set this to the name of the cluster. ■ If OSM is deployed to a single administration server, set this to the name of the administration server.

Configuring Oracle AIA Connection Settings

The Oracle AIA connection settings, listed in Table 3–6, must be configured if you intend to connect to a live instance of Oracle AIA.

Table 3–6 Oracle AIA Connection Settings in the build.properties File

Parameter Name	Sample Value	Description
solution.saf.userName	aiauser	Set this value to the name of the user for opening a JMS connection to Oracle AIA.
solution.saf.password	{AES}sw97PfpHI0uCrpGYIXHDAYivSHo3iBTDX3tslkPR1xA=	Set this to the value generated in the "Encrypting the Oracle AIA JMS Connection Password" section.

solution.saf.serverURLs	host1.example.com	If Oracle AIA is deployed to the administration server, enter the host name and port of the administration server for Oracle AIA.
		If Oracle AIA is deployed to a single managed server, enter the host name and port of the managed server for Oracle AIA.
		If Oracle AIA is deployed to a cluster, enter the host names and ports of all of the managed servers in the Oracle AIA cluster in the following format:
		`hostname:port,hostname:port,hostname:port`
		For example:
		`server1.host.com:7101,server1.host.com:7201,server2.host.com:7101`

Configuring UIM Connection Settings

The UIM connection settings, listed in Table 3–7, must be configured if you intend to connect to a live instance of UIM and are configuring both central order management and service order management or service order management only.

Table 3–7 UIM Connection Settings in the build.properties File

Parameter Name	Sample Value	Description
o2a.som.cso.inv.architecture.bridge.type	s.n	**Note:** The first character in this value must match the first character of the value of **o2a.architecture.bridge.type**. Set this to one of the following values: ■ **s.n** if OSM is in a standalone environment and you would like to set up local queues for UIM (for example, if you are using the UIM emulator) ■ **c.n** if OSM is in a clustered environment and you would like to set up local queues for UIM (for example, if you are using the UIM emulator) ■ **s.s** if OSM is in a standalone environment and you would like to set up a SAF agent to connect to a live UIM instance ■ **c.s** if OSM is in a clustered environment and you would like to set up a SAF agent to connect to a live UIM instance ■ **s.x** if OSM is in a standalone environment and you would not like to set up any queues for UIM ■ **c.x** if OSM is in a clustered environment and you would not like to set up any queues for UIM
o2a.som.cso.inv.mdb.mode	d	Set this value to **n** if **o2a.som.cso.inv.architecture.bridge.type** is **s.s**, **c.s**, **s.x**, or **c.x**. If **o2a.som.cso.inv.architecture.bridge.type** is **s.n** or **c.n**, set to **d** to deploy the emulators, or set to **n** if you want to create the local queues and not deploy the emulators.
solution.uim.saf.userName	uimuser	Set this value to the name of the user for opening a JMS connection to UIM.
solution.uim.saf.password	{AES}sw97PfpHI0uCrpGYIXHDAYivSHo3iBTDX3tslkPR1xA=	Set this to the value generated in the "Encrypting the UIM JMS Connection Password" section.

solution.uim.saf.serverURLs	host1.example.com	If UIM is deployed to the administration server, enter the host name and port of the administration server for UIM.
		If UIM is deployed to a single managed server, enter the host name and port of the managed server for UIM.
		If UIM is deployed to a cluster, enter the host names and ports of all of the managed servers in the UIM cluster in the following format:
		`hostname:port,hostname:port,hostname:port`
		For example:
		`server1.host.com:7101,server1.host.com:7201,server2.host.com:7101`

Configuring Technical Order Management Connection Settings

The technical order management connection settings, listed in Table 3–8, must be configured if you intend to connect to a live instance of technical order management and are configuring both central order management and service order management or service order management only.

Table 3–8 Technical Order Management Connection Settings in the build.properties File

Parameter Name	Sample Value	Description
o2a.som.cso.tom.architecture.bridge.type	s.n	**Note:** The first character in this value must match the first character of the value of **o2a.architecture.bridge.type**. Set this to one of the following values: ■ **s.n** if OSM is in a standalone environment and you would like to set up local queues for technical order management (for example, if you are using the technical order management emulator) ■ **c.n** if OSM is in a clustered environment and you would like to set up local queues for technical order management (for example, if you are using the technical order management emulator) ■ **s.s** if OSM is in a standalone environment and you would like to set up a SAF agent to connect to a live technical order management instance ■ **c.s** if OSM is in a clustered environment and you would like to set up a SAF agent to connect to a live technical order management instance ■ **s.x** if OSM is in a standalone environment and you would not like to set up any queues for technical order management ■ **c.x** if OSM is in a clustered environment and you would not like to set up any queues for technical order management
o2a.som.cso.tom.mdb.mode	d	Set this value to **n** if **o2a.som.cso.tom.architecture.bridge.type** is **s.s**, **c.s**, **s.x**, or **c.x**. If **o2a.som.cso.tom.architecture.bridge.type** is **s.n** or **c.n**, set to **d** to deploy the emulators, or set to **n** if you want to create the local queues and not deploy the emulators.

Table 3–8 (Cont.) Technical Order Management Connection Settings in the build.properties File

Parameter Name	Sample Value	Description
solution.tom.saf.userName	tomuser	Set this value to the name of the user for opening a JMS connection to technical order management.
solution.tom.saf.password	{AES}sw97PfpHI0uCrpGYIXHDAYivSHo3iBTDX3tslkPR1xA=	Set this to the value generated in the "Encrypting the Technical Order Management JMS Connection Password" section.

solution.tom.saf.serverURLs	host1.example.com	If technical order management is deployed to the administration server, enter the host name and port of the administration server for technical order management.
		If technical order management is deployed to a single managed server, enter the host name and port of the managed server for technical order management.
		If technical order management is deployed to a cluster, enter the host names and ports of all of the managed servers in the technical order management cluster in the following format:
		`hostname:port,hostname:port,hostname:port`
		For example:
		`server1.host.com:7101,server1.host.com:7201,server2.host.com:7101`

Performing the Silent Installation

To build and deploy the Order-to-Activate cartridges you run Ant scripts that build the solution cartridges, configure the WebLogic Server resources, and deploy the cartridges.

Building the Solution Cartridges

If you have not yet configured the WebLogic Server queues, resources, and users in the OSM domain, do not perform this procedure, but instead see "Building the Solution Cartridges and Configuring the WebLogic Server Resources." You build the cartridges by running Ant.

To build the solution cartridges:

1. In a command shell on the Windows system, change to the *silent_install_dir* directory.
2. Enter the following command:

   ```
   ant headless.build.deploy
   ```

   The script will finish and create a file in the *silent_install_dir*/**workspace/***solution_cart_name*/**cartridgeBuild** directory, where *solution_cart_name* is the name of the composite cartridge corresponding to the settings you have selected. See Table 2–1 for a list of the composite cartridges and the situations to which they apply. The file that is created is named *solution_cart_name***.par**.

Building the Solution Cartridges and Configuring the WebLogic Server Resources

You must build the solution cartridges and configure the WebLogic Server resources before you can use the Order-to-Activate cartridges. If you have already configured the WebLogic Server queues, resources, and users in the OSM domain, do not perform this procedure, but instead see "Building the Solution Cartridges." You build the cartridges and configure the WebLogic Server resources by running Ant.

To build the solution cartridges and configure the WebLogic Server resources:

1. In a command shell on the Windows system, change to the *silent_install_dir* directory.
2. Enter the following command:

   ```
   ant headless.build.resource.deploy
   ```

   The script will finish after a few minutes and in addition to creating resources in the WebLogic Server domain, will create a file in the *silent_install_dir*/**workspace/***solution_cart_name*/**cartridgeBuild** directory, where *solution_cart_name* is the name of the composite cartridge corresponding to the settings you have selected. See Table 2–1 for a list of the composite cartridges and the situations to which they apply. The file that is created is named *solution_cart_name***.par**.

Deploying the Cartridges

You deploy the Order-to-Activate cartridges by running Ant.

To deploy the cartridges:

1. In a command shell on the Windows system, change to the *silent_install_dir* directory.
2. Enter the following command:

   ```
   ant headless.deploy
   ```

Testing the Order-to-Activate Cartridges

To test the Order-to-Activate cartridges, you must open Design Studio. For information about how to do this, see "Testing the Order-to-Activate Cartridges."

Switching Between Live and Emulator Configurations

Using the interactive installer, you can change your configuration between connecting to emulators and connecting to live systems. For information about how to do this, see "Switching Between Live and Emulator Configurations."

Configuring a Workspace Without Configuring WebLogic Server

Using the interactive installer, you can configure a workspace to contain the Order-to-Activate cartridges without configuring a WebLogic Server domain. This operation cannot be performed silently. For information about how to do this, see "Configuring a Workspace Without Configuring WebLogic Server."

Uninstalling Order-to-Activate Components

You can uninstall the Order-to-Activate components that are no longer needed by undeploying the cartridges and deleting the Oracle AIA emulators. For information about how to do this, see "Uninstalling Order-to-Activate Components."

4 Order-to-Activate Cartridge Contents

This chapter describes the various Order-to-Activate cartridges, their entities, and how these entities can be extended.

Note: In the Oracle Communications Order to Cash solution, Oracle Communications Order and Service Management (OSM) does not directly interact with Oracle Communications Billing and Revenue Management (BRM), Siebel Customer Relationship Management (Siebel CRM), or provisioning systems. OSM uses Oracle Application Integration Architecture (Oracle AIA), which in turn uses BRM Application Business Connector Service (ABCS) for billing and CRM ABCS for Siebel CRM.

Cartridge Overview

Following is an overview of the cartridges. You should not modify productized cartridges, but demonstration cartridges are sample cartridges provided so that you can modify them to meet your needs.

Common Order Management Cartridges

Common order management cartridges contain data and entities that are available in all solution options and topologies.

Table 4–1 lists and describes the common order management cartridges.

Table 4–1 Common Order Management Cartridges

Cartridge Name	Description
OracleComms_OSM_CommonDataDictionary	Productized cartridge. Orchestration Common ControlData dictionary (core Oracle Communications Design Studio product cartridge)
OracleComms_OSM_O2A_ AIAEBMDataDictionary	Productized cartridge. This data dictionary cartridge contains the data schema that defines the data elements from the Oracle AIA Enterprise Business Message (EBM) schema.
OracleComms_OSM_O2A_CommonUtility	Productized cartridge. This data dictionary cartridge contains the data schema that defines the data elements for modeling orchestration entities in OSM.

Table 4–1 (Cont.) Common Order Management Cartridges

Cartridge Name	Description
OracleComms_OSM_O2A_ControlMap	Productized cartridge. This cartridge provides testing utilities including breakpoints, point of no return disabling, and support for fault simulation.
OracleComms_OSM_O2A_RecognitionFallout	Productized cartridge. This cartridge generates Oracle AIA trouble ticket creation request messages for unrecognizable customer order messages.
OracleComms_OSM_O2A_SystemAdmin	Productized cartridge. This cartridge works in conjunction with the Inbound Message Recovery message-driven bean (MDB) to create fallout tasks that help you recover from inbound message processing errors.

Central Order Management Cartridges

Central order management cartridges contain the processes for the central order management functionality and also the Oracle AIA interaction mechanism which in turn interacts with Siebel CRM, BRM, and OSM in its service order management role.

Some central order management cartridges are common to both the solution option that uses calculate service order and the solution option that does not. Other cartridges are specific to one solution option or the other.

Common Central Order Management Cartridges

Table 4–2 lists the central order management cartridges that are used by both the calculate service order solution option and the service option without calculate service order.

Table 4–2 Common Central Order Management Cartridges

Cartridge Name	Description
OracleComms_OSM_O2A_COM_Base	Productized cartridge. This cartridge supports the orchestration of customer orders from Oracle AIA.
OracleComms_OSM_O2A_COM_SalesOrderFulfillment	Productized cartridge. This cartridge supports the communications between central order management and fulfillment systems.
OracleComms_OSM_O2A_COM_Billing	Productized cartridge. This cartridge supports billing fulfillment functions.
OracleComms_OSM_O2A_COM_Provisioning	Productized cartridge. This cartridge supports provisioning fulfillment functions.
OracleComms_OSM_O2A_COM_Shipping_Sample	Demonstration cartridge. This cartridge supports shipping fulfillment functions.
OracleComms_OSM_O2A_COM_Install_Sample	Demonstration cartridge. This cartridge supports installation fulfillment functions.
OracleComms_OSM_O2A_COM_Recognition_Sample	Demonstration cartridge. This cartridge recognizes a customer order from Oracle AIA and triggers the creation of a COM_SalesOrderFulfillment order. It is used for both solution options when the workspace contains only central order management.

Central Order Management Cartridges for the Calculate Service Order Solution Option

Table 4–3 lists and describes the central order management cartridges for the Calculate Service Order solution option.

Table 4–3 Central Order Management Cartridges for the Calculate Service Order Solution Option

Cartridge Name	Description
OracleComms_OSM_O2A_COM_CSO_Base	Productized cartridge. This cartridge contains entities, like orchestration processes, order items, and transformation sequences, that support the orchestration of orders for the calculate service order solution option.
OracleComms_OSM_O2A_COM_CSO_Broadband_Internet_Access_CFS	Demonstration cartridge. This cartridge contains the mapping rules and order item parameter bindings associated with the customer facing service for broadband internet access.
OracleComms_OSM_O2A_COM_CSO_Email_CFS	Demonstration cartridge. This cartridge contains the mapping rules and order item parameter bindings associated with the customer facing service for email service.
OracleComms_OSM_O2A_COM_CSO_FulfillmentPattern	Productized cartridge. This cartridge contains fulfillment patterns and orchestration dependencies for the calculate service order solution option.
OracleComms_OSM_O2A_COM_CSO_FulfillmentStateMap	Productized cartridge. This cartridge contains fulfillment state maps and transformed order item fulfillment state composition rule sets specific to the calculate service order solution option.
OracleComms_OSM_O2A_COM_CSO_Internet_Media_CFS	Productized cartridge. This cartridge contains the mapping rules and order item parameter bindings associated with the customer facing service for Internet media service.
OracleComms_OSM_O2A_COM_CSO_IP_Fax_CFS	Productized cartridge. This cartridge contains the mapping rules and order item parameter bindings associated with the customer facing service for IP fax service.
OracleComms_OSM_O2A_COM_CSO_Model_Container	Demonstration cartridge. This cartridge defines the common model projects that contain elements that might need to be included in the deployment and contains the transformation manager for the calculate service order solution option.
OracleComms_OSM_O2A_COM_CSO_Provisioning	Demonstration cartridge. This cartridge contains order components for provisioning that are specific to the calculate service order solution option.
OracleComms_OSM_O2A_COM_CSO_SalesOrders	Demonstration cartridge. This cartridge contains sample customer orders for use with the calculate service order solution option.
OracleComms_OSM_O2A_COM_CSO_Solution	Demonstration composite cartridge. This cartridge references all cartridges required when the calculate service order solution option is used and the current workspace is for central order management only.
OracleComms_OSM_O2A_COM_CSO_Topology	Productized cartridge. This cartridge contains decomposition rules and order components for the topology you selected when installing the Order-to-Activate cartridges.

Table 4–3 (Cont.) Central Order Management Cartridges for the Calculate Service Order Solution Option

Cartridge Name	Description
OracleComms_OSM_O2A_COM_CSO_VoIP_Access_CFS	Demonstration cartridge. This cartridge contains the mapping rules and order item parameter bindings associated with the customer facing service for Voice over Internet Protocol (VoIP) access.
OracleComms_OSM_O2A_COM_CSO_Web_Conferencing_CFS	Productized cartridge. This cartridge contains the mapping rules and order item parameter bindings associated with the customer facing service for web conferencing service.
OracleComms_OSM_O2A_COM_FulfillmentPattern	Productized cartridge. This cartridge contains the base fulfillment pattern from which other fulfillment patterns can inherit.
OracleComms_OSM_O2A_COM_FulfillmentStateMap_Sample	Demonstration cartridge. This cartridge contains fulfillment state entities used by the solution.
OracleComms_OSM_O2A_COMSOM_CSO_Recognition	Demonstration cartridge. This cartridge recognizes a customer order from Oracle AIA and triggers the creation of a COM_SalesOrderFulfillment order. It is used when the current workspace includes both central order management and service order management.
OracleComms_OSM_O2A_COMSOM_CSO_Solution	Demonstration composite cartridge. This cartridge references all cartridges required when the calculate service order solution option is used and the current workspace is for both central order management and service order management.

Central Order Management Cartridges for the Solution Option Without Calculate Service Order

Table 4–4 lists and describes the central order management cartridges for the solution option without Calculate Service Order.

Table 4–4 Central Order Management Cartridges for the Solution Option Without Calculate Service Order

Cartridge Name	Description
OracleComms_OSM_O2A_BBVoIP_FP_NP_Danube_Sample	Demonstration cartridge. This cartridge contains fulfillment patterns and orchestration dependencies for the Simple topology.
OracleComms_OSM_O2A_BBVoIP_FP_NP_Nile_Sample	Demonstration cartridge. This cartridge contains fulfillment patterns and orchestration dependencies for the Typical or Complex topologies.
OracleComms_OSM_O2A_COM_NCSO_Base	Productized cartridge. This cartridge supports the orchestration of customer orders from Oracle AIA.
OracleComms_OSM_O2A_COM_NCSO_Provisioning	Productized cartridge. This cartridge supports provisioning fulfillment functions.
OracleComms_OSM_O2A_COM_Simple_NP_Soln	Demonstration composite cartridge. This cartridge references all cartridges required for the Simple topology for central order management.
OracleComms_OSM_O2A_COM_Typical_NP_Soln	Demonstration composite cartridge. This cartridge references all cartridges required for the Typical or Complex topologies for central order management.

Table 4–4 (Cont.) Central Order Management Cartridges for the Solution Option Without Calculate Service

Cartridge Name	Description
OracleComms_OSM_O2A_COMSOM_Recognition_Sample	Demonstration cartridge. This cartridge recognizes a customer order from Oracle AIA and triggers the creation of a COM_SalesOrderFulfillment order. It is used when the current workspace includes both central order management and service order management.
OracleComms_OSM_O2A_COMSOM_Simple_NP_Soln	Demonstration composite cartridge. This cartridge references all cartridges required for the Simple topology. It is used when the current workspace includes both central order management and service order management.
OracleComms_OSM_O2A_COMSOM_Typical_NP_Soln	Demonstration composite cartridge. This cartridge references all cartridges required for the Typical or Complex topologies. It is used when the current workspace includes both central order management and service order management.
OracleComms_OSM_O2A_FulfillmentPatternMap_Sample	Demonstration cartridge. This cartridge contains the mappings between product specifications and fulfillment patterns.
OracleComms_OSM_O2A_SalesOrders_NP_Sample	Demonstration cartridge. This cartridge contains sample customer orders.
OracleComms_OSM_O2A_SimpleTopology_Sample	Demonstration cartridge. This cartridge contains decomposition rules and order components for the Simple topology.
OracleComms_OSM_O2A_TypicalTopology_Sample	Demonstration cartridge. This cartridge contains decomposition rules and order components for the Typical or Complex topology

Note: Danube and Nile are the names for the process fulfillment patterns in the Simple and Typical topologies, respectively. See "About Fulfillment Topologies" for more details on topologies.

Service Order Management Cartridges

Service order management cartridges contain the OSM functionality that handles a provisioning request as a service order from central order management and completes the predetermined tasks to fulfill the service order.

Service order management cartridges are specific to either the solution option that uses calculate service order or the solution option that does not.

Service Order Management Cartridges for the Calculate Service Order Solution Option

Table 4–5 lists and describes the service order management cartridges for the Calculate Service Order solution option.

Table 4–5 Service Order Management Cartridges for the Calculate Service Order Solution Option

Cartridge Name	Description
OracleComms_OSM_O2A_SOM_CSO_Base	Productized cartridge. This cartridge supports the orchestration of service orders.
OracleComms_OSM_O2A_SOM_CSO_Broadband_Internet_Access_CFS	Demonstration cartridge. This cartridge contains the order item parameter bindings associated with the customer facing service for broadband internet access.
OracleComms_OSM_O2A_SOM_CSO_Common	Productized cartridge. This cartridge contains data elements and fulfillment modes for service order management with the calculate service order solution option.
OracleComms_OSM_O2A_SOM_CSO_CompleteProvisioning	Demonstration cartridge. This cartridge supports provisioning fulfillment functions for service order management.
OracleComms_OSM_O2A_SOM_CSO_DeliverOrder	Demonstration cartridge. This cartridge supports order delivery fulfillment functions for service order management.
OracleComms_OSM_O2A_SOM_CSO_DesignService	Demonstration cartridge. This cartridge supports service design functions for service order management.
OracleComms_OSM_O2A_SOM_CSO_Email_CFS	Demonstration cartridge. This cartridge contains the order item parameter bindings associated with the customer facing service for email service.
OracleComms_OSM_O2A_SOM_CSO_FulfillmentPattern	Productized cartridge. This cartridge contains fulfillment patterns for service order management.
OracleComms_OSM_O2A_SOM_CSO_FulfillmentStateMap	Productized cartridge. This cartridge contains fulfillment state entities for service order management.
OracleComms_OSM_O2A_SOM_CSO_Internet_Media_CFS	Demonstration cartridge. This cartridge contains the order item parameter bindings associated with the customer facing service for Internet media service.
OracleComms_OSM_O2A_SOM_CSO_IP_Fax_CFS	Demonstration cartridge. This cartridge contains the order item parameter bindings associated with the customer facing service for IP fax service.
OracleComms_OSM_O2A_SOM_CSO_ModelContainer	Demonstration cartridge. This cartridge defines the common model projects that contain elements that might need to be included in the deployment.
OracleComms_OSM_O2A_SOM_CSO_PlanDelivery	Demonstration cartridge. This cartridge supports delivery planning functions for service order management
OracleComms_OSM_O2A_SOM_CSO_Recognition	Demonstration cartridge. This cartridge recognizes a service order from central order management and triggers the creation of a SOM_ProvisionServiceOrderFulfillment order. Catches all in-bound messages not recognized by any other provisioning recognition rules.
OracleComms_OSM_O2A_SOM_CSO_Solution	Demonstration composite cartridge. This cartridge references all cartridges required when the calculate service order solution option is used and the current workspace is for service order management only.

Table 4–5 (Cont.) Service Order Management Cartridges for the Calculate Service Order Solution Option

Cartridge Name	Description
OracleComms_OSM_O2A_SOM_CSO_Topology	Productized cartridge. This cartridge contains entities, such as decomposition rules and order components, for service order management.
OracleComms_OSM_O2A_SOM_CSO_VoIP_Access_CFS	Demonstration cartridge. This cartridge contains the order item parameter bindings associated with the customer facing service for VoIP access.
OracleComms_OSM_O2A_SOM_CSO_Web_Conferencing_CFS	Demonstration cartridge. This cartridge contains the order item parameter bindings associated with the customer facing service for web conferencing service.

Service Order Management Cartridges for the Solution Option Without Calculate Service Order

Table 4–6 lists and describes the service order management cartridges for the solution option without Calculate Service Order.

Table 4–6 Service Order Management Cartridges for the Solution Option Without Calculate Service Order

Cartridge Name	Description
OracleComms_OSM_O2A_SOM_Base	Productized cartridge. This cartridge supports the orchestration of service orders, including handling status and data updates from fulfillment requests back to central order management.
OracleComms_OSM_O2A_SOM_Provisioning	Productized cartridge. This cartridge supports provisioning fulfillment functions in service order management.
OracleComms_OSM_O2A_SOM_Solution	Demonstration composite cartridge. This cartridge references all cartridges required for service order management.
OracleComms_OSM_O2A_SOM_Recognition_Sample	Demonstration cartridge. This cartridge recognizes a service order from central order management and triggers the creation of a SOM_ProvisionServiceOrderFulfillment order. Catches all in-bound messages not recognized by any other provisioning recognition rules.
OracleComms_OSM_O2A_SomBBVoIP_FP_NP_Sample	Demonstration cartridge. This cartridge contains fulfillment patterns, decomposition rules, and order components for service order management functions.
OracleComms_OSM_O2A_SomProvisionBroadband_Sample	Demonstration cartridge. This cartridge supports service orders for broadband services.
OracleComms_OSM_O2A_SomProvisionVoIP_Sample	Demonstration cartridge. This cartridge supports service orders for VoIP services.

Conceptual Model Projects

Conceptual model projects contain the relationships between your commercial products, the services that they represent, and the resources that are required to implement the services. These projects are not deployed, but the information in them that is needed for deployment is deployed with the OracleComms_OSM_O2A_COM_CSO_Model_Container cartridge.

For more information about the conceptual model, see *Design Studio Concepts*.

Common Conceptual Model Projects

Table 4–7 lists the conceptual model projects that are present in both central order management and service order management workspaces.

Table 4–7 Common Conceptual Model Projects

Cartridge Name	Description
OracleComms_Model_Base	Productized cartridge. This cartridge contains entities, like provider functions and functional areas, that support conceptual modeling.
OracleComms_Model_BaseCatalog	Productized cartridge. This cartridge contains conceptual model fulfillment patterns.
OracleComms_Model_Common	Productized cartridge. This cartridge contains a data schema with common data element definitions.
OracleComms_Model_O2A_Broadband_Internet_Access_CFS	Productized cartridge. This cartridge contains the customer facing services for broadband Internet access.
OracleComms_Model_O2A_Broadband_Internet_Access_SA	Productized cartridge. This cartridge contains the actions for broadband Internet access.
OracleComms_Model_O2A_Broadband_Internet_DataModel	Productized cartridge. This cartridge contains a data schema for data specific to broadband Internet access.
OracleComms_Model_O2A_Email_CFS	Productized cartridge. This cartridge contains the customer facing services for email.
OracleComms_Model_O2A_Email_DataModel	Productized cartridge. This cartridge contains a data schema for data specific to email.
OracleComms_Model_O2A_Email_SA	Productized cartridge. This cartridge contains the actions for email.
OracleComms_Model_O2A_Internet_Media_CFS	Productized cartridge. This cartridge contains the customer facing services for Internet media.
OracleComms_Model_O2A_Internet_Media_DataModel	Productized cartridge. This cartridge contains a data schema for data specific to Internet media.
OracleComms_Model_O2A_Internet_Media_SA	Productized cartridge. This cartridge contains the actions for Internet media.
OracleComms_Model_O2A_VoIP_Access_CFS	Productized cartridge. This cartridge contains the customer facing services for VoIP.
OracleComms_Model_O2A_VoIP_Access_SA	Productized cartridge. This cartridge contains the actions for VoIP.
OracleComms_Model_O2A_VoIP_DataModel	Productized cartridge. This cartridge contains a data schema for data specific to VoIP.

Conceptual Model Projects for Central Order Management

Table 4–8 lists the conceptual model projects that are only present in workspaces with central order management installed.

Table 4–8 Conceptual Model Projects for Central Order Management

Cartridge Name	Description
OracleComms_Model_O2A_Billing_PS	Productized cartridge. This cartridge contains the domains and products for billing services.
OracleComms_Model_O2A_Broadband_Internet_PS	Productized cartridge. This cartridge contains the domains and products for broadband Internet access services.
OracleComms_Model_O2A_Email_PS	Productized cartridge. This cartridge contains the domains and products for email services.
OracleComms_Model_O2A_Install_PS	Productized cartridge. This cartridge contains the domains and products for installation services.
OracleComms_Model_O2A_Internet_Media_PS	Productized cartridge. This cartridge contains the domains and products for Internet media services.
OracleComms_Model_O2A_VoIP_PS	Productized cartridge. This cartridge contains the domains and products for VoIP services.

Conceptual Model Projects for Service Order Management

Table 4–9 lists the conceptual model projects that are only present in workspaces with service order management installed.

Table 4–9 Conceptual Model Projects for Service Order Management

Cartridge Name	Description
OracleComms_Model_O2A_SOM_PS	Productized cartridge. This cartridge contains the products for service order management. This cartridge is only present when the service option without calculate service order is used.

Common Order Management Cartridges

The following cartridges provide common data dictionary elements that are used by or referenced by other Order-to-Activate cartridges.

OracleComms_OSM_CommonDataDictionary

The OracleComms_OSM_CommonDataDictionary cartridge is a productized data dictionary cartridge. It contains the data schema that defines the data elements for modeling orchestration entities in OSM.

This cartridge is referenced by many other cartridges, including OracleComms_OSM_O2A_COM_Base, OracleComms_OSM_O2A_SOM_CSO_Base, and OracleComms_OSM_O2A_SOM_Base.

OracleComms_OSM_O2A_AIAEBMDataDictionary

The OracleComms_OSM_O2A_AIAEBMDataDictionary cartridge is a productized data dictionary cartridge that is part of the core OSM product. It contains the data schema that defines the data elements from the Oracle AIA EBM schema. Cartridges that must include data elements from Oracle AIA EBM can reuse the elements defined in this cartridge.

This cartridge is referenced by the OracleComms_OSM_O2A_COM_Base cartridge.

OracleComms_OSM_O2A_CommonUtility

The OracleComms_OSM_O2A_CommonUtility cartridge is a productized component cartridge. It contains the data schema that defines the data elements for Order-to-Activate cartridges. Cartridges that extend Order-to-Activate cartridges can reuse the elements defined in this cartridge.

This cartridge is referenced by many other cartridges, including OracleComms_OSM_O2A_COM_Base, OracleComms_OSM_O2A_SOM_CSO_Base, and OracleComms_OSM_O2A_SOM_Base.

Table 4–10 describes the data schema elements that can be reused when extending Order-to-Activate cartridges:

Table 4–10 OracleComms_OSM_O2A_CommonUtility Extensible Data Dictionary Elements

Data Dictionary Element	Extension
Order Component	New fulfillment functions should use this data dictionary element or its extended type as the base fulfillment function type.

Table 4–11 describes the XQuery modules in the cartridge.

Table 4–11 OracleComms_OSM_O2A_CommonUtility XQuery Modules

XQuery Module	Extendable	Description
AIAEBMUtilityModule	No	Provides utilities for manipulating Oracle AIA EBM.
AIAFaultMsgEBMUtilityModule	No	Provides utilities for handling Oracle AIA fault messages.
BreakpointControlModule	No	Provides services related to breakpoint control in Order-to-Activate. Please refer to the OracleComms_OSM_O2A_ControlMap cartridge for an extensible way to control breakpoints.
ComponentDataManagementModule	No	Manipulates data structures for fulfillment functions.
ExtensionPointModule	No	Defines XQuery extension points for fulfillment functions.
ExtensionPointSelector	No	Sets the order of the fulfillment function's extension points based on priority.
FalloutLifecycleModule	No	Provides services related to message fallout and trouble ticket tracking.
FalloutSimulationModule	No	Simulates fallout.
FulfillmentOrderEventModule	No	Manages the fulfillment request's events in central order management and service order management.
FulfillmentOrderLifecycle-Management Module	No	Provides services related to the fulfillment request's order lifecycle management.
LogModule	No	Provides logging facility for Order-to-Activate cartridges.
OrderComponentMetadataBuilder	No	Provides an internal framework for data discrepancy detection.
OrderExtensionPointModule	No	Provides support for order event extension points.
OrderExtensionPointSelector	No	Provides support for order event extension point selection.

Table 4–11 (Cont.) OracleComms_OSM_O2A_CommonUtility XQuery Modules

XQuery Module	Extendable	Description
OrderLifecycleModule	No	Provides services related to the fulfillment request's external fulfillment state, milestone tracking, upstream order status map, and status context calculation.
OSMEBMUtilityModule	No	Provides utilities for manipulating EBM in central order management.
OsmWebServiceModule	No	Provides services for OSM web service requests.
PerspectiveModule	No	Provides utilities for retrieving historical perspectives.
ProductClassToFulfillmentPatternModule	No	Provides utilities for order line retrieval for both fulfillment request and service order.
ProductClassToProductSpec	Yes	Provides services related to the mapping between product specifications and fulfillment patterns.
ProvisionOrderLifecycle-ManagementModule	No	Provides services related to service order management-to-fulfillment request lifecycle management.
ServiceActionCodeModule	No	Provides services related to service action calculation.
SOMEBMUtilityModule	No	Provides utilities for manipulating the EBM in service order management.
SomProductClassToProductSpec	Yes	Provides services related to the mapping between product specifications and fulfillment patterns in SOM.
SystemConfigModule	No	Provides constants and utilities for the calculate service order solution option.
SystemInteractionModule	No	Provides services related to message sequencing and generation and order locking.
TargetMapping	No	Returns the target system name for a given active interaction ID during fallout handling.
TargetSystemManagementModule	No	Provides services related to target system information such as target system identifier and code.
Topology	No	Calculates the system topology for the calculate service order solution option.
TroubleTicket	No	Provides utilities for trouble ticketing during fallout handling.
UpdateServiceOrderStatusFunctionsModule	No	Provides services to create EBM message for the service provisioning order to update the service order management order.

OracleComms_OSM_O2A_ControlMap

The OracleComms_OSM_O2A_ControlMap cartridge is a productized cartridge that provides the ability to:

- Stop at a breakpoint when OSM executes central order management tasks
- Disable a point of no return
- Simulate a fallout scenario
- Configure the processing granularity of billing fulfillment functions dynamically
- Manage the frequency of order updates to the upstream system for debugging

To use these functions:

1. Create a control file in XML, using the parameters listed in the following sections. The same control file can contain more than one control function.
2. Validate your control file against the **BFPMap.xsd** schema located in **OracleComms_OSM_O2A_ControlMap\resource**.

   You must validate your control file against the schema because OSM will not validate control files and report errors during order processing. If the file is not a valid XML file, the entire file will be ignored by OSM. If a file contains an invalid element or value, the control function containing the invalid element or value will be ignored.
3. Put your control file into the **OracleComms_OSM_O2A_ControlMap\resource** directory.
4. When you want to use the control file for an order, preface the order number with the name of the control file inside square brackets and without the **.xml** extension. For example, if you have a control file named **control001.xml** in the **OracleComms_OSM_O2A_ControlMap\resource** directory, and you want to use it with an order numbered VOIP01, send the order in with the orderID **[control001]VOIP01**.

Configuring Breakpoints for Central Order Management and for Service Order Management Without Calculate Service Order

This section applies to configuring breakpoints for central order management for both the calculate service order solution option and the service option without calculate service order and configuring breakpoints for service order management for the solution option without calculate service order. For information about configuring breakpoints for service order management with the calculate service order solution option, see "Configuring Breakpoints for Service Order Management with Calculate Service Order."

If you enable breakpoint control, you can set a breakpoint in the order process to cause the order to go through a particular manual task or a special automated task before or after an interaction with an external system. This enables you to do things like check status and define data.

A breakpoint task for central order management or for service order management for the solution option without calculate service order is defined by:

- BreakComponent: This is an order component name; for example, **SyncCustomerFunction**
- ExecutionMode: **do**, **redo**, **undo**, and **amend_do**
- Event: **Component_PRESTART** (before the component has started), only applicable to FulfillBillingFunction), **Component_START** (after the component has started), and **Component_COMPLETE** (after the component has completed)
- TargetSystem: **ANY**, or a particular target system such as **BRM-ALL**

Example 4–1 contains a control file to configure a breakpoint before FulfillBillingFunction starts.

Example 4–1 Control File to Configure a Breakpoint in FulfillBillingFunction

```
<?xml version="1.0" encoding="UTF-8"?>
<oms:ControlMap xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="urn:com:metasolv:oms:xmlapi:1 BFPMap.xsd"
 xmlns="urn:com:metasolv:oms:xmlapi:1"
 xmlns:oms="urn:com:metasolv:oms:xmlapi:1">
```

```
    <oms:BreakPointControlMap>
        <oms:BreakComponent>FulfillBillingFunction</oms:BreakComponent>
        <oms:ExecutionMode>do</oms:ExecutionMode>
        <Event>Component_PRESTART</Event>
        <TargetSystem>ANY</TargetSystem>
    </oms:BreakPointControlMap>
</oms:ControlMap>
```

The valid component names for central order management are:

- SyncCustomerFunction
- InitiateBillingFunction
- ProvisionOrderFunction
- FulfillBillingFunction

The valid component names for service order management for the solution option without calculate service order are:

- SomProvisionOrderFunction

Configuring Breakpoints for Service Order Management with Calculate Service Order

This section applies to configuring breakpoints for service order management for the calculate service order solution option. For information about configuring breakpoints for other situations, see "Configuring Breakpoints for Central Order Management and for Service Order Management Without Calculate Service Order."

If you enable breakpoint control, you can set a breakpoint in the order process to cause the order to go through a particular manual task or a special automated task before or after an interaction with an external system. This enables you to do things like check status and define data.

A breakpoint task for service order management for the calculate service order solution option is defined by:

- BreakComponent: This is an order component name; for example, **DesignServiceFunction**
- ExecutionMode: **do**, **redo**, **undo**, and **amend_do**
- Event: The task name in the sub-process of the component
- TargetSystem: **ANY**, or a particular target system such as **SOM_DeliverySystem**

Table 4–12 describes the components and events used for breakpoints for service order management with the calculate service order option.

Table 4–12 Breakpoint Events for Service Order Management with the Calculate Service Order Solution Option

Component Name	Events
DesignServiceFunction	CaptureBITask ProcessBITask ApproveBITask
PlanDeliveryFunction	IssueBITask CalculateTechnicalActionsTask
DeliverOrderFunction	CreateTechnicalOrderTask

Table 4–12 (Cont.) Breakpoint Events for Service Order Management with the Calculate Service Order Solution Option

Component Name	Events
CompleteProvisioningFunction	CompleteBITask

Example 4–2 contains a control file to configure a breakpoint before FulfillBillingFunction starts.

Example 4–2 Control File to Configure a Breakpoint in DesignServiceFunction

```
<?xml version="1.0" encoding="UTF-8"?>
<oms:ControlMap xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="urn:com:metasolv:oms:xmlapi:1 BFPMap.xsd"
 xmlns="urn:com:metasolv:oms:xmlapi:1"
 xmlns:oms="urn:com:metasolv:oms:xmlapi:1">
    <oms:BreakPointControlMap>
        <oms:BreakComponent>DesignServiceFunction</oms:BreakComponent>
        <oms:ExecutionMode>do</oms:ExecutionMode>
        <Event>CaptureBITask</Event>
        <TargetSystem>ANY</TargetSystem>
    </oms:BreakPointControlMap>
</oms:ControlMap>
```

Controlling Point of No Return

During testing, a user may want to disable OSM's ability to set or check for points of no return. This allows the user to submit an amendment order successfully in situations that would not be allowed under normal circumstances.

You can disable point of no return processing either for all order components or selectively using the following field:

- PONRComponent: This can contain any of the following
  - Order Component Name - For example, **SyncCustomerFunction**
  - **ALL** - Disable point of no return setting and checking for all order components
  - **CONFIG** - Disable point of no return checking for all order components when an order amendment is received

Example 4–3 contains a control file that disables point of no return setting and checking for all order components.

Example 4–3 Control File to Disable Points of No Return

```
<?xml version="1.0" encoding="UTF-8"?>
<oms:ControlMap xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="urn:com:metasolv:oms:xmlapi:1 BFPMap.xsd"
 xmlns="urn:com:metasolv:oms:xmlapi:1"
 xmlns:oms="urn:com:metasolv:oms:xmlapi:1">
    <oms:PONRControlMap>
        <oms:PONRComponent>ALL</oms:PONRComponent>
    </oms:PONRControlMap>
</oms:ControlMap>
```

Controlling Fault Simulation

If you are using the Oracle AIA emulators included with the Order-to-Activate cartridges, you can simulate fallout scenarios using control files.

There are two ways to trigger the emulator to generate a fault response rather than a successful response:

- Using the VerbCode field in the EBM
- Using a control file

Both methods are discussed in the following sections.

Simulating Faults in Central Order Management

To simulate a fault in central order management using the VerbCode field in the EBM, populate the following XPath location:

```
ProcessSalesOrderFulfillmentEBM/EBMHeader/VerbCode
```

Table 4–13 lists the valid values to put in the field and their descriptions.

Table 4–13 VerbCode Values for Central Order Management

VerbCode Value	Description
SIMULATE_FAIL_SYNCCUST_FAULT	The SyncCustomer emulator generates a fault to the AIA error handler.
SIMULATE_FAIL_SYNCCUST_RESP	The SyncCustomer emulator sends back an invalid response.
SIMULATE_FAIL_SYNCCUST_NOTIF	The SyncCustomer emulator sends back a response with a fault indicator.
SIMULATE_FAIL_INITBILL_FAULT	The InitiateBilling emulator generates a fault to the AIA error handler.
SIMULATE_FAIL_INITBILL_RESP	The InitiateBilling emulator sends back an invalid response.
SIMULATE_FAIL_INITBILL_NOTIF	The InitiateBilling emulator sends back a response with a fault indicator.
SIMULATE_FAIL_FULFILLBILL_FAULT	The FulfillBilling emulator generates a fault to the AIA error handler.
SIMULATE_FAIL_FULFILLBILL_RESP	The FulfillBilling emulator sends back an invalid response.
SIMULATE_FAIL_FULFILLBILL_NOTIF	The FulfillBilling emulator sends back a response with a fault indicator.
SIMULATE_FAIL_PROV_FAULT	The ProvisionOrder emulator generates a fault to the AIA error handler.
SIMULATE_FAIL_PROV_RESP	The ProvisionOrder emulator sends back an invalid response.
SIMULATE_FAIL_PROV_NOTIF	The ProvisionOrder emulator sends back a response with a fault indicator.

To create a control file to simulate fault situations in central order management, use the following parameters:

- FaultComponent: this is an order component name, for example, **SyncCustomerFunction**
- ExecutionMode: **do**, **redo**, **undo**, and **amend_do**
- FaultMode:
  - **Fault**: generate a fault to the AIA error handler

– **InvalidRESP**: send back an invalid response
– **FailNotification**: send back a response with a fault indicator

Example 4–4 contains a control file that simulates an invalid response for SyncCustomerFunction.

Example 4–4 Control File to Simulate a Fault in Central Order Management

```
<?xml version="1.0" encoding="UTF-8"?>
<oms:ControlMap xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="urn:com:metasolv:oms:xmlapi:1 BFPMap.xsd"
 xmlns="urn:com:metasolv:oms:xmlapi:1"
 xmlns:oms="urn:com:metasolv:oms:xmlapi:1">
    <oms:FaultControlMap>
        <oms:FaultComponent>SyncCustomerFunction</oms:FaultComponent>
        <oms:ExecutionMode>do</oms:ExecutionMode>
        <FaultMode>InvalidRESP</FaultMode>
    </oms:FaultControlMap>
</oms:ControlMap>
```

Simulating Faults in Service Order Management Without Calculate Service Order

To simulate a fault in service order management without the calculate service order solution option using the VerbCode field in the EBM, populate the following XPath location:

```
ProcessSalesOrderFulfillmentEBM/EBMHeader/VerbCode
```

Table 4–14 lists the valid values to put in the field and their descriptions.

Table 4–14 VerbCode Values for Service Order Management Without Calculate Service Order

VerbCode Value	Description
SIMULATE_FAIL_BRD_SERVICEBUNDLE_FAULT	Service order management generates a fault to the AIA error handler during broadband service bundle provisioning.
SIMULATE_FAIL_BRD_EMAILSERVICEBUNDLE_FAULT	Service order management generates a fault to the AIA error handler during broadband email service provisioning.
SIMULATE_FAIL_BRD_MEDIASERVICEBUNDLE_FAULT	Service order management generates a fault to the AIA error handler during broadband media service provisioning.
SIMULATE_FAIL_BRD_CPE_FAULT	Service order management generates a fault to the AIA error handler during broadband customer premise equipment provisioning.
SIMULATE_FAIL_VOIP_SERVICEBUNDLE_FAULT	Service order management generates a fault to the AIA error handler during VoIP service bundle provisioning.
SIMULATE_FAIL_VOIP_CPE_FAULT	Service order management generates a fault to the AIA error handler during VoIP customer premise equipment provisioning.

To create a control file to simulate fault situations in service order management without the calculate service order solution option, use the following parameters:

- ExecutionMode: **do**, **redo**, **undo**, and **amend_do**
- VerbCode: This can contain any of the values in Table 4–14.

Example 4–5 contains a control file that simulates a failure in provisioning broadband customer premise equipment in execution mode **redo**.

Example 4–5 Control File to Simulate a Fault in Service Order Management Without Calculate Service Order

```
<?xml version="1.0" encoding="UTF-8"?>
<oms:ControlMap xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="urn:com:metasolv:oms:xmlapi:1 BFPMap.xsd"
 xmlns="urn:com:metasolv:oms:xmlapi:1"
 xmlns:oms="urn:com:metasolv:oms:xmlapi:1">
    <oms:LF_FaultControlMap>
        <oms:ExecutionMode>redo</oms:ExecutionMode>
        <VerbCode>SIMULATE_FAIL_BRD_CPE_FAULT</VerbCode>
    </oms:LF_FaultControlMap>
</oms:ControlMap>
```

Simulating Faults in Service Order Management With Calculate Service Order

To simulate a fault in service order management with the calculate service order solution option using the VerbCode field in the EBM, populate the following XPath location:

```
ProcessSalesOrderFulfillmentEBM/EBMHeader/VerbCode
```

Table 4–15 lists the valid values to put in the field and their descriptions.

Table 4–15 VerbCode Values for Service Order Management With Calculate Service Order

VerbCode Value	Description
SIMULATE_FAIL_CAPTURE_BI_RESP	The CaptureBI emulator generates a SOAP fault response message.
SIMULATE_FAIL_CAPTURE_BI_NOTIF	The CaptureBI emulator generates a normal response message with missing mandatory data.
SIMULATE_FAIL_PROCESS_BI_RESP	The ProcessBI emulator generates a SOAP fault response message.
SIMULATE_FAIL_PROCESS_BI_NOTIF	The ProcessBI emulator generates a normal response message with missing mandatory data.
SIMULATE_FAIL_APPROVE_BI_RESP	The ApproveBI emulator generates a SOAP fault response message.
SIMULATE_FAIL_APPROVE_BI_NOTIF	The ApproveBI emulator generates a normal response message with missing mandatory data.
SIMULATE_FAIL_ISSUE_BI_RESP	The IssueBI emulator generates a SOAP fault response message.
SIMULATE_FAIL_ISSUE_BI_NOTIF	The IssueBI emulator generates a normal response message with missing mandatory data.
SIMULATE_FAIL_CALCULATE_TA_RESP	The CalculateTA emulator generates a SOAP fault response message.
SIMULATE_FAIL_CALCULATE_TA_NOTIF	The CalculateTA emulator generates a normal response message with missing mandatory data.
SIMULATE_FAIL_CREATE_TO_RESP	The CreateTO emulator generates a SOAP fault response message.
SIMULATE_FAIL_CREATE_TO_NOTIF	The CreateTO emulator generates a normal response message with missing mandatory data.

Table 4–15 (Cont.) VerbCode Values for Service Order Management With Calculate Service Order

VerbCode Value	Description
SIMULATE_FAIL_COMPLETE_ BI_RESP	The CompleteBI emulator generates a SOAP fault response message.
SIMULATE_FAIL_COMPLETE_ BI_NOTIF	The CompleteBI emulator generates a normal response message with missing mandatory data.

To create a control file to simulate fault situations in service order management with the calculate service order solution option, use the following parameters:

- ExecutionMode: **do**, **redo**, **undo**, and **amend_do**
- VerbCode: This can contain any of the values in Table 4–15.

Example 4–6 contains a control file that simulates a failure in provisioning broadband customer premise equipment in execution mode **redo**.

Example 4–6 Control File to Simulate a Fault in Service Order Management Without Calculate Service Order

```
<?xml version="1.0" encoding="UTF-8"?>
<oms:ControlMap xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="urn:com:metasolv:oms:xmlapi:1 BFPMap.xsd"
 xmlns="urn:com:metasolv:oms:xmlapi:1"
 xmlns:oms="urn:com:metasolv:oms:xmlapi:1">
    <oms:LF_FaultControlMap>
        <oms:ExecutionMode>redo</oms:ExecutionMode>
        <VerbCode>SIMULATE_FAIL_CAPTURE_BI_RESP</VerbCode>
    </oms:LF_FaultControlMap>
</oms:ControlMap>
```

Controlling Order Updates

In Order-to-Activate, order updates are sent by central order management to upstream systems at every milestone update on the order, unless the order is in the Canceling state. The milestones are defined in **OracleComms_OSM_O2A_ Configuration\solution-config\ComponentMilestoneMap.xml**. You can use a control file to disable the order updates for one or more specific milestones.

Following are the attributes and elements to use in your control file to disable updates for a breakpoint:

- system: OracleComms_OSM_O2A_SystemAdmintarget system name defined in the **<oms:targetSystem>** element of the **resources/SolutionConfig/TargetSystemMap.xml** file in the Order-to-Activate composite cartridge.
- execMode: **do**, **redo**, and **amend_do**
- ComponentMilestone: **COMPONENT-START**, **COMPONENT-UPDATE**, or **COMPONENT-COMPLETE**
- Milestone: Milestone defined by external system such as **PROVISION DESIGNED**. This field is optional and only applicable to the **COMPONENT-UPDATE** component milestone.
- UpdateUpstreamSystem: Set this to **false** to disable the event

Example 4–7 contains a control file that disables the sending of updates for the PROVISION START milestone.

Example 4–7 Control File to Disable Updates for PROVISION START

```
<?xml version="1.0" encoding="UTF-8"?>
<oms:ControlMap xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="urn:com:metasolv:oms:xmlapi:1 BFPMap.xsd"
 xmlns="urn:com:metasolv:oms:xmlapi:1"
 xmlns:oms="urn:com:metasolv:oms:xmlapi:1">
    <oms:MilestoneMap system="OSMPROV" systemName="*" execMode="do">
        <oms:ComponentMilestone>COMPONENT-START</oms: ComponentMilestone >
        <oms:Milestone>PROVISION START</oms:Milestone>
        <oms:UpdateUpstreamSystem>false</oms:UpdateUpstreamSystem>
    </oms: MilestoneMap>
</oms:ControlMap>
```

Controlling Processing Granularity for FulfillBillingFunction

In Order-to-Activate, the granularity decomposition rule for **FulfillBillingFunction** uses **ServiceBundle** granularity by default. If you want to test **Order** granularity on **FulfillBillingFunction**, use a control file.

Following are the elements in the control file to change processing granularity:

- GranularityFunction: Only **FulfillBillingFunction** is supported for this element. Any other value is ignored.
- Granularity: Only **Order** and **ServiceBundle** are supported for this element. Any other value is ignored.

Example 4–8 contains a control file that changes the processing granularity to **Order**:

Example 4–8 Control File to Change Processing Granularity

```
<?xml version="1.0" encoding="UTF-8"?>
<oms:ControlMap xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="urn:com:metasolv:oms:xmlapi:1 BFPMap.xsd"
 xmlns="urn:com:metasolv:oms:xmlapi:1"
 xmlns:oms="urn:com:metasolv:oms:xmlapi:1">
    <oms:GranularityControlMap>
        <oms:GranularityFunction>FulfillBillingFunction</oms:GranularityFunction>
        <oms:Granularity>Order</oms: Granularity>
    </oms:GranularityControlMap>
</oms:ControlMap>
```

OracleComms_OSM_O2A_RecognitionFallout

The OracleComms_OSM_O2A_RecognitionFallout cartridge is a productized cartridge that generates Oracle AIA trouble ticket creation request messages for unrecognizable customer order messages.

Table 4–16 lists entities that are defined in this cartridge.

Table 4–16 OracleComms_OSM_O2A_RecognitionFallout Entities

Entity Name	Entity Type	Description
ORPFalloutPIPOrder	Order	The order that is created when an unrecognizable message is received.
CreationORPFalloutTask	Manual Task	Creation task that is used to create an ORPFalloutPIPOrder.
ORPQueryTask	Manual Task	Query task used by a manual user to view the fallout order.

Table 4–16 (Cont.) OracleComms_OSM_O2A_RecognitionFallout Entities

Entity Name	Entity Type	Description
ORPFalloutProcessErrorTask	Manual Task	Task that handles error when creating a fault message in service order management or when creating a fulfillment request for the trouble ticketing system.
ORPFalloutProcess	Process and tasks	Fallout process that creates a trouble ticket for Oracle AIA.
orpfalloutrole	Role	Role with privileges to create and view fallout orders.

Table 4–17 lists XQuery modules defined in this cartridge.

Table 4–17 OracleComms_OSM_O2A_RecognitionFallout XQuery Modules

XQuery Module	Extendable	Description
CreateErrorFault	No	Used by service order management to create a fault notification.
CreateORPFalloutTroubleTicket	No	Creates a trouble ticket EBM for Oracle AIA indicating an ORP error.
CreateORPFalloutTroubleTicketResponse	No	Handles responses from the ORP trouble ticket request.
DetectORPFalloutHandlingType	No	Determines the fallout mode depending on whether this is used in a central order management or service order management context. For central order management, the fallout processing requires creation of a trouble ticket request to the upstream system. For service order management, fallout processing requires creation of a fault notification that is sent to the Oracle AIA error handling queue.

Table 4–18 lists the automation modules (with their associated automated tasks) defined in this cartridge.

Table 4–18 OracleComms_OSM_O2A_RecognitionFallout Automation Modules

Automation Module	Automated Task	Description
CreateErrorFaultBean	CreateFaultErrorTask	Calls the CreateErrorFault XQuery.
CreateORPFalloutTroubleTicketRequestBean	CreateORPFalloutTroubleTicketTask	Calls the CreateORPFalloutTroubleTicket XQuery.
ORPTTResponseBean	CreateORPFalloutTroubleTicketTask	External event receiver to invoke CreateORPFalloutTroubleTicketResponse XQuery.
DetectORPFalloutHandlingTypeBean	DetectORPFalloutHandlingTypeTask	Calls the DetectORPFalloutHandlingType XQuery.

OracleComms_OSM_O2A_SystemAdmin

The OracleComms_OSM_O2A_SystemAdmin cartridge is a productized cartridge that works in conjunction with the Inbound Message Recovery MDB to create fallout tasks that help you recover from inbound message processing errors. The OracleComms_OSM_O2A_SystemAdmin cartridge and Inbound Message Recovery MDB handle errors caused by the following:

- Suspended orders (See "Recovering from Inbound Message Errors Due to Suspended Orders")

- Order-to-Activate resource issues (See "Recovering from Inbound Message Errors Due to Resource Issues")

Table 4–19 describes the XQuery modules in the cartridge.

Table 4–19 OracleComms_OSM_O2A_SystemAdmin XQuery Modules

XQuery Module	Extendable	Description
InboundMessageHandlerDetection	No	Utility module for fallout recovery. Used to find out the handling component type from the response message based on the EBM type.
InBoundMessageRecovery	No	Routes the inbound message for recovery to appropriate target system.
ResumePendingInBoundMessage_OrderDataRule	No	Module used as order data rule when recognizing the inbound message to create the ResumePendingInBoundMessage order.

How the Inbound Message Recovery MDB Works

The Inbound Message Recovery MDB works with the OracleComms_OSM_O2A_SystemAdmin cartridge to handle inbound message errors. When the message cannot be delivered to the response queue due to JMS system errors, the response queue uses the OSM Integration Pack Fallout Queue. The Inbound Message Recovery MDB listens to the OSM Integration Pack Fallout Queue and does the following:

1. Routes the message (response) to the OSM InBoundMessageRecoveryQueue queue.
2. Produces a request to OSM to create the ResumePendingInBoundMessage order (using the OracleComms_OSM_O2A_SystemAdmin cartridge) by doing the following:
   a. Running a manual task that enables order management personnel to confirm that resource or XQuery logic errors have been resolved.

      This step can be configured to redeliver the inbound message automatically, by setting the JVM parameter **pip.require.ack** to **NO**. The number of automatic redelivery attempts for inbound messages is configured in the model variable named **IB_MSG_MAX_RETRY** in the Order-to-Activate composite cartridge.
   b. Running an automated task that moves the message from the recovery queue to the response queue.

OSM recognizes the ResumePendingInBoundMessage order, and the OracleComms_OSM_O2A_SystemAdmin cartridge begins to process.

Recovering from Inbound Message Errors Due to Suspended Orders

The following steps demonstrate how the Inbound Message Recovery MDB and the OracleComms_OSM_O2A_SystemAdmin cartridge recover from an error scenario where an inbound message is not processed because its associated order is suspended.

1. OSM sends a message to an external system using an automation task.
2. Before the automation task receives the inbound response message from the external system, the order associated with the automation task is suspended.
3. OSM receives the response message from the external system.
4. The automation task that receives the inbound response message does the following:

   a. Checks the order state while processing the response.
   b. If the order state is Suspended, moves the message to the OSM Inbound Message Recovery Queue (passing required order information such as order ID and FulfillmentFunction name, etc.).

      This logic is implemented in the resource **AIAEBMResponse.xqy**, which is located in **OracleComms_OSM_O2A_COM_SalesOrderFulfillmentPIP/resources/ComponentInteraction**.

5. When the order state changes from Suspended to In Progress, the order state change event handler creates the ResumePendingInBoundMessage order (using the OracleComms_OSM_O2A_SystemAdmin cartridge).
6. The ResumePendingInBoundMessage automation task of the process associated with the ResumePendingInBoundMessage order moves the original response message from the OSM Inbound Message Recovery Queue back to the response queue.
7. The automation task that could not process the inbound message in the response queue originally (due to its associated order being suspended) processes the inbound message successfully.

Recovering from Inbound Message Errors Due to Resource Issues

The following steps demonstrate how the Inbound Message Recovery MDB and the OracleComms_OSM_O2A_SystemAdmin cartridge recover from an error scenario where an inbound message processing error occurs due to a resource issue such as a Global Transaction Error (GTX) or an incorrect XQuery script.

1. OSM sends a message to an external system using an automation task and receives a response back from the external system.
2. The automation task receives the inbound response message in the response queue but cannot process the message because a resource required to process the message is currently locked by another task.

   For example, the resource could be locked because of a GTX timeout or because of an error in the logic of an XQuery script.
3. After a few retries, the automation task raises a fallout.
4. The fallout message goes to the OSM Integration Pack Fallout Queue.
5. The Inbound Message Recovery MDB, listening on the OSM Integration Pack Fallout Queue, moves the response message to the OSM Inbound Message Recovery Queue.
6. The MDB creates the ResumePendingInBoundMessage order (using the OracleComms_OSM_O2A_SystemAdmin cartridge), and its process begins to run.
7. The order process assigns a manual task (a fallout or confirmation task) to order management personnel who manage fallout.
8. The fallout task is displayed on the worklist of the Task web client.

Note: OSM does not raise a fallout notification to inform order management personnel that a fallout task has been created on the worklist.

9. Order management personnel resolve the resource error. For example, they correct the XQuery script logic and restart the system.
10. In the worklist, order management personnel click the **Confirm** button on the task and (optionally) specify the name of the response queue of the automation task that could not originally process the inbound message. If the name of the response queue is not specified, the ResumePendingInBoundMessage automated task uses the EBM type to detect which queue is the response queue and routes the message accordingly.
11. After the confirmation task completes, the process in the OracleComms_OSM_O2A_SystemAdmin cartridge runs the ResumePendingInBoundMessage automation task, which moves the original response message back to the response queue.
12. The original automation task that could not process the inbound message in the response queue is retried, and the message processes successfully.

Common Central Order Management Cartridges

The following cartridges operate in the central order management role, which coordinates fulfillment functions across the Business Support Systems (BSS) and Operations Support Systems (OSS) such as Siebel CRM, BRM, and provisioning.

OracleComms_OSM_O2A_COM_Base

The OracleComms_OSM_O2A_COM_Base cartridge is a productized cartridge supporting the orchestration of customer orders from Oracle AIA. It includes communication to and from fulfillment providers and handles status and data updates.

Order Events

When the COM_SalesOrderFulfillment order reaches one of the order events listed in Table 4–20, it triggers the listed XQuery module to send an order update to the upstream system.

Table 4–20 OracleComms_OSM_O2A_COM_Base Order Events

Order Event	Description
stateChange	Calls the OrderStateChangeHandler XQuery module to send an order update to the Siebel CRM system.
completion	Calls the OrderCompletionHandler XQuery module to send the order completion to the Siebel CRM system.

Processing Granularity Rules

There are four orchestration stages defined in the orchestration sequence to decompose the order line items. The result of each stage of decomposition is the source for the next stage of decomposition.

- In the first stage, the order line items are decomposed by fulfillment function.
- In the second stage, the order line items are decomposed by fulfillment provider.
- In the third stage, the order line items are decomposed by granularity rule.
- In the fourth stage, depending on the fulfillment function process, central order management will use the fulfillment function process to determine whether to

create an executable order component with all of the order line items if a significant change is detected on any order line item.

Granularity rules provide the configuration for the third stage of decomposition. During orchestration plan generation at run time, the granularity rule takes as input the order line items that have already been grouped by fulfillment function and subdivided by fulfillment provider.

The behavior of granularity rules varies between design time and run time.

For example, during design time, a granularity rule such as ServiceBundleGranularity or BundleGranularity is selected per fulfillment function by creating one decomposition rule per fulfillment function for use in the third stage of decomposition.

During run time, granularity rules group the order line items into one or more fulfillment requests. Granularity rules group the order line items that are targeted at the same fulfillment function and are specific to a fulfillment provider.

Table 4–21 lists the processing granularity rules.

Table 4–21 OracleComms_OSM_O2A_COM_Base Processing Granularity Rules

Name	Entity Type	Description
BundleGranularity	Order Component Specification	This granularity rule selects: ▪ An order line item that represents a bundle along with bundle components and related order line items Nested bundles are considered components of the root bundle and are processed in the same fulfillment request. In Siebel CRM, a bundle is referred to as a Commercial Bundle. ▪ Order line items of any other root node on the order along with their related order line items
OfferGranularity	Order Component Specification	This granularity rule selects: ▪ An order line item that represents an offer along with offer components and Related order line items In Siebel CRM, an offer is referred to as a promotion. ▪ Order line items of any other root node on the order along with their related order line items

Table 4–21 (Cont.) OracleComms_OSM_O2A_COM_Base Processing Granularity Rules

Name	Entity Type	Description
OrderGranularity	Order Component Specification	This granularity rule selects all lines targeted at the same fulfillment function and specific to a fulfillment provider make a single fulfillment request.
ServiceBundleGranularity	Order Component Specification	This granularity rule selects: ▪ An order line item that represents a service bundle along with service bundle components and related order line items Nested service bundles and their components make separate fulfillment requests. ▪ Order line items of any other root node on the order along with their related order line items This granularity rule implements an optimization to group together offers and non-service billing items into a single fulfillment request to be fulfilled at the same time
WholeItemGranularity	Order Component Specification	This granularity rule selects: ▪ An order line item that represents a whole item along with whole item components and related order line items Nested whole items and their components make separate fulfillment requests. ▪ Order line items of any other root node on the order along with their related order line items

Abstract Orchestration Entities

Table 4–22 lists the orchestration entities that are used as base entities for fulfillment function, fulfillment system, process granularity rule, and fulfillment function updates.

Table 4–22 OracleComms_OSM_O2A_COM_Base Abstract Orchestration Entities

Name	Type	Description
COM_FulfillmentFunction	Order Component Specification	This order component specification represents the base fulfillment function in central order management. All fulfillment functions, such as SyncCustomerFunction, should extend from it. This order component also contains the external fulfillment state definitions.
COM_FulfillmentGranularity	Order Component Specification	This order component specification represents the processing granularity rule used in the orchestration stage. All processing granularity rules should extend from it.
COM_FulfillmentSystem	Order Component Specification	This order component specification represents the base fulfillment system in central order management. All fulfillment systems, such as BRM-VOIP, should extend from it.
COM_FulfillmentSignificantUpdates	Order Component Specification	This order component specification represents the base fulfillment function with significant updates in the fourth orchestration stage.

Order Lifecycle Manager Configuration

The Order-to-Activate order lifecycle manager is configured with the header values for the Order Lifecycle Management user interface. It also contains mappings between Order-to-Activate central order management fulfillment states and standard order lifecycle manager states.

Table 4–23 displays the mappings that are configured. The high-level fulfillment states are mapped, which causes the child states to be mapped as well.

Table 4–23 Fulfillment State to Order Lifecycle Manager State Mapping

Fulfillment State	Order Lifecycle Manager State
CANCELLED	Canceled
COMPLETE	Complete
FAILED	Failed
IN_PROGRESS	In Progress

Note: If you have both central order management and service order management in the same Design Studio workspace, you will see service order management fulfillment states in the list in the order lifecycle manager. The names of the high-level fulfillment states for service order management all start with **SOM_**. The service order management fulfillment states do not need to be mapped here, because they are mapped in the order lifecycle manager in the service order management configuration. See "Order Lifecycle Manager Configuration" for information about service order management state mappings.

XQuery Modules in the OracleComms_OSM_O2A_COM_Base Cartridge

Table 4–24 through Table 4–32 list the different types of XQuery modules in this cartridge.

No table is included for the Order Item Property XQuery modules because none are extendable and each XQuery module does the same thing: retrieves the specified order item property from the appropriate location in the order data.

Table 4–24 OracleComms_OSM_O2A_COM_Base XQuery Modules for Constants

Constants XQuery Module	Extendable	Description
O2AConstants	No	Defines overall solution constants.
PromotionGroupConstants	No	Defines constants for processing promotion groups in central order management.
QueryViewConstants	No	Defines constants for querying views in central order management.

Table 4–25 OracleComms_OSM_O2A_COM_Base XQuery Modules for Fallout Handling

Fallout Handling XQuery Module	Extendable	Description
AbortOrderRequest	No	Sends an order termination request for the fulfillment request through the web service API.
AbortOrderResponse	No	Receives the response to the order termination request for the fulfillment request through the web service API.
CFwsResponseHandler	No	Utility module for providing retrieval and update to central order management order.
CloseCreationFailedTroubleTickets	No	Sends a request to the trouble ticketing system to close the trouble tickets for orders with the same Oracle AIA sales order key.
CloseTroubleTicket	No	Creates a request to the trouble ticketing system to close a trouble ticket.
CreateSIFalloutTroubleTicket	No	Creates a trouble ticket for system interaction.
CreateTroubleTicket	No	Creates a trouble ticket for both system interactions and Order Request Processor (ORP) errors.
FalloutNotificationRouter	No	Routes fallout notifications to different fallout process to handle updating EBM, creating a trouble ticket, and keeping track of the trouble ticket in the OSM order.
FalloutNotificationToCFTask	No	Directs fallout notifications to central order management.
FindFulfillmentOrderData	No	Retrieves the fulfillment request data in fallout.
FindOrderCreationFailedTroubleTickets	No	Finds an order that failed at creation with the Oracle AIA sales order key.
GetCreationFailFulfillmentOrder	No	Retrieves the order data for an order that failed at creation.
GetFulfillmentOrderResponse	No	Utility module to handle the web service response for the find order request.
GetTroubleTicketData	No	Updates trouble ticket data on the OSM order.
OrderAbortPropagation	No	Sends an order termination request to the service order through the Oracle AIA provisioning order queue.
OrderAbortPropagationCheck	No	Checks the status of the order termination request for the service order.
OrderAbortPropagationResp	No	Handles the response of the order termination request to the service order.
SuspendCFOrder	No	Suspends a central order management order.
SuspendCFOrderResponse	No	Utility module to handle the web service response for the suspend order request.
UpdateCreationFailFulfillmentOrder	No	Updates the trouble ticket information back to the OSM order that failed at creation.
UpdateFulfillmentOrder	No	Updates the trouble ticket data on the OSM order. This is used by both UpdateCreationFailFulfillmentOrder and UpdateSIFalloutTroubleTicket XQuery file.

Table 4–25 (Cont.) OracleComms_OSM_O2A_COM_Base XQuery Modules for Fallout Handling

Fallout Handling XQuery Module	Extendable	Description
UpdateSIFalloutTroubleTicket	No	Updates the trouble ticket information back to the OSM order that has system interaction fallout.
UpdateStatusRequest	No	Creates an update status EBM to the Siebel CRM system for fallout.
UpdateTroubleTicket	No	Creates the trouble ticket payload for the trouble ticketing system.

Table 4–26 OracleComms_OSM_O2A_COM_Base Orchestration Sequence XQuery Modules

Orchestration Sequence XQuery Module	Extendable	Description
FulfillmentModeExpression	No	Marshals the fulfillment mode code from the customer order.
OrderItemSelector	No	Selects all order line items from the customer order.

Table 4–27 OracleComms_OSM_O2A_COM_Base Order Data Change XQuery Modules

Order Data Change XQuery Module	Extendable	Description
CloseFalloutTroubleTicket	No	Creates a request to close a trouble ticket.
CreateFalloutOrderNotification	No	Creates a fallout order notification to handle fallout for an order that failed at creation.
UpdateSalesOrderFalloutStatusRequest	No	Creates an EBM with order status context being populated with fallout information.

Table 4–28 OracleComms_OSM_O2A_COM_Base Order Item Hierarchy XQuery Modules

Order Item XQuery Module	Extendable	Description
InterOrderDependency	No	Determines the inter-order dependency based on the order item's dependencies across different orders.
LineIdKey	No	Retrieves the order line item's ID.
ParentLineIdKey	No	Retrieves the parent order line item's ID.
PromotionGroupKey	No	Retrieves the order line item's promotion group.
PromotionGroupParentKey	No	Retrieves the parent order line item's promotion group.
RelatedSalesOrderLineIdKey	No	Retrieves the related sales order line item's ID.
RootParentSalesOrderLineIdKey	No	Retrieves the root order item business component ID.

Table 4–29 OracleComms_OSM_O2A_COM_Base Order Recognition XQuery Modules

Order Recognition XQuery Module	Extendable	Description
AIAOrderData	No	Transforms the customer order to an OSM order.
AIAOrderPriority	No	Retrieves the priority of the customer order.

Table 4–29 (Cont.) OracleComms_OSM_O2A_COM_Base Order Recognition XQuery Modules

Order Recognition XQuery Module	Extendable	Description
AIAOrderRecognition	No	Recognizes the customer order.
AIAOrderReference	No	Adds the **-TSQ** suffix to order identification if the order is to be processed in Technical Service Qualification mode. (This allows the same customer order to be sent later as a DELIVER order with the same order ID.)
AIAOrderValidation	No	Validates the customer order.

Table 4–30 OracleComms_OSM_O2A_COM_Base Order State XQuery Modules

Order State XQuery Module	Extendable	Description
OrderAbortedStateHandler	No	Creates an EBM to update the Siebel CRM order with an aborted status context.
OrderCancelledStateHandler	No	Creates an EBM to update the Siebel CRM order with a canceled status context.
OrderCompletedStateHandler	No	Creates an EBM to update the Siebel CRM order with a completed status context.
OrderCompletionHandler	No	Responds to the central order management order completion event and triggers the OrderCompletedStateHandler module.
OrderFailedStateHandler	No	Creates an EBM to update the Siebel CRM order with a failed status context.
OrderInProgressStateHandler	No	Creates an EBM to update the Siebel CRM order with an in-progress status context.
OrderStateChangeHandler	No	Responds to the central order management order state change event and triggers the appropriate OrderStateHandler according to the order state.
OrderStateUtilityModule	No	Provides utility functions related to order state.

Table 4–31 OracleComms_OSM_O2A_COM_Base Order Transformation Manager XQuery Modules

Order Transformation Manager XQuery Module	Extendable	Description
OTMMappingModule	No	Utilities to support the order transformation manager.

Table 4–32 OracleComms_OSM_O2A_COM_Base Processing Granularity XQuery Modules

Processing Granularity XQuery Module	Extendable	Description
BundleGranularity	No	Groups related order items as a bundle for processing.
GranularityModule	No	Utility module to group order items based on service action code.
OfferBundleGranularity	No	Groups related order items as an offer for processing.
OrderGranularity	No	Groups related order items as an order for processing.
ServiceBundleGranularity	No	Groups related order items as a service for processing.
WholeItemGranularity	No	Groups related order items as a whole item for processing.

Automation Modules in the OracleComms_OSM_O2A_COM_Base Cartridge

Table 4–33 lists the automation modules in the cartridge with their associated automated tasks.

Table 4–33 OracleComms_OSM_O2A_COM_Base Automation Modules

Automation Module	Automated Task	Description
AbortOrderRequestBean	AbortFulfillmentOrderTask	Calls the AbortOrderRequest XQuery.
CloseOrderCreationFailedTroubleTicketsBean	CloseCreationFailedTroubleTicketTask	External event receiver to invoke CloseCreationFailedTroubleTicket XQuery.
FindOrderCreationFailedTroubleTicketsBean	CloseCreationFailedTroubleTicketTask	Calls the FindOrderCreationFailedTroubleTickets XQuery.
CreateSIFalloutTroubleTicketRequestBean	CreateSIFalloutTroubleTicket Task	Calls the CreateSIFalloutTroubleTicket XQuery.
GetSIFalloutTroubleTicketResponse	CreateSIFalloutTroubleTicket Task	External event receiver to invoke GetTroubleTicketData XQuery.
FalloutnotificationRouterBean	FalloutNotificationRouterTask	Calls the FalloutNotificationRouter XQuery.
SIFalloutNotificationToCF	FalloutNotificationToCFTask	Internal event receiver to invoke FalloutNotificationToCFTask XQuery.
GetCreationFailFulfillmentOrderBean	GetCreationFailFulfillmentOrderTask	Internal event receiver to invoke GetCreationFailFulfillmentOrder XQuery.
GetFulfillmentOrder	GetFulfillmentOrderTask	Calls the FindFulfillmentOrderData XQuery.
GetFulfillmentOrderResponse	GetFulfillmentOrderTask	External event receiver to invoke CFwsResponseHandler XQuery.
OrderAbortPropagationCheck Plugin	OrderAbortPropagationCheck	Calls the OrderAbortPropagationCheck XQuery.
OrderAbortPropagationPlugin	OrderAbortPropagationTask	Calls the OrderAbortPropagation XQuery.
OrderAbortPropagationRespPlugin	OrderAbortPropagationTask	External event receiver to invoke OrderAbortPropagationResp XQuery.
SuspendCFOrderPlugin	SetCFOrderAbortInProgress Task	Calls the SuspendCFOrder XQuery.
SuspendCFOrderRespPlugin	SetCFOrderAbortInProgress Task	External event receiver to invoke CFwsResponseHandler XQuery.
UpdateCreationFailFulfillment OrderBean	UpdateCreationFailFulfillmentOrderTask	Calls the UpdateCreationFailFulfillmentOrder XQuery.
UpdateFulfillmentOrderBean	UpdateFulfillmentOrderTask	Calls the UpdateFulfillmentOrder XQuery.
UpdateSIFalloutTroubleTicket Request	UpdateSIFalloutTroubleTicketTask	Calls the UpdateSIFalloutTroubleTicket XQuery.
UpdateStatusRequestBean	UpdateStatusToCRMTask	Calls the UpdateStatusRequest XQuery.

External Fulfillment States

External fulfillment states in the OracleComms_OSM_O2A_COM_Base cartridge are defined in the **COM_FulfillmentFunction** order component specification, representing the base fulfillment function. All fulfillment functions, such as **SyncCustomerFunction**, extend from **COM_FulfillmentFunction**.

The following external fulfillment states are defined in this cartridge:

- OPEN

- IN_PROGRESS
- IN_PROGRESS-FULFILL_BILLING_START
- IN_PROGRESS-INITIATE_BILLING_START
- IN_PROGRESS-INSTALL_START
- IN_PROGRESS-INSTALL_PLANNED
- IN_PROGRESS-INSTALL_COMMITTED
- IN_PROGRESS-PROVISION_START
- IN_PROGRESS-PROVISION_DESIGNED
- IN_PROGRESS-SHIP_ORDER_START
- IN_PROGRESS-SHIP_ORDER_PLANNED
- IN_PROGRESS-SYNC_CUSTOMER_START
- COMPLETE
- COMPLETE-FULFILL_BILLING_COMPLETE
- COMPLETE-INITIATE_BILLING_COMPLETE
- COMPLETE-INSTALL_COMPLETE
- COMPLETE-PROVISION_COMPLETE
- COMPLETE-SHIP_ORDER_SHIPPED
- COMPLETE-SYNC_CUSTOMER_COMPLETE
- CANCELLED
- CANCELLED-FULFILL_BILLING_COMPLETE
- CANCELLED-FULFILL_BILLING_START
- CANCELLED-INITIATE_BILLING_COMPLETE
- CANCELLED-INITIATE_BILLING_START
- CANCELLED-INSTALL_COMMITTED
- CANCELLED-INSTALL_COMPLETE
- CANCELLED-INSTALL_PLANNED
- CANCELLED-INSTALL_START
- CANCELLED-PROVISION_COMPLETE
- CANCELLED-PROVISION_DESIGNED
- CANCELLED-PROVISION_START
- CANCELLED-SHIP_ORDER_PLANNED
- CANCELLED-SHIP_ORDER_SHIPPED
- CANCELLED-SHIP_ORDER_START
- CANCELLED-SYNC_CUSTOMER_COMPLETE
- CANCELLED-SYNC_CUSTOMER_START
- FAILED
- FAILED-FULFILL_BILLING_COMPLETE

- FAILED-FULFILL_BILLING_START
- FAILED-INITIATE_BILLING_COMPLETE
- FAILED-INITIATE_BILLING_START
- FAILED-INSTALL_COMMITTED
- FAILED-INSTALL_COMPLETE
- FAILED-INSTALL_PLANNED
- FAILED-INSTALL_START
- FAILED-PROVISION_COMPLETE
- FAILED-PROVISION_DESIGNED
- FAILED-PROVISION_START
- FAILED-SHIP_ORDER_PLANNED
- FAILED-SHIP_ORDER_SHIPPED
- FAILED-SHIP_ORDER_START
- FAILED-SYNC_CUSTOMER_COMPLETE
- FAILED-SYNC_CUSTOMER_START
- TSQ_Passed
- TSQ_Passed-PROVISION_DESIGNED
- TSQ_Failed
- TSQ_Failed-PROVISION_DESIGNED

OracleComms_OSM_O2A_COM_SalesOrderFulfillment

The OracleComms_OSM_O2A_COM_SalesOrderFulfillment cartridge is a productized cartridge supporting the communications between central order management and fulfillment systems. It includes resources to generate requests to fulfillment providers and consume their responses and to do validation and condition evaluation.

Table 4–34 lists the XQuery modules defined in this cartridge.

Table 4–34 OracleComms_OSM_O2A_COM_SalesOrderFulfillment XQuery Modules

XQuery Module	Extendable	Description
AIAEBMRequest_do	No	Generates Oracle AIA EBM requests to a fulfillment provider.
AIAEBMRequest_redo	No	Generates Oracle AIA EBM requests to a fulfillment provider for redo.
AIAEBMRequest_undo	No	Generates Oracle AIA EBM requests to a fulfillment provider for undo.
AIAEBMResponse_ValidationModule	No	Validates Oracle AIA EBM responses from a fulfillment provider.
AIAEBMResponse	No	Consumes Oracle AIA EBM responses from a fulfillment provider.
DoublePlayComponentDependency	No	Order component dependency rule used to create the orchestration plan.
FalseRevision	No	Utility module to detect a false revision order.

Table 4–34 (Cont.) OracleComms_OSM_O2A_COM_SalesOrderFulfillment XQuery Modules

XQuery Module	Extendable	Description
InitiateWaitForProvisioningResponse	No	Initiates the wait for a provisioning response.
OrderLifecycleManagementModule	No	Utility module to support order lifecycle management for order and order line items.
PostSIBreakpoint	No	Supports breakpoints for the automated task *Function*PostSIBreakTask during undo.
PreSIBreakpoint	No	Supports breakpoints for the automated task *Function*PreSIBreakTask during undo.
SIEntryPoint	No	Provides lifecycle management for both customer order and service order. Used by automated task *Function*EntryPointTask. (This task should be used as the entry point task for a new fulfillment function.)
SIExitPoint	No	Module used by automated task *Function*ExitPointTask to simulate order item data updates for the InstallOrder and ShipOrder fulfillment functions.
SIMilestone_doredo	No	Simulates milestone updates for the automated task *Function*PlannedTask for both InstallOrder and ShipOrder fulfillment function.
SIStartPoint	No	Module used by automated task FulfillBillingStartTask to provide a breakpoint before the start of the FulfillBilling function.
UpdateSalesOrderStatusFunctions	No	Provides functions for updating customer order’s status.
FulfillmentStateModule	No	Contains configuration for calculating order and order item fulfillment state.
PointOfNoReturn	No	Checks whether an order component or order item has reached the point of no return.

OracleComms_OSM_O2A_COM_Billing

The OracleComms_OSM_O2A_COM_Billing cartridge is a productized cartridge that supports billing fulfillment functions. These functions specify subprocesses to handle delivery of a relevant subset of order data to the BRM ABCS, and handle responses from BRM ABCS. The modeled interaction includes coping with fallout, order change management, and status or data updates back to the CRM ABCS.

Table 4–35 lists the fulfillment functions defined in the cartridge.

Table 4–35 OracleComms_OSM_O2A_COM_Billing Fulfillment Functions

Fulfillment Function	Billing Pattern	Description
SyncCustomerFunction	Single-phase, two-phase	Provides the ability to synchronize only accounts from CRM to Billing as part of an order. SyncCustomerFunction is used in both single-phase and two-phase billing patterns. This function is essential because the pre-existence of a customer account in the billing provider is assumed before billing.
InitiateBillingFunction	Two-phase	Provides the ability to start the usage cycle. InitiateBillingFunction is the first phase in the two-phase billing pattern. In two-phase billing patterns, the first phase invokes InitiateBillingFunction, and the second phase invokes FulfillBillingFunction.
FulfillBillingFunction	Single-phase, two-phase	Provides the ability to start the billing cycle. FulfillBillingFunction is the single phase in the single-phase billing pattern and the second phase in the two-phase billing pattern.

Table 4–36 lists the XQuery modules in the cartridge that support component interaction.

Table 4–36 OracleComms_OSM_O2A_COM_Billing Component Interaction XQuery Modules

Component Interaction XQuery Module	Extendable	Description
BillingPatternModule	No	Determines the billing pattern based on order item's billing product type and product specification.
BillingInteractionModule	No	Creates a fulfillment request (in the format of an Oracle AIA EBM) to send to the billing system.
SyncCustomerInteractionModule	No	Provides functions to support SyncCustomerFunction.
InitiateBillingInteractionModule	No	Provides functions to support InitiateBillingFunction, using the BillingInteractionModule utility module.
FulfillBillingInteractionModule	No	Provides functions to support FulfillBillingFunction, using the BillingInteractionModule utility module.

SyncCustomerFunction

This section provides details of SyncCustomerFunction, one of the three fulfillment functions in the OracleComms_OSM_O2A_COM_Billing.

SyncCustomerFunction and Decomposition Rules

For the fulfillment request to be relevant for the billing provider to process, there must be at least one order line item with a service action that is relevant for the SyncCustomerFunction function to process. The decomposition rules in Table 4–37 ensure that SyncCustomerFunction is called only if relevant.

Table 4–37 Decomposition Rules for SyncCustomerFunction

Configuration	Cartridge	Decomposition Rule
Solution option without calculate service order, Simple Topology	OracleComms_OSM_O2A_SimpleTopology_Sample	Simple_DetermineSignificantUpdates_For_SyncCustomer
Solution option without calculate service order, Typical or Complex Topology	OracleComms_OSM_O2A_TypicalTopology_Sample	Typical_DetermineSignificantUpdates_For_SyncCustomer
Calculate service order solution option, all topologies	OracleComms_OSM_O2A_COM_CSO_Topology	Typical_DetermineSignificantUpdates_For_SyncCustomer

The decomposition conditions in the rules above return true() if fromOrderComponent (the order component being decomposed from) has at least one order line item that is relevant for SyncCustomerFunction to process. An order line item property is initialized to YES if the order line item is relevant for the billing provider's SyncCustomerFunction to process. By default, the service actions that are relevant for SyncCustomerFunction to process are:

- Order line items with ServiceActionCode=**ADD**
- SyncCustomer compensation-significant updates: Order line items with ServiceActionCode=**UPDATE** or **MOVE-ADD** with compensation-significant updates as determined by a comparison of the new and prior values from Siebel CRM in the customer order

SyncCustomerFunction and Fulfillment Patterns

The **SyncCustomerFunction** order component for fulfillment mode **DELIVER** is selected for each fulfillment pattern that supports the single and two-phase billing patterns. The order component **SyncCustomerFunction** is included in the **BaseProductSpec** fulfillment pattern. As a result, all fulfillment patterns which inherit from **BaseProductSpec** include **SyncCustomerFunction** as part of their fulfillment flow. This applies to each entity in the cartridge that configures fulfillment patterns, including any custom cartridge specifying the **COM_SalesOrderFulfillment** namespace.

SyncCustomerFunction XQuery Modules

Table 4–38 lists the XQuery modules defined for the SyncCustomerFunction fulfillment function. Customers can provide their own implementation of the XQuery modules in this fulfillment function indicated extension points. See "Extending XQuery Modules" for more information about XQuery extension points.

Table 4–38 SyncCustomerFunction XQuery Modules

XQuery Module	XQuery Extension Point	Description
SyncCustomerComplete_Event	COMPONENT-COMPLETE	Invoked when the SyncCustomerFunction component is completed.
SyncCustomerCreateAllOrderItemsEBM_Event	CREATE-EBM-ALL-ORDERITEMS	Invoked after CREATE-EBM-CUSTOM for the SyncCustomerFunction component to create all order items. This should always be invoked unless you want to create only the order header without any order items.
SyncCustomerCreateCustomEBM_Event	CREATE-EBM-CUSTOM	Invoked after CREATE-EBM for the SyncCustomerFunction component. Invokes extension logic on the order-level CUSTOM element.
SyncCustomerCreateEBM_DoEvent	CREATE-EBM (execution mode: *do*)	Invoked when the EBM is created in the **do** execution mode for the SyncCustomerFunction component.
SyncCustomerCreateEBM_ReDoEvent	CREATE-EBM (execution mode: *redo*)	Invoked when the EBM is created in the **redo** execution mode for the SyncCustomerFunction component.
SyncCustomerCreateEBM_UnDoEvent	CREATE-EBM (execution mode: *undo*)	Invoked when the EBM is created in the **undo** execution mode for the SyncCustomerFunction component.
SyncCustomerCreateOrderItemCustomEBM_Event	CREATE-EBM-ORDERITEM-CUSTOM	Invoked after CREATE-EBM-ORDERITEM for the SyncCustomerFunction component. Invokes extension logic on the order-item-level CUSTOM element.
SyncCustomerCreateOrderItemEBM_DoEvent	CREATE-EBM-ORDERITEM (execution mode: *do*)	Invoked after CREATE-EBM-ALL-ORDERITEMS for the SyncCustomerFunction component in **do** execution mode. Invokes extension logic on the order item element.
SyncCustomerCreateOrderItemEBM_ReDoEvent	CREATE-EBM-ORDERITEM (execution mode: *redo*)	Invoked after CREATE-EBM-ALL-ORDERITEMS for the SyncCustomerFunction component in **redo** execution mode. Invokes extension logic on the order item element.
SyncCustomerCreateOrderItemEBM_UnDoEvent	CREATE-EBM-ORDERITEM (execution mode: *undo*)	Invoked after CREATE-EBM-ALL-ORDERITEMS for the SyncCustomerFunction component in **undo** execution mode. Invokes extension logic on the order item element.
SyncCustomerCreatePriorOrderItemCustomEBM_Event	CREATE-EBM-PRIORORDERITEM-CUSTOM	Invoked after CREATE-EBM-PRIORORDERITEM for the SyncCustomerFunction component. Invokes extension logic on the prior-order-item-level CUSTOM element.
SyncCustomerCreatePriorOrderItemEBM_Event	CREATE-EBM-PRIORORDERITEM	Invoked after CREATE-EBM-ORDERITEM-CUSTOM for the SyncCustomerFunction component. Invokes extension logic on the prior order item element.

Table 4–38 (Cont.) SyncCustomerFunction XQuery Modules

XQuery Module	XQuery Extension Point	Description
SyncCustomerStart_Event	COMPONENT-START	Invoked when the SyncCustomerFunction component is started.
SyncCustomerUpdate_Event	COMPONENT-RESPONSE-UPDATE	Invoked when the SyncCustomerFunction updates are received to process order items on the billing response.
SyncCustomerValidateResponseEBM_Event	VALIDATE-RESPONSE-EBM	Invoked to validate the EBM response for the SyncCustomerFunction component.

SyncCustomerFunction Automation Modules

Table 4–39 lists the automation modules (with their associated automated tasks) defined in the OracleComms_OSM_O2A_COM_Billing cartridge for the SyncCustomerFunction fulfillment function.

Table 4–39 SyncCustomerFunction Automation Modules

Automation Module	Automated Task	Description
SyncCustomerEntryPointBean_doredo	SyncCustomerEntryPointTask	Calls the SIEntryPoint XQuery.
SyncCustomerRequestBean_do	SyncCustomerSITask	Calls the AIAEBMRequest_do XQuery.
SyncCustomerRequestBean_redo	SyncCustomerSITask	Calls the AIAEBMRequest_redo XQuery.
SyncCustomerRequestBean_undo	SyncCustomerSITask	Calls the AIAEBMRequest_undo XQuery.
SyncCustomerResponseBean	SyncCustomerSITask	External event receiver to invoke AIAEBMResponse XQuery.

InitiateBillingFunction

This section provides details of InitiateBillingFunction, one of the three fulfillment functions in the OracleComms_OSM_O2A_COM_Billing.

InitiateBillingFunction and Decomposition Rules

For the fulfillment request to be relevant for the billing provider to process, there must be at least one order line item with a service action that is relevant for the InitiateBillingFunction function to process. The decomposition rules in Table 4–40 ensure that InitiateBillingFunction is called only if relevant.

Table 4–40 Decomposition Rules for InitiateBillingFunction

Topology	Cartridge	Decomposition Rule
Solution option without calculate service order, Simple Topology	OracleComms_OSM_O2A_SimpleTopology_Sample	Simple_DetermineSignificantUpdates_For_InitiateBilling
Solution option without calculate service order, Typical or Complex Topology	OracleComms_OSM_O2A_TypicalTopology_Sample	Typical_DetermineSignificantUpdates_For_InitiateBilling
Calculate service order solution option, all topologies	OracleComms_OSM_O2A_COM_CSO_Topology	Typical_DetermineSignificantUpdates_For_InitiateBilling

The decomposition conditions in the rules above return true() if fromOrderComponent (the order component being decomposed from) has at least one order line item that is relevant for InitiateBillingFunction to process. By default, the service actions that are relevant for InitiateBillingFunction to process are:

- Order line items with ServiceActionCode=**ADD**

InitiateBillingFunction and Fulfillment Patterns

The **InitiateBillingFunction** order component for fulfillment mode **DELIVER** is selected for each fulfillment pattern that supports the two-phase billing pattern. This includes the OracleComms_OSM_O2A_BBVoIPFulfillmentPatternNileFlow_Sample, OracleComms_OSM_O2A_BBVoIPFulfillmentPatternDanubeFlow_Sample, or any custom cartridge specifying the **COM_SalesOrderFulfillment** namespace.

Some of the sample fulfillment patterns included in the InitiateBillingFunction in the fulfillment flow are:

- Service.VoIP
- Service.CPE.VoIP (The IntiateBillingFunction order component is conditional based on whether the VoIP CPE is contained in a VoIP service. This condition is included in decomposition rules InitiateBillingFunction_To_*YourSystemInstanceName1*… InitiateBillingFunction_To_*YourSystemInstanceNameN*)
- NonService.Offer (The IntiateBillingFunction order component is conditional based on whether the Offer contains VoIP services. This condition is included in decomposition rules InitiateBillingFunction_To_YourSystemInstanceName1… InitiateBillingFunction_To_YourSystemInstanceNameN)
- NonService.BillingInitiatedItem
- Non.Service.BillingItem

InitiateBillingFunction XQuery Modules

Table 4–41 lists the XQuery modules defined for the InitiateBillingFunction fulfillment function. Customers can provide their own implementation of the XQuery modules in this fulfillment function indicated extension points. See "Extending XQuery Modules" for more information about XQuery extension points.

Table 4–41 InitiateBillingFunction XQuery Modules

XQuery Module	XQuery Extension Point	Description
InitiateBillingComplete_Event	COMPONENT-COMPLETE	Invoked when the InitiateBillingFunction component is completed.
InitiateBillingCreateAllOrderItemsEBM_Event	CREATE-EBM-ALL-ORDERITEMS	Invoked after CREATE-EBM-CUSTOM for the InitiateBillingFunction component to create all order items. This should always be invoked unless you want to create only the order header without any order items.
InitiateBillingCreateCustomEBM_Event	CREATE-EBM-CUSTOM	Invoked after CREATE-EBM for the InitiateBillingFunction component. Invokes extension logic on the order-level CUSTOM element.
InitiateBillingCreateEBM_DoEvent	CREATE-EBM (execution mode: *do*)	Invoked when the EBM is created in the do execution mode for the InitiateBillingFunction component.
InitiateBillingCreateEBM_ReDoEvent	CREATE-EBM (execution mode: *redo*)	Invoked when the EBM is created in the redo execution mode for the InitiateBillingFunction component.
InitiateBillingCreateEBM_UnDoEvent	CREATE-EBM (execution mode: *undo*)	Invoked when the EBM is created in the undo execution mode for the InitiateBillingFunction component.
InitiateBillingCreateOrderItemCustomEBM_Event	CREATE-EBM-ORDERITEM-CUSTOM	Invoked after CREATE-EBM-ORDERITEM for the InitiateBillingFunction component. Invokes extension logic on the order-item-level CUSTOM element.
InitiateBillingCreateOrderItemEBM_DoEvent	CREATE-EBM-ORDERITEM (execution mode: *do*)	Invoked after CREATE-EBM-ALL-ORDERITEMS for the InitiateBillingFunction component in **do** execution mode. Invokes extension logic on the order item element.
InitiateBillingCreateOrderItemEBM_ReDoEvent	CREATE-EBM-ORDERITEM (execution mode: *redo*)	Invoked after CREATE-EBM-ALL-ORDERITEMS for the InitiateBillingFunction component in **redo** execution mode. Invokes extension logic on the order item element.
InitiateBillingCreateOrderItemEBM_UnDoEvent	CREATE-EBM-ORDERITEM (execution mode: *undo*)	Invoked after CREATE-EBM-ALL-ORDERITEMS for the InitiateBillingFunction component in **undo** execution mode. Invokes extension logic on the order item element.
InitiateBillingCreatePriorOrderItemCustomEBM_Event	CREATE-EBM-PRIORORDERITEM-CUSTOM	Invoked after CREATE-EBM-PRIORORDERITEM for the InitiateBillingFunction component. Invokes extension logic on the prior-order-item-level CUSTOM element.
InitiateBillingCreatePriorOrderItemEBM_Event	CREATE-EBM-PRIORORDERITEM	Invoked after CREATE-EBM-ORDERITEM-CUSTOM for the InitiateBillingFunction component. Invokes extension logic on the prior order item element.

Table 4–41 (Cont.) InitiateBillingFunction XQuery Modules

XQuery Module	XQuery Extension Point	Description
InitiateBillingStart_Event	COMPONENT-START	Invoked when the InitiateBillingFunction component is started.
InitiateBillingUpdate_Event	COMPONENT-RESPONSE-UPDATE	Invoked when the InitiateBillingFunction updates are received to process order items on the billing response.
InitiateBillingValidateResponseEBM_Event	VALIDATE-RESPONSE-EBM	Invoked to validate the EBM response for the InitiateBillingFunction component.

InitiateBillingFunction Automation Modules

Table 4–42 lists the automation modules (with their associated automated tasks) defined in the OracleComms_OSM_O2A_COM_Billing cartridge for the InitiateBillingFunction fulfillment function.

Table 4–42 InitiateBillingFunction Automation Modules

Automation Module	Automated Task	Description
InitiateBillingEntryPointBean_doredo	InitiateBillingEntryPointTask	Calls the SIEntryPoint XQuery.
InitiateBillingRequestBean_do	InitiateBillingSITask	Calls the AIAEBMRequest_do XQuery.
InitiateBillingRequestBean_redo	InitiateBillingSITask	Calls the AIAEBMRequest_redo XQuery.
InitiateBillingRequestBean_undo	InitiateBillingSITask	Calls the AIAEBMRequest_undo XQuery.
InitiateBillingResponseBean	InitiateBillingSITask	External event receiver to invoke AIAEBMResponse XQuery.

FulfillBillingFunction

This section provides details of FulfillBillingFunction, one of the three fulfillment functions in the OracleComms_OSM_O2A_COM_Billing.

FulfillBillingFunction and Decomposition Rules

For the fulfillment request to be relevant for the billing provider to process, there must be at least one order line item with a service action that is relevant for the FulfillBillingFunction function to process. The decomposition rules in Table 4–43 ensure that FulfillBillingFunction is called only if relevant.

Table 4–43 Decomposition Rules for FulfillBillingFunction

Topology	Cartridge	Decomposition Rule
Solution option without calculate service order, Simple Topology	OracleComms_OSM_O2A_SimpleTopology_Sample	Simple_DetermineSignificantUpdates_For_FulfillBilling
Solution option without calculate service order, Typical or Complex Topology	OracleComms_OSM_O2A_TypicalTopology_Sample	Typical_DetermineSignificantUpdates_For_FulfillBilling
Calculate service order solution option, all topologies	OracleComms_OSM_O2A_COM_CSO_Topology	Typical_DetermineSignificantUpdates_For_FulfillBilling

The decomposition conditions in the rules above return true() if fromOrderComponent (the order component being decomposed from) has at least one order line item that is relevant for FulfillBillingFunction to process. By default, the service actions that are relevant for FulfillBillingFunction to process are:

- Order line items with ServiceActionCode=**ADD**, **DELETE**, **UPDATE**, **SUSPEND**, **RESUME**, **MOVE-ADD**, or **MOVE-DELETE**

FulfillBillingFunction and Fulfillment Patterns

The order component **FulfillBillingFunction** is included in the orchestration plan for the **BaseProductSpec** fulfillment pattern. This ensures that FulfillBillingFunction is included in the fulfillment flow for all fulfillment pattern entities that extend from **BaseProductSpec**. This includes any cartridge specifying the **COM_SalesOrderFulfillment** namespace.

All sample fulfillment patterns include FulfillBillingFunction.

Sample fulfillment patterns that include FulfillBillingFunction in a single-phase billing pattern (without InitiateBillingFunction) in the fulfillment flow are:

- Service.Broadband
- Service.CPE.Broadband
- NonService.BillingItem

FulfillBillingFunction XQuery Modules

Table 4–44 lists the XQuery modules defined for the FulfillBillingFunction fulfillment function. Customers can provide their own implementation of the XQuery modules in this fulfillment function indicated extension points. See "Extending XQuery Modules" for more information about XQuery extension points.

Table 4–44 FulfillBillingFunction XQuery Modules

XQuery Module	XQuery Extension Point	Description
FulfillBillingComplete_Event	COMPONENT-COMPLETE	Invoked when the FulfillBillingFunction component is completed.
FulfillBillingCreateAllOrderItemsEBM_Event	CREATE-EBM-ALL-ORDERITEMS	Invoked after CREATE-EBM-CUSTOM for the FulfillBillingFunction component to create all order items. This should always be invoked unless you want to create only the order header without any order items.
FulfillBillingCreateCustomEBM_Event	CREATE-EBM-CUSTOM	Invoked after CREATE-EBM for the FulfillBillingFunction component. Invokes extension logic on the order-level CUSTOM element.
FulfillBillingCreateEBM_DoEvent	CREATE-EBM (execution mode: *do*)	Invoked when the EBM is created in the do execution mode for the FulfillBillingFunction component.
FulfillBillingCreateEBM_ReDoEvent	CREATE-EBM (execution mode: *redo*)	Invoked when the EBM is created in the redo execution mode for the FulfillBillingFunction component.
FulfillBillingCreateEBM_UnDoEvent	CREATE-EBM (execution mode: *undo*)	Invoked when the EBM is created in the undo execution mode for the FulfillBillingFunction component.
FulfillBillingCreateOrderItemCustomEBM_Event	CREATE-EBM-ORDERITEM-CUSTOM	Invoked after CREATE-EBM-ORDERITEM for the FulfillBillingFunction component. Invokes extension logic on the order-item-level CUSTOM element.
FulfillBillingCreateOrderItemEBM_DoEvent	CREATE-EBM-ORDERITEM (execution mode: *do*)	Invoked after CREATE-EBM-ALL-ORDERITEMS for the FulfillBillingFunction component in **do** execution mode. Invokes extension logic on the order item element.
FulfillBillingCreateOrderItemEBM_ReDoEvent	CREATE-EBM-ORDERITEM (execution mode: *redo*)	Invoked after CREATE-EBM-ALL-ORDERITEMS for the FulfillBillingFunction component in **redo** execution mode. Invokes extension logic on the order item element.
FulfillBillingCreateOrderItemEBM_UnDoEvent	CREATE-EBM-ORDERITEM (execution mode: *undo*)	Invoked after CREATE-EBM-ALL-ORDERITEMS for the FulfillBillingFunction component in **undo** execution mode. Invokes extension logic on the order item element.
FulfillBillingCreatePriorOrderItemCustomEBM_Event	CREATE-EBM-PRIORORDERITEM-CUSTOM	Invoked after CREATE-EBM-PRIORORDERITEM for the FulfillBillingFunction component. Invokes extension logic on the prior-order-item-level CUSTOM element.
FulfillBillingCreatePriorOrderItemEBM_Event	CREATE-EBM-PRIORORDERITEM	Invoked after CREATE-EBM-ORDERITEM-CUSTOM for the FulfillBillingFunction component. Invokes extension logic on the prior order item element.

Table 4–44 (Cont.) FulfillBillingFunction XQuery Modules

XQuery Module	XQuery Extension Point	Description
FulfillBillingStart_Event	COMPONENT-START	Invoked when the FulfillBillingFunction component is started.
FulfillBillingUpdate_Event	COMPONENT-RESPONSE-UPDATE	Invoked when the FulfillBillingFunction updates are received to process order items on the billing response.
FulfillBillingValidateResponseEBM_Event	VALIDATE-RESPONSE-EBM	Invoked to validate the EBM response for the FulfillBillingFunction component.

FulfillBillingFunction Automation Modules

Table 4–45 lists the automation modules (with their associated automated tasks) defined in the OracleComms_OSM_O2A_COM_Billing cartridge for the FulfillBillingFunction fulfillment function.

Table 4–45 FulfillBillingFunction Automation Modules

Automation Module	Automated Task	Description
FulfillBillingEntryPointBean_doredo	FulfillBillingEntryPointTask	Calls the SIEntryPoint XQuery.
FulfillBillingRequestBean_do	FulfillBillingSITask	Calls the AIAEBMRequest_do XQuery.
FulfillBillingRequestBean_redo	FulfillBillingSITask	Calls the AIAEBMRequest_redo XQuery.
FulfillBillingRequestBean_undo	FulfillBillingSITask	Calls the AIAEBMRequest_undo XQuery.
FulfillBillingResponseBean	FulfillBillingSITask	External event receiver to invoke AIAEBMResponse XQuery.
FulfillBillingStart	FulfillBillingStartTask	Calls the SIStartPoint XQuery.

Billing Dates for Billing Patterns

Billing dates are set when usage events start being rated (usage start date time), when cycle charges start being billed (cycle start date time), and when one time purchase charges should be billed (purchase start time). This section discusses how billing dates are generated.

Default Billing Dates for Two-Phase Billing Patterns

In a two-phase billing pattern, the billing dates are calculated based on data from the customer order and the determination of whether the LATENCY or VALIDATION pattern is used.

The billing dates are reset between the two billing phases (InitiateBillingFunction and FulfillBillingFunction).Billing dates are only set for order line items having the ServiceActionCode of **ADD**. The following order item properties are updated with the billing dates:

- For phase 1, order item properties in **ControlData/Functions/IntiateBillingFunction/orderItem/orderItemRef/WorkLineItemData/SalesOrderSchedule**
  - ServiceUsageStartDate
  - PurchaseDate
  - CycleStartDate

Note: The three elements above have a data type of DateTime. However the names of the data elements as defined in the Oracle AIA EBM do not end with DateTime but Date only. O2A is following the names defined in the Oracle AIA EBM.

- For phase 2, the order item properties from phase 1 are copied into properties with the same names in **ControlData/Functions/FulfillBillingFunction/orderItem/orderItemRef/WorkPriorLineItemData/SalesOrderSchedule** and all three Date fields are re-calculated as indicated in the Default Billing Dates for Phase 2 column in Table 4–46.

Table 4–46 lists the default date calculations applicable to two-phase billing patterns.

Table 4–46 Two-Phase Billing Pattern Date Calculations

Billing Pattern	Default Billing Dates for Phase 1	Default Billing Dates for Phase 2

LATENCY	ServiceUsageStartDate= OrderingBaseDateTime
	
	PurchaseDate = OrderingBaseDateTime
	
	CycleStartDate = OrderingBaseDateTime + 1 year
		CycleStartDate = SalesOrderLine/CycleStartDate if populated; otherwise compute as Actual Delivery Date Time or Requested Delivery Date Time, whichever is later.

VALIDATION	ServiceUsageStartDate= OrderingBaseDateTime + 1 year
	
	PurchaseDate=OrderingBaseDateTime + 1 year
	
	CycleStartDate = OrderingBaseDateTime + 1 year
		ServiceUsageStartDate = SalesOrderLine/ ServiceUsageStartDate if populated; otherwise Actual Delivery Date Time
		
		PurchaseDate = SalesOrderLine/PurchaseDate if populated; otherwise Actual Delivery Date Time
		
		CycleStartDate = SalesOrderLine/ CycleStartDate if populated; otherwise Actual Delivery Date Time or Requested Delivery Date Time, whichever is later.

Default Billing Dates for Single-Phase Billing Patterns

In a single-phase billing pattern, the billing dates are calculated based on data from the customer order.

Billing dates are only set for order line items having the ServiceActionCode of **ADD**. The following order item properties are updated with the billing dates:

- Order item properties in **ControlData/Functions/IntiateBillingFunction/orderItem/orderItemRef/WorkLineItemData/SalesOrderSchedule**
  - ServiceUsageStartDate: This is set to SalesOrderLine/ ServiceUsageStartDate if populated; otherwise ActualDeliveryDateTime
  - PurchaseDate: This is set to SalesOrderLine/PurchaseDate if populated; otherwise ActualDeliveryDateTime
  - CycleStartDateTime: This is set to SalesOrderLine/CycleStartDateTime if populated; otherwise Actual Delivery Date Time or Requested Delivery Date Time, whichever is later.

OracleComms_OSM_O2A_COM_Provisioning

The OracleComms_OSM_O2A_COM_Provisioning cartridge is a productized cartridge that supports the provisioning fulfillment functions. These functions specify a subprocess to handle delivery of a relevant subset of order data to the provisioning ABCS and to handle responses from the provisioning ABCS.

Table 4–47 lists the XQuery modules in the cartridge that support component interaction.

Table 4–47 OracleComms_OSM_O2A_COM_Provisioning Component Interaction XQuery Modules

Component Interaction XQuery Module	Extendable	Description
ProvisionOrderInteractionModule	No	Provides functions to support ProvisionOrderFunction when the calculate service order solution option is not being used.
ProvisionOrderInteractionModule_base	No	Provides base functions to support ProvisionOrderFunction when the calculate service order solution option is being used.
ProvisionOrderInteractionModule_do	No	Provides functions to support ProvisionOrderFunction for the do execution mode when the calculate service order solution option is being used.
ProvisionOrderInteractionModule_ events	No	Provides event-related functions to support ProvisionOrderFunction when the calculate service order solution option is being used.
ProvisionOrderInteractionModule_redo	No	Provides functions to support ProvisionOrderFunction for the redo execution mode when the calculate service order solution option is being used.
ProvisionOrderInteractionModule_undo	No	Provides functions to support ProvisionOrderFunction for the undo execution mode when the calculate service order solution option is being used.

Table 4–48 lists the XQuery modules defined for the ProvisionOrderFunction fulfillment function. Customers can provide their own implementation of the XQuery modules in this fulfillment function indicated extension points. See "Extending XQuery Modules" for more information about XQuery extension points.

Table 4–48 ProvisionOrderFunction XQuery Modules

XQuery Module	XQuery Extension Point	Description
ProvisionOrderComplete_Event	COMPONENT-COMPLETE	Invoked when the ProvisionOrderFunction component is completed.
ProvisionOrderCreateEBM_ DoEvent	CREATE-EBM (execution mode: do)	Invoked when the EBM is created in the **do** execution mode for the ProvisionOrderFunction component.
ProvisionOrderCreateEBM_ ReDoEvent	CREATE-EBM (execution mode: redo)	Invoked when the EBM is created in the **redo** execution mode for the ProvisionOrderFunction component.
ProvisionOrderCreateEBM_ UnDoEvent	CREATE-EBM (execution mode: undo)	Invoked when the EBM is created in the **undo** execution mode for the ProvisionOrderFunction component.
ProvisionOrderStart_Event	COMPONENT-START	Invoked when the ProvisionOrderFunction component is started.
ProvisionOrderValidateRespons eEBM_Event	VALIDATE-RESPONSE-EBM	Invoked to validate the EBM response for the ProvisionOrderFunction component.
ProvisionOrderUpdate_Event	COMPONENT-RESPONSE-U PDATE	Invoked when the EBM response for ProvisionOrderFunction component is updated.

Table 4–49 lists the automation modules (with their associated automated tasks) defined in the OracleComms_OSM_O2A_COM_Provisioning cartridge for the ProvisionOrderFunction fulfillment function.

Table 4–49 ProvisionOrderFunction Automation Modules

Automation Module	Automated Task	Description
ProvisionOrderEntryPointBean_doredo	ProvisionOrderSIEntryPointTask	Calls the SIEntryPoint XQuery.
ProvisionOrderRequestBean_do	ProvisionOrderSITask	Calls the AIAEBMRequest_do XQuery.
ProvisionOrderRequestBean_redo	ProvisionOrderSITask	Calls the AIAEBMRequest_redo XQuery.
ProvisionOrderRequestBean_undo	ProvisionOrderSITask	Calls the AIAEBMRequest_undo XQuery.
ProvisionOrderResponseBean	ProvisionOrderSITask	External event receiver to invoke AIAEBMResponse XQuery.

OracleComms_OSM_O2A_COM_Shipping_Sample

The OracleComms_OSM_O2A_COM_Shipping_Sample cartridge is a demonstration cartridge that supports the shipping fulfillment functions. These functions specify subprocesses to handle delivery of a relevant subset of order data to supply chain management.

Table 4–50 lists he XQuery modules in the cartridge that support component interaction.

Table 4–50 OracleComms_OSM_O2A_COM_Shipping_Sample Component Interaction XQuery Modules

Component Interaction XQuery Module	Extendable	Description
ShipOrderInteractionModule	Yes	Provides functions to support ShipOrderFunction.

Table 4–51 lists the XQuery modules defined for the ShipOrderFunction fulfillment function. Customers can provide their own implementation of the XQuery modules in this fulfillment function indicated extension points. See "Extending XQuery Modules" for more information about XQuery extension points.

Table 4–51 ShipOrderFunction XQuery Modules

XQuery Module	XQuery Extension Point	Description
ShipOrderComplete_Event	COMPONENT-COMPLETE	Invoked when the ShipOrderFunction component is completed.
ShipOrderStart_Event	COMPONENT-START	Invoked when the ShipOrderFunction component is started.

Table 4–52 lists the automation modules (with their associated automated tasks) defined in the OracleComms_OSM_O2A_COM_Shipping_Sample cartridge for the ShipOrderFunction fulfillment function.

Table 4–52 ShipOrderFunction Automation Modules

Automation Module	Automated Task	Description
ShipOrderEntryPointBean	ShipOrderEntryPointTask	Calls the SIEntryPoint XQuery.
ShipOrderPlannedBean	ShipOrderPlannedTask	Calls the SIMilestone_doredo XQuery to return the SHIP ORDER PLANNED milestone.
ShipOrderExitPointBean	ShipOrderExitPointTask	Calls the SIExitPoint XQuery.

OracleComms_OSM_O2A_COM_Install_Sample

The OracleComms_OSM_O2A_COM_Install_Sample cartridge is a demonstration cartridge that supports the installation fulfillment functions for High-Speed Internet. These functions specify subprocesses to handle delivery of a relevant subset of order data to supply chain management.

Table 4–53 lists he XQuery modules in the cartridge that support component interaction.

Table 4–53 OracleComms_OSM_O2A_COM_Install_Sample Component Interaction XQuery Modules

Component Interaction XQuery Module	Extendable	Description
InstallOrderInteractionModule	Yes	Provides functions to support InstallOrderFunction.

Table 4–54 lists the XQuery modules defined for the InstallOrderFunction fulfillment function. Customers can provide their own implementation of the XQuery modules in this fulfillment function indicated extension points. See "Extending XQuery Modules" for more information about XQuery extension points.

Table 4–54 InstallOrderFunction XQuery Modules

XQuery Module	XQuery Extension Point	Description
InstallOrderComplete_Event	COMPONENT-COMPLETE	Invoked when the InstallOrderFunction component is completed.
InstallOrderStart_Event	COMPONENT-START	Invoked when the InstallOrderFunction component is started.

Table 4–55 lists the automation modules (with their associated automated tasks) defined in the OracleComms_OSM_O2A_COM_Install_Sample cartridge for the InstallOrderFunction fulfillment function.

Table 4–55 InstallOrderFunction Automation Modules

Automation Module	Automated Task	Description
InstallOrderEntryPointBean	InstallOrderEntryPointTask	Calls the SIEntryPoint XQuery.
InstallOrderPlannedBean	InstallOrderPlannedTask	Calls the SIMilestone_doredo XQuery to return the INSTALL PLANNED milestone.
InstallOrderCommittedBean	InstallOrderCommittedTask	Calls the SIMilestone_doredo XQuery to return the INSTALL COMMITTED milestone.
InstallOrderExitPointBean	InstallOrderExitPointTask	Calls the SIExitPoint XQuery.

OracleComms_OSM_O2A_COM_Recognition_Sample

The OracleComms_OSM_O2A_COM_Recognition_Sample cartridge is a demonstration cartridge that recognizes a customer order from Oracle AIA and triggers the creation of a COM_SalesOrderFulfillment order. In addition, this cartridge recognizes order fallout notifications, trouble ticket requests, and inbound message errors due to suspended orders or resource issues. It also catches all unrecognizable messages.

Table 4–56 lists the order recognition rules defined in this cartridge.

Table 4–56 OracleComms_OSM_O2A_COM_Recognition_Sample Order Recognition Rules

Order Recognition Rule	Description
COM_CloseTroubleTicketWorkOrder_Recognition	Recognizes a request to close an order as a result of a trouble ticket.
COM_FaultNotificationOrder_Recognition	Recognizes order fallout notifications from Oracle AIA.
COM_ORPFallout_CTT_OrderRecognitionRule	Recognizes an ORP fallout and triggers creation of a fulfillment request for a trouble ticketing system.
COM_ResumePendingIbMsg_ OrderRecognitionRule	Recognizes an inbound message and triggers creation of a ResumePendingInBoundMessage order.
COM_SalesOrderFulfillment_Recognition	Recognizes an Oracle AIA customer order and triggers the creation of a COM_SalesOrderFulfillment order.

Revision Number Update for Canceled Orders

When AIA receives an order that has a mode of CANCEL, or when an order is received in which all of the order line items have an action code of NONE, AIA does not update the revision number. Because OSM will ignore a revision order if its revision number is the same as on a previously received order revision, the OTA recognition cartridges update the revision number on orders of this type to 999999999. This ensures that the cancelation is processed.

Central Order Management Cartridges for the Calculate Service Order Solution Option

The following cartridges operate in the central order management role, which coordinates fulfillment functions across the Business Support Systems (BSS) and Operations Support Systems (OSS) such as Siebel CRM, BRM, and provisioning. These cartridges are used in the calculate service order solution option.

OracleComms_OSM_O2A_COM_CSO_Base

The OracleComms_OSM_O2A_COM_CSO_Base cartridge is a productized cartridge. It contains entities that support the orchestration of orders for the calculate service order solution option. It includes the following entities:

- Order Item Specification: **COM_TransformedServiceLine**: This order item specification defines the order item information for transformed order items.
- Data Schema: **OracleComms_OSM_O2A_COM_CSO_Data**: This data schema contains elements relating to transformed order items.
- Orchestration Process: **COM_SalesOrderFulfillmentOrchestrationProcess**: This orchestration process invokes the order transformation manager.

- Transformation Sequence: **COM_OTM_Sequence**: This orchestration sequence has four stages. For more information, see the discussion of transformation sequences in *OSM Concepts*.

OracleComms_OSM_O2A_COM_CSO_Broadband_Internet_Access_CFS

The OracleComms_OSM_O2A_COM_CSO_Broadband_Internet_Access_CFS cartridge is a demonstration cartridge that contains the mapping rules and order item parameter bindings associated with the customer facing service for broadband internet access. It also contains XQuery modules to support the order item parameter bindings.

OracleComms_OSM_O2A_COM_CSO_Email_CFS

The OracleComms_OSM_O2A_COM_CSO_Email_CFS cartridge is a demonstration cartridge that contains the mapping rules and order item parameter bindings associated with the customer facing service for email service. It also contains XQuery modules to support the order item parameter bindings.

OracleComms_OSM_O2A_COM_CSO_FulfillmentPattern

The OracleComms_OSM_O2A_COM_CSO_FulfillmentPattern cartridge is a productized cartridge that contains fulfillment patterns and orchestration dependencies for the calculate service order solution option.

OracleComms_OSM_O2A_COM_CSO_FulfillmentStateMap

The OracleComms_OSM_O2A_COM_CSO_FulfillmentStateMap cartridge is a productized cartridge. It fulfillment state maps and transformed order item fulfillment state composition rule sets specific to the calculate service order solution option.

OracleComms_OSM_O2A_COM_CSO_Internet_Media_CFS

The OracleComms_OSM_O2A_COM_CSO_Internet_Media_CFS cartridge is a demonstration cartridge that contains the mapping rules and order item parameter bindings associated with the customer facing service for Internet media service. It also contains XQuery modules to support the order item parameter bindings.

OracleComms_OSM_O2A_COM_CSO_IP_Fax_CFS

The OracleComms_OSM_O2A_COM_CSO_IP_Fax_CFS cartridge is a demonstration cartridge that contains the mapping rules and order item parameter bindings associated with the customer facing service for IP fax service. It also contains XQuery modules to support the order item parameter bindings.

OracleComms_OSM_O2A_COM_CSO_Model_Container

The OracleComms_OSM_O2A_COM_CSO_Model_Container cartridge is a demonstration cartridge. It defines the common model projects that contain elements that might need to be included in the deployment and contains the transformation manager for the calculate service order solution option.

To see the common model projects that are contained by this cartridge, open the **Properties** tab of the cartridge editor. For more information about the common model projects included with the Order-to-Activate cartridges, see "Conceptual Model Projects."

OracleComms_OSM_O2A_COM_CSO_Provisioning

The OracleComms_OSM_O2A_COM_CSO_Provisioning cartridge is a demonstration cartridge. It contains order components for provisioning that are specific to the calculate service order solution option.

External Fulfillment States

External fulfillment states in the OracleComms_OSM_O2A_COM_CSO_Provisioning cartridge are defined in the **ProvisioningOrderFunction** order component specification for use withy the calculate service order solution option.

The following external fulfillment states are defined in this cartridge:

- IN_PROGRESS-BROADBANDINTERNETDOMAIN_PROVISION_DESIGNED
- IN_PROGRESS-BROADBANDINTERNETDOMAIN_PROVISION_ISSUED
- IN_PROGRESS-VOIPDOMAIN_PROVISION_DESIGNED
- IN_PROGRESS-VOIPDOMAIN_PROVISION_ISSUED
- FAILED-VOIPDOMAIN_PROVISION_FAILED
- FAILED-BROADBANDINTERNETDOMAIN_PROVISION_FAILED
- IN_PROGRESS-BROADBANDINTERNETDOMAIN_PROVISION_START
- IN_PROGRESS-VOIPDOMAIN_PROVISION_START
- CANCELLED-VOIPDOMAIN_PROVISION_DESIGNED
- CANCELLED-VOIPDOMAIN_PROVISION_START
- CANCELLED-VOIPDOMAIN_PROVISION_ISSUED
- CANCELLED-BROADBANDINTERNETDOMAIN_PROVISION_DESIGNED
- CANCELLED-BROADBANDINTERNETDOMAIN_PROVISION_START
- CANCELLED-BROADBANDINTERNETDOMAIN_PROVISION_ISSUED
- COMPLETE-VOIPDOMAIN_PROVISION_COMPLETE
- COMPLETE-BROADBANDINTERNETDOMAIN_PROVISION_COMPLETE
- FAILED-BROADBANDINTERNETDOMAIN_PROVISION_AUXILIARY_FAILED
- FAILED-VOIPDOMAIN_PROVISION_AUXILIARY_FAILED
- IN_PROGRESS-INTERNETMEDIADOMAIN_PROVISION_START
- IN_PROGRESS-INTERNETMEDIADOMAIN_PROVISION_DESIGNED
- IN_PROGRESS-INTERNETMEDIADOMAIN_PROVISION_ISSUED
- IN_PROGRESS-EMAILDOMAIN_PROVISION_START
- IN_PROGRESS-EMAILDOMAIN_PROVISION_DESIGNED
- IN_PROGRESS-EMAILDOMAIN_PROVISION_ISSUED
- CANCELLED-EMAILDOMAIN_PROVISION_DESIGNED
- CANCELLED-EMAILDOMAIN_PROVISION_START
- CANCELLED-EMAILDOMAIN_PROVISION_ISSUED
- CANCELLED-INTERNETMEDIADOMAIN_PROVISION_DESIGNED
- CANCELLED-INTERNETMEDIADOMAIN_PROVISION_START

- CANCELLED-INTERNETMEDIADOMAIN_PROVISION_ISSUED
- COMPLETE-EMAILDOMAIN_PROVISION_COMPLETE
- COMPLETE-INTERNETMEDIADOMAIN_PROVISION_COMPLETE
- FAILED-EMAILDOMAIN_PROVISION_FAILED
- FAILED-EMAILDOMAIN_PROVISION_AUXILIARY_FAILED
- FAILED-INTERNETMEDIADOMAIN_PROVISION_FAILED
- FAILED-INTERNETMEDIADOMAIN_PROVISION_AUXILIARY_FAILED

OracleComms_OSM_O2A_COM_CSO_SalesOrders

The OracleComms_OSM_O2A_COM_CSO_SalesOrders cartridge is a demonstration cartridge that contains sample customer orders for use with the calculate service order solution option. These orders are in the same format as orders that are received in an integrated environment with Oracle AIA, Siebel CRM, and CRM ABCS. In a standalone OSM environment, you can submit them to central order management to generate and execute an orchestration plan. In a standalone OSM environment, EBMs are placed on OSM JMS queues for pickup by Oracle AIA.

Note: Each customer order that you send must contain a unique EBM ID. For example, the EBM ID of a cancel order request (revision order) cannot be the same as the EBM ID of the original base order (new order).

Table 4–57 describes the order numbers for new and change orders:

Table 4–57 OracleComms_OSM_O2A_COM_CSO_SalesOrders Order Descriptions

Order ID	Order XML File	Description
testcso-doubleplay-voip-broadband	testcso-all.xml	Adds all domain services (Broadband, VoIP, Email and Internet Media) and creates all CFS services lines.
testcso-bandwidth	testcso-bandwidth.xml	Add primary broadband line with auxiliary line that defines upload/download bandwidth.
testcso-broadband	testcso-broadband.xml	Add primary basic broadband line only.
testcso-cme-voip	testcso-cme-voip_base.xml	Add VoIP feature services.
testcso-cme-voip	testcso-cme-voip_revision.xml	Revision to add extra VoIP services
testcso-doubleplay-voip-delete	testcso-doubleplay-voip-broadband.xml	Add Broadband and VoIP services.
testcso-email	testcso-email.xml	Add Email service.
testcso-firewall	testcso-firewall.xml	Add Firewall service.
TestCSO_InternetMedia_1	testcso-internetmedia.xml	Add Internet Media service.
TestCSO_IP_FAX	testcso-ip-fax.xml	Add IP Fax service.
testcso-modem	testcso-modem.xml	Add Broadband Modem service.
testcso-modem	testcso-modem-cancel.xml	Cancel Broadband Modem service.
testcso-router	testcso-router.xml	Add Broadband Router service.

Table 4–57 (Cont.) OracleComms_OSM_O2A_COM_CSO_SalesOrders Order Descriptions

Order ID	Order XML File	Description
testcso-voip-callerid	testcso-voip-callerid.xml	Add VoIP Caller ID service via Value Added Services.
testcso-voip	testcso-voip.xml	Add VoIP Services.
testcso-web-conference	testcso-web-conference.xml	Add Web Conference service.

OracleComms_OSM_O2A_COM_CSO_Solution

The OracleComms_OSM_O2A_COM_CSO_Solution cartridge is a demonstration composite cartridge that references all cartridges required for central order management in the topology you selected when installing the Order-to-Activate cartridges.

To see the component cartridges referenced in this cartridge for your solution, open the **Dependencies** tab in the composite cartridge editor.

OracleComms_OSM_O2A_COM_CSO_Topology

The OracleComms_OSM_O2A_COM_CSO_Topology cartridge is a demonstration cartridge containing decomposition rules and order component specifications to decompose billing, provisioning, shipping, and install fulfillment functions into the topology you selected when installing the Order-to-Activate cartridges.

Table 4–58 contains a list of the order component specifications defined in this cartridge.

Table 4–58 OracleComms_OSM_O2A_COM_CSO_Topology Order Component Specifications

Order Component Specification	Description
BRM-ALL	Represents the billing fulfillment system if there is a single billing system.
BRM-BIZBDB	Represents the billing fulfillment system for business broadband customers.
BRM-REZBDB	Represents the billing fulfillment system for residential broadband customers.
BRM-VIRTUAL	Represents the billing fulfillment system for non-service billing functions.
BRM-VoIP	Represents the billing fulfillment system for VoIP customers.
Provisioning-ALL	Represents the provisioning fulfillment system if there is a single provisioning system.
Provisioning-Broadband	Represents the provisioning system for broadband customers.
Provisioning-VoIP	Represents the provisioning system for VoIP customers.
Shipping-ALL	Represents the shipping fulfillment system if there is a single shipping system.
Shipping-InHouse	Represents the shipping system for shipments without partner involvement.
Shipping-PartnerInc	Represents the shipping system for shipments with partner involvement.
WFM-A	Represents the first workflow management fulfillment system if there is more than one workflow management system.
WFM-ALL	Represents the workflow management fulfillment system if there is a single workflow management system.
WFM-B	Represents the second workflow management fulfillment system if there is more than one workflow management system.

OracleComms_OSM_O2A_COM_CSO_VoIP_Access_CFS

The OracleComms_OSM_O2A_COM_CSO_VoIP_Access_CFS cartridge is a demonstration cartridge that contains the mapping rules and order item parameter bindings associated with the customer facing service for VoIP access. It also contains XQuery modules to support the order item parameter bindings.

OracleComms_OSM_O2A_COM_CSO_Web_Conferencing_CFS

The OracleComms_OSM_O2A_COM_CSO_Web_Conferencing_CFS cartridge is a demonstration cartridge that contains the mapping rules and order item parameter bindings associated with the customer facing service for web conferencing service. It also contains XQuery modules to support the order item parameter bindings.

OracleComms_OSM_O2A_COM_FulfillmentPattern

The OracleComms_OSM_O2A_COM_FulfillmentPattern cartridge is a productized cartridge. It contains the base fulfillment pattern, **BaseProductSpec**, from which other fulfillment patterns can inherit.

OracleComms_OSM_O2A_COM_FulfillmentStateMap_Sample

The OracleComms_OSM_O2A_COM_FulfillmentStateMap_Sample cartridge is a productized cartridge that contains fulfillment state entities used by the solution.

This cartridge contains the following common fulfillment state definitions, which are used in composition rules. Listed under each main fulfillment state are its child states.

- OPEN
- IN_PROGRESS
  - IN_PROGRESS-FULFILL_BILLING_START
  - IN_PROGRESS-INITIATE_BILLING_START
  - IN_PROGRESS-INSTALL_COMMITTED
  - IN_PROGRESS-INSTALL_PLANNED
  - IN_PROGRESS-INSTALL_START
  - IN_PROGRESS-PROVISION_DESIGNED
  - IN_PROGRESS-PROVISION_ISSUED
  - IN_PROGRESS-PROVISION_START
  - IN_PROGRESS-SHIP_ORDER_PLANNED
  - IN_PROGRESS-SHIP_ORDER_START
  - IN_PROGRESS-SYNC_CUSTOMER_START
- COMPLETE
  - COMPLETE-FULFILL_BILLING_COMPLETE
  - COMPLETE-FULFILL_BILLING_START
  - COMPLETE-INITIATE_BILLING_COMPLETE
  - COMPLETE-INITIATE_BILLING_START
  - COMPLETE-INSTALL_COMPLETE

  - COMPLETE-INSTALL_START
  - COMPLETE-PROVISION_COMPLETE
  - COMPLETE-PROVISION_DESIGNED
  - COMPLETE-PROVISION_ISSUED
  - COMPLETE-PROVISION_START
  - COMPLETE-SHIP_ORDER_SHIPPED
  - COMPLETE-SHIP_ORDER_START
  - COMPLETE-SYNC_CUSTOMER_COMPLETE
  - COMPLETE-SYNC_CUSTOMER_START
- CANCELLED
  - CANCELLED-FULFILL_BILLING_COMPLETE
  - CANCELLED-FULFILL_BILLING_START
  - CANCELLED-INITIATE_BILLING_COMPLETE
  - CANCELLED-INITIATE_BILLING_START
  - CANCELLED-INSTALL_COMMITTED
  - CANCELLED-INSTALL_COMPLETE
  - CANCELLED-INSTALL_PLANNED
  - CANCELLED-INSTALL_START
  - CANCELLED-PROVISION_COMPLETE
  - CANCELLED-PROVISION_DESIGNED
  - CANCELLED-PROVISION_ISSUED
  - CANCELLED-PROVISION_START
  - CANCELLED-SHIP_ORDER_PLANNED
  - CANCELLED-SHIP_ORDER_SHIPPED
  - CANCELLED-SHIP_ORDER_START
  - CANCELLED-SYNC_CUSTOMER_COMPLETE
  - CANCELLED-SYNC_CUSTOMER_START
- FAILED
  - FAILED-FULFILL_BILLING_COMPLETE
  - FAILED-FULFILL_BILLING_START
  - FAILED-INITIATE_BILLING_COMPLETE
  - FAILED-INITIATE_BILLING_START
  - FAILED-INSTALL_COMMITTED
  - FAILED-INSTALL_COMPLETE
  - FAILED-INSTALL_PLANNED
  - FAILED-INSTALL_START
  - FAILED-PROVISION_AUXILIARY

- FAILED-PROVISION_COMPLETE
- FAILED-PROVISION_DESIGNED
- FAILED-PROVISION_FAILED
- FAILED-PROVISION_ISSUED
- FAILED-PROVISION_START
- FAILED-SHIP_ORDER_PLANNED
- FAILED-SHIP_ORDER_SHIPPED
- FAILED-SHIP_ORDER_START
- FAILED-SYNC_CUSTOMER_COMPLETE
- FAILED-SYNC_CUSTOMER_START

OracleComms_OSM_O2A_COMSOM_CSO_Recognition

The OracleComms_OSM_O2A_COMSOM_CSO_Recognition cartridge is a demonstration cartridge that recognizes a customer order from Oracle AIA and triggers the creation of a COM_SalesOrderFulfillment order. In addition, this cartridge recognizes order fallout notifications, trouble ticket requests, and inbound message errors due to suspended orders or resource issues. It also catches all unrecognizable messages.

Table 4–59 lists the order recognition rules defined in this cartridge.

Table 4–59 OracleComms_OSM_O2A_COMSOM_CSO_Recognition Order Recognition Rules

Order Recognition Rule	Description
COM_CloseTroubleTicketWorkOrder_Recognition	Recognizes a request to close an order as a result of a trouble ticket.
COM_FaultNotificationOrder_Recognition	Recognizes order fallout notifications from Oracle AIA.
COM_ORPFallout_CTT_OrderRecognitionRule	Recognizes an ORP fallout and triggers creation of a fulfillment request for a trouble ticketing system.
COMSOM_CSO_Recognize_ AbortPropagationServiceOrder	Recognizes a termination request for an order.
COMSOM_CSO_RecognizeEBM_ ProvisioningOrder	Recognizes and EBM provisioning order.
COM_ResumePendingIbMsg_ OrderRecognitionRule	Recognizes an inbound message and triggers creation of a ResumePendingInBoundMessage order.
COM_SalesOrderFulfillment_Recognition	Recognizes an Oracle AIA customer order and triggers the creation of a COM_SalesOrderFulfillment order.

Revision Number Update for Canceled Orders

When AIA receives an order that has a mode of CANCEL, or when an order is received in which all of the order line items have an action code of NONE, AIA does not update the revision number. Because OSM will ignore a revision order if its revision number is the same as on a previously received order revision, the OTA recognition cartridges update the revision number on orders of this type to 999999999. This ensures that the cancelation is processed.

OracleComms_OSM_O2A_COMSOM_CSO_Solution

The OracleComms_OSM_O2A_COMSOM_CSO_Solution cartridge is a demonstration composite cartridge that references all cartridges required for central order management and service order management in the topology you selected when installing the Order-to-Activate cartridges.

To see the component cartridges referenced in this cartridge for your solution, open the **Dependencies** tab in the composite cartridge editor.

Central Order Management Cartridges for the Solution Option Without Calculate Service Order

The following cartridges operate in the central order management role, which coordinates fulfillment functions across the Business Support Systems (BSS) and Operations Support Systems (OSS) such as Siebel CRM, BRM, and provisioning. These cartridges are used in the solution option without calculate service order.

OracleComms_OSM_O2A_BBVoIP_FP_NP_Danube_Sample

The OracleComms_OSM_O2A_BBVoIP_FP_NP_Danube_Sample cartridge contains fulfillment patterns and orchestration dependencies for the Simple topology.

The following fulfillment patterns are configured in this cartridge:

- BaseProductSpec – All other fulfillment patterns extend from this.
- NonService.BillingInitiatedItem
- NonService.BillingItem
- NonService.Offer
- Service.Broadband
- Service.CPE.Broadband
- Service.CPE.VoIP
- Service.Install
- Service.VoIP

OracleComms_OSM_O2A_BBVoIP_FP_NP_Nile_Sample

The OracleComms_OSM_O2A_BBVoIP_FP_NP_Nile_Sample cartridge contains fulfillment patterns and orchestration dependencies for the Typical or Complex topologies.

The following fulfillment patterns are configured in this cartridge:

- BaseProductSpec – All other fulfillment patterns extend from this.
- NonService.BillingInitiatedItem
- NonService.BillingItem
- NonService.Offer
- Service.Broadband
- Service.CPE.Broadband
- Service.CPE.VoIP

- Service.Install
- Service.VoIP

OracleComms_OSM_O2A_COM_NCSO_Base

The OracleComms_OSM_O2A_COM_NCSO_Base cartridge is a productized cartridge. It contains the orchestration process, **COM_ SalesOrderFulfillmentOrchestrationProcess**, that supports the orchestration of orders for the solution option without calculate service order.

OracleComms_OSM_O2A_COM_NCSO_Provisioning

The OracleComms_OSM_O2A_COM_NCSO_Provisioning cartridge is a demonstration cartridge. It contains an order component for provisioning, **ProvisionOrderFunction**, that is specific to the service option without calculate service order.

Table 4–60 lists the XQuery modules defined for the ProvisionOrderFunction fulfillment function. Customers can provide their own implementation of the XQuery modules in this fulfillment function indicated extension points. See "Extending XQuery Modules" for more information about XQuery extension points.

Table 4–60 ProvisionOrderFunction XQuery Modules

XQuery Module	XQuery Extension Point	Description
ProvisionOrderComplete_Event	COMPONENT-COMPLETE	Invoked when the ProvisionOrderFunction component is completed.
ProvisionOrderCreateEBM_ DoEvent	CREATE-EBM (execution mode: do)	Invoked when the EBM is created in the **do** execution mode for the ProvisionOrderFunction component.
ProvisionOrderCreateEBM_ ReDoEvent	CREATE-EBM (execution mode: redo)	Invoked when the EBM is created in the **redo** execution mode for the ProvisionOrderFunction component.
ProvisionOrderCreateEBM_ UnDoEvent	CREATE-EBM (execution mode: undo)	Invoked when the EBM is created in the **undo** execution mode for the ProvisionOrderFunction component.
ProvisionOrderStart_Event	COMPONENT-START	Invoked when the ProvisionOrderFunction component is started.
ProvisionOrderValidateRespons eEBM_Event	VALIDATE-RESPONSE-EBM	Invoked to validate the EBM response for the ProvisionOrderFunction component.
ProvisionOrderUpdate_Event	COMPONENT-RESPONSE-U PDATE	Invoked when the EBM response for ProvisionOrderFunction component is updated.

Table 4–61 lists the automation modules (with their associated automated tasks) defined in the OracleComms_OSM_O2A_COM_Provisioning cartridge for the ProvisionOrderFunction fulfillment function.

Table 4–61 ProvisionOrderFunction Automation Modules

Automation Module	Automated Task	Description
ProvisionOrderEntryPointBean_doredo	ProvisionOrderSIEntryPointTask	Calls the SIEntryPoint XQuery.
ProvisionOrderRequestBean_do	ProvisionOrderSITask	Calls the AIAEBMRequest_do XQuery.
ProvisionOrderRequestBean_redo	ProvisionOrderSITask	Calls the AIAEBMRequest_redo XQuery.
ProvisionOrderRequestBean_undo	ProvisionOrderSITask	Calls the AIAEBMRequest_undo XQuery.
ProvisionOrderResponseBean	ProvisionOrderSITask	External event receiver to invoke AIAEBMResponse XQuery.
InitiateWaitforProvisioningResponseBean	ProvisionOrderSIResponseTask	Calls the InitiateWaitforProvisioningResponse XQuery.
ProvisioningResponseBean	ProvisionOrderSIResponseTask	External event receiver to invoke AIAEBMResponse XQuery for do, redo and amend-do mode.

OracleComms_OSM_O2A_COM_Simple_NP_Soln

The OracleComms_OSM_O2A_COM_Simple_NP_Soln cartridge is a demonstration composite cartridge that references all cartridges required for central order management in the Simple topology.

To see the component cartridges referenced in this cartridge for your solution, open the **Dependencies** tab in the composite cartridge editor.

OracleComms_OSM_O2A_COM_Typical_NP_Soln

The OracleComms_OSM_O2A_COM_Typical_NP_Soln cartridge is a demonstration composite cartridge that references all cartridges required for central order management in the Typical or Complex topologies.

The OracleComms_OSM_O2A_COM_Simple_NP_Soln cartridge is a demonstration composite cartridge that references all cartridges required for central order management in the Simple topology.

OracleComms_OSM_O2A_COMSOM_Recognition_Sample

The OracleComms_OSM_O2A_COMSOM_Recognition_Sample cartridge is a demonstration cartridge that is used when central order management and service order management are deployed together in the same OSM instance. It recognizes a customer order from Oracle AIA and triggers the creation of a COM_SalesOrderFulfillment order. In addition, this cartridge recognizes order fallout notifications, trouble ticket requests, and inbound message errors due to suspended orders or resource issues. It also catches all unrecognizable messages.

Table 4–62 lists the order recognition rules defined in this cartridge.

Table 4–62 OracleComms_OSM_O2A_COMSOM_Recognition_Sample Order Recognition Rules

Order Recognition Rule	Description
COMSOM_CloseTroubleTicketWorkOrder	Recognizes a request to close an order as a result of a trouble ticket.
COMSOM_FaultNotificationOrder	Recognizes order fallout notifications from Oracle AIA.
COMSOM_LFAbortOrderPropagationOrder	Recognizes a termination request for an order.
COMSOM_ORPFallout_CTT_Order	Recognizes an ORP fallout and triggers creation of a fulfillment request for a trouble ticketing system.
COMSOM_ProvisionOrderFulfillment	Recognizes a service order that must be executed and creates a SOM_ProvisionOrderFulfillment order.
COMSOM_ResumePendingIbMsg	Recognizes an inbound message and triggers creation of a ResumePendingInBoundMessage order.
COMSOM_SalesOrderFulfillment	Recognizes an Oracle AIA customer order and triggers the creation of a COM_SalesOrderFulfillment order.

Revision Number Update for Canceled Orders

For information about special revision number processing for canceled orders, see "Revision Number Update for Canceled Orders."

OracleComms_OSM_O2A_COMSOM_Simple_NP_Soln

The OracleComms_OSM_O2A_COMSOM_Simple_NP_Soln cartridge is a demonstration composite cartridge that references all cartridges required for central order management and service order management in the Simple topology.

To see the component cartridges referenced in this cartridge for your solution, open the **Dependencies** tab in the composite cartridge editor.

OracleComms_OSM_O2A_COMSOM_Typical_NP_Soln

The OracleComms_OSM_O2A_COMSOM_Typical_NP_Soln cartridge is a demonstration composite cartridge that references all cartridges required for central order management and service order management in the Typical or Complex topology

To see the component cartridges referenced in this cartridge for your solution, open the **Dependencies** tab in the composite cartridge editor.

OracleComms_OSM_O2A_FulfillmentPatternMap_Sample

The OracleComms_OSM_O2A_FulfillmentPatternMap_Sample cartridge is a demonstration cartridge. It contains the mappings between product specifications and fulfillment patterns, where the product specifications are either imported from customer's Siebel CRM system or manually created. In either case, this cartridge can be extended and can contain custom product specification information.

This cartridge also contains the following common fulfillment state definitions, which are used in composition rules. Listed under each main fulfillment state are its child states.

- OPEN
- IN_PROGRESS
  - IN_PROGRESS-FULFILL_BILLING_START

  - IN_PROGRESS-INITIATE_BILLING_START
  - IN_PROGRESS-INSTALL_COMMITTED
  - IN_PROGRESS-INSTALL_PLANNED
  - IN_PROGRESS-INSTALL_START
  - IN_PROGRESS-PROVISION_DESIGNED
  - IN_PROGRESS-PROVISION_START
  - IN_PROGRESS-SHIP_ORDER_PLANNED
  - IN_PROGRESS-SHIP_ORDER_START
  - IN_PROGRESS-SYNC_CUSTOMER_START
- COMPLETE
  - COMPLETE-FULFILL_BILLING_COMPLETE
  - COMPLETE-FULFILL_BILLING_START
  - COMPLETE-INITIATE_BILLING_COMPLETE
  - COMPLETE-INITIATE_BILLING_START
  - COMPLETE-INSTALL_COMPLETE
  - COMPLETE-INSTALL_START
  - COMPLETE-PROVISION_COMPLETE
  - COMPLETE-PROVISION_START
  - COMPLETE-SHIP_ORDER_SHIPPED
  - COMPLETE-SHIP_ORDER_START
  - COMPLETE-SYNC_CUSTOMER_COMPLETE
  - COMPLETE-SYNC_CUSTOMER_START
- CANCELLED
  - CANCELLED-FULFILL_BILLING_COMPLETE
  - CANCELLED-FULFILL_BILLING_START
  - CANCELLED-INITIATE_BILLING_COMPLETE
  - CANCELLED-INITIATE_BILLING_START
  - CANCELLED-INSTALL_COMMITTED
  - CANCELLED-INSTALL_COMPLETE
  - CANCELLED-INSTALL_PLANNED
  - CANCELLED-INSTALL_START
  - CANCELLED-PROVISION_COMPLETE
  - CANCELLED-PROVISION_DESIGNED
  - CANCELLED-PROVISION_START
  - CANCELLED-SHIP_ORDER_PLANNED
  - CANCELLED-SHIP_ORDER_SHIPPED
  - CANCELLED-SHIP_ORDER_START

  - CANCELLED-SYNC_CUSTOMER_COMPLETE
  - CANCELLED-SYNC_CUSTOMER_START
- FAILED
  - FAILED-FULFILL_BILLING_COMPLETE
  - FAILED-FULFILL_BILLING_START
  - FAILED-INITIATE_BILLING_COMPLETE
  - FAILED-INITIATE_BILLING_START
  - FAILED-INSTALL_COMMITTED
  - FAILED-INSTALL_COMPLETE
  - FAILED-INSTALL_PLANNED
  - FAILED-INSTALL_START
  - FAILED-PROVISION_COMPLETE
  - FAILED-PROVISION_DESIGNED
  - FAILED-PROVISION_START
  - FAILED-SHIP_ORDER_PLANNED
  - FAILED-SHIP_ORDER_SHIPPED
  - FAILED-SHIP_ORDER_START
  - FAILED-SYNC_CUSTOMER_COMPLETE
  - FAILED-SYNC_CUSTOMER_START

OracleComms_OSM_O2A_SalesOrders_NP_Sample

The OracleComms_OSM_O2A_SalesOrders_NP_Sample cartridge is a demonstration cartridge that contains sample customer orders. These orders are in the same format as orders that are received in an integrated environment with Oracle AIA, Siebel CRM, and CRM ABCS. In a standalone OSM environment, you can submit them to central order management to generate and execute an orchestration plan. In a standalone OSM environment, EBMs are placed on OSM JMS queues for pickup by Oracle AIA.

This cartridge contains sample orders in XML files. The names of the XML files use the following conventions:

- NSalesOrder: Order to add services
- CSalesOrder: Change order, otherwise known as a Move Add Change Delete (MACD) order
- FSalesOrder: Follow-on order, used to update an order that has passed the point of no return
- R1, R2, R3, R4: revision order for submission after the original base order with the same name
- TBO: An order specifying time-based offerings
- Cancel: Cancel order

Note: Each customer order that you send must contain a unique EBM ID. For example, the EBM ID of a cancel order request (revision order) cannot be the same as the EBM ID of the original base order (new order).

Table 4–63 describes the order numbers for new and change orders:

Table 4–63 OracleComms_OSM_O2A_SalesOrders_NP_Sample Order Descriptions

Order Number	Description
Sales Order 10000	Double Play First-Time Purchase
Sales Order 10010	Double Play Promotion change orders for broadband
Sales Order 10020	Double Play Promotion change orders for VoIP
Sales Order 10030	Double Play Change Purchased Products
Sales Order 10040	Double Play Update Attributes of a Product
Sales Order 10050	Double Play Suspend Services
Sales Order 10060	Double Play Suspend and Resume on the Same Order
Sales Order 10070	Double Play Move Services to Different Address
Sales Order 10080	Double Play Disconnect Optional Products

Following is a list of the order XML files included in the cartridge:

- NEWPROD_NSalesOrderTBOEBM.xml
- NEWPROD_CSalesOrderTBOEBM.xml
- NEWPROD_GoldFSalesOrder10000F-1FFEBM.xml
- NEWPROD_GoldNSalesOrder10000-V2EBM-Predecessor.xml
- NEWPROD_GoldNSalesOrder10000-V2EBM-Successor.xml
- NEWPROD_GoldNSalesOrder10000-V2EBM.xml
- NEWPROD_GoldNSalesOrder10000F-1EBM.xml
- NEWPROD_GoldNSalesOrder10000F-1FO1EBM.xml
- NEWPROD_GoldNSalesOrder10000WithAdditionalFulfillmentItemCodeEBM.xml
- NEWPROD_GoldNSalesOrder10010-V2EBM.xml
- NEWPROD_GoldCSalesOrder10010-V2EBM.xml
- NEWPROD_GoldCSalesOrder10020-2-1EBM.xml
- NEWPROD_GoldCSalesOrder10020-2-1R1EBM.xml
- NEWPROD_GoldCSalesOrder10020-V2EBM.xml
- NEWPROD_GoldNSalesOrder10020-2-1EBM.xml
- NEWPROD_GoldNSalesOrder10020-V2EBM.xml
- NEWPROD_GoldNSalesOrder10030-V2EBM.xml
- NEWPROD_GoldNSalesOrder10030_2V1EBM.xml
- NEWPROD_GoldCSalesOrder10030-V2EBM.xml

- NEWPROD_GoldCSalesOrder10030R1_2V1EBM.xml
- NEWPROD_GoldCSalesOrder10030R3_2V1EBM.xml
- NEWPROD_GoldCSalesOrder10030R4_2V1EBM.xml
- NEWPROD_GoldCSalesOrder10030_2V1EBM.xml
- NEWPROD_GoldNSalesOrder10040-V2EBM.xml
- NEWPROD_GoldNSalesOrder10040_2V1EBM.xml
- NEWPROD_GoldCSalesOrder10040-V2EBM.xml
- NEWPROD_GoldR1SalesOrder10040-V2EBM.xml
- NEWPROD_GoldCSalesOrder10040R2_2V1EBM.xml
- NEWPROD_GoldR3SalesOrder10040-V2EBM.xml
- NEWPROD_GoldCSalesOrder10040R4_2V1EBM.xml
- NEWPROD_GoldCSalesOrder10040_2V1EBM.xml
- NEWPROD_GoldNSalesOrder10050-V2EBM.xml
- NEWPROD_GoldCSalesOrder10050-V2EBM.xml
- NEWPROD_GoldNSalesOrder10060-V2EBM.xml
- NEWPROD_GoldCSalesOrder10060-V2EBM.xml
- NEWPROD_GoldNSalesOrder10070-V2EBM.xml
- NEWPROD_GoldCSalesOrder10070-V2EBM.xml
- NEWPROD_GoldCSalesOrder10070R1-V2EBM.xml
- NEWPROD_GoldNSalesOrder10080-V2EBM.xml
- NEWPROD_GoldCSalesOrder10080-V2EBM.xml
- NEWPROD_GoldR1SalesOrder10080-V2EBM.xml
- NEWPROD_GoldR2SalesOrder10080-V2EBM.xml
- NEWPROD_NSalesOrderWirelessProductsEBM.xml
- NEWPROD_SalesOrder10000CancelEBM.xml
- NEWPROD_SalesOrder10000DeliverEBM.xml

Note: The **NEWPROD_testfalloutnotification.xml** file is a sample order fallout notification but not a sample customer order. This XML file is used to send a particular task to fallout manually by pausing the corresponding queue.

Table 4–64 contains information about the changes included in the specific revision orders above:

Table 4–64 OracleComms_OSM_O2A_SalesOrders_NP_Sample Order Revision Details

Order Number	**Description**
GoldCSalesOrder10030R1_2V1EBM	ADD canceled on revision, DELETE canceled on revision
GoldCSalesOrder10030R3_2V1EBM	ADD modified on revision
GoldCSalesOrder10030R4_2V1EBM	New ADD on revision

Table 4–64 (Cont.) OracleComms_OSM_O2A_SalesOrders_NP_Sample Order Revision Details

Order Number	Description
GoldR1SalesOrder10040-V2EBM	UPDATE modified on revision
GoldCSalesOrder10040R2_2V1EBM	UPDATE canceled on revision
GoldR3SalesOrder10040-V2EBM	New ADD on revision
GoldCSalesOrder10040R4_2V1EBM	DELETE added on revision
GoldCSalesOrder10070R1-V2EBM	MOVE canceled on revision
GoldR1SalesOrder10080-V2EBM	DELETE modified on revision (future date)
GoldR2SalesOrder10080-V2EBM	UPDATE added on revision

OracleComms_OSM_O2A_SimpleTopology_Sample

The OracleComms_OSM_O2A_SimpleTopology_Sample cartridge is a demonstration cartridge containing decomposition rules and order component specifications to decompose billing and provisioning fulfillment functions into the Simple topology: a single billing instance and a single local fulfillment instance.

Table 4–65 contains a list of the order component specifications defined in this cartridge.

Table 4–65 OracleComms_OSM_O2A_SimpleTopology_Sample Order Component Specifications

Order Component Specification	Description
BRM-ALL	Represents the billing fulfillment system.
Provisioning-ALL	Represents the provisioning fulfillment system.

OracleComms_OSM_O2A_TypicalTopology_Sample

The OracleComms_OSM_O2A_TypicalTopology_Sample cartridge is a demonstration cartridge containing decomposition rules and order component specifications to decompose billing, provisioning, shipping, and install fulfillment functions into the Typical topology: multiple billing, local fulfillment, supply chain management, and workforce management instances.

Table 4–66 contains a list of the order component specifications defined in this cartridge.

Table 4–66 OracleComms_OSM_O2A_TypicalTopology_Sample Order Component Specifications

Order Component Specification	Description
BRM-ALL	Represents the billing fulfillment system if there is a single billing system.
BRM-BIZBDB	Represents the billing fulfillment system for business broadband customers.
BRM-REZBDB	Represents the billing fulfillment system for residential broadband customers.
BRM-VIRTUAL	Represents the billing fulfillment system for non-service billing functions.
BRM-VoIP	Represents the billing fulfillment system for VoIP customers.
Provisioning-ALL	Represents the provisioning fulfillment system if there is a single provisioning system.
Provisioning-DSL	Represents the provisioning system for DSL customers outside the UK.

Table 4–66 (Cont.) OracleComms_OSM_O2A_TypicalTopology_Sample Order Component Specifications

Order Component Specification	Description
Provisioning-UKDSL	Represents the provisioning system for DSL customers inside the UK.
Provisioning-VOIP	Represents the provisioning system for VoIP customers.
Shipping-ALL	Represents the shipping fulfillment system if there is a single shipping system.
Shipping-InHouse	Represents the shipping system for shipments without partner involvement.
Shipping-PartnerInc	Represents the shipping system for shipments with partner involvement.
WFM-A	Represents the first workflow management fulfillment system if there is more than one workflow management system.
WFM-ALL	Represents the workflow management fulfillment system if there is a single workflow management system.
WFM-B	Represents the second workflow management fulfillment system if there is more than one workflow management system.

Service Order Management Cartridges for the Calculate Service Order Solution Option

The following cartridges operate in the service order management role, which translates Oracle AIA service orders into OSM service orders and processes those orders. These cartridges are used in the calculate service order solution option.

OracleComms_OSM_O2A_SOM_CSO_Base

The OracleComms_OSM_O2A_SOM_CSO_Base cartridge is a productized cartridge that supports the provisioning of service orders.

Order Events

When the COM_SalesOrderFulfillment order reaches one of the order events listed in Table 4–67, it triggers the listed XQuery module to send an order update to the upstream system.

Table 4–67 OracleComms_OSM_O2A_SOM_CSO_Base Order Events

Order Event	Description
creation	Calls the SOMOrderCreationFailure XQuery module to determine if a failure has occurred. If so, generates a message to central order management through the Oracle AIA error handling framework.
completion	Calls the SOMCompletionStatusSender XQuery module to send the order completion to central order management.

Order Lifecycle Manager Configuration

The Order-to-Activate order lifecycle manager is configured with the header values for the Order Lifecycle Management user interface. It also contains mappings between Order-to-Activate central order management fulfillment states and standard order lifecycle manager states.

Table 4–68 displays the mappings that are configured. The high-level fulfillment states are mapped, which causes the child states to be mapped as well.

Table 4–68 Fulfillment State to Order Lifecycle Manager State Mapping

Fulfillment State	Order Lifecycle Manager State
SOM_CANCELLED	Canceled
SOM_COMPLETE	Complete
SOM_FAILED	Failed
SOM_INPROGRESS	In Progress

Note: If you have both central order management and service order management in the same Design Studio workspace, you will see service order management fulfillment states in the list in the order lifecycle manager. The names of the high-level fulfillment states for central order management do not start with **SOM_**. The central order management fulfillment states do not need to be mapped here, because they are mapped in the order lifecycle manager in the central order management configuration. See "Order Lifecycle Manager Configuration" for information about central order management state mappings.

XQuery Modules in the OracleComms_OSM_O2A_SOM_CSO_Base Cartridge

Table 4–69 through Table 4–72 list the different types of XQuery modules in this cartridge.

Table 4–69 OracleComms_OSM_O2A_SOM_CSO_Base XQuery Modules for Fallout Handling

Fallout Handling XQuery Module	Extendable	Description
SendAbortLFOrderFailure	No	Sends a fulfillment order failure update when the order termination request failed.
SendAbortLFOrderSuccess	No	Sends a fulfillment order success update when the order termination request succeeded.
SomAbortOrder	No	Sends an order termination request for a given fulfillment request.
SomFindOrder	No	Creates a find order request for a fulfillment request with a given order key.
SomSuspendOrder	No	Creates a suspend order request for a fulfillment request with a given order key.
SomWebServicesResponseHandler	No	Utility module for providing retrieval and update to service order.

Table 4–70 OracleComms_OSM_O2A_SOM_CSO_Base Orchestration Sequence XQuery Modules

Orchestration Sequence XQuery Module	Extendable	Description
SOM_FulfillmentModeExpression	No	Marshals the fulfillment mode code from the service order.
SOM_OrderItemSelector	No	Module to select all order line items from the service order.

Table 4–71 OracleComms_OSM_O2A_SOM_CSO_Base Order Recognition XQuery Modules

Order Recognition XQuery Module	Extendable	Description
SOM_DataTransform	No	Transforms the Oracle AIA service order to an OSM service order.
SOM_DataValidation	No	Validates the Oracle AIA service order.

Table 4–72 OracleComms_OSM_O2A_SOM_CSO_Base Order State XQuery Modules

Order State XQuery Module	Extendable	Description
SOM_Reference	No	Accesses the sales order reference.

Automation Modules in the OracleComms_OSM_O2A_SOM_CSO_Base Cartridge

Table 4–73 lists the automation modules in the cartridge with their associated automated tasks.

Table 4–73 OracleComms_OSM_O2A_SOM_CSO_Base Automation Modules

Automation Module	Automated Task	Description
AbortSomOrderRequest	AbortSomOrderTask	Calls the SomAbortOrder XQuery.
AbortSomOrderResponse	AbortSomOrderTask	External event receiver to invoke SomWebServicesResponseHandler XQuery.
FindSomOrderRequest	FindSomOrderTask	Calls the SomFindOrder XQuery.
FindSomOrderResponse	FindSomOrderTask	External event receiver to invoke SomWebServicesResponseHandler XQuery.
SendAbortStatusToUpstream	SendAbortStatusToUpstream Task	Calls the SendAbortLFOrderSuccess XQuery
SuspendSomOrderRequest	SuspendSomOrderTask	Calls the SomSuspendOrder XQuery.
SuspendSomOrderResponse	SuspendSomOrderTask	External event receiver to invoke SomWebServicesResponseHandler XQuery

OracleComms_OSM_O2A_SOM_CSO_Broadband_Internet_Access_CFS

The OracleComms_OSM_O2A_SOM_CSO_Broadband_Internet_Access_CFS cartridge is a demonstration cartridge that contains the order item parameter bindings associated with the customer facing service for broadband Internet access. It also contains XQuery modules to support the order item parameter bindings.

OracleComms_OSM_O2A_SOM_CSO_Common

The OracleComms_OSM_O2A_SOM_CSO_Common cartridge is a productized cartridge that contains data elements and fulfillment modes for service order management with the calculate service order solution option.

The following fulfillment modes are defined in this cartridge:

- SOM_CANCEL
- SOM_DELIVER
- SOM_TSQ

OracleComms_OSM_O2A_SOM_CSO_CompleteProvisioning

The OracleComms_OSM_O2A_COM_CSO_Base cartridge is a demonstration cartridge that supports provisioning fulfillment functions for service order management.

OracleComms_OSM_O2A_SOM_CSO_DeliverOrder

The OracleComms_OSM_O2A_SOM_CSO_DeliverOrder cartridge is a demonstration cartridge that supports order delivery fulfillment functions for service order management.

OracleComms_OSM_O2A_SOM_CSO_DesignService

The OracleComms_OSM_O2A_SOM_CSO_DesignService cartridge is a demonstration cartridge that supports service design fulfillment functions for service order management.

OracleComms_OSM_O2A_SOM_CSO_Email_CFS

The OracleComms_OSM_O2A_SOM_CSO_Email_CFS cartridge is a demonstration cartridge that contains the order item parameter bindings associated with the customer facing service for email service. It also contains XQuery modules to support the order item parameter bindings.

OracleComms_OSM_O2A_SOM_CSO_FulfillmentPattern

The OracleComms_OSM_O2A_SOM_CSO_FulfillmentPattern cartridge is a productized cartridge that contains service order management fulfillment patterns for the calculate service order solution option.

OracleComms_OSM_O2A_SOM_CSO_FulfillmentStateMap

The OracleComms_OSM_O2A_COM_CSO_Base cartridge is a productized cartridge that contains fulfillment state entities used by the solution.

This cartridge contains the following common fulfillment state definitions, which are used in composition rules. Listed under each main fulfillment state are its child states.

- SOM_PENDING
- SOM_INPROGRESS
  - IN_PROGRESS-BI_CAPTURED
  - IN_PROGRESS-BI_ISSUED
  - IN_PROGRESS-BI_PROCESSED
  - IN_PROGRESS-TO_CREATE
- SOM_COMPLETED
  - COMPLETE-BI_APPROVED
  - COMPLETE-BI_COMPLETED
  - COMPLETE-TA_CALCULATED
  - COMPLETE-TO_COMPLETED
- SOM_CANCELLED

  - CANCELLED-BI_APPROVED
  - CANCELLED-BI_CAPTURED
  - CANCELLED-BI_COMPLETED
  - CANCELLED-BI_ISSUED
  - CANCELLED-BI_PROCESSED
  - CANCELLED-TA_CALCULATED
  - CANCELLED-TO_COMPLETED
  - CANCELLED-TO_CREATE
- SOM_FAILED
  - FAILED-BI_APPROVED
  - FAILED-BI_CAPTURED
  - FAILED-BI_COMPLETED
  - FAILED-BI_ISSUED
  - FAILED-BI_PROCESSED
  - FAILED-SERVICE
  - FAILED-TO_COMPLETED
  - FAILED-TO_CREATE

OracleComms_OSM_O2A_SOM_CSO_Internet_Media_CFS

The OracleComms_OSM_O2A_SOM_CSO_Internet_Media_CFS cartridge is a demonstration cartridge that contains the order item parameter bindings associated with the customer facing service for Internet media service. It also contains XQuery modules to support the order item parameter bindings.

OracleComms_OSM_O2A_SOM_CSO_IP_Fax_CFS

The OracleComms_OSM_O2A_COM_CSO_Base cartridge is a demonstration cartridge that contains the order item parameter bindings associated with the customer facing service for IP fax service. It also contains XQuery modules to support the order item parameter bindings.

OracleComms_OSM_O2A_SOM_CSO_ModelContainer

The OracleComms_OSM_O2A_SOM_CSO_ModelContainer cartridge defines the common model projects that contain elements that might need to be included in the deployment.

To see the common model projects that are contained by this cartridge, open the **Properties** tab of the cartridge editor. For more information about the common model projects included with the Order-to-Activate cartridges, see "Conceptual Model Projects."

OracleComms_OSM_O2A_SOM_CSO_PlanDelivery

The OracleComms_OSM_O2A_COM_CSO_Base cartridge is a demonstration cartridge that supports delivery planning fulfillment functions for service order management.

OracleComms_OSM_O2A_SOM_CSO_Recognition

The OracleComms_OSM_O2A_SOM_CSO_Recognition cartridge is a demonstration cartridge that recognizes a service order and triggers the creation of a SOM_ProvisionOrderFulfillment order. In addition, this cartridge catches all in-bound messages not recognized by any other provisioning recognition rules.

Table 4–74 lists the order recognition rules defined in this cartridge.

Table 4–74 OracleComms_OSM_O2A_SOM_CSO_Recognition Recognition Rules

Order Recognition Rule	Description
Recognize_AbortPropagationServiceOrder	Recognizes a termination request for an order.
SOM_ORPFallout_CFM_OrderRecognitionRule	Recognizes an ORP fallout to create a fault message to be sent to Oracle AIA error handling queue.
RecognizeEBM_ProvisioningOrder	Recognizes a service order that must be executed and creates a SOM_ProvisionOrderFulfillment order.
SOM_ResumePendingIbMsg_OrderRecognitionRule	Recognizes an inbound message to create a ResumePendingInBoundMessage order in service order management.

Revision Number Update for Canceled Orders

For information about special revision number processing for canceled orders, see "Revision Number Update for Canceled Orders."

OracleComms_OSM_O2A_SOM_CSO_Solution

The OracleComms_OSM_O2A_SOM_CSO_Solution cartridge is a demonstration composite cartridge that references all cartridges required for service order management.

To see the component cartridges referenced in this cartridge for your solution, open the **Dependencies** tab in the composite cartridge editor.

OracleComms_OSM_O2A_SOM_CSO_Topology

The OracleComms_OSM_O2A_SOM_CSO_Topology cartridge is a demonstration cartridge. This cartridge contains entities, such as decomposition rules and order components, for service order management.

OracleComms_OSM_O2A_SOM_CSO_VoIP_Access_CFS

The OracleComms_OSM_O2A_SOM_CSO_VoIP_Access_CFS cartridge is a demonstration cartridge that contains the order item parameter bindings associated with the customer facing service for VoIP access. It also contains XQuery modules to support the order item parameter bindings.

OracleComms_OSM_O2A_SOM_CSO_Web_Conferencing_CFS

The OracleComms_OSM_O2A_SOM_CSO_Internet_Media_CFS cartridge is a demonstration cartridge that contains the order item parameter bindings associated with the customer facing service for web conferencing service. It also contains XQuery modules to support the order item parameter bindings.

Service Order Management Cartridges for the Solution Option Without Calculate Service Order

The following cartridges operate in the service order management role, which translates Oracle AIA service orders into OSM service orders and processes those orders. These cartridges are used in the solution option without calculate service order.

OracleComms_OSM_O2A_SOM_Base

The OracleComms_OSM_O2A_SOM_Base cartridge is a productized cartridge supporting the orchestration of service orders that have come from Oracle AIA. It includes handling status and data updates from fulfillment requests back to central order management.

Order Events

When the COM_SalesOrderFulfillment order reaches one of the order events listed in Table 4–75, it triggers the listed XQuery module to send an order update to the upstream system.

Table 4–75 OracleComms_OSM_O2A_SOM_Base Order Events

Order Event	Description
creation	Calls the LFCheckCreationOrderFailure XQuery to determine if a failure has occurred. If so, generates a message to central order management through the Oracle AIA error handling framework.
completion	Calls the ProvisionOrderCompleteEventHandler XQuery module to send the order completion to central order management.

Processing Granularity Rules

There are three orchestration stages defined in the orchestration sequence to decompose the order line items. The result of each stage of decomposition is the source for the next stage of decomposition.

- In the first stage, the order line items are decomposed by fulfillment function.
- In the second stage, the order line items are decomposed by fulfillment provider.
- In the third stage, the order line items are decomposed by granularity rule.

Granularity rules provide the configuration for the third stage of decomposition. During orchestration plan generation at run time, the granularity rule takes as input the order line items that have already been grouped by fulfillment function and subdivided by fulfillment provider.

Table 4–76 lists the processing granularity rule entities.

Table 4–76 OracleComms_OSM_O2A_SOM_Base Processing Granularity Rules

Entity Name	Entity Type	Description
ServiceGranularity	Order Component Specification	This granularity rule selects: ■ An order line item that represents a service along with service components and related order line items ■ Order line items of any other root node on the order along with their related order line items

XQuery Modules in the OracleComms_OSM_O2A_SOM_Base Cartridge

Table 4–77 through Table 4–83 list the different types of XQuery modules in this cartridge.

No table is included for the Order Item Property XQuery modules because none are extendable and each XQuery module does the same thing: retrieves the specified order item property from the appropriate location in the order data.

Table 4–77 OracleComms_OSM_O2A_SOM_Base XQuery Modules for Constants

Constants XQuery Module	Extendable	Description
SomQueryViewConstants	No	Defines constants for querying views in service order management.

Table 4–78 OracleComms_OSM_O2A_SOM_Base XQuery Modules for Fallout Handling

Fallout Handling XQuery Module	Extendable	Description
AbortLFOrderRequest	No	Sends an order termination request for the fulfillment request through the web service API.
FindLFOrder	No	Creates a find order request for a fulfillment request with a given order key.
LFAbortOrderPropagation	No	Sends an order termination request for a given fulfillment request.
LFAbortOrderPropagationCheck	No	Checks the status of the order termination request for fulfillment request.
LFAbortOrderPropagationResp	No	Handles the response of the order termination request for the fulfillment request.
LFwsResponseHandler	No	Utility module for providing retrieval and update to service order.
SendAbortLFOrderFailure	No	Sends a fulfillment order failure update when the order termination request failed.
SendAbortLFOrderSuccess	No	Sends a fulfillment order success update when the order termination request succeeded.
SuspendLFOrder	No	Creates a suspend order request for a fulfillment request with a given order key.

Table 4–79 OracleComms_OSM_O2A_SOM_Base Orchestration Sequence XQuery Modules

Orchestration Sequence XQuery Module	Extendable	Description
FulfillmentModeExpression	No	Marshals the fulfillment mode code from the service order.
OrderItemSelector	No	Module to select all order line items from the service order.

Table 4–80 OracleComms_OSM_O2A_SOM_Base Order Data Change XQuery Modules

Order Data Change XQuery Module	Extendable	Description
CreateLFFaultToAIAEH	No	Creates an error message to be sent to the Oracle AIA error handling queue.

Table 4–81 OracleComms_OSM_O2A_SOM_Base Order Item Hierarchy XQuery Modules

Order Item XQuery Module	Extendable	Description
LineIdKey	No	Retrieves the order line item's ID.
ParentLineIdKey	No	Retrieves the parent order line item's ID.

Table 4–82 OracleComms_OSM_O2A_SOM_Base Order Recognition XQuery Modules

Order Recognition XQuery Module	Extendable	Description
ProvisionOrderData	No	Transforms the Oracle AIA service order to an OSM service order.
ProvisionOrderPriority	No	Retrieves the priority of the Oracle AIA service order.
ProvisionOrderRecognition	No	Recognizes the Oracle AIA service order.
ProvisionOrderValidation	No	Validates the Oracle AIA service order.

Table 4–83 OracleComms_OSM_O2A_SOM_Base Order State XQuery Modules

Order State XQuery Module	Extendable	Description
LFCheckCreationOrderFailure	No	Determines if a failure has occurred. If so, generates an error message to central order management through the Oracle AIA error handling framework.
ProvisionOrderCompleteEventHandler	No	Sends service order status update with COMPLETE status code back to central order management.

Automation Modules in the OracleComms_OSM_O2A_SOM_Base Cartridge

Table 4–84 lists the automation modules in the cartridge with their associated automated tasks.

Table 4–84 OracleComms_OSM_O2A_SOM_Base Automation Modules

Automation Module	Automated Task	Description
AbortLFOrderPlugin	AbortLFOrderTask	Calls the AbortLFOrderRequest XQuery.
AbortLFOrderRespPlugin	AbortLFOrderTask	External event receiver to invoke LFwsResponseHandler XQuery.
FindLFOrderPlugin	GetLFOrderTask	Calls the FindLFOrder XQuery.
GetLFOrderDataPlugin	GetLFOrderTask	External event receiver to invoke LFwsResponseHandler XQuery.
SendAbortLFOrderFailurePlugin	LFAbortOrderFailureTask	Calls the SendAbortLFOrderFailure XQuery.
SendAbortLFOrderSuccessPlugin	LFAbortOrderSuccessTask	Calls the SendAbortLFOrderSuccess XQuery.
LFOrderAbortPropagationCheckPlugin	LFOrderAbortPropagationCheck	Internal event receiver to invoke LFAbortOrderPropagationCheck XQuery.
LFAbortOrderPropagationPlugin	LFOrderAbortPropagationTask	Internal event receiver to invoke LFAbortOrderPropagation XQuery.

Table 4–84 (Cont.) OracleComms_OSM_O2A_SOM_Base Automation Modules

Automation Module	Automated Task	Description
LFAbortOrderPropagationRespPlugin	LFOrderAbortPropagationTask	External event receiver to invoke LFAbortOrderPropagationResp XQuery.
SuspendLFOrderPlugin	SuspendLFOrderTask	Internal event receiver to invoke SuspendLFOrder XQuery.
SuspendLFOrderRespPlugin	SuspendLFOrderTask	Calls the LFwsResponseHandler XQuery.

OracleComms_OSM_O2A_SOM_Provisioning

The OracleComms_OSM_O2A_SOM_Provisioning cartridge is a productized cartridge that supports provisioning fulfillment functions. These functions specify a subprocess to handle delivery of a relevant subset of order data to the provisioning system and handle responses from the provisioning system.

Table 4–85 lists he XQuery modules in the cartridge that support component interaction.

Table 4–85 OracleComms_OSM_O2A_SOM_Provisioning Component Interaction XQuery Modules

Component Interaction XQuery Module	Extendable	Description
SomProvisionOrderInteractionModule	Yes	Provides functions to support ProvisionOrderFunction in service order management.

Table 4–86 lists the XQuery modules defined for the SomProvisionOrderFunction fulfillment function.

Table 4–86 SomProvisionOrderFunction XQuery Modules

XQuery Module	Extendable	Description
CreateProvisioningOrderRequest_do	No	Generates the Oracle AIA EBM requests to the external provisioning system.
CreateProvisioningOrderRequest_redo	No	Generates the Oracle AIA EBM requests to the external provisioning system.
CreateProvisioningOrderRequest_undo	No	Generates the Oracle AIA EBM requests to external provisioning system for undo.
CreateProvisioningOrderResponse	No	Consumes the Oracle AIA EBM response from the external provisioning system.
InitialSomProvisionOrderSIResponse	No	Sets the correlation context before consuming Oracle AIA EBM response from the external provisioning system.
RetryCreateProvisioningOrderResponse	No	Handles fallout in service order management's provisioning order by re-creating the response from the external provisioning system.
SomProvisionOrderSIEntryPoint	No	Handles extension point COMPONENT-START, updates order item properties, and reports milestones to lifecycle management for service order management and then to central order management.

Table 4–86 (Cont.) SomProvisionOrderFunction XQuery Modules

XQuery Module	Extendable	Description
SomProvisionOrderSIResponse	No	Consumes the Oracle AIA EBM response from the external provisioning system.
SomProvisionOrderSIResponseFalloutPrepare	No	Handles manual retry process to re-submit EBM to the external provisioning system.
UpdateProvisionOrderStatusFunctions	No	Provides functions to create an EBM that contains the fulfillment request's update and send the EBM to central order management.

Table 4–87 lists the automation modules (with their associated automated tasks) defined in the OracleComms_OSM_O2A_SOM_Provisioning cartridge for the SomProvisionOrderFunction fulfillment function.

Table 4–87 SomProvisionOrderFunction Automation Modules

Automation Module	Automated Task	Description
SomProvisionOrderSIEntryPointBean	SomProvisionOrderSIEntryPointTask	Calls the SomProvisionOrderSIEntryPoint XQuery.
SomProvisionOrderSIResponseFalloutPrepareBean	SomProvisionOrderSIResponseFalloutPrepareTask	Calls the SomProvisionOrderSIResponseFalloutPrepare XQuery.
InitialSomProvisionOrderSIResponseBean	SomProvisionOrderSIResponseTask	Calls the InitialSomProvisionOrderSIResponse XQuery.
SomProvisionOrderSIResponseBean	SomProvisionOrderSIResponseTask	External event receiver to invoke SomProvisionOrderSIResponse XQuery.
RetryCreateProvisioningOrderRequestBean	SomProvisionOrderSIRetryTask	Calls the CreateProvisioningOrderRequest_do XQuery.
RetryCreateProvisioningOrderResponseBean	SomProvisionOrderSIRetryTask	External event receiver to invoke RetryCreateProvisoningOrderResponse XQuery.
CreateProvisiongOrderRequestBean_do	SomProvisionOrderSITask	Calls the CreateProvisioningOrderRequest_do XQuery.
CreateProvisiongOrderRequestBean_redo	SomProvisionOrderSITask	Calls the CreateProvisioningOrderRequest_redo XQuery.
CreateProvisiongOrderRequestBean_undo	SomProvisionOrderSITask	Calls the CreateProvisioningOrderRequest_undo XQuery.
CreateProvisioningOrderResponseBean	SomProvisionOrderSITask	External event receiver to invoke CreateProvisoningOrderResponse XQuery.
SomProvisionOrderSIResponseUndoBean	SomProvisionOrderSITask	External event receiver to invoke SomProvisionOrderSIResponse XQuery.

OracleComms_OSM_O2A_SOM_Solution

The OracleComms_OSM_O2A_SOM_Solution cartridge is a demonstration composite cartridge that references all cartridges required for service order management.

To see the component cartridges referenced in this cartridge for your solution, open the **Dependencies** tab in the composite cartridge editor.

OracleComms_OSM_O2A_SOM_Recognition_Sample

The OracleComms_OSM_O2A_SOM_Recognition_Sample cartridge is a demonstration cartridge that recognizes a service order and triggers the creation of a SOM_ProvisionOrderFulfillment order. In addition, this cartridge catches all in-bound messages not recognized by any other provisioning recognition rules.

Table 4–88 lists the order recognition rules defined in this cartridge.

Table 4–88 OracleComms_OSM_O2A_SOM_Recognition_Sample Recognition Rules

Order Recognition Rule	Description
SOM_LFAbortOrderPropagationOrder_ Recognition	Recognizes a termination request for an order.
SOM_ORPFallout_CFM_OrderRecognitionRule	Recognizes an ORP fallout to create a fault message to be sent to Oracle AIA error handling queue.
SOM_ProvisionOrderFulfillment_Recognition	Recognizes a service order that must be executed and creates a SOM_ProvisionOrderFulfillment order.
SOM_ResumePendingIbMsg_ OrderRecognitionRule	Recognizes an inbound message to create a ResumePendingInBoundMessage order in service order management.

Revision Number Update for Canceled Orders

For information about special revision number processing for canceled orders, see "Revision Number Update for Canceled Orders."

OracleComms_OSM_O2A_SomBBVoIP_FP_NP_Sample

The OracleComms_OSM_O2A_SomBBVoIP_FP_NP_Sample cartridge is a demonstration cartridge containing fulfillment patterns, each of which configures a fulfillment flow for provisioning fulfillment functions. The demonstration VoIP and Broadband products map to the fulfillment patterns.

The following fulfillment patterns are configured in this cartridge:

- SOM_Service.Provision – All other fulfillment patterns extend from this.
- SOM_Service.Broadband
- SOM_Service.CPE.Broadband
- SOM_Service.CPE.VoIP
- SOM_Service.VoIP

OracleComms_OSM_O2A_SomProvisionBroadband_Sample

The OracleComms_OSM_O2A_SomProvisionBroadband_Sample cartridge is a demonstration cartridge supporting service orders for broadband services.

Table 4–89 to Table 4–91 list the entities in the OracleComms_OSM_O2A_ SomProvisionBroadband_Sample cartridge.

Table 4–89 OracleComms_OSM_O2A_SomProvisionBroadband_Sample Entities

Name	Type	Description
OSM_O2A_SomProvisionBroadband_ Recognition	Order Recognition Rule	Recognizes a broadband service order and creates an OracleComms_OSM_O2A_SomProvisionBroadband_ SampleOrder service order to manage its fulfillment.
OracleComms_OSM_O2A_ SomProvisionBroadband_SampleOrder	Order	Local service order structure for managing a service order for broadband services.
BroadbandProvisioningOrderLifeCycle	Lifecycle Policy	Defines the security permissions for order transactions.
BroadbandProvisioningRole	Role	Role with permissions to create and view OracleComms_OSM_O2A_SomProvisionBroadband_ SampleOrder.
BroadbandServicesProvisioningProcess	Process and Tasks	Process to handle provisioning of broadband services such as email, Internet and customer premise equipment.
CreateBroadbandServicesProvisioningOr derTask	Manual Task	Creation task to create an OracleComms_OSM_O2A_ SomProvisionBroadband_SampleOrder.
OracleComms_OSM_O2A_ SomProvisionBroadband_Sample	Data Schema	Data structures for managing broadband services.

Table 4–90 OracleComms_OSM_O2A_SomProvisionBroadband_Sample XQuery Modules

XQuery Module	Extendable	Description
ActivityRouterTask	Yes	Transit to the next task with different task exit status depending on broadband provisioning service.
BroadbandServiceErrorFault	Yes	Creates error fault for broadband provisioning service.
BroadbandServiceOrderCompleteEventH andler	Yes	Sends broadband service order status update with COMPLETE status code back to service order management.
BroadbandServiceProcessEntryUndoBea n	Yes	Updates the provisioning order and sends broadband service order status.
BroadbandServiceProvisioningOrderDat aRule	Yes	Transforms the Oracle AIA service order to an OSM service order.
BroadbandServiceUtilityModule	Yes	Utility module to provide functions to support provisioning broadband service.
ProvisionTaskComplete	Yes	Completes a provisioning task.
ProvisionTaskStart	Yes	Starts a provisioning task.

Table 4–91 OracleComms_OSM_O2A_SomProvisionBroadband_Sample Automation Modules

Automation Module	Automated Task	Description
ActivityRouterBean	ActivityRouterTask	Calls the ActivityRouterTask XQuery.
BroadbandServiceErrorFaultBe an	BroadbandServiceErrorFault Task	Calls the BroadbandServiceErrorFault XQuery.
BroadbandServiceErrorFaultBe an_redo	BroadbandServiceErrorFault Task	Calls the BroadbandServiceErrorFault XQuery for redo mode.
BroadbandServiceProcessEntr yUndoBean	BroadbandServiceProcessEnt ryTask	Calls the BroadbandServiceProcessEntryUndoBean XQuery.

Table 4–91 (Cont.) OracleComms_OSM_O2A_SomProvisionBroadband_Sample Automation Modules

Automation Module	Automated Task	Description
BroadbandServiceProcessExitBean	BroadbandServiceProcessExitTask	Calls the BroadbandServiceProcessExitBean XQuery.
ProvisionCPEEntryPointBean	ProvisionCPEEntryPointTask	Calls the ProvisionTaskStart XQuery.
ProvisionCPEExitPointBean	ProvisionCPEExitPointTask	Calls the ProvisionTaskComplete XQuery.
ProvisionInternetEmailServiceBundleEntryPointBean	ProvisionInternetEmailServiceBundleEntryPointTask	Calls the ProvisionTaskStart XQuery.
ProvisionInternetEmailServiceBundleExitPointBean	ProvisionInternetEmailServiceBundleExitPointTask	Calls the ProvisionTaskComplete XQuery.
ProvisionInternetMediaServiceBundleEntryPointBean	ProvisionInternetMediaServiceBundleEntryPointTask	Calls the ProvisionTaskStart XQuery.
ProvisionInternetMediaServiceBundleExitPointBean	ProvisionInternetMediaServiceBundleExitPointTask	Calls the ProvisionTaskComplete XQuery.
ProvisionInternetServiceBundleEntryPointBean	ProvisionInternetServiceBundleEntryPointTask	Calls the ProvisionTaskStart XQuery.
ProvisionInternetServiceBundleExitPointBean	ProvisionInternetServiceBundleExitPointTask	Calls the ProvisionTaskComplete XQuery.

OracleComms_OSM_O2A_SomProvisionVoIP_Sample

The OracleComms_OSM_O2A_SomProvisionVoIP_Sample cartridge is a demonstration cartridge supporting service orders for VoIP services.

Table 4–92 to Table 4–94 list the entities in the OracleComms_OSM_O2A_SomProvisionVoIP_Sample cartridge.

Table 4–92 OracleComms_OSM_O2A_SomProvisionVoIP_Sample Entities

Name	Type	Description
OSM_O2A_SomProvisionVoIP_Recognition	Order Recognition Rule	Recognizes a VoIP service order and creates an OracleComms_OSM_O2A_SomProvisionVoIP_SampleOrder service order to manage its fulfillment.
OracleComms_OSM_O2A_SomProvisionVoIP_SampleOrder	Order	Local service order structure for managing a service order for VoIP services.
VoIPProvisioningOrderLifeCycle	Lifecycle Policy	Defines the security permissions for order transactions.
VoIPProvisioningRole	Role	Role with permissions to create and view OracleComms_OSM_O2A_SomProvisionVoIP_SampleOrder.
VoIPServicesProvisioningProcess	Process and Tasks	Process to handle provisioning of VoIP services such as VoIP service and customer premise equipment.
CreateVoIPServicesProvisioningOrderTask	Manual Task	Creation task to create an OracleComms_OSM_O2A_SomProvisionVoIP_SampleOrder.
OracleComms_OSM_O2A_SomProvisionVoIP_Sample	Data Schema	Data structures for managing VoIP services.

Table 4–93 OracleComms_OSM_O2A_SomProvisionVoIP_Sample XQuery Modules

XQuery Module	Extendable	Description
VoIPActivityRouterTask	Yes	Transit to the next task with different task exit status depending on VoIP provisioning service.
VoIPServiceErrorFault	Yes	Creates error fault for VoIP provisioning service.
VoIPServiceOrderCompleteEventHandler	Yes	Sends VoIP service order status update with COMPLETE status code back to service order management.
VoIPServiceProcessEntryBean	Yes	Updates the provisioning order and sends VoIP service order status.
VoIPServiceProvisioningOrderDataRule	Yes	Transforms the Oracle AIA service order to an OSM service order.
VoIPServiceUtilityModule	Yes	Utility module to provide functions to support provisioning VoIP service.
VoIPProvisionTaskComplete	Yes	Completes a provisioning task.
VoIPProvisionTaskStart	Yes	Starts a provisioning task.
VoIPServiceProcessExitBean	Yes	Completes a task using a successful status.

Table 4–94 OracleComms_OSM_O2A_SomProvisionVoIP_Sample Automation Modules

Automation Module	Automated Task	Description
VoIPActivityRouterBean	VoIPActivityRouterTask	Calls the VoIPActivityRouterTask XQuery.
VoIPServiceErrorFaultBean	VoIPServiceErrorFaultTask	Calls the VoIPServiceErrorFault XQuery.
VoIPServiceErrorFaultBean_redo	VoIPServiceErrorFaultTask	Calls the VoIPServiceErrorFault XQuery for redo mode.
VoIPServiceProcessEntryBean	VoIPServiceProcessEntryTask	Calls the VoIPServiceProcessEntryBean XQuery.
VoIPServiceProcessExitBean	VoIPServiceProcessExitTask	Calls the VoIPServiceProcessExitBean XQuery.
ProvisionVoIPCPEEntryPointBean	ProvisionVoIPCPEEntryPointTask	Calls the VoIPProvisionTaskStart XQuery.
ProvisionVoIPCPEExitPointBean	ProvisionVoIPCPEExitPointTask	Calls the VoIPProvisionTaskComplete XQuery.
ProvisionVoIPServiceBundleEntryPointBean	ProvisionVoIPServiceBundleEntryPointTask	Calls the VoIPProvisionTaskStart XQuery.
ProvisionVoIPServiceBundleExitPointBean	ProvisionVoIPServiceBundleExitPointTask	Calls the VoIPProvisionTaskComplete XQuery.

Common Conceptual Model Projects

The following cartridges provide entities that are used by or referenced by other Order-to-Activate cartridges.

For more information about the conceptual model, see *Design Studio Concepts*.

OracleComms_Model_Base

The OracleComms_Model_Base project contains entities that are used for multiple services, including provider functions, functional areas, relationship types, action codes, and units of measure.

OracleComms_Model_BaseCatalog

The OracleComms_Model_BaseCatalog project contains conceptual model fulfillment patterns. It also contains the data schema that defines the data elements from the Oracle AIA EBM schema. Cartridges that must include data elements from Oracle AIA EBM can reuse the elements defined in this cartridge.

OracleComms_Model_Common

The OracleComms_Model_Common project contains common data elements for all of the services.

OracleComms_Model_O2A_Broadband_Internet_Access_CFS

The OracleComms_Model_O2A_Broadband_Internet_Access_CFS project contains the customer facing services for broadband Internet access.

The following customer facing services are defined in this project:

- Broadband_Internet_Access_CFS

OracleComms_Model_O2A_Broadband_Internet_Access_SA

The OracleComms_Model_O2A_Broadband_Internet_Access_SA project contains the actions for broadband Internet access.

The following action codes are defined in this project for the customer facing service in the OracleComms_Model_O2A_Broadband_Internet_Access_CFS project:

- Add
- Change
- Disconnect
- Modify
- Move-Add
- Move-Delete
- Query
- Remove
- Resume
- Suspend

OracleComms_Model_O2A_Broadband_Internet_DataModel

The OracleComms_Model_O2A_Broadband_Internet_DataModel project contains the data schema for data specific to broadband Internet access.

OracleComms_Model_O2A_Email_CFS

The OracleComms_Model_O2A_Email_CFS project contains the customer facing services for email service.

The following customer facing services are defined in this project:

- Email_CFS

OracleComms_Model_O2A_Email_DataModel

The OracleComms_Model_O2A_Email_CFS project contains the data schema for data specific to email service.

OracleComms_Model_O2A_Email_SA

The OracleComms_Model_O2A_Email_CFS project contains the actions for email service.

The following action codes are defined in this project for the customer facing service in the OracleComms_Model_O2A_Email_CFS project:

- Add
- Change
- Disconnect
- Modify
- Move-Add
- Move-Delete
- Query
- Remove
- Resume
- Suspend

OracleComms_Model_O2A_Internet_Media_CFS

The OracleComms_Model_O2A_Internet_Media_CFS project contains the customer facing services for Internet media service.

The following customer facing services are defined in this project:

- InternetMedia_CFS

OracleComms_Model_O2A_Internet_Media_DataModel

The OracleComms_Model_O2A_Internet_Media_DataModel project contains the data schema for data specific to Internet media service.

OracleComms_Model_O2A_Internet_Media_SA

The OracleComms_Model_O2A_Internet_Media_SA project contains the actions for internet media service.

The following action codes are defined in this project for the customer facing service in the OracleComms_Model_O2A_Internet_Media_CFS project:

- Add
- Change
- Disconnect
- Modify
- Move-Add
- Move-Delete

- Query
- Remove
- Resume
- Suspend

OracleComms_Model_O2A_VoIP_Access_CFS

The OracleComms_Model_O2A_VoIP_Access_CFS project contains the customer facing services for VoIP access.

The following customer facing services are defined in this project:

- IP_Fax_CFS
- VoIP_Access_CFS
- Web_Conferencing_CFS

OracleComms_Model_O2A_VoIP_Access_SA

The OracleComms_Model_O2A_VoIP_Access_CFS project contains the actions for VoIP access.

The following action codes are defined in this project for each of the customer facing services in the OracleComms_Model_O2A_VoIP_Access_CFS project:

- Add
- Change
- Delete
- Disconnect
- Modify
- Move-Add
- Move-Delete
- None
- Query
- Remove
- Resume
- Suspend
- Update

OracleComms_Model_O2A_VoIP_DataModel

The OracleComms_Model_O2A_VoIP_Access_CFS project contains the data schema for data specific to VoIP access.

Conceptual Model Projects for Central Order Management

The following cartridges provide entities that are used by or referenced by central order management Order-to-Activate cartridges.

For more information about the conceptual model, see *Design Studio Concepts*.

OracleComms_Model_O2A_Billing_PS

The OracleComms_Model_O2A_Billing_PS project contains the products for billing services.

The following products are defined in this project:

- Broadband_Pricing_Event_PS
- Group_Member_PS
- Group_Owner_PS
- Offer_Sponsorship_PS
- Pricing_Event_PS
- Promotion_Group_PS
- VoIP_Pricing_Event_Billing_Validation_PS
- VoIP_Pricing_Event_PS

OracleComms_Model_O2A_Broadband_Internet_Access_PS

The OracleComms_Model_O2A_Broadband_Internet_Access_PS project contains the domains and products for broadband Internet services.

The following domains are defined in this project:

- BroadbandInternetDomain

The following products are defined in this project:

- Broadband_Bandwidth_PS
- Broadband_Modem_PS
- Broadband_Offer_Charge_Class
- Broadband_PS
- Broadband_Router_PS
- Firewall_PS

OracleComms_Model_O2A_Email_PS

The OracleComms_Model_O2A_Email_PS project contains the domains and products for email services.

The following domains are defined in this project:

- EmailDomain

The following products are defined in this project:

- Email_Service_PS

OracleComms_Model_O2A_Install_PS

The OracleComms_Model_O2A_Install_PS project contains the products for installation services.

The following products are defined in this project:

- High_Speed_Internet_Installation_PS

OracleComms_Model_O2A_Internet_Media_PS

The OracleComms_Model_O2A_Internet_Media_PS project contains the domains and products for Internet media services.

The following domains are defined in this project:

- InternetMediaDomain

The following products are defined in this project:

- Internet_Media_PS

OracleComms_Model_O2A_VoIP_PS

The OracleComms_Model_O2A_VoIP_PS project contains the domains and products for VoIP services.

The following domains are defined in this project:

- VoIPDomain

The following products are defined in this project:

- Value_Added_Features_PS
- VoIP_Adaptor_PS
- VoIP_Fax_Service_PS
- VoIP_Offer_Charge_Class
- VoIP_Phone_PS
- VoIP_PS
- VoIP_Soft_Phone_PS
- VoIP_Visual_Voicemail_PS
- VoIP_Voicemail_PS
- Web_Conferencing_PS

Conceptual Model Projects for Service Order Management

The following cartridges provide entities that are used by or referenced by service order management Order-to-Activate cartridges when the service option without calculate service order is used.

For more information about the conceptual model, see *Design Studio Concepts*.

OracleComms_Model_O2A_SOM_PS

The OracleComms_Model_O2A_SOM_PS project contains the products for service order management services when the service option without calculate service order is used.

The following products are defined in this project:

- SOM_Broadband_Bandwidth_PS
- SOM_Broadband_Modem_PS
- SOM_Broadband_PS
- SOM_Broadband_Router_PS

- SOM_Email_Service_PS
- SOM_Firewall_PS
- SOM_Internet_Media_PS
- SOM_Value_Added_Features_PS
- SOM_VoIP_Adaptor_PS
- SOM_VoIP_Fax_Service_PS
- SOM_VoIP_Phone_PS
- SOM_VoIP_PS
- SOM_VoIP_Service_Feature_Billing_Validation_PS
- SOM_VoIP_Service_Plan_Billing_Validation_PS
- SOM_VoIP_Soft_Phone_PS
- SOM_VoIP_Visual_Voicemail_PS
- SOM_VoIP_Voicemail_PS
- SOM_Web_Conferencing_PS

Oracle AIA Emulators

The Oracle AIA emulators are used in development and testing when Oracle AIA is not available.

Table 4–95 lists and describes the emulators contained in the OracleComms_OSM_O2A_Install project.

Note: In the Order to Cash solution, OSM interacts with billing, CRM, and Provisioning systems using Oracle AIA. It does not directly interact with Siebel CRM, BRM, and provisioning systems.

Table 4–95 Emulators in OSM

Name	Description
osm_AIASyncCustomerEmulator	Emulates Oracle AIA billing service (for example, BRM ABCS by generating response messages in EBM format for requests targeted at a billing provider to synchronize customer account details.
osm_AIAInitiateFulfillBillingEmulator	Emulates Oracle AIA billing service by generating response messages in EBM format for requests targeted at a billing provider to initiate or fulfill billing.
osm_AIAFalloutNotificationToOrderEmulator	Emulates Oracle AIA error handling by generating order fallout notification messages for faults targeted at Oracle AIA error handling. These are error faults generated by the external systems (such as Provisioning). Error faults are sent to Oracle AIA which then translate them into fallout notifications and sent to OSM central order management.
osm_AIATroubleTicketEmulator	Emulates an Oracle AIA trouble ticket Siebel CRM service by generating response messages in EBM format for requests targeted at Siebel CRM to create trouble tickets. Note that no trouble ticket response is generated for Update Trouble ticket EBMs but only for Create Trouble ticket EBMs.

Table 4–95 (Cont.) Emulators in OSM

Name	Description
osm_CF2LFProvisionOrderCreateEmulator	Emulates an OSM service (for example, OSM ABCS) for service order creation by wrapping EBM format messages in OSM format for requests targeted at OSM service order management fulfillment to process service orders.
osm_LF2CFProvisionOrderUpdateEmulator	Emulates an OSM service (for example, OSM ABCS) for order update by wrapping EBM format messages in OSM format for messages targeted at OSM central order management fulfillment to update service orders.
osm_AIAProvisionOrderEmulator	Emulates Oracle AIA Provisioning service (for example, order management) fulfillment by generating response messages in EBM format for requests targeted at OSM service order management fulfillment to process service orders.
osm_InventoryOrderEmulator	Emulates UIM by setting simulated enriched data from inventory such as Service ID and MAC Address.
osm_TomOrderEmulator	Emulates a technical order management system by returning a successful status to requests.

5 Extending Order-to-Activate Cartridges

This chapter describes how to extend the Order-to-Activate cartridges for Oracle Communications Order and Service Management (OSM).

The Order-to-Activate cartridges are provided as a working foundation which you can extend to design and build a solution. This chapter provides details and guidelines on how to extend the base model entities.

Adding Custom Data Elements

To add custom data elements to the Order-to-Activate cartridges, please see knowledge article 1514936.1, **Data Enrichment - Extending Order to Activate Cartridges**, on the Oracle support website:

`https://support.oracle.com`

Adding Custom Order Item Properties

You can add custom order item properties to your order template without unsealing any cartridges. The following XML-type variable is available in the COM_Sales_OrderFulfillment order template:

```
ControlData
    OrderItem
        CustomXmlData
```

This data element allows the addition of custom properties, but it does not support significance or revision, and it cannot be used for component wait dependencies.

You can populate custom properties into that element by using the following URI for the CustomXmlData property in the Order Item Specification editor:

```
http://xmlns.oracle.com/communications/ordermanagement/o2a/customextensions/Custom
XmlData.xquery
```

After installation, the shell for the **CustomXmlData.xquery** file is located in the *SolutionCartridge*/**resources/CustomExtensions** folder, where *SolutionCartridge* is the cartridge for your solution, as listed in Table 5–1:

Table 5–1 Solution Configurations and Corresponding Solution Cartridge Names

Using Calculate Service Order Option?	Current Workspace Is for:	Topology	Solution Cartridge
Yes	COM only	All	OracleComms_OSM_O2A_COM_CSO_Solution
Yes	SOM only	All	OracleComms_OSM_O2A_SOM_CSO_Solution
Yes	COM and SOM	All	OracleComms_OSM_O2A_COMSOM_CSO_Solution
No	COM only	Simple	OracleComms_OSM_O2A_COM_Simple_NP_Soln
No	COM only	Typical or Complex	OracleComms_OSM_O2A_COM_Typical_NP_Soln
No	SOM only	All	OracleComms_OSM_O2A_SOM_NP_Soln
No	COM and SOM	Simple	OracleComms_OSM_O2A_COMSOM_Simple_NP_Soln
No	COM and SOM	Typical or Complex	OracleComms_OSM_O2A_COMSOM_Typical_NP_Soln

Note: If you edit the **CustomXmlData.xquery** file, save a copy of the updated file to the **custom-extension** folder in the OracleComms_ OSM_O2A_Configuration cartridge. This ensures that your changes will not be overwritten if you reconfigure your workspace.

Following is a sample of a configured **CustomXmlData.xquery** file:

Example 5–1 Sample CustomXmlData.xquery File

```
import module namespace fulfillmentmodecodefn =
"http://xmlns.oracle.com/communications/ordermanagement/o2a/customextensions/fulfi
llmentmodecodefn" at
"http://xmlns.oracle.com/communications/ordermanagement/o2a/customextensions/Fulfi
llmentModeCode.xqy";
import module namespace fulfillmentcondprovfn =
"http://xmlns.oracle.com/communications/ordermanagement/o2a/customextensions/fulfi
llmentcondprovfn" at
"http://xmlns.oracle.com/communications/ordermanagement/o2a/customextensions/Fulfi
llmentCondProv.xqy";
import module namespace fulfillmentcondfn =
"http://xmlns.oracle.com/communications/ordermanagement/o2a/customextensions/fulfi
llmentcondfn" at
"http://xmlns.oracle.com/communications/ordermanagement/o2a/customextensions/Fulfi
llmentCond.xqy";
import module namespace completeshippingfn =
"http://xmlns.oracle.com/communications/ordermanagement/o2a/customextensions/compl
eteshippingfn" at
"http://xmlns.oracle.com/communications/ordermanagement/o2a/customextensions/Compl
eteShipping.xqy";

declare namespace prop = "COM_SalesOrderFulfillment";
declare namespace
salesord="http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/SalesOrder/V2";

declare variable $inputDoc as document-node() external;

let $ebm := $inputDoc/GetOrder.Response/_
root/messageXmlData/salesord:ProcessSalesOrderFulfillmentEBM
```

```
let $line := .

return
(
    <prop:CustomProperties>
        <prop:CompleteShipping>{completeshippingfn:CompleteShipping($line,
$ebm)}</prop:CompleteShipping>
        <prop:FulfillmentCond>{fulfillmentcondfn:FulfillmentCond($line,
$ebm)}</prop:FulfillmentCond>

<prop:FulfillmentCondProv>{fulfillmentcondprovfn:FulfillmentCondProv($line,
$ebm)}</prop:FulfillmentCondProv>

<prop:FulfillmentModeCode>{fulfillmentmodecodefn:FulfillmentModeCode($line,
$ebm)}</prop:FulfillmentModeCode>
    </prop:CustomProperties>
)
```

Changing Durations for Order Components

You can change the Optimistic, Most Likely, and Pessimistic order component durations. If you create custom functional order components, you can change the values in the Order Component Specification editor, like you would for any non-Order-to-Activate configuration. However, since the standard Order-to-Activate functional order components are in a sealed cartridge, you should not edit the order component durations in the normal way, but instead using the instructions in this section.

Note: As in non-Order-to-Activate scenarios, the duration settings for fulfillment patterns override the duration settings for order components. Since Order-to-Activate fulfillment patterns are not in sealed cartridges, you can edit durations for the fulfillment patterns in the normal way, using the Fulfillment Pattern editor in Design Studio. See Design Studio Modeling OSM Orchestration Help for more information about setting durations for fulfillment patterns.

To change durations for standard Order-to-Activate functional order components:

1. Edit the *workspace***OracleComms_OSM_O2A_Configuration\solution-config\ComponentDurationMap.xml** file, where *workspace* is the directory containing the files for your Order-to-Activate workspace.

   You can edit this file in any text or XML editor or in Eclipse. If you use Eclipse, use the Package Explorer view to find the file.

2. Find the entry for the order component you would like to modify. The name of the order component is located in the OrderComponentSpec tag. The example below shows the element tag for the SyncCustomerFunction functional order component:

   ```
   <model:orderComponentSpec name="SyncCustomerFunction" namespace="COM_
   SalesOrderFulfillment">
   ```

3. Update the durations you would like to change. The durations are in the standard XML date/time format:

   P*n*Y*n*M*n*DT*n*H*n*M*n*S

for example:

```
P0Y0M0DT0H2M0S
```

which indicates a duration of two minutes.

The individual elements of the format are:

- **P** indicates the period
- *n***Y** indicates the number of years
- *n***M** indicates the number of months
- *n***D** indicates the number of days
- **T** indicates the start of a time section
- *n***H** indicates the number of hours
- *n***M** indicates the number of minutes
- *n***S** indicates the number of seconds

4. Save and close the *workspace***\OracleComms_OSM_O2A_Configuration\solution-config\ComponentDurationMap.xml** file.
5. In Design Studio, go to the Ant view and look for the **SolutionConfig.xml** build file you added when you installed the Order-to-Activate cartridges.

   Note: If you have removed this file from your workspace, see "Configuring WebLogic Server Resources" and follow steps 1 through 4.

6. Expand the **SolutionConfig.xml** build file and double-click the **config_Metadata_And_ModelVariable** target.

   This updates the metadata so that the system will use the new values you have configured.

Adding a New Fulfillment Function

A fulfillment function represents an activity, for example billing or provisioning, that must be performed to process an order item. You can add a new fulfillment function for a new action or you can extend an existing fulfillment function to add data elements and entities without unsealing the productized Order-to-Activate cartridges.

Planning the Addition of a New Fulfillment Function

This section contains planning considerations for adding a new fulfillment function to a solution.

- Is the new fulfillment function for a system type that is already modeled in the Order-to-Activate cartridge, or is it for a new system type?
  - If the new fulfillment function is for a new system type, you must know the naming convention configured in Oracle Application Integration Architecture (Oracle AIA) deployments for logical identifiers of instances of the new system type. See the coverage of **EBMHeader/Sender/ID and EBMHeader/Target/ID** elements for the various system interactions in *Fusion*

  Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack. The Sender IDs and Target IDs in the EBM messages must match the logical identifiers for the system instances configured in the Oracle AIA deployment. See "Considerations When Integrating with Oracle AIA."

  – Following is a summary of naming conventions for fulfillment functions in Oracle AIA:

    * Naming convention used for OSM central order management instances: OSMCFS_01, OSMCFS_02, and so on.
    * Naming convention used for OSM service order management instances: OSMPROV_01, OSMPROV_02, and so on.
    * Naming convention used for Billing and Revenue Management instances: BRM_01, BRM_02, and so on.
    * Naming convention used for Siebel Customer Relationship Management (Siebel CRM) instances: SEBL_01, SEBL_02, and so on.

- What is the message format for the fulfillment function request and response?
  - The fulfillment function request must conform to an Enterprise Business Message (EBM).
  - The fulfillment function response can be either an EBM or an OrderFalloutNotification. Oracle AIA specifies XML schemas for the EBMs which describe the request and response format.
- What are the JNDI names for the source and destination queues that are used to exchange request and response messages between OSM and Oracle AIA for the new fulfillment function?
- What are the data elements on the order that are compensation-significant for the new fulfillment function? Compensation-significant data elements trigger compensation if a revision order contains changes to the values.
- What service action codes will apply to the current fulfillment function? The predefined service action codes include:
  - ADD: Add a new service.
  - UPDATE: Make a change to an existing service.
  - DELETE: Remove a service.
  - SUSPEND: Suspend a service.
  - RESUME: Resume a suspended service.
  - MOVE-ADD: Add service as part of a service location move process.
  - MOVE-DELETE: Remove service as part of a service location move process.
  - NONE: Make no service change.
- What processing is required for the different execution modes? The following execution modes supported in the existing Order-to-Activate fulfillment function and are mandatory:
  - *do* is required to handle new orders and change orders
  - *redo* is required for amendment processing to fulfill an order revision
  - *undo* is required for order cancellation

- What data elements from the fulfillment request XML schema must be included in the outbound fulfillment request?
  - Identify the data elements that must be copied as-is by OSM from the customer order to the fulfillment request. This data exists in the OSM order data in the **messageXMLData** element and does not need to be modeled separately.
  - Identify the data that must be generated by OSM and inserted into the fulfillment request. Consider data generation for all OSM execution modes: *do*, *redo*, and *undo*.
- What data elements from the fulfillment response XML schema must be included in data updates to Siebel CRM or included in subsequent fulfillment function requests? For example, you might include a service ID returned from provisioning that must be included in the fulfillment-data updates to Siebel CRM and in the FulfillBilling request that takes place after provisioning is complete.

Response Patterns in System Interactions

There are two patterns of handling responses in system interactions, the single-response pattern and the multiple-response pattern. Single-response patterns receive a single response in a system interaction with the Oracle AIA billing service, while multiple-response patterns can receive multiple responses.

Single Response Pattern

The single-response pattern is used in billing functions. Each billing function is transactional. In a system interaction with the Oracle AIA billing service, a single response is expected which is either a response EBM or an OrderFalloutNotification created by Oracle AIA error handling framework, and not both.

If a revision order with compensation-significant updates for the billing function arrives while the request is in progress in OSM central order management (that is, after an EBM has been put on the queue for the Oracle AIA billing service to pick up but before the arrival and processing of a response), the amendment is queued until the response is processed.

In this case, a single automated task in the subprocess for the fulfillment function (such as SyncCustomerSITask, InitiateBillingSITask, or FulfillBillingSITask) includes automation to generate the EBM and put it on a queue in *do*, *redo*, and *undo* execution modes, and the same automated task also includes automation to correlate and process the response from Oracle AIA.

Multiple Response Pattern

The multiple-response pattern is used in the ProvisionOrderFunction. In a system interaction with the Oracle AIA order provisioning service, multiple responses are expected in the form of a sequence of response EBMs of type ProcessFulfillmentOrderUpdateEBM or an OrderFalloutNotification created by the Oracle AIA error-handling framework.

If a revision order with compensation-significant updates for the ProvisionOrderFunction arrives while a request is in progress in OSM central order management (that is, after a request EBM has been put on the queue for the Oracle AIA provisioning service to pick up but before the arrival and processing of a response), then a separate request for the revision is sent to the Oracle AIA order provisioning service.

In this case, a separate automated task, such as ProvisionOrderSIResponseTask, is needed in the subprocess to correlate and process the responses. The success flow without debugging breakpoints then becomes:

1. *YourFunctionName*EntryPointTask
2. *YourFunctionName*SITask
3. *YourFunctionName*SIResponseTask

Entities to Create, Modify, or Reuse

Table 5–2 provides an overview of the entities that may be created, modified, or reused in the creation of a fulfillment function. Further information about many of these entities is provided in the following sections.

Table 5–2 Entities to be Created, Modified, or Reused

Name	Type	Remarks
OracleComms_OSM_CommonDataDictionary OracleComms_OSM_O2A_AIAEBMDataDictionary	Data Dictionary	See "Data Dictionary and Order Templates" for information about how to make changes for the new fulfillment function.
COM_SalesOrderFulfillment_CreationTask	Manual Task	See "About Creation Tasks" for information about adding to task data for the new fulfillment function.
COM_SalesOrderFulfillment_OrderDetails	Manual Task	See "About Query Tasks" for information about adding to task data for the new fulfillment function.
FUNCTION/*YourSystemType*Function extends FulfillmentFunction FUNCTION/*YourFunctionName*Function extends *YourSystemType*Function SYSTEM/*YourSystemType*System extends FulfillmentSystem UPDATES/*YourFunctionName*SignificantUpdates extends SignificantUpdates	Order Component Specification	If your system type is an already modeled system type such as Billing, you reuse the existing entities, for example: FUNCTION/BillingFunction and SYSTEM/BillingSystem Existing GRANULARITY order component specifications are reusable for new fulfillment functions without any modifications.
SUBPROCESS/*YourFunctionName*SubProcess	Process	See "About Subprocesses."
TASK/*YourFunctionName*EntryPointTask TASK/*YourFunctionName*SITask	Automated Task	See "About Subprocesses" for automated tasks to be created for the new subprocess for the new fulfillment function.
TASK/*YourFunctionName*PreSITask TASK/*YourFunctionName*PostSITask TASK/*YourFunctionName*FalloutRecoverTask TASK/*YourFunctionName*WaitForAmendmentTask TASK/*YourFunctionName*EPQTask extends TASK/*YourFunctionName*EntryPointTask TASK/*YourFunctionName*SIQTask extends TASK/*YourFunctionName*SITask	Manual Task	See "About Subprocesses" for manual tasks to be created for the new subprocess for the new fulfillment function.

Data Dictionary and Order Templates

Additional data fields for the new fulfillment function can be defined in the cartridge created for it. All order template entities can be placed in a separate cartridge, in an order that extends from COM_SalesOrderFulfillment. Orchestration sequences, processes, and tasks for the new fulfillment function can also be placed in this separate cartridge.

An XML data type element named **messageXMLdata** is used to store the incoming customer order data in an XML format inside the OracleComms_OSM_O2A_COM_Base cartridge. This element is defined in the OracleCgbuCommonDataDictionary data dictionary and is included in the order template. It should be added to any new tasks that require access to the raw customer order data. The raw data is used by automated tasks that copy some data as-is to the fulfillment request.

Model additional fields in the **LineItemData** structure in the data dictionary. The element names, types, and sub-structures in **LineItemData** mimic the structure of the SalesOrderLine structure in the Oracle AIA SalesOrderEBM schema. The following sections contain information about these changes.

Order Change Management Configuration

You may need to model data for order change management configuration including keys and data significance. Add all data elements from the customer order line that are compensation-significant for the fulfillment function to the **LineItemData** structure (if they do not already exist in the **LineItemData** structure).

Customize a copy of the **BaseLineItemData.xquery** XQuery file located in **OracleComms_OSM_O2A_COM_Base/resources/OrderItemProperties** to add code to copy compensation-significant data for the new fulfillment function from the customer order line to the order line item property **BaseLineItemData**.

Note: Use XML catalogs to specify the location of XQuery files. Use a unique namespace prefix to avoid naming conflicts.

Data Required for Sending the Fulfillment Request or Processing the Fulfillment Request Response

You may need to model data that must be generated by OSM and copied to the fulfillment request, not including service order identification and service order line identification. Add these to the **LineItemData** structure (if they do not already exist in the **LineItemData** structure). An example of this is new and prior values for customized billing date calculations to send to a billing provider.

You may also need to model data that can be updated back to central order management from the fulfillment system.

Additional Control Data Required for Orchestration Logic

If the new fulfillment function can process order items with a service action code of UPDATE or MOVE-ADD, add an element such as **OrderItemHas***YourFunctionName***Updates** to the OrderItemControl structure. You can make a copy of **CommunicationsSalesOrderItemProperties_OrderItemControl.xqy** and edit the XQuery to set the property value for each order item. A value of **YES** means that this order item has relevant changes for your fulfillment function to process. A service action code has relevant changes if it is a service action code that the fulfillment function can process, or in the case of UPDATE or MOVE-ADD, a service action code with compensation-significant updates.

Add a condition to the decomposition rule that decomposes from *YourFunctionName*Function to *YourFunctionName* SignificantUpdates. The condition checks for the existence of at least one order item in the fromOrderComponent that has **OrderItemHas***YourFunctionName***Updates** set to **YES**. This avoids creating an executable order component (which avoids generating and sending a fulfillment request) when there are no order items with relevant changes for the fulfillment function to process.

Data that Must be Modeled in the Order Template

If any additional data must be modeled in the order template:

- Follow the same pattern as for the existing fulfillment functions model ControlData/Functions/*YourFunctionName*Function
- Any data that must be generated by OSM and copied to the fulfillment request or data that can be updated back to central order management from the fulfillment system is added to:

  ControlData/OrderItem/WorkLineItemData for OSM-generated new values, or new values from fulfillment function response

  ControlData/OrderItem/WorkPriorLineItemData for OSM-generated prior values

  ControlData/OrderItem/BaseLineItemData for significance information for compensation
- Any additional control data required for the orchestration logic

About Creation Tasks

COM_SalesOrderFulfillment_CreationTask is the creation task for the OSM **COM_SalesOrderFulfillment** order. With the Order-to-Activate composite cartridge, you can add data to this creation task through the task data contribution tab in the Order-to-Activate composite cartridge for the new fulfillment function.

About Query Tasks

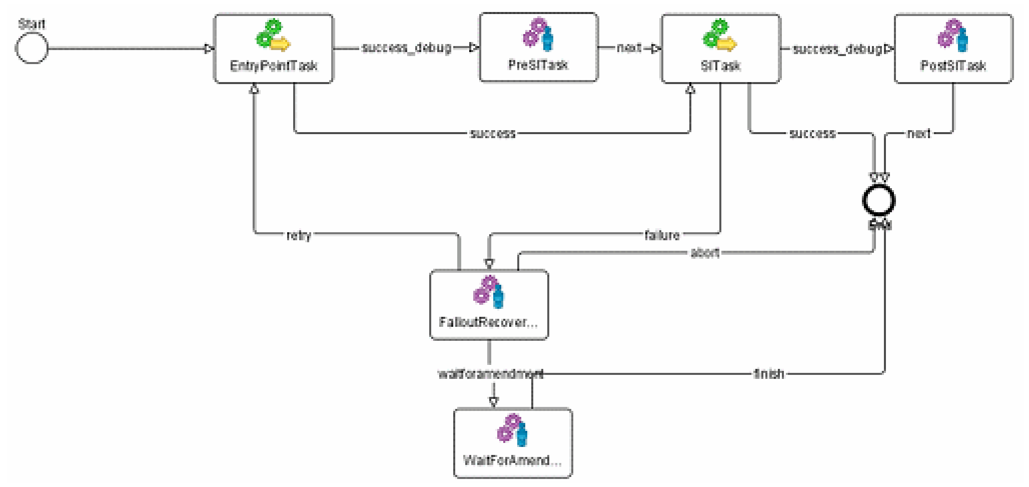
COM_SalesOrderFulfillment_OrderDetails is the query task for the OSM **COM_SalesOrderFulfillment** order. With the Order-to-Activate composite cartridge, you can add data to this query task through the task data contribution tab in the Order-to-Activate composite cartridge for the new fulfillment function.

About Subprocesses

A system interaction configured in the **OracleComms_OSM_O2A_COM_Base** cartridge handles the asynchronous communication of service order data to a fulfillment system instance. A fulfillment system instance is also referred to as a fulfillment provider. The system interaction should handle the delivery of a relevant subset of service order data to the fulfillment provider. When triggered, the system interaction also invokes the correct subprocess which represents the fulfillment function for the order component. Additionally, the system interaction must handle responses from the fulfillment provider and cope with messaging, fallout, status and data updates, and order change management.

Model the subprocess following the pattern established for the existing fulfillment functions as depicted in Figure 5–1. Prefix task names with *YourFunctionName*. The flow of the process is described in Table 5–3.

Figure 5–1 Creating Subprocesses



The manual tasks *YourFunctionName***PreSITask**, and *YourFunctionName***PostSITask** are optional. They are useful as cartridge breakpoints for a number of purposes including providing the user the ability to control process flow before and after functions and to examine data in the process flow for revision testing. Cartridge breakpoints stop at manual tasks in subprocess flows for system interactions. In a success flow, the process flows distinguish whether to exit with 'success_debug' status (to include manual tasks in the subprocess flow) or **success** status (to skip the manual tasks in the subprocess flow).

Table 5–3 lists the tasks and flows associated with subprocess.

Table 5–3 Flows and Tasks

Flow patterns	Tasks associated
Success flow without debug breakpoints	Start --> *YourFunctionName*EntryPointTask --> *YourFunctionName*SITask --> End
Success flow with debug breakpoints	Start --> *YourFunctionName*EntryPointTask --> *YourFunctionName*PreSITask (manual) --> *YourFunctionName*SITask --> *YourFunctionName*PostSITask (manual) --> End
Failure flows	Start --> *YourFunctionName*EntryPointTask --> *YourFunctionName*SITask -->*YourFunctionName*FalloutRecoverTask (manual) --> ▪ Option 1: abort End ▪ Option 2: wait for amendment *YourFunctionName*WaitForAmendmentTask (manual) --> End ▪ Option 3: retry

Significance must be set in the task data for the nodes in **CreateCommunicationsSalesOrderTask** and the tasks in the subprocess for the fulfillment function *YourFunctionName*.

The system interaction for the fulfillment function is implemented by a subprocess named *YourFunctionName*SubProcess in the Order-to-Activate cartridges. The automated tasks in the subprocess accomplish the following:

1. Accept input data from the original EBM message destined for the outbound message and properties for each of the configurable data elements.

- Input data to pass through from the original message (ProcessSalesOrderFulfillmentEBM) to the outbound message:
  - Reference to the message header (EBMHeader) from the original EBM
  - Reference to the customer order header (DataArea/ProcessSalesOrderFulfillment) from the original EBM
  - References to line items from the original EBM destined for the outbound message

  The inbound message is expected to conform to the schema **AIAComponents\EnterpriseObjectLibrary\Industry\Communications\EBO\SalesOrder\V2\SalesOrderEBM.xsd**

- Input data for the configurable elements of the outbound message:
  - EBMHeader/Sender/ID and all EBMHeader/Sender element values
  - EBMHeader/Target/ID

2. Automation actions in the automated task *YourFunctionName*EntryPointTask update the order item data required for the fulfillment function system interaction. Copy the **SIEntryPoint.xqy** file, and customize it as needed.

   Use XML catalogs to specify the location of the XQuery files. Use a unique namespace prefix to avoid naming conflicts.

   Table 5–4 lists the XML catalogs.

Table 5–4 XML Catalogs

Automation Action	Execution Modes	Automation Type	Event Type	Customize
'*YourFunctionName*EntryPointBean_doredo'	*do*, *redo*	XQuery Sender	Internal	Customize a copy of the SIEntryPoint.xqy XQuery file to add code to update order data before message generation for the new fulfillment function.

- Logic for *do* and *redo* execution modes (if applicable) include updating the order data in ControlData/Functions/*YourFunctionName*/orderItem/orderItemRef/WorkLineItemData. Because orderItemRef is a reference, subsequent functions on the same order line reference the same instance of the data. Any additional data that is generated by OSM for the fulfillment function request should be updated.

3. Automation actions *YourFunctionName*RequestBean_do, *YourFunctionName*RequestBean_redo, *YourFunctionName*RequestBean_undo in automated task *YourFunctionName*SITask are configured as internal XQuery Senders to use the XQuery automation plug-in to construct the payload for the outbound message, in *do*, *redo*, and *undo* modes, respectively (if applicable).

- Outbound message format: Identify the EBM
- Outbound message schema: Identify the Oracle AIA schema. For example, the schema for the billing functions is: **AIAComponents\EnterpriseObjectLibrary\Industry\Communications\EBO\FulfillmentOrder\V1\FulfillmentOrderEBM.xsd**.
- Relevant line items: all line items targeted at the same fulfillment provider

4. Configure automation actions in the automated task *YourFunctionName*SITask to generate messages in the EBM format and send the XML payload over JMS. You must specify the JNDI name of the JMS destination.

   Table 5–5 lists the automation actions and the XQueries to be customized.

Table 5–5 Automation Actions

Automation Action	Execution Modes	Automation Type	Event Type	Customize
YourFunctionName RequestBean_do	*do*	XQuery Sender	Internal	Customize a copy of the **AIAEBMRequest_do.xqy** XQuery file to add code to update the order data before message generation for the new fulfillment function.
YourFunctionName RequestBean_redo	*redo*	XQuery Sender	Internal	Customize a copy of the AIAEBMRequest_redo.xqy XQuery file to add code to update the order data before message generation for the new fulfillment function.
YourFunctionName RequestBean_ undo	*undo*	XQuery Sender	Internal	Customize a copy of the AIAEBMRequest_undo.xqy XQuery file to add code to update the order data before message generation for the new fulfillment function.

5. Configure automation action *YourFunctionName*ResponseBean in automated task *YourFunctionName*SITask as an external XQuery Automator to process responses. You must specify the JNDI name of the JMS source.

   Table 5–6 lists the automated XQueries.

Table 5–6 Automated XQueries

Automation action	Execution Modes	Automation Type	Event Type	Customize
YourFunctionName ResponseBean	N/A	XQuery Automater	External	Customize a copy of the AIAEBMResponse.xqy XQuery file to add code to update the order data before message generation for the new fulfillment function.

- Success response: Recognize and process a successful response. Set the reached milestone to *YOUR FUNCTION NAME*COMPLETE. A successful response is a well-formed response message, that conforms to the response EBM format, with an empty or non-existent EBMHeader/FaultNotification and FaultMessage Code.
- Failure response: Recognize and process failure responses that OSM expects to be either a response EBM, or an OrderFalloutNotification.
- No response: In this case OSM expects an OrderFalloutNotification from Oracle AIA.

Fulfillment Function Extension Point Interface

The Order-to-Activate cartridges use XQuery resources to perform functions including setting order item properties, mapping product specifications to fulfillment patterns, managing fulfillment function dependencies, and managing the order life cycle. One way to customize XQueries is to rewrite or add to a provided XQuery module and use the XML catalog to enable URI reference mapping. Fulfillment function extension points have different input parameters depending on whether you are using the calculate service order solution option or the solution option without calculate service order. Both sets of input parameters are provided in the tables in this section. Extension points are defined for both fulfillment functions and fulfillment states. This section contains information about the fulfillment function extension points. For information about the fulfillment state extension points, see "Fulfillment State Extension Point Interface."

XML catalogs are system-wide entities, which means an XML Catalog specified in one cartridge will be used when processing requests for orders on other cartridges. With the use of solution cartridges, multiple solutions can be deployed to a single system and coexist with each other.

Each fulfillment function extension point has one XQuery API except for CREATE-EBM, which has three: one for each execution mode (*do*, *redo*, and *undo*).

An XQuery extension script must be implemented in a standalone file. The file URI must be registered to the extension configuration.

Fulfillment Function Extension Point Overview

Table 5–7 lists the XQuery extension points for fulfillment functions in the Order-to-Activate cartridges.

Table 5–7 Fulfillment Function Extension Points

Fulfillment Function Extension Point	Description
COMPONENT-START	Fulfillment function start extension point. The extension is expected to return a list of OrderItem properties to be updated when the fulfillment function is started.
COMPONENT-COMPLETE	Fulfillment function complete extension point. The extension is expected to return a list of OrderItem properties to be updated when the fulfillment function is completed.
COMPONENT-UPDATE	Fulfillment function update extension point. The extension is expected to return a list of OrderItem properties to be updated when the fulfillment function is updated.
CREATE-EBM	Fulfillment function create payload extension point. The extension is expected to return the EBM to be sent to the external system in *do*, *redo*, or *undo* mode operations.
CREATE-EBM-CUSTOM	Fulfillment function create payload extension point with order level Custom XML element in the EBM.
CREATE-EBM-ALL-ORDERITEMS	Fulfillment function create payload extension point for all order items. The extension is expected to return the EBM to be sent to the external system.
CREATE-EBM-ORDERITEM	Fulfillment function extension point to create an XML fragment for a single order item in *do*, *redo*, or *undo* mode operations.
CREATE-EBM-ORDERITEM-CUSTOM	Fulfillment function extension point to create an XML fragment for a single order item that has an order-item-level Custom XML element in the EBM.
CREATE-EBM-PRIORORDERITEM	Fulfillment function extension point to create an XML fragment for a single prior order item.
CREATE-EBM-PRIORORDERITEM-CUSTOM	Fulfillment function extension point to create a payload with prior-order-item-level Custom XML element in the EBM.

Table 5–7 (Cont.) Fulfillment Function Extension Points

Fulfillment Function Extension Point	Description
VALIDATE-RESPONSE-EBM	Fulfillment function response validation extension point. The extension is expected to validate the EBM response coming back from the external system.
COMPONENT-RESPONSE-UPDATE	Fulfillment function response update extension point. The extension is expected to return a list of OrderItem properties to be updated when a valid EBM response comes back from the external system.
ORDER-EXTENSION-UPDATE-STATUS-EBM	Fulfillment function update extension point for status updates from central order management to the upstream system. The extension is expected to return an EBM containing sales order status and other information. Other systems can also listen to the output of this extension point to create or update asset information.

When a fulfillment function is introduced, you can create an ExtensionPointMap entry for each applicable fulfillment function extension point (such as Component Start) in the **resources\SolutionConfig\ComponentExtensionPointMap.xml** of the Order-to-Activate composite cartridge. You must create a separate XQuery file for each fulfillment function extension point.

COMPONENT-START Extension Point

This section describes the XQuery script that implements the logic to handle the COMPONENT-START extension point.

Table 5–8 lists the input parameters for the extension point XQuery when you are using the calculate service order solution option.

Table 5–8 COMPONENT-START Input Parameters for the Calculate Service Order Option

Name	Type	Scope	Description
$extensionVersion	xs:string	External variable	Version number of the extension framework
$log	Java Object	External variable	Java Type org.apache.commons.logging.Log Logging level related to server log.
$executionMode	xs:string	External variable	Task execution mode
$breakpointDebugControl	element()	External variable	Break Point Control XML fragment
$taskInputData	element()	External variable	Task data XML fragment with the schema for GetOrder.Response
$component	element()	External variable	XML fragment containing the fulfillment function data
.	element()	Context node	Order item data XML fragment

Table 5–9 lists the input parameters for the extension point XQuery when you are using the solution option without calculate service order.

Table 5–9 COMPONENT-START Input Parameters for the Option Without Calculate Service Order

Name	Type	Scope	Description
$extensionVersion	xs:string	External variable	Version number of the extension framework
$log	Java Object	External variable	Java Type org.apache.commons.logging.Log Logging level related to server log.
$executionMode	xs:string	External variable	Task execution mode
$breakpointDebugControl	element()	External variable	Break Point Control XML fragment
$taskInputData	element()	External variable	Task data XML fragment with the schema for GetOrder.Response
.	element()	Context node	Order item data XML fragment

Table 5–10 lists the return parameters for the extension point XQuery.

Table 5–10 COMPONENT-START Return Parameters

Output Parameter Type	Description
element()*	XML wrapper element that contains all the order item properties to be updated

Example 5–2 is a sample XQuery code fragment for the COMPONENT-START fulfillment function extension point.

Example 5–2 COMPONENT-START XQuery Code Fragment

```
import module namespace YourFunctionNamefn =
"http://xmlns.oracle.com/communications/ordermanagement/pip/YourFunctionNamefn" at
"http://xmlns.oracle.com/communications/ordermanagement/pip/YourFunctionNamefn/You
rFunctionNameInteractionModule.xquery";
declare namespace oms = "urn:com:metasolv:oms:xmlapi:1";

declare variable $executionMode external;
declare variable $breakpointDebugControl external;
declare variable $taskInputData external;

(: This function registers to the YourFunctionName/START event.
 : It returns a list of elements which are the properties to be updated for
 : the given order item. :)
declare function YourFunctionNamefn:onYourFunctionNameStart(
    $execMode as xs:string,
    $lineItem  as element(),
    $taskData as element(),
    $debugControl as element()) as element()*
{
    let $id := $lineItem/oms:BaseLineId
    return
        <BaseLineId>{ $id/text() }</BaseLineId>,
        (: list of order item properties to be updated :)
};

let $lineItem := .
return
    <OrderItem>
    {
        YourFunctionNamefn:onYourFunctionNameStart($executionMode, $lineItem,
$taskInputData, $breakpointDebugControl)
    }
    </OrderItem>
```

COMPONENT-COMPLETE Extension Point

This section describes the XQuery script that implements the logic to handle the COMPONENT-COMPLETE extension point.

Table 5–11 lists the input parameters for the extension point XQuery when you are using the calculate service order solution option.

Table 5–11 COMPONENT-COMPLETE Input Parameters for the Calculate Service Order Option

Name	Type	Scope	Description
$extensionVersion	xs:string	External variable	Version number of the extension framework
$log	Java Object	External variable	Java Type org.apache.commons.logging.Log Logging level related to server log
$component	element()	External variable	XML fragment containing the fulfillment function data
$executionMode	xs:string	External variable	Task execution mode
$orderItemFromResponse	element()	External variable	Order item data from the response message
$hasFallout	xs:boolean	External variable	Boolean indicator of whether the previous EBM request sent to the external system has had fallout
$falloutMessage	xs:string	External variable	The fallout error message of this order item
$breakpointDebugControl	element()	External variable	Break Point Control XML fragment
$taskInputData	element()	External variable	Task data XML fragment with the schema for GetOrder.Response
.	element()	Context node	Order item data XML fragment

Table 5–12 lists the input parameters for the extension point XQuery when you are using the solution option without calculate service order.

Table 5–12 COMPONENT-COMPLETE Input Parameters for the Option Without Calculate Service Order

Name	Type	Scope	Description
$extensionVersion	xs:string	External variable	Version number of the extension framework
$log	Java Object	External variable	Java Type org.apache.commons.logging.Log Logging level related to server log
$executionMode	xs:string	External variable	Task execution mode
$hasFallout	xs:boolean	External variable	Boolean indicator of whether the previous EBM request sent to the external system has had fallout
$falloutMessage	xs:string	External variable	The fallout error message of this order item

Table 5–12 (Cont.) COMPONENT-COMPLETE Input Parameters for the Option Without Calculate Service

Name	Type	Scope	Description
$breakpointDebugControl	element()	External variable	Break Point Control XML fragment
$taskInputData	element()	External variable	Task data XML fragment with the schema for GetOrder.Response
.	element()	Context node	Order item data XML fragment

Table 5–13 lists the return parameters for the extension point XQuery.

Table 5–13 COMPONENT-COMPLETE Return Parameters

Output Parameter Type	Description
element()*	XML wrapper element which contains all the order item properties to be updated

Example 5–3 is a sample XQuery code fragment for the COMPONENT-COMPLETE fulfillment function extension point.

Example 5–3 COMPONENT-COMPLETE XQuery Code Fragment

```
import module namespace pipextensionmodule =
"http://xmlns.oracle.com/communications/ordermanagement/pip/pipextensionmodule" at
"http://xmlns.oracle.com/communications/ordermanagement/pip/pipextensionmodule/Ext
ensionPointModule.xquery";
import module namespace YourFunctionNamefn =
"http://xmlns.oracle.com/communications/ordermanagement/pip/YourFunctionNamefn" at
"http://xmlns.oracle.com/communications/ordermanagement/pip/YourFunctionNamefn/You
rFunctionNameInteractionModule.xquery";

declare namespace oms = "urn:com:metasolv:oms:xmlapi:1";

declare variable $hasFallout external;
declare variable $falloutMessage external;

(: This function register to the YourFunctionName/COMPLETE event.
 : It return a list of elements that are the properties to be updated for
 : the given order item. :)
declare function YourFunctionNamefn:onYourFunctionNameComplete(
    $lineItem as element(),
    $hasFallout as xs:boolean,
    $falloutMessage as xs:string*) as element()*
{
    let $id := $lineItem/oms:BaseLineId
    return
        <BaseLineId>{ $id/text() }</BaseLineId>,
        (: list of order item properties to be updated :)
};

let $lineItem := .
return
    <OrderItem>
    {
        YourFunctionNamefn:onYourFunctionNameComplete($lineItem, $hasFallout,
pipextensionmodule:unWrapStringParameter($falloutMessage))
    }
    </OrderItem>
```

COMPONENT-UPDATE Extension Point

This section describes the XQuery script that implements the logic to handle the COMPONENT-UPDATE extension point.

Table 5–14 lists the input parameters for the extension point XQuery when you are using the calculate service order solution option.

Table 5–14 COMPONENT-UPDATE Input Parameters for the Calculate Service Order Option

Name	Type	Scope	Description
$extensionVersion	xs:string	External variable	Version number of the extension framework
$log	Java Object	External variable	Java Type org.apache.commons.logging.Log Logging level related to server log
$executionMode	xs:string	External variable	Task execution mode
$hasFallout	xs:boolean	External variable	Boolean indicator of whether the previous EBM request sent to the external system has had fallout
$falloutMessage	xs:string	External variable	The fallout error message of this order item
$breakpointDebugControl	element()	External variable	Break Point Control XML fragment
$taskInputData	element()	External variable	Task data XML fragment with the schema for GetOrder.Response
$milestoneCode	xs:string	External variable	Injected milestone code
.	element()	Context node	Order item data XML fragment

Table 5–15 lists the input parameters for the extension point XQuery when you are using the solution option without calculate service order.

Table 5–15 COMPONENT-UPDATE Input Parameters for the Option Without Calculate Service Order

Name	Type	Scope	Description
$extensionVersion	xs:string	External variable	Version number of the extension framework
$log	Java Object	External variable	Java Type org.apache.commons.logging.Log Logging level related to server log
$executionMode	xs:string	External variable	Task execution mode
$hasFallout	xs:boolean	External variable	Boolean indicator of whether the previous EBM request sent to the external system has had fallout
$falloutMessage	xs:string	External variable	The fallout error message of this order item
$breakpointDebugControl	element()	External variable	Break Point Control XML fragment
$taskInputData	element()	External variable	Task data XML fragment with the schema for GetOrder.Response
$milestoneCode	xs:string	External variable	Injected milestone code
.	element()	Context node	Order item data XML fragment

Table 5–16 lists the return parameters for the extension point XQuery.

Table 5–16 COMPONENT-UPDATE Return Parameters

Output Parameter Type	Description
element()*	XML wrapper element which contains all the order item properties to be updated

Example 5–4 is a sample XQuery code fragment for the COMPONENT-UPDATE fulfillment function extension point.

Example 5–4 COMPONENT-UPDATE XQuery Code Fragment

```
import module namespace pipextensionmodule =
"http://xmlns.oracle.com/communications/ordermanagement/pip/pipextensionmodule" at
"http://xmlns.oracle.com/communications/ordermanagement/pip/pipextensionmodule/Ext
ensionPointModule.xquery";
import module namespace YourFunctionNamefn =
"http://xmlns.oracle.com/communications/ordermanagement/pip/YourFunctionNamefn" at
"http://xmlns.oracle.com/communications/ordermanagement/pip/YourFunctionNamefn/You
rFunctionNameInteractionModule.xquery";
import module namespace pipbreakpointfn =
"http://xmlns.oracle.com/communications/ordermanagement/pip/pipbreakpointmodule"
at
"http://xmlns.oracle.com/communications/ordermanagement/pip/pipbreakpointmodule/Br
eakpointControlModule.xquery";

declare namespace oms = "urn:com:metasolv:oms:xmlapi:1";
declare namespace
salesord="http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/SalesOrder/V2";

declare variable $executionMode external;
declare variable $breakpointDebugControl external;
declare variable $taskInputData external;
declare variable $hasFallout external;
declare variable $falloutMessage external;
declare variable $milestoneCode external;

declare function YourFunctionNamefn:onYourFunctionNameUpdate(
    $execMode as xs:string,
    $lineItem as element(),
    $taskData as element(),
    $debugControl as element(),
    $milestoneCode as xs:string) as element()*
{
    let $ponrOverride := pipbreakpointfn:checkPONROverride(YourFunctionNamefn,
$debugControl)
    let $revisionPermissibleCode := if ($ponrOverride=fn:false()) then "HARD" else
"NOT YET"
    let $updateRevisionPermissibleCode := ($taskData/oms:_
root/oms:CustomerHeaders/oms:FulfillmentModeCode/text()!="TSQ")
    return
        YourFunctionNamefn:onYourFunctionNameUpdate($execMode, $lineItem,
$taskData, $milestoneCode, $updateRevisionPermissibleCode,
$revisionPermissibleCode)
};

let $lineItem := .
return
```

```
    <OrderItem>
    {
        YourFunctionNamefn:onYourFunctionNameUpdate($executionMode, $lineItem,
$taskInputData, $breakpointDebugControl, $milestoneCode)
    }
    </OrderItem>
```

CREATE-EBM Extension Point for *do* Execution Mode

This section describes the XQuery script that implements the logic to handle the CREATE-EBM extension point for *do* execution mode.

Table 5–17 lists the input parameters for the extension point XQuery when you are using the calculate service order solution option.

Table 5–17 CREATE-EBM Input Parameters for do Execution Mode for the Calculate Service Order Option

Name	Type	Scope	Description
$extensionVersion	xs:string	External variable	Version number of the extension framework
$log	Java Object	External variable	Java Type org.apache.commons.logging.Log Logging level related to server log
$orderId	xs:string	External variable	OSM Order ID of the current order
$orderKey	xs:string	External variable	AIA Order Number
$ebmHeader	element()	External variable	SalesOrder request EBM header: This element can be used as a reference to populate the request EBM header
$salesOrderLine	element()	External variable	XML fragment of all order lines belonging to the current fulfillment function
$transformedOrderLines	element()	External variable	XML fragment of all transformed order lines belonging to the current fulfillment function
$mappingContext	element()	External variable	XML fragment describing the mapping context between all sales order items and transformed order items belonging to the current fulfillment function
$priorSalesOrderLine	element()	External variable	All prior order lines belonging to the current fulfillment function
$customerHeaders	element()	External variable	XML fragment describing the mapping of the CustomerHeader structure
$component	element()	External variable	XML fragment containing the fulfillment function data
$targetIdentifier	element()	External variable	XML fragment describing the target system information
$idMap	element()	External variable	XML fragment describing the mapping between the original order line's BusinessComponentID and the newly generated BusinessComponentID to be populated into the EBM request message
$ebmId	xs:string	External variable	EBM ID to be populated into the EBM request message

Table 5–17 (Cont.) CREATE-EBM Input Parameters for do Execution Mode for the Calculate Service Order

Name	Type	Scope	Description
$fulfillmentOrderId	xs:string	External variable	BusinessComponentID to be populated into the EBM request message as the Order ID
$fulfillmentOrderNumber	xs:string	External variable	Cross-system order number reference
$faultMode	xs:string	External variable	FaultMode code to control how the emulator generates the response message: this parameter is in effect only if the request EBM is sent to an external system emulator rather than a real system.
$verbCode	xs:string	External variable	FaultMode code to control how the service order management orchestration order generates the response message. This element only applies if the request EBM is sent to service order management (SOM).
.	element()	Context node	Fulfillment order header for the SalesOrder request EBM

Table 5–18 lists the input parameters for the extension point XQuery when you are using the solution option without calculate service order.

Table 5–18 CREATE-EBM Input Parameters for do Execution Mode for the Option Without Calculate Service Order

Name	Type	Scope	Description
$extensionVersion	xs:string	External variable	Version number of the extension framework
$log	Java Object	External variable	Java Type org.apache.commons.logging.Log Logging level related to server log.
$ebmHeader	element()	External variable	SalesOrder request EBM header: This element can be used as a reference to populate the request EBM header
$salesOrderLine	element()	External variable	XML fragment of all order lines belonging to the current fulfillment function
$priorSalesOrderLine	element()	External variable	All prior order lines belonging to the current fulfillment function
$component	element()	External variable	XML fragment containing the fulfillment function data
$targetIdentifier	element()	External variable	XML fragment describing the target system information
$idMap	element()	External variable	XML fragment describing the mapping between the original order line's BusinessComponentID and the newly generated BusinessComponentID to be populated into the EBM request message
$ebmId	xs:string	External variable	EBM ID to be populated into the EBM request message
$fulfillmentOrderId	xs:string	External variable	BusinessComponentID to be populated into the EBM request message as the Order ID

Table 5–18 (Cont.) CREATE-EBM Input Parameters for do Execution Mode for the Option Without Calculate Service Order

Name	Type	Scope	Description
$fulfillmentOrderNumber	xs:string	External variable	Cross-system order number reference
$faultMode	xs:string	External variable	FaultMode code to control how the emulator generates the response message: this parameter is in effect only if the request EBM is sent to an external system emulator rather than a real system.
$verbCode	xs:string	External variable	FaultMode code to control how the service order management orchestration order generates the response message. This element only applies if the request EBM is sent to service order management (SOM).
.	element()	Context node	Fulfillment order header for the SalesOrder request EBM

Table 5–19 lists the return parameters for the extension point XQuery.

Table 5–19 CREATE-EBM for do Execution Mode Return Parameters

Output Parameter Type	Description
element()?	XML wrapper element that contains the EBM: The EBM format depends on external fulfillment provider requirements.

Example 5–5 is a sample XQuery code fragment for the CREATE-EBM fulfillment function extension point implementation for the *do* execution mode.

Example 5–5 CREATE-EBM XQuery Code Fragment for do Execution Mode

```
import module namespace YourFunctionNamefn =
"http://xmlns.oracle.com/communications/ordermanagement/pip/YourFunctionNamefn" at
"http://xmlns.oracle.com/communications/ordermanagement/pip/YourFunctionNamefn/You
rFunctionNameInteractionModule.xquery";

declare namespace oms = "urn:com:metasolv:oms:xmlapi:1";

declare variable $ebmHeader external;
declare variable $salesOrderLine external;
declare variable $priorSalesOrderLine external;
declare variable $component external;
declare variable $targetIdentifier external;
declare variable $idMap external;
declare variable $ebmId external;
declare variable $fulfillmentOrderId external;
declare variable $fulfillmentOrderNumber external;
declare variable $faultMode external;

let $fulfillmentOrder := .
return
    <Ebm>
    {
        YourFunctionNamefn:createDoYourFunctionNamePayload(
            $ebmHeader,
            $fulfillmentOrder,
            $salesOrderLine,
            $priorSalesOrderLine,
            $targetIdentifier,
            $idMap,
            $ebmId,
            $fulfillmentOrderId,
            $fulfillmentOrderNumber,
            $faultMode)
    }
    </Ebm>
```

CREATE-EBM Extension Point for *redo* Execution Mode

This section describes the XQuery script that implements the logic to handle the CREATE-EBM extension point for the *redo* execution mode.

Table 5–20 lists the input parameters for the extension point XQuery when you are using the calculate service order solution option.

Table 5–20 CREATE-EBM Input Parameters for redo Execution Mode for the Calculate Service Order Option

Name	Type	Scope	Description
$extensionVersion	xs:string	External variable	Version number of the extension framework
$log	Java Object	External variable	Java Type org.apache.commons.logging.Log Logging level related to server log
$orderId	xs:string	External variable	OSM Order ID of the current order
$orderKey	xs:string	External variable	AIA Order Number
$ebmHeader	element()	External variable	SalesOrder request EBM header: This element can be used as a reference to populate the request EBM header
$salesOrderLine	element()	External variable	XML fragment of all order lines belonging to the current fulfillment function
$transformedOrderLines	element()	External variable	XML fragment of all transformed order lines belonging to the current fulfillment function
$mappingContext	element()	External variable	XML fragment describing the mapping context between all sales order items and transformed order items belonging to the current fulfillment function
$histTransformedOrderLines	element()	External variable	XML fragment of all transformed order items belonging to the current fulfillment function before the revision
$histMappingContext	element()	External variable	XML fragment describing the pre-revision mapping context between all sales order items and transformed order items belonging to the current fulfillment function
$priorSalesOrderLine	element()	External variable	All prior order lines belonging to the current fulfillment function
$histSalesOrderLine	element()	External variable	All order lines belonging to the current fulfillment function before amendment
$histPriorSalesOrderLine	element()	External variable	All prior order lines belonging to the current fulfillment function before amendment

Table 5–20 (Cont.) CREATE-EBM Input Parameters for redo Execution Mode for the Calculate Service Order Option

Name	Type	Scope	Description
$deletedlineItems	element()	External variable	Order lines that were deleted by the amendment
$deletedTransformedLineItems	element()	External variable	XML fragment of all deleted transformed order lines belonging to the current fulfillment function
$deletedpriorlineItems	element()	External variable	Prior order line data that was deleted by the amendment
$deletedMappingContext	element()	External variable	XML fragment describing the deleted mapping context for all sales order items and transformed order items belonging to the current fulfillment function
$addedlineItems	element()	External variable	Order line data that was added by the amendment
$component	element()	External variable	XML fragment containing the fulfillment function data
$histComponent	element()	External variable	XML fragment with the pre-amendment fulfillment function data
$targetIdentifier	element()	External variable	XML fragment describing the target system information
$idMap	element()	External variable	XML fragment describing the mapping between the original order line's BusinessComponentID and the newly generated BusinessComponentID to be populated into the EBM request message
$histIdMap	element()	External variable	XML fragment describing the mapping between the original order line's BusinessComponentID and the BusinessComponentID populated into the earlier EBM request message
$ebmId	xs:string	External variable	EBM ID to be populated into the EBM request message
$fulfillmentOrderId	xs:string	External variable	BusinessComponentID to be populated into the EBM request message as the Order ID
$fulfillmentOrderNumber	xs:string	External variable	Cross-system order number reference
$hasFallout	xs:boolean	External variable	Boolean indicator of whether the previous EBM request sent to the external system has had fallout
$faultMode	xs:string	External variable	FaultMode code to control how the emulator generates the response message: this parameter is in effect only if the request EBM is sent to an external system emulator rather than a real system.

Table 5–20 (Cont.) CREATE-EBM Input Parameters for redo Execution Mode for the Calculate Service Order Option

Name	Type	Scope	Description
$verbCode	xs:string	External variable	FaultMode code to control how the service order management orchestration order generates the response message. This element only applies if the request EBM is sent to service order management (SOM).
$customerHeaders	element()	External variable	XML fragment describing the mapping of the CustomerHeader structure
.	element()	Context node	Fulfillment order header for the SalesOrder request EBM

Table 5–21 lists the input parameters for the extension point XQuery when you are using the solution option without calculate service order.

Table 5–21 CREATE-EBM Input Parameters for redo Execution Mode for the Option Without Calculate Service Order

Name	Type	Scope	Description
$extensionVersion	xs:string	External variable	Version number of the extension framework
$log	Java Object	External variable	Java Type org.apache.commons.logging.Log Logging level related to server log
$ebmHeader	element()	External variable	SalesOrder request EBM header: This element can be used as a reference to populate the request EBM header
$salesOrderLine	element()	External variable	XML fragment of all order lines belonging to the current fulfillment function
$priorSalesOrderLine	element()	External variable	All prior order lines belonging to the current fulfillment function
$histSalesOrderLine	element()	External variable	All order lines belonging to the current fulfillment function before amendment
$histPriorSalesOrderLine	element()	External variable	All prior order lines belonging to the current fulfillment function before amendment
$deletedlineItems	element()	External variable	Order lines that were deleted by the amendment
$deletedpriorlineItems	element()	External variable	Prior order line data that was deleted by the amendment
$addedlineItems	element()	External variable	Order line data that was added by the amendment
$component	element()	External variable	XML fragment containing the fulfillment function data
$histComponent	element()	External variable	XML fragment with the pre-amendment fulfillment function data
$targetIdentifier	element()	External variable	XML fragment describing the target system information

Table 5–21 (Cont.) CREATE-EBM Input Parameters for redo Execution Mode for the Option Without Calculate Service Order

Name	Type	Scope	Description
$idMap	element()	External variable	XML fragment describing the mapping between the original order line's BusinessComponentID and the newly generated BusinessComponentID to be populated into the EBM request message
$histIdMap	element()	External variable	XML fragment describing the mapping between the original order line's BusinessComponentID and the BusinessComponentID populated into the earlier EBM request message
$ebmId	xs:string	External variable	EBM ID to be populated into the EBM request message
$fulfillmentOrderId	xs:string	External variable	BusinessComponentID to be populated into the EBM request message as the Order ID
$fulfillmentOrderNumber	xs:string	External variable	Cross-system order number reference
$fulfillmentOrderStatus	xs:string	External variable	The child order's current status (IN_ PROGRESS or COMPLETE). This element controls how the EBM should be generated. The EBM is expected to be generated as a cancel order if the child order is IN_PROGRESS or as a disconnect order if the child order is COMPLETE.
$faultMode	xs:string	External variable	FaultMode code to control how the emulator generates the response message: this parameter is in effect only if the request EBM is sent to an external system emulator rather than a real system.
$verbCode	xs:string	External variable	FaultMode code to control how the service order management orchestration order generates the response message: This element only applies if the request EBM is sent to service order management (SOM).
$hasFallout	xs:boolean	External variable	Boolean indicator of whether the previous EBM request sent to the external system has had fallout
.	element()	Context node	Fulfillment order header for the SalesOrder request EBM

Table 5–22 lists the return parameters for the extension point XQuery.

Table 5–22 CREATE-EBM for redo Execution Mode Return Parameters

Output Parameter Type	Description
element()?	XML wrapper element that contains the EBM: The EBM format depends on external fulfillment provider requirements.

Example 5–6 is a sample XQuery code fragment for the CREATE-EBM fulfillment function extension point implementation for the *redo* execution mode.

Example 5–6 CREATE-EBM XQuery Code Fragment for redo Execution Mode

```
import module namespace pipextensionmodule =
"http://xmlns.oracle.com/communications/ordermanagement/pip/pipextensionmodule" at
"http://xmlns.oracle.com/communications/ordermanagement/pip/pipextensionmodule/Ext
ensionPointModule.xquery";
import module namespace YourFunctionNamefn =
"http://xmlns.oracle.com/communications/ordermanagement/pip/YourFunctionNamefn" at
"http://xmlns.oracle.com/communications/ordermanagement/pip/YourFunctionNamefn/You
rFunctionNameInteractionModule.xquery";

declare namespace oms = "urn:com:metasolv:oms:xmlapi:1";

declare variable $ebmHeader external;
declare variable $salesOrderLine external; (:check null:)
declare variable $priorSalesOrderLine external; (:check null:)
declare variable $histSalesOrderLine external; (:check null:)
declare variable $histPriorSalesOrderLine external; (:check null:)
declare variable $deletedlineItems external; (:check null:)
declare variable $deletedpriorlineItems external; (:check null:)
declare variable $addedlineItems external; (:check null:)
declare variable $component external;
declare variable $targetIdentifier external;
declare variable $idMap external;
declare variable $histIdMap external;
declare variable $ebmId external;
declare variable $fulfillmentOrderId external;
declare variable $fulfillmentOrderNumber external;
declare variable $faultMode external;
declare variable $hasFallout external;

let $fulfillmentOrder := .
return
    <Ebm>
    {
        YourFunctionNamefn:createRedoYourFunctioNamePayload(
            $ebmHeader,
            $fulfillmentOrder,
            pipextensionmodule:unWrapParameter($salesOrderLine),
            pipextensionmodule:unWrapParameter($priorSalesOrderLine),
            if ($hasFallout = fn:true()) then () else
pipextensionmodule:unWrapParameter($histSalesOrderLine),
            if ($hasFallout = fn:true()) then () else
pipextensionmodule:unWrapParameter($deletedlineItems),
            $targetIdentifier,
            $idMap,
            $histIdMap,
            $ebmId,
            $fulfillmentOrderId,
            $fulfillmentOrderNumber,
            $faultMode,
            $hasFallout)
    }
    </Ebm>
```

CREATE-EBM Extension Point for *undo* Execution Mode

This section describes the XQuery script that implements the logic to handle the CREATE-EBM extension point for the *undo* execution mode.

Table 5–23 lists the input parameters for the extension point XQuery when you are using the calculate service order solution option.

Table 5–23 CREATE-EBM Input Parameters for undo Execution Mode for the Calculate Service Order Option

Name	Type	Scope	Description
$extensionVersion	xs:string	External variable	Version number of the extension framework
$log	Java Object	External variable	Java Type org.apache.commons.logging.Log Logging level related to server log
$orderId	xs:string	External variable	OSM Order ID of the current order
$orderKey	xs:string	External variable	AIA Order Number
$ebmHeader	element()	External variable	SalesOrder request EBM header: This element can be used as a reference to populate the request EBM header
$salesOrderLine	element()	External variable	XML fragment of all order lines belonging to the current fulfillment function
$transformedOrderLines	element()	External variable	XML fragment of all transformed order lines belonging to the current fulfillment function
$mappingContext	element()	External variable	XML fragment describing the mapping context between all sales order items and transformed order items belonging to the current fulfillment function
$priorSalesOrderLine	element()	External variable	All prior order lines belonging to the current fulfillment function
$customerHeaders	element()	External variable	XML fragment describing the mapping of the CustomerHeader structure
$component	element()	External variable	XML fragment containing the fulfillment function data
$targetIdentifier	element()	External variable	XML fragment describing the target system information
$idMap	element()	External variable	XML fragment describing the mapping between the original order line's BusinessComponentID and the newly generated BusinessComponentID to be populated into the EBM request message
$ebmId	xs:string	External variable	EBM ID to be populated into the EBM request message
$fulfillmentOrderId	xs:string	External variable	BusinessComponentID to be populated into the EBM request message as the Order ID
$fulfillmentOrderNumber	xs:string	External variable	Cross-system order number reference
$fulfillmentOrderStatus	xs:string	External variable	The child order's current status (IN_PROGRESS or COMPLETE). This element controls how the EBM should be generated. The EBM is expected to be generated as a cancel order if the child order is IN_PROGRESS or as a disconnect order if the child order is COMPLETE

Table 5–23 (Cont.) CREATE-EBM Input Parameters for undo Execution Mode for the Calculate Service Order Option

Name	Type	Scope	Description
$hasFallout	xs:boolean	External variable	Boolean indicator of whether the previous EBM request sent to the external system has had fallout
$faultMode	xs:string	External variable	FaultMode code to control how the emulator generates the response message. this parameter is in effect only if the request EBM is sent to an external system emulator rather than a real system.
$verbCode	xs:string	External variable	FaultMode code to control how the service order management orchestration order generates the response message: This element only applies if the request EBM is sent to service order management (SOM).
$priorFulfillmentOrder	element()	External variable	Prior Fulfillment order header for the SalesOrder request EBM
.	element()	Context node	Fulfillment order header for the SalesOrder request EBM

Table 5–24 lists the input parameters for the extension point XQuery when you are using the solution option without calculate service order.

Table 5–24 CREATE-EBM Input Parameters for undo Execution Mode for the Option Without Calculate Service Order

Name	Type	Scope	Description
$extensionVersion	xs:string	External variable	Version number of the extension framework
$log	Java Object	External variable	Java Type org.apache.commons.logging.Log Logging level related to server log
$ebmHeader	element()	External variable	SalesOrder request EBM header: This element can be used as a reference to populate the request EBM header
$salesOrderLine	element()	External variable	XML fragment of all order lines belonging to the current fulfillment function
$priorSalesOrderLine	element()	External variable	All prior order lines belonging to the current fulfillment function
$component	element()	External variable	XML fragment containing the fulfillment function data
$targetIdentifier	element()	External variable	XML fragment describing the target system information
$idMap	element()	External variable	XML fragment describing the mapping between the original order line's BusinessComponentID and the newly generated BusinessComponentID to be populated into the EBM request message
$ebmId	xs:string	External variable	EBM ID to be populated into the EBM request message

Table 5–24 (Cont.) CREATE-EBM Input Parameters for undo Execution Mode for the Option Without Calculate Service Order

Name	Type	Scope	Description
$fulfillmentOrderId	xs:string	External variable	BusinessComponentID to be populated into the EBM request message as the Order ID
$fulfillmentOrderNumber	xs:string	External variable	Cross-system order number reference
$fulfillmentOrderStatus	xs:string	External variable	The child order's current status (IN_ PROGRESS or COMPLETE): This element controls how the EBM should be generated. The EBM is expected to be generated as a cancel order if the child order is IN_PROGRESS or as a disconnect order if the child order is COMPLETE.
$hasFallout	xs:boolean	External variable	Boolean indicator of whether the previous EBM request sent to the external system has had fallout
$faultMode	xs:string	External variable	FaultMode code to control how the emulator generates the response message. this parameter is in effect only if the request EBM is sent to an external system emulator rather than a real system.
$verbCode	xs:string	External variable	FaultMode code to control how the service order management orchestration order generates the response message: This element only applies if the request EBM is sent to service order management (SOM).
$priorFulfillmentOrder	element()	External variable	Prior Fulfillment order header for the SalesOrder request EBM
.	element()	Context node	Fulfillment order header for the SalesOrder request EBM

Table 5–25 lists the return parameters for the extension point XQuery.

Table 5–25 CREATE-EBM for undo Execution Mode Return Parameters

Output Parameter Type	Description
element()?	XML wrapper element that contains the EBM: The EBM format depends on external fulfillment provider requirements.

Example 5–7 is a sample XQuery code fragment for the CREATE-EBM fulfillment function extension point implementation for the *undo* execution mode.

Example 5–7 CREATE-EBM XQuery Code Fragment for undo Execution Mode

```
import module namespace pipextensionmodule =
"http://xmlns.oracle.com/communications/ordermanagement/pip/pipextensionmodule" at
"http://xmlns.oracle.com/communications/ordermanagement/pip/pipextensionmodule/Ext
ensionPointModule.xquery";
import module namespace YourFunctionNamefn =
"http://xmlns.oracle.com/communications/ordermanagement/pip/YourFunctionNamefn" at
"http://xmlns.oracle.com/communications/ordermanagement/pip/YourFunctionNamefn/You
rFunctionNameInteractionModule.xquery";

declare namespace oms = "urn:com:metasolv:oms:xmlapi:1";
declare variable $ebmHeader external;
declare variable $salesOrderLine external; (:check null:)
declare variable $priorSalesOrderLine external; (:check null:)
declare variable $component external;
declare variable $targetIdentifier external;
declare variable $idMap external;
declare variable $ebmId external;
declare variable $fulfillmentOrderId external;
declare variable $fulfillmentOrderNumber external;
declare variable $fulfillmentOrderStatus external;
declare variable $faultMode external;
declare variable $hasFallout external;
declare variable $verbCode external;

let $fulfillmentOrder := .
return
    <Ebm>
    {
        YourFunctionNamefn:createUndoYourFunctionNamePayload(
            $ebmHeader,
            $fulfillmentOrder,
            pipextensionmodule:unWrapParameter($salesOrderLine),
            pipextensionmodule:unWrapParameter($priorSalesOrderLine),
            $component,
            $targetIdentifier,
            $idMap,
            $ebmId,
            $fulfillmentOrderId,
            $fulfillmentOrderNumber,
            $faultMode,
            $hasFallout,
            $verbCode)
    }
    </Ebm>
```

CREATE-EBM-CUSTOM Extension Point

This section describes the XQuery script that implements the logic to handle the CREATE-EBM-CUSTOM extension point.

Table 5–26 lists the input parameters for the extension point XQuery. If any parameters do not apply to the solution option without Calculate Service Order, that will be indicated in the parameter description.

Table 5–26 CREATE-EBM-CUSTOM Input Parameters

Name	Type	Scope	Description
$log	Java Object	External variable	Java Type org.apache.commons.logging.Log Logging level related to server log
$aiaEbmHelper	Java Object	External variable	java:oracle.communications.ordermanagement.util.ebm.AiaEbmHelper For more information about this object, see knowledge article 2086727.1, **Javadocs for Order to Activate (O2A) Cartridge**, on the Oracle Support website: `https://support.oracle.com`
$taskContext	Java Object	External variable	Java Type java:com.mslv.oms.automation.TaskContext OSM-provided interface into the task. See the OSM SDK for more information.
$orderId	xs:string	External variable	OSM order ID of the current order
$orderKey	xs:string	External variable	AIA order number
$componentName	xs:string	External variable	Name of the component from which the extension point was called.
$systemType	xs:string	External variable	Name of the target system for the component, for example, BRM-BIZBDB
$execMode	xs:string	External variable	Task execution mode
$ebmHeader	element()	External variable	SalesOrder request EBM header: This element can be used as a reference to populate the request EBM header
$component	element()	External variable	XML fragment containing the fulfillment function data
$ebmId	xs:string	External variable	EBM ID to be populated into the EBM request message
$customerHeaders	element()	External variable	XML fragment describing the mapping of the CustomerHeader structure
.	element()	Context node	OSM component element

Table 5–27 lists the return parameters for the extension point XQuery.

Table 5–27 CREATE-EBM-CUSTOM Return Parameters

Output Parameter Type	Description
element()?	XML wrapper element that contains the order-level custom EBM fragment

Example 5–8 is a sample XQuery code fragment for the CREATE-EBM-CUSTOM fulfillment function extension point.

Example 5–8 CREATE-EBM-CUSTOM XQuery Code Fragment

```
declare namespace log = "java:org.apache.commons.logging.Log";
declare variable $log external;
declare variable $ebmHeader external;
declare variable $component external;
declare variable $ebmId external;
declare variable $customerHeaders external;

let $fulfillmentOrder := .
return
(
    <Ebm>
    {
        yourFuntionNamefn:createProvisioningOrderCustom(
            $log,
            $ebmHeader,
            $fulfillmentOrder,
            $component,
            $ebmId,
            $customerHeaders)
    }
    </Ebm>
)
```

CREATE-EBM-ALL-ORDERITEMS Extension Point

This section describes the XQuery script that implements the logic to handle the CREATE-EBM-ALL-ORDERITEMS extension point.

Table 5–28 lists the input parameters for the extension point XQuery. If any parameters do not apply to the solution option without Calculate Service Order, that will be indicated in the parameter description.

Table 5–28 CREATE-EBM-ALL-ORDERITEMS Input Parameters

Name	Type	Scope	Description
$log	Java Object	External variable	Java Type org.apache.commons.logging.Log Logging level related to server log
$aiaEbmHelper	Java Object	External variable	java:oracle.communications.ordermanagement.util.ebm.AiaEbmHelper For more information about this object, see knowledge article 2086727.1, **Javadocs for Order to Activate (O2A) Cartridge**, on the Oracle Support website: `https://support.oracle.com`
$taskContext	Java Object	External variable	Java Type java:com.mslv.oms.automation.TaskContext OSM-provided interface into the task. See the OSM SDK for more information.
$orderId	xs:string	External variable	OSM order ID of the current order
$orderKey	xs:string	External variable	AIA order number
$componentName	xs:string	External variable	Name of the component from which the extension point was called.
$systemType	xs:string	External variable	Name of the target system for the component, for example, BRM-BIZBDB
$execMode	xs:string	External variable	Task execution mode

Table 5–28 (Cont.) CREATE-EBM-ALL-ORDERITEMS Input Parameters

Name	Type	Scope	Description
$ebmHeader	element()	External variable	SalesOrder request EBM header: This element can be used as a reference to populate the request EBM header
$salesOrderHeader	element()	External variable	SalesOrder request EBM header: This element can be used as a reference to populate the request EBM header.
$salesOrderLine	element()	External variable	XML fragment of all order lines belonging to the current fulfillment function
$transformedOrderLines	element()	External variable	XML fragment of all transformed order lines belonging to the current fulfillment function (applies to Calculate Service Order only)
$mappingContext	element()	External variable	XML fragment describing the mapping context between all sales order items and transformed order items belonging to the current fulfillment function (applies to Calculate Service Order only)
$priorSalesOrderLine	element()	External variable	All prior order lines belonging to the current fulfillment function (has the same value as $priorLineItems)
$priorLineItems	element()	External variable	All prior order lines belonging to the current fulfillment function (has the same value as $priorSalesOrderLine)
$component	element()	External variable	XML fragment containing the fulfillment function data
$targetIdentifier	element()	External variable	XML fragment describing the target system information
$idMap	element()	External variable	XML fragment describing the mapping between the original order line's BusinessComponentID and the newly generated BusinessComponentID to be populated into the EBM request message
$ebmId	xs:string	External variable	EBM ID to be populated into the EBM request message
$fulfillmentOrderId	xs:string	External variable	BusinessComponentID to be populated into the EBM request message as the Order ID
$fulfillmentOrderNumber	xs:string	External variable	Cross-system order number reference
$faultMode	xs:string	External variable	FaultMode code to control how the emulator generates the response message: This parameter in effect only if the request EBM is sent to an external system emulator rather than a real system.
$lfVerbCode	xs:string	External variable	FaultMode code to control how the service order management orchestration order generates the response message. This element only applies if the request EBM is sent to service order management (SOM).
$customerHeaders	element()	External variable	XML fragment describing the mapping of the CustomerHeader structure

Table 5–28 (Cont.) CREATE-EBM-ALL-ORDERITEMS Input Parameters

Name	Type	Scope	Description
$orderMode	xs:string	Internal variable	This parameter has the following possible values: ■ In *do* mode, this parameter has the value of **DELIVER**. ■ In *redo* mode, this parameter has the value **CHANGE** if the service order management order has completed, so the service order management order is being revised by sending a change order. It has the value of **REVISE** if the service order management order has not completed, so the service order management order is being revised by sending an amendment order. ■ In undo mode, this parameter has the value **CANCEL** if the action codes are going to be set to **None** for the order items. It has the value of **DELIVER** if the action codes are going to be set to **Disconnect** for the order items.
$changeMode	xs:string	Internal variable	This parameter is used to set action codes appropriately. It only has a value if the value of $orderMode is **CHANGE**. Available values for this parameter are **DO**, **REDO**, and **UNDO**.
.	element()	Context node	Fulfillment order header for the SalesOrder request EBM

Table 5–29 lists the return parameters for the extension point XQuery.

Table 5–29 CREATE-EBM-ALL-ORDERITEMS Return Parameters

Output Parameter Type	Description
element()?	XML wrapper element that contains the XML fragment for the all order items EBM. The EBM format depends on external fulfillment provider requirements.

Example 5–9 is a sample XQuery code fragment for the CREATE-EBM-ALL-ORDERITEMS fulfillment function extension point.

Example 5–9 CREATE-EBM-ALL-ORDERITEMS XQuery Code Fragment

```
declare namespace log = "java:org.apache.commons.logging.Log";
declare namespace orderActivity =
"java:oracle.communications.ordermanagement.log.LogOrderActivity";
declare namespace taskExecutionMode =
"java:oracle.communications.ordermanagement.automation.OsmPipTaskConstant";
declare namespace oms = "urn:com:metasolv:oms:xmlapi:1";
declare namespace solutionconfig =
"java:oracle.communications.ordermanagement.config.OsmPipConfigProvider";

declare variable $ebmHeader external;
declare variable $salesOrderHeader external;
declare variable $salesOrderLine external;
declare variable $component external;
declare variable $idMap external;
declare variable $fulfillmentOrderId external;
declare variable $fulfillmentOrder external;
declare variable $fulfillmentOrderNumber external;
declare variable $orderId external;
declare variable $orderKey external;
declare variable $log external;
declare variable $customerHeaders external;
declare variable $aiaEbmHelper external;
declare variable $transformedOrderLines external;
declare variable $histTransformedOrderLines external;
declare variable $mappingContext external;
declare variable $orderMode external;
declare variable $changeMode external;
declare variable $execMode external;
declare variable $lineItems external;
declare variable $priorLineItems external;

let $input := .
let $isCSOEnabled := if (solutionconfig:getVariable("O2A_CSO_ENABLE_FLAG") =
"enable") then fn:true() else fn:false()

return
(
    <OrderItemEbm>
    {
        if ($isCSOEnabled = fn:true()) then
        (
            yourFuntionNamefn:createTransformedLines(
                $log,
                $aiaEbmHelper,
                $orderId,
                $orderKey,
                $ebmHeader,
                $salesOrderHeader,
                pipextensionmodule:unWrapParameter($salesOrderLine),
                pipextensionmodule:unWrapParameter($transformedOrderLines),
                pipextensionmodule:unWrapParameter($histTransformedOrderLines),
                pipextensionmodule:unWrapParameter($mappingContext),
                $component,
                $idMap,
                $fulfillmentOrder,
                $fulfillmentOrderId,
                $fulfillmentOrderNumber,
                $customerHeaders,
                $execMode,
                $orderMode,
                $changeMode)
        )
        else
        (
            yourFuntionNamefn:createNormalLines(
                $log,
                $aiaEbmHelper,
                $orderId,
                $orderKey,
                $ebmHeader,
                $salesOrderHeader,
                pipextensionmodule:unWrapParameter($lineItems),
                pipextensionmodule:unWrapParameter($priorLineItems),
                $component,
                $idMap,
                $fulfillmentOrder,
                $fulfillmentOrderId,
                $fulfillmentOrderNumber,
                $customerHeaders,
                $execMode,
                $orderMode,
                $changeMode)
        )
    }
    </OrderItemEbm>
)
```

CREATE-EBM-ORDERITEM Extension Point for *do* Execution Mode

This section describes the XQuery script that implements the logic to handle the CREATE-EBM-ORDERITEM extension point for *do* execution mode.

Table 5–30 lists the input parameters for the extension point XQuery. If any parameters do not apply to the solution option without Calculate Service Order, that will be indicated in the parameter description.

Table 5–30 CREATE-EBM-ORDERITEM Input Parameters for do Execution Mode

Name	Type	Scope	Description
$log	Java Object	External variable	Java Type org.apache.commons.logging.Log Logging level related to server log
$aiaEbmHelper	Java Object	External variable	java:oracle.communications.ordermanagement.util.ebm.AiaEbmHelper For more information about this object, see knowledge article 2086727.1, **Javadocs for Order to Activate (O2A) Cartridge**, on the Oracle Support website: `https://support.oracle.com`
$taskContext	Java Object	External variable	Java Type java:com.mslv.oms.automation.TaskContext OSM-provided interface into the task. See the OSM SDK for more information.
$orderId	xs:string	External variable	OSM order ID of the current order
$orderKey	xs:string	External variable	AIA order number
$componentName	xs:string	External variable	Name of the component from which the extension point was called.
$systemType	xs:string	External variable	Name of the target system for the component, for example, BRM-BIZBDB
$execMode	xs:string	External variable	Task execution mode
$ebmHeader	element()	External variable	SalesOrder request EBM header: This element can be used as a reference to populate the request EBM header
$salesOrderHeader	element()	External variable	SalesOrder request EBM header: This element can be used as a reference to populate the request EBM header.

Table 5–30 (Cont.) CREATE-EBM-ORDERITEM Input Parameters for do Execution Mode

Name	Type	Scope	Description
$serviceActionCode	element()	External variable	Service action for order item or transformed order item
$orderItem	element()	External variable	XML fragment for a single order item in the current Fulfillment function
$lineItem	element()	External variable	Single sales order line in the current fulfillment function
$salesOrderLine	element()	External variable	XML fragment of all order lines belonging to the current fulfillment function
$transformedOrderLines	element()	External variable	XML fragment of all transformed order lines belonging to the current fulfillment function (applies to Calculate Service Order only)
$mappingContext	element()	External variable	XML fragment describing the mapping context between all sales order items and transformed order items belonging to the current fulfillment function (applies to Calculate Service Order only)
$priorSalesOrderLine	element()	External variable	All prior order lines belonging to the current fulfillment function
$component	element()	External variable	XML fragment containing the fulfillment function data
$idMap	element()	External variable	XML fragment describing the mapping between the original order line's BusinessComponentID and the newly generated BusinessComponentID to be populated into the EBM request message
$fulfillmentOrderId	xs:string	External variable	BusinessComponentID to be populated into the EBM request message as the Order ID
$fulfillmentOrderNumber	xs:string	External variable	Cross-system order number reference
$fulfillmentOrderStatus	xs:string	External variable	The child order's current status (IN_PROGRESS or COMPLETE). This element controls how the EBM should be generated. The EBM is expected to be generated as a cancel order if the child order is IN_PROGRESS or as a disconnect order if the child order is COMPLETE.
$customerHeaders	element()	External variable	XML fragment describing the mapping of the CustomerHeader structure
$orderMode	xs:string	Internal variable	In *do* mode, this parameter has the value of **DELIVER**.
$changeMode	xs:string	Internal variable	This parameter has no value in *do* mode.
.	element()	Context node	Fulfillment order item

Table 5–31 lists the return parameters for the extension point XQuery.

Table 5–31 CREATE-EBM-ORDERITEM for do Execution Mode Return Parameters

Output Parameter Type	Description
element()?	XML wrapper element that contains the XML fragment for single order item.

Example 5–10 is a sample XQuery code fragment for the CREATE-EBM-ORDERITEM fulfillment function extension point implementation for the *do* execution mode.

Example 5–10 CREATE-EBM-ORDERITEM XQuery Code Fragment for do Execution Mode

```
declare namespace log = "java:org.apache.commons.logging.Log";
declare namespace orderActivity =
"java:oracle.communications.ordermanagement.log.LogOrderActivity";
declare namespace taskExecutionMode =
"java:oracle.communications.ordermanagement.automation.OsmPipTaskConstant";
declare namespace myContext =
"java:oracle.communications.ordermanagement.extensionpoint.XQueryExtensionUtil";
declare namespace solutionconfig =
"java:oracle.communications.ordermanagement.config.OsmPipConfigProvider";

declare variable $idMap external;
declare variable $fulfillmentOrderId external;
declare variable $orderId external;
declare variable $log external;
declare variable $transformedOrderLines external;
declare variable $transformedOrderLine external;
declare variable $serviceActionCode external;
declare variable $lineItems external;
declare variable $lineItem external;

let $input := .
let $isCSOEnabled := if (solutionconfig:getVariable("O2A_CSO_ENABLE_FLAG") =
"enable") then fn:true() else fn:false()

return
(
    let $tag := myContext:getString("tag")
    return
    (
        <OrderItemEbm>
        {
            if ($isCSOEnabled = fn:true()) then
            (
                YourFunctionNamefn:createProvisionOrderLineItemFromTransformLine(
                    $log,
                    $orderId,
                    $tag,
                    $fulfillmentOrderId,
                    $lineItem,
                    $transformedOrderLines,
                    $transformedOrderLine,
                    aiaebmfn:hasParentLine($lineItems, $lineItem),
                    aiaebmfn:getRootLineItem($lineItems, $lineItem),
                    $idMap,
                    $serviceActionCode)
            )
            else
            (
                YourFunctionNamefn:createProvisionOrderLineItem(
                    $lineItems,
                    $lineItem,
                    aiaebmfn:hasParentLine($lineItems, $lineItem),
                    aiaebmfn:getRootLineItem($lineItems, $lineItem),
                    $idMap,
                    $serviceActionCode,
                    $lineItem)
            )
        }
        </OrderItemEbm>
    )
)
```

CREATE-EBM-ORDERITEM Extension Point for *redo* Execution Mode

This section describes the XQuery script that implements the logic to handle the CREATE-EBM-ORDERITEM extension point for the *redo* execution mode.

Table 5–32 lists the input parameters for the extension point XQuery. If any parameters do not apply to the solution option without Calculate Service Order, that will be indicated in the parameter description.

Table 5–32 CREATE-EBM-ORDERITEM Input Parameters for redo Execution Mode

Name	Type	Scope	Description
$log	Java Object	External variable	Java Type org.apache.commons.logging.Log Logging level related to server log
$aiaEbmHelper	Java Object	External variable	java:oracle.communications.ordermanagement.util.ebm.AiaEbmHelper For more information about this object, see knowledge article 2086727.1, **Javadocs for Order to Activate (O2A) Cartridge**, on the Oracle Support website: `https://support.oracle.com`
$taskContext	Java Object	External variable	Java Type java:com.mslv.oms.automation.TaskContext OSM-provided interface into the task. See the OSM SDK for more information.
$orderId	xs:string	External variable	OSM order ID of the current order
$orderKey	xs:string	External variable	AIA order number
$componentName	xs:string	External variable	Name of the component from which the extension point was called.
$systemType	xs:string	External variable	Name of the target system for the component, for example, BRM-BIZBDB
$execMode	xs:string	External variable	Task execution mode
$ebmHeader	element()	External variable	SalesOrder request EBM header: This element can be used as a reference to populate the request EBM header
$salesOrderHeader	element()	External variable	SalesOrder request EBM header: This element can be used as a reference to populate the request EBM header.
$orderItem	element()	External variable	XML fragment for a single order item in the current Fulfillment function
$lineItem	element()	External variable	Single sales order line in the current fulfillment function
$salesOrderLine	element()	External variable	XML fragment of all order lines belonging to the current fulfillment function

Table 5–32 (Cont.) CREATE-EBM-ORDERITEM Input Parameters for redo Execution Mode

Name	Type	Scope	Description
$transformedOrderLines	element()	External variable	XML fragment of all transformed order lines belonging to the current fulfillment function (applies to Calculate Service Order only)
$mappingContext	element()	External variable	XML fragment describing the mapping context between all sales order items and transformed order items belonging to the current fulfillment function (applies to Calculate Service Order only)
$histTransformedOrderLines	element()	External variable	XML fragment of all transformed order items belonging to the current fulfillment function before the revision (applies to Calculate Service Order only)
$histMappingContext	element()	External variable	XML fragment describing the pre-revision mapping context between all sales order items and transformed order items belonging to the current fulfillment function (applies to Calculate Service Order only)
$priorSalesOrderLine	element()	External variable	All prior order lines belonging to the current fulfillment function
$histSalesOrderLine	element()	External variable	All order lines belonging to the current fulfillment function before amendment (has the same value as $histLineItem)
$histLineItem	element()	External variable	All order lines belonging to the current fulfillment function before amendment (has the same value as $histSalesOrderLine)
$histPriorSalesOrderLine	element()	External variable	All prior order lines belonging to the current fulfillment function before amendment
$deletedlineItem	element()	External variable	Deleted order item that is currently being processed.
$deletedlineItems	element()	External variable	List of the order items that were deleted by the amendment
$deletedTransformedLineItems	element()	External variable	XML fragment of all deleted transformed order lines belonging to the current fulfillment function (applies to Calculate Service Order only)
$deletedMappingContext	element()	External variable	XML fragment describing the deleted mapping context for all sales order items and transformed order items belonging to the current fulfillment function (applies to Calculate Service Order only)
$deletedpriorlineItems	element()	External variable	Prior order line data that was deleted by the amendment
$addedlineItems	element()	External variable	Order line data that was added by the amendment
$component	element()	External variable	XML fragment containing the fulfillment function data
$histComponent	element()	External variable	XML fragment with the pre-amendment fulfillment function data

Table 5–32 (Cont.) CREATE-EBM-ORDERITEM Input Parameters for redo Execution Mode

Name	Type	Scope	Description
$targetIdentifier	element()	External variable	XML fragment describing the target system information
$idMap	element()	External variable	XML fragment describing the mapping between the original order line's BusinessComponentID and the newly generated BusinessComponentID to be populated into the EBM request message
$histIdMap	element()	External variable	XML fragment describing the mapping between the original order line's BusinessComponentID and the BusinessComponentID populated into the earlier EBM request message
$ebmId	xs:string	External variable	EBM ID to be populated into the EBM request message
$fulfillmentOrderId	xs:string	External variable	BusinessComponentID to be populated into the EBM request message as the Order ID
$fulfillmentOrderNumber	xs:string	External variable	Cross-system order number reference
$fulfillmentOrderStatus	xs:string	External variable	The child order's current status (IN_ PROGRESS or COMPLETE). This element controls how the EBM should be generated. The EBM is expected to be generated as a cancel order if the child order is IN_PROGRESS or as a disconnect order if the child order is COMPLETE.
$hasFallout	xs:boolean	External variable	Boolean indicator of whether the previous EBM request sent to the external system has had fallout
$faultMode	xs:string	External variable	FaultMode code to control how the emulator generates the response message: this parameter is in effect only if the request EBM is sent to an external system emulator rather than a real system.
$lfVerbCode	xs:string	External variable	FaultMode code to control how the service order management orchestration order generates the response message. This element only applies if the request EBM is sent to service order management (SOM).
$customerHeaders	element()	External variable	XML fragment describing the mapping of the CustomerHeader structure
.	element()	Context node	Fulfillment order item

Table 5–33 lists the return parameters for the extension point XQuery.

Table 5–33 CREATE-EBM-ORDERITEM for redo Execution Mode Return Parameters

Output Parameter Type	Description
element()?	XML wrapper element that contains the XML fragment for single order item.

Example 5–11 is a sample XQuery code fragment for the CREATE-EBM-ORDERITEM fulfillment function extension point implementation for the *redo* execution mode.

Example 5–11 CREATE-EBM-ORDERITEM XQuery Code Fragment for redo Execution Mode

```
declare namespace log = "java:org.apache.commons.logging.Log";
declare namespace orderActivity =
"java:oracle.communications.ordermanagement.log.LogOrderActivity";
declare namespace taskExecutionMode =
"java:oracle.communications.ordermanagement.automation.OsmPipTaskConstant";
declare namespace myContext =
"java:oracle.communications.ordermanagement.extensionpoint.XQueryExtensionUtil";
declare namespace solutionconfig =
"java:oracle.communications.ordermanagement.config.OsmPipConfigProvider";

declare variable $idMap external;
declare variable $fulfillmentOrderId external;
declare variable $orderId external;
declare variable $log external;
declare variable $transformedOrderLines external;
declare variable $transformedOrderLine external;
declare variable $serviceActionCode external;
declare variable $lineItems external;
declare variable $lineItem external;

let $input := .
let $isCSOEnabled := if (solutionconfig:getVariable("O2A_CSO_ENABLE_FLAG") =
"enable") then fn:true() else fn:false()

return
(
    let $tag := myContext:getString("tag")
    return
    (
        <OrderItemEbm>
        {
            if ($isCSOEnabled = fn:true()) then
            (
                YourFunctionNamefn:createProvisionOrderLineItemFromTransformLine(
                    $log,
                    $orderId,
                    $tag,
                    $fulfillmentOrderId,
                    $lineItem,
                    $transformedOrderLines,
                    $transformedOrderLine,
                    aiaebmfn:hasParentLine($lineItems, $lineItem),
                    aiaebmfn:getRootLineItem($lineItems, $lineItem),
                    $idMap,
                    $serviceActionCode)
            )
            else
            (
                YourFunctionNamefn:createProvisionOrderLineItem(
                    $lineItems,
                    $lineItem,
                    aiaebmfn:hasParentLine($lineItems, $lineItem),
                    aiaebmfn:getRootLineItem($lineItems, $lineItem),
                    $idMap,
                    $serviceActionCode,
                    $lineItem)
            )
        }
```

```
            </OrderItemEbm>
        )
)
```

CREATE-EBM-ORDERITEM Extension Point for *undo* Execution Mode

This section describes the XQuery script that implements the logic to handle the CREATE-EBM-ORDERITEM extension point for the *undo* execution mode.

Table 5–34 lists the input parameters for the extension point XQuery. If any parameters do not apply to the solution option without Calculate Service Order, that will be indicated in the parameter description.

Table 5–34 CREATE-EBM-ORDERITEM Input Parameters for undo Execution Mode

Name	Type	Scope	Description
$log	Java Object	External variable	Java Type org.apache.commons.logging.Log Logging level related to server log
$aiaEbmHelper	Java Object	External variable	java:oracle.communications.ordermanagement.util.ebm.AiaEbmHelper For more information about this object, see knowledge article 2086727.1, **Javadocs for Order to Activate (O2A) Cartridge**, on the Oracle Support website: `https://support.oracle.com`
$taskContext	Java Object	External variable	Java Type java:com.mslv.oms.automation.TaskContext OSM-provided interface into the task. See the OSM SDK for more information.
$orderId	xs:string	External variable	OSM order ID of the current order
$orderKey	xs:string	External variable	AIA order number
$componentName	xs:string	External variable	Name of the component from which the extension point was called.
$systemType	xs:string	External variable	Name of the target system for the component, for example, BRM-BIZBDB
$execMode	xs:string	External variable	Task execution mode
$ebmHeader	element()	External variable	SalesOrder request EBM header: This element can be used as a reference to populate the request EBM header
$salesOrderHeader	element()	External variable	SalesOrder request EBM header: This element can be used as a reference to populate the request EBM header.
$priorSalesOrderHeader	element()	External variable	Prior SalesOrder request EBM header: This element can be used as a reference to populate the request EBM header.
$component	element()	External variable	XML fragment containing the fulfillment function data
$orderItem	element()	External variable	XML fragment for a single order item in the current Fulfillment function
$lineItem	element()	External variable	Single sales order line in the current fulfillment function

Table 5–34 (Cont.) CREATE-EBM-ORDERITEM Input Parameters for undo Execution Mode

Name	Type	Scope	Description
$salesOrderLine	element()	External variable	XML fragment of all order lines belonging to the current fulfillment function
$transformedOrderLines	element()	External variable	XML fragment of all transformed order lines belonging to the current fulfillment function (applies to Calculate Service Order only)
$mappingContext	element()	External variable	XML fragment describing the mapping context between all sales order items and transformed order items belonging to the current fulfillment function (applies to Calculate Service Order only)
$priorSalesOrderLine	element()	External variable	All prior order lines belonging to the current fulfillment function
$targetIdentifier	element()	External variable	XML fragment describing the target system information
$idMap	element()	External variable	XML fragment describing the mapping between the original order line's BusinessComponentID and the newly generated BusinessComponentID to be populated into the EBM request message
$fulfillmentOrderId	xs:string	External variable	BusinessComponentID to be populated into the EBM request message as the Order ID
$fulfillmentOrderNumber	xs:string	External variable	Cross-system order number reference
$fulfillmentOrderStatus	xs:string	External variable	The child order's current status (IN_ PROGRESS or COMPLETE). This element controls how the EBM should be generated. The EBM is expected to be generated as a cancel order if the child order is IN_PROGRESS or as a disconnect order if the child order is COMPLETE.
$customerHeaders	element()	External variable	XML fragment describing the mapping of the CustomerHeader structure
$histComponent	element()	External variable	XML fragment with the pre-amendment fulfillment function data
$histTransformedOrderLines	element()	External variable	XML fragment of all transformed order items belonging to the current fulfillment function before the revision (applies to Calculate Service Order only)
$histMappingContext	element()	External variable	XML fragment describing the pre-revision mapping context between all sales order items and transformed order items belonging to the current fulfillment function (applies to Calculate Service Order only)
$histSalesOrderLine	element()	External variable	All order lines belonging to the current fulfillment function before amendment
$histPriorSalesOrderLine	element()	External variable	All prior order lines belonging to the current fulfillment function before amendment

Table 5–34 (Cont.) CREATE-EBM-ORDERITEM Input Parameters for undo Execution Mode

Name	Type	Scope	Description
$histIdMap	element()	External variable	XML fragment describing the mapping between the original order line's BusinessComponentID and the BusinessComponentID populated into the earlier EBM request message
$deletedlineItems	element()	External variable	Order lines that were deleted by the amendment
$deletedTransformedLineItems	element()	External variable	XML fragment of all deleted transformed order lines belonging to the current fulfillment function (applies to Calculate Service Order only)
$deletedMappingContext	element()	External variable	XML fragment describing the deleted mapping context for all sales order items and transformed order items belonging to the current fulfillment function (applies to Calculate Service Order only)
$deletedpriorlineItems	element()	External variable	Prior order line data that was deleted by the amendment
$addedlineItems	element()	External variable	Order line data that was added by the amendment
.	element()	Context node	Fulfillment order item

Table 5–35 lists the return parameters for the extension point XQuery.

Table 5–35 CREATE-EBM-ORDERITEM for undo Execution Mode Return Parameters

Output Parameter Type	Description
element()?	XML wrapper element that contains the XML fragment for single order item.

Example 5–12 is a sample XQuery code fragment for the CREATE-EBM-ORDERITEM fulfillment function extension point implementation for the *undo* execution mode.

Example 5–12 CREATE-EBM-ORDERITEM XQuery Code Fragment for undo Execution Mode

```
declare namespace log = "java:org.apache.commons.logging.Log";
declare namespace orderActivity =
"java:oracle.communications.ordermanagement.log.LogOrderActivity";
declare namespace taskExecutionMode =
"java:oracle.communications.ordermanagement.automation.OsmPipTaskConstant";
declare namespace myContext =
"java:oracle.communications.ordermanagement.extensionpoint.XQueryExtensionUtil";
declare namespace solutionconfig =
"java:oracle.communications.ordermanagement.config.OsmPipConfigProvider";

declare variable $idMap external;
declare variable $fulfillmentOrderId external;
declare variable $orderId external;
declare variable $log external;
declare variable $transformedOrderLines external;
declare variable $transformedOrderLine external;
declare variable $serviceActionCode external;
declare variable $lineItems external;
declare variable $lineItem external;
```

```
let $input := .
let $isCSOEnabled := if (solutionconfig:getVariable("O2A_CSO_ENABLE_FLAG") =
"enable") then fn:true() else fn:false()

return
(
    let $tag := myContext:getString("tag")
    return
    (
        <OrderItemEbm>
        {
            if ($isCSOEnabled = fn:true()) then
            (
                YourFunctionNamefn:createProvisionOrderLineItemFromTransformLine(
                    $log,
                    $orderId,
                    $tag,
                    $fulfillmentOrderId,
                    $lineItem,
                    $transformedOrderLines,
                    $transformedOrderLine,
                    aiaebmfn:hasParentLine($lineItems, $lineItem),
                    aiaebmfn:getRootLineItem($lineItems, $lineItem),
                    $idMap,
                    $serviceActionCode)
            )
            else
            (
                YourFunctionNamefn:createProvisionOrderLineItem(
                    $lineItems,
                    $lineItem,
                    aiaebmfn:hasParentLine($lineItems, $lineItem),
                    aiaebmfn:getRootLineItem($lineItems, $lineItem),
                    $idMap,
                    $serviceActionCode,
                    $lineItem)
            )
        }
        </OrderItemEbm>
    )
)
```

CREATE-EBM-ORDERITEM-CUSTOM Extension Point

This section describes the XQuery script that implements the logic to handle the CREATE-EBM-ORDERITEM-CUSTOM extension point.

Table 5–36 lists the input parameters for the extension point XQuery. If any parameters do not apply to the solution option without Calculate Service Order, that will be indicated in the parameter description.

Table 5–36 CREATE-EBM-ORDERITEM-CUSTOM Input Parameters

Name	Type	Scope	Description
$log	Java Object	External variable	Java Type org.apache.commons.logging.Log Logging level related to server log
$aiaEbmHelper	Java Object	External variable	java:oracle.communications.ordermanage ment.util.ebm.AiaEbmHelper For more information about this object, see knowledge article 2086727.1, **Javadocs for Order to Activate (O2A) Cartridge**, on the Oracle Support website: `https://support.oracle.com`
$taskContext	Java Object	External variable	Java Type java:com.mslv.oms.automation.TaskContex t OSM-provided interface into the task. See the OSM SDK for more information.
$orderId	xs:string	External variable	OSM order ID of the current order
$orderKey	xs:string	External variable	AIA order number
$componentName	xs:string	External variable	Name of the component from which the extension point was called.
$systemType	xs:string	External variable	Name of the target system for the component, for example, BRM-BIZBDB
$execMode	xs:string	External variable	Task execution mode
$ebmHeader	element()	External variable	SalesOrder request EBM header: This element can be used as a reference to populate the request EBM header
$component	element()	External variable	XML fragment containing the fulfillment function data
$salesOrderLine	element()	External variable	XML fragment of all order lines belonging to the current fulfillment function
$transformedOrderLines	element()	External variable	XML fragment of all transformed order lines belonging to the current fulfillment function (applies to Calculate Service Order only)
.	element()	Context node	OSM Component element

Table 5–37 lists the return parameters for the extension point XQuery.

Table 5–37 CREATE-EBM-ORDERITEM-CUSTOM Return Parameters

Output Parameter Type	Description
element()?	XML wrapper element that contains the order-item-level custom EBM fragment

Example 5–13 is a sample XQuery code fragment for the CREATE-EBM-ORDERITEM-CUSTOM fulfillment function extension point.

Example 5–13 CREATE-EBM-ORDERITEM-CUSTOM XQuery Code Fragment

```
declare namespace log = "java:org.apache.commons.logging.Log";
declare namespace orderActivity =
"java:oracle.communications.ordermanagement.log.LogOrderActivity";
declare namespace taskExecutionMode =
"java:oracle.communications.ordermanagement.automation.OsmPipTaskConstant";
declare namespace myContext =
"java:oracle.communications.ordermanagement.extensionpoint.XQueryExtensionUtil";
declare namespace solutionconfig =
"java:oracle.communications.ordermanagement.config.OsmPipConfigProvider";

declare variable $sourceLineItem external;
declare variable $transformedOrderLine external;

let $input := .

let $isCSOEnabled := if (solutionconfig:getVariable("O2A_CSO_ENABLE_FLAG") =
"enable") then fn:true() else fn:false()
let $result :=
       if ($isCSOEnabled = fn:true()) then
       (

YourFunctionNamefn:createTransformedLineCustom($sourceLineItem,$transformedOrderLi
ne)
       )
       else
       (
            YourFunctionNamefn:createLineCustom($sourceLineItem)
       )
return
(
    <result>
    {
        $result
    }
    </result>
```

CREATE-EBM-PRIORORDERITEM Extension Point

This section describes the XQuery script that implements the logic to handle the CREATE-EBM-PRIORORDERITEM extension point.

Table 5–38 lists the input parameters for the extension point XQuery. If any parameters do not apply to the solution option without Calculate Service Order, that will be indicated in the parameter description.

Table 5–38 CREATE-EBM-PRIORORDERITEM Input Parameters

Name	Type	Scope	Description
$log	Java Object	External variable	Java Type org.apache.commons.logging.Log Logging level related to server log
$aiaEbmHelper	Java Object	External variable	java:oracle.communications.ordermanagement.util.ebm.AiaEbmHelper For more information about this object, see knowledge article 2086727.1, **Javadocs for Order to Activate (O2A) Cartridge**, on the Oracle Support website: `https://support.oracle.com`
$taskContext	Java Object	External variable	Java Type java:com.mslv.oms.automation.TaskContext OSM-provided interface into the task. See the OSM SDK for more information.
$orderId	xs:string	External variable	OSM order ID of the current order
$orderKey	xs:string	External variable	AIA order number
$componentName	xs:string	External variable	Name of the component from which the extension point was called.
$systemType	xs:string	External variable	Name of the target system for the component, for example, BRM-BIZBDB
$execMode	xs:string	External variable	Task execution mode
$ebmHeader	element()	External variable	SalesOrder request EBM header: This element can be used as a reference to populate the request EBM header
$component	element()	External variable	XML fragment containing the fulfillment function data
$salesOrderLine	element()	External variable	XML fragment of all order lines belonging to the current fulfillment function
$transformedOrderLines	element()	External variable	XML fragment of all transformed order lines belonging to the current fulfillment function (applies to Calculate Service Order only)
$transformedOrderLine	element()	External variable	XML fragment of all transformed order lines belonging to the current fulfillment function (applies to Calculate Service Order only)
$mappingContext	element()	External variable	XML fragment describing the mapping context between all sales order items and transformed order items belonging to the current fulfillment function (applies to Calculate Service Order only)
$priorSalesOrderLine	element()	External variable	All prior order lines belonging to the current fulfillment function
$idMap	element()	External variable	XML fragment describing the mapping between the original order line's BusinessComponentID and the newly generated BusinessComponentID to be populated into the EBM request message

Table 5–38 (Cont.) CREATE-EBM-PRIORORDERITEM Input Parameters

Name	Type	Scope	Description
$fulfillmentOrderId	xs:string	External variable	BusinessComponentID to be populated into the EBM request message as the Order ID
$fulfillmentOrderNumber	xs:string	External variable	Cross-system order number reference
$customerHeaders	element()	External variable	XML fragment describing the mapping of the CustomerHeader structure
.	element()	Context node	EBM Header

Table 5–39 lists the return parameters for the extension point XQuery.

Table 5–39 CREATE-EBM-PRIORORDERITEM Return Parameters

Output Parameter Type	Description
element()?	XML wrapper element that contains the XML fragment for the prior single order item custom EBM

Example 5–14 is a sample XQuery code fragment for the CREATE-EBM-PRIORORDERITEM fulfillment function extension point.

Example 5–14 CREATE-EBM-PRIORORDERITEM XQuery Code Fragment

```
declare namespace solutionconfig =
"java:oracle.communications.ordermanagement.config.OsmPipConfigProvider";

declare variable $priorSalesOrderLine external;

let $isCSOEnabled := if (solutionconfig:getVariable("O2A_CSO_ENABLE_FLAG") =
"enable") then fn:true() else fn:false()

let $input := .

return
(
    <OrderItemEbm>
    {
        if ($isCSOEnabled = fn:true()) then
        (
            YourFunctionNamefn:createSingleTransformedLine(
                $priorSalesOrderLine
            )
        else
        (
            YourFunctionNamefn:createProvisionOrderLineItem(
                $priorSalesOrderLine
        )
    }

    </OrderItemEbm>
)
```

CREATE-EBM-PRIORORDERITEM-CUSTOM Extension Point

This section describes the XQuery script that implements the logic to handle the CREATE-EBM-PRIORORDERITEM-CUSTOM extension point.

Table 5–40 lists the input parameters for the extension point XQuery. If any parameters do not apply to the solution option without Calculate Service Order, that will be indicated in the parameter description.

Table 5–40 CREATE-EBM-PRIORORDERITEM-CUSTOM Input Parameters

Name	Type	Scope	Description
$log	Java Object	External variable	Java Type org.apache.commons.logging.Log Logging level related to server log
$aiaEbmHelper	Java Object	External variable	java:oracle.communications.ordermanagement.util.ebm.AiaEbmHelper For more information about this object, see knowledge article 2086727.1, **Javadocs for Order to Activate (O2A) Cartridge**, on the Oracle Support website: `https://support.oracle.com`
$taskContext	Java Object	External variable	Java Type java:com.mslv.oms.automation.TaskContext OSM-provided interface into the task. See the OSM SDK for more information.
$orderId	xs:string	External variable	OSM order ID of the current order
$orderKey	xs:string	External variable	AIA order number
$componentName	xs:string	External variable	Name of the component from which the extension point was called.
$systemType	xs:string	External variable	Name of the target system for the component, for example, BRM-BIZBDB
$execMode	xs:string	External variable	Task execution mode
$ebmHeader	element()	External variable	SalesOrder request EBM header: This element can be used as a reference to populate the request EBM header
$component	element()	External variable	XML fragment containing the fulfillment function data
$salesOrderLine	element()	External variable	XML fragment of all order lines belonging to the current fulfillment function
$transformedOrderLines	element()	External variable	XML fragment of all transformed order lines belonging to the current fulfillment function (applies to Calculate Service Order only)
.	element()	Context node	OSM Component element

Table 5–41 lists the return parameters for the extension point XQuery.

Table 5–41 CREATE-EBM-PRIORORDERITEM-CUSTOM Return Parameters

Output Parameter Type	Description
element()?	XML wrapper element that contains the prior-order-item-level custom EBM fragment

Example 5–15 is a sample XQuery code fragment for the CREATE-EBM-PRIORORDERITEM-CUSTOM fulfillment function extension point.

Example 5–15 CREATE-EBM-PRIORORDERITEM-CUSTOM XQuery Code Fragment

```
declare namespace log = "java:org.apache.commons.logging.Log";
declare namespace orderActivity =
"java:oracle.communications.ordermanagement.log.LogOrderActivity";
declare namespace taskExecutionMode =
"java:oracle.communications.ordermanagement.automation.OsmPipTaskConstant";
declare namespace myContext =
"java:oracle.communications.ordermanagement.extensionpoint.XQueryExtensionUtil";
declare namespace solutionconfig =
"java:oracle.communications.ordermanagement.config.OsmPipConfigProvider";

declare variable $sourceLineItem external;
declare variable $transformedOrderLine external;

let $input := .

let $isCSOEnabled := if (solutionconfig:getVariable("O2A_CSO_ENABLE_FLAG") =
"enable") then fn:true() else fn:false()
let $result :=
    if ($isCSOEnabled = fn:true()) then
    (

YourFunctionNamefn:createTransformedLineCustom($sourceLineItem,$transformedOrderLi
ne)
    )
    else
    (
        YourFunctionNamefn:createLineCustom($sourceLineItem)
    )
return
(
    <result>
    {
        $result
    }
    </result>
)
```

VALIDATE-RESPONSE-EBM Extension Point

This section describes the XQuery script that implements the logic to handle the VALIDATE-RESPONSE-EBM extension point.

Table 5–42 lists the input parameters for the extension point XQuery when you are using the calculate service order solution option.

Table 5–42 VALIDATE-RESPONSE-EBM Input Parameters for the Calculate Service Order Option

Name	Type	Scope	Description
$log	Java Object	External variable	Java Type org.apache.commons.logging.Log Logging level related to server log
$executionMode	xs:string	External variable	Task execution mode
.	element()	Context node	Response EBM message to be validated

Table 5–43 lists the input parameters for the extension point XQuery when you are using the solution option without calculate service order.

Table 5–43 VALIDATE-RESPONSE-EBM Input Parameters for the Option Without Calculate Service Order

Name	Type	Scope	Description
.	element()	Context node	Response EBM message to be validated

Table 5–44 lists the return parameters for the extension point XQuery.

Table 5–44 VALIDATE-RESPONSE-EBM Return Parameters

Output Parameter Type	Description
element()?	XML wrapper element which contains an empty sequence if no error was found or a list of XML fragments that describe the validation error if an error was found

Example 5–16 is a code fragment from **OracleComms_OSM_O2A_COM_Billing/resources/ExtensionPoint/SyncCustomerValidateResponseEBM_Event.xquery** demonstrates the extension implementation.

Example 5–16 VALIDATE-RESPONSE-EBM XQuery Code Fragment

```
import module namespace aiaebmvalidationfn =
"http://xmlns.oracle.com/communications/ordermanagement/pip/aiaebmvalidationfn" at
"http://xmlns.oracle.com/communications/ordermanagement/pip/aiaebmvalidationfn/AIA
EBMResponse_ValidationModule.xquery";

declare namespace oms = "urn:com:metasolv:oms:xmlapi:1";
declare variable $SYNCCUST_RESPONSE_EBM :=
"ProcessFulfillmentOrderBillingAccountListResponseEBM";

declare function local:validateSyncCustomerResponse(
    $ebm as element() *) as element()
{
    if(fn:local-name($ebm) = $SYNCCUST_RESPONSE_EBM)
    then
        <oms:validationReport>
        {
            aiaebmvalidationfn:validateSyncCustomerResponse($ebm)
        }
        </oms:validationReport>
    else
        <oms:validationReport>{ $aiaebmvalidationfn:NO_VALID_EBM
}</oms:validationReport>
};

let $ebm := .
return
    <Validation>
    {
        local:validateSyncCustomerResponse($ebm)
    }
    </Validation>
```

COMPONENT-RESPONSE-UPDATE Extension Point

This section describes the XQuery script that implements the logic to handle the COMPONENT-RESPONSE-UPDATE extension point.

Table 5–45 lists the input parameters for the extension point XQuery when you are using the calculate service order solution option.

Table 5–45 COMPONENT-RESPONSE-UPDATE Input Parameters for the Calculate Service Order Option

Name	Type	Scope	Description
$extensionVersion	xs:string	External variable	Version number of the extension framework
$log	Java Object	External variable	Java Type org.apache.commons.logging.Log Logging level related to server log
$orderId	xs:string	External variable	OSM Order ID of the current order
$orderKey	xs:string	External variable	AIA Order Number
$component	element()	External variable	XML fragment containing the fulfillment function data
$executionMode	xs:string	External variable	Task execution mode
$hasFallout	xs:boolean	External variable	Boolean indicator of whether the previous EBM request sent to the external system has had fallout
$falloutMessage	xs:string	External variable	The fallout error message of this order item
$orderItemFromResponse	element()	External variable	Order item data from the response message
$mappingContext	element()	External variable	XML fragment describing the mapping context between all sales order items and transformed order items belonging to the current fulfillment function
.	element()	Context node	The order item data XML fragment

Table 5–46 lists the input parameters for the extension point XQuery when you are using the solution option without calculate service order.

Table 5–46 COMPONENT-RESPONSE-UPDATE Input Parameters for the Option Without Calculate Service Order

Name	Type	Scope	Description
$extensionVersion	xs:string	External variable	Version number of the extension framework
$log	Java Object	External variable	Java Type org.apache.commons.logging.Log Logging level related to server log
$executionMode	xs:string	External variable	Task execution mode
$hasFallout	xs:boolean	External variable	Boolean indicator of whether the previous EBM request sent to the external system has had fallout
$orderId	xs:string	External variable	OSM Order ID of the current order
$orderKey	xs:string	External variable	AIA Order Number
$falloutMessage	xs:string	External variable	The fallout error message of this order item
$orderItemFromResponse	element()	External variable	Order item data from the response message
.	element()	Context node	The order item data XML fragment

Table 5–47 lists the return parameters for the extension point XQuery.

Table 5–47 COMPONENT-RESPONSE-UPDATE Return Parameters

Output Parameter Type	Description
element()*	XML wrapper element that contains all the order item properties to be updated

Example 5–17 is a sample XQuery code fragment for the COMPONENT-RESPONSE-UPDATE fulfillment function extension point.

Example 5–17 COMPONENT-RESPONSE-UPDATE XQuery Code Fragment

```
import module namespace YourFunctionNamefn =
"http://xmlns.oracle.com/communications/ordermanagement/pip/YourFunctionNamefn" at
"http://xmlns.oracle.com/communications/ordermanagement/pip/YourFunctionNamefn/You
rFunctionNameInteractionModule.xquery";

declare namespace oms = "urn:com:metasolv:oms:xmlapi:1";

declare variable $hasFallout external;
declare variable $falloutMessage external;
declare variable $orderItemFromResponse external;

declare function YourFunctionNamefn:onYourFunctionNameResponseUpdate(
    $lineItem as element(),
    $orderItemFromResponse as element()) as element()*
{
    let $id := $lineItem/oms:BaseLineId
    return
        <BaseLineId>{ $id/text() }</BaseLineId>,
        (: list of order item properties to be updated :)
};

let $lineItem := .
return
    <OrderItem>
    {
        YourFunctionNamefn:onYourFunctionNameResponseUpdate($lineItem,
$orderItemFromResponse)
    }
    </OrderItem>
```

ORDER-EXTENSION-UPDATE-STATUS-EBM Extension Point

This section describes the XQuery script that implements the logic to handle the ORDER-EXTENSION-UPDATE-STATUS-EBM extension point.

Table 5–48 lists the input parameters for the extension point XQuery when you are using the calculate service order solution option.

Table 5–48 ORDER-EXTENSION-UPDATE-STATUS-EBM Input Parameters for the Calculate Service Order Option

Name	Type	Scope	Description
$controlData	element()	External variable	XML data fragment of the control data from the current task that calls to this extension point
$taskData	element()	External variable	XML data fragment of the current task that calls to this extension point
$ebmId	xs:string	External variable	EBM ID to be populated into the EBM request message
$ebm	element()	External variable	XML data fragment of the EBM header from the CRM system
$fulfillmentOrder	element()	External variable	XML data fragment of the Fulfillment Order from the CRM system
$extensionVersion	xs:string	External variable	Version number of the extension framework
$orderId	xs:string	External variable	OSM Order ID of the current order
$sequenceNumber	xs:string	External variable	EBM message unique sequence tracking number to be populated in the update EBM
$changeLinesStatus	element()*	External variable	XML data fragment which contains multiple order items that must be included in the update EBM
$hasFalloutFlag	xs:boolean	External variable	Boolean indicator of whether the order is in Fallout state
$isCancelFlag	xs:boolean	External variable	Boolean indicator for the order is being cancelled
$log	Java Object	External variable	Java Type org.apache.commons.logging.Log Logging level related to server log
$execMode	xs:string	External variable	Task execution mode
$debugControl	element()	External variable	XML data fragment of the order execution control for break point and debug
$orderStatusContext	xs:string	External variable	Order status description to be populated in the update EBM
$includeActualDelDateTime	xs:boolean	External variable	Boolean indicator for whether to populate the actual delivery date/time field of the update EBM
$includeFulfillmentData	xs:boolean	External variable	Boolean indicator for whether to include detailed order item data in the update EBM

Table 5–49 lists the input parameters for the extension point XQuery when you are using the solution option without calculate service order.

Table 5–49 ORDER-EXTENSION-UPDATE-STATUS-EBM Input Parameters for the Option Without Calculate Service Order

Name	Type	Scope	Description
$controlData	element()	External variable	XML data fragment of the control data from the current task that calls to this extension point
$taskData	element()	External variable	XML data fragment of the current task that calls to this extension point
$ebmId	xs:string	External variable	EBM ID to be populated into the EBM request message
$ebm	element()	External variable	XML data fragment of the EBM header from the CRM system
$fulfillmentOrder	element()	External variable	XML data fragment of the Fulfillment Order from the CRM system
$extensionVersion	xs:string	External variable	Version number of the extension framework
$orderId	xs:string	External variable	OSM Order ID of the current order.
$sequenceNumber	xs:string	External variable	EBM message unique sequence tracking number to be populated in the update EBM
$changeLinesStatus	element()*	External variable	XML data fragment which contains multiple order items that must be included in the update EBM
$hasFalloutFlag	xs:boolean	External variable	Boolean indicator of whether the order is in a fallout state
$isCancelFlag	xs:boolean	External variable	Boolean indicator of whether the order is being cancelled
$log	Java Object	External variable	Java Type org.apache.commons.logging.Log Logging level related to server log
$execMode	xs:string	External variable	Task execution mode
$debugControl	element()	External variable	XML data fragment of the order execution control for break point and debug
$orderStatusContext	xs:string	External variable	Order status description to be populated in the update EBM
$includeActualDelDateTime	xs:boolean	External variable	Boolean indicator for whether to populate the actual delivery date/time field of the update EBM
$includeFulfillmentData	xs:boolean	External variable	Boolean indicator for whether to include detailed order item data in the update EBM

Table 5–50 lists the return parameters for the extension point XQuery.

Table 5–50 ORDER-EXTENSION-UPDATE-STATUS-EBM Return Parameters

Output Parameter Type	Description
element()?	XML wrapper element that contains the Update Sales Order EBM. The EBM format depends on external fulfillment provider requirements.

Example 5–18 is a sample XQuery code fragment for the ORDER-EXTENSION-UPDATE-STATUS-EBM fulfillment function extension point when you are using the calculate service order solution option.

Example 5–18 ORDER-EXTENSION-UPDATE-STATUS-EBM XQuery Code Fragment for the Calculate Service Order Option

```
declare namespace log = "java:org.apache.commons.logging.Log";
declare namespace oms="urn:com:metasolv:oms:xmlapi:1";
declare namespace
salesord="http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/SalesOrder/V2";
declare namespace
corecom="http://xmlns.oracle.com/EnterpriseObjects/Core/Common/V2";

declare variable $controlData external;
declare variable $taskData external;
declare variable $ebmId external;
declare variable $ebm external;
declare variable $fulfillmentOrder external;
declare variable $extensionVersion external;
declare variable $orderId external;
declare variable $sequenceNumber external;
declare variable $changeLinesStatus external;
declare variable $hasFalloutFlag external;
declare variable $isCancelFlag external;
declare variable $log external;
declare variable $execMode external;
declare variable $debugControl external;
declare variable $orderStatusContext external;
declare variable $includeActualDelDateTime external;
declare variable $includeFulfillmentData external;

if ($extensionVersion=$pipextensionmodule:EXTENSION_VERSION_2) then
(

yourOrderFunctionfn:createUpdateSalesOrderPayloadWithUserProvideOrderStatusContext
(
        $log,
        $orderId,
        $ebm,
        $fulfillmentOrder,
        $ebmId,
        $sequenceNumber,
        $controlData,
        $changeLinesStatus,
        $hasFalloutFlag,
        $isCancelFlag,
        $orderStatusContext,
        $includeActualDelDateTime,
        $includeFulfillmentData)
)
else
(
    log:warn($log, fn:concat("UpdateEBM Extension Point V2 is receiving the wrong
version! extensionVersion:[",$extensionVersion,"]"))
)
```

Example 5–19 is a sample XQuery code fragment for the ORDER-EXTENSION-UPDATE-STATUS-EBM fulfillment function extension point when you are using the solution option without calculate service order.

Example 5–19 ORDER-EXTENSION-UPDATE-STATUS-EBM XQuery Code Fragment for the Calculate Service Order Option

```
declare namespace log = "java:org.apache.commons.logging.Log";
declare namespace oms="urn:com:metasolv:oms:xmlapi:1";
declare namespace
salesord="http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/SalesOrder/V1";
declare namespace
corecom="http://xmlns.oracle.com/EnterpriseObjects/Core/Common/V1";

declare variable $controlData external;
declare variable $taskData external;
declare variable $ebmId external;
declare variable $ebm external;
declare variable $fulfillmentOrder external;
declare variable $extensionVersion external;
declare variable $orderId external;
declare variable $sequenceNumber external;
declare variable $changeLinesStatus external;
declare variable $hasFalloutFlag external;
declare variable $isCancelFlag external;
declare variable $log external;
declare variable $execMode external;
declare variable $debugControl external;
declare variable $orderStatusContext external;
declare variable $includeActualDelDateTime external;
declare variable $includeFulfillmentData external;

if ($extensionVersion=$pipextensionmodule:EXTENSION_VERSION_1) then
(

yourOrderFunctionfn:createUpdateSalesOrderPayloadWithUserProvideOrderStatusContext
(
        $log,
        $orderId,
        $ebm,
        $fulfillmentOrder,
        $ebmId,
        $sequenceNumber,
        $controlData,
        $changeLinesStatus,
        $hasFalloutFlag,
        $isCancelFlag,
        $orderStatusContext,
        $includeActualDelDateTime,
        $includeFulfillmentData)
)
else
(
    log:warn($log, fn:concat("UpdateEBM Extension Point V1 is receiving the wrong
version! extensionVersion:[",$extensionVersion,"]"))
)
```

About Fallout

When creating a new fulfillment function, you must consider the fallout handling needed. Information about the fallout considerations is contained in the following sections.

Fallout Customization

The **AIAResponseEBM.xqy** XQuery file is called from automated tasks, for example, SyncCustomerSITask and ProvisionOrderSITask. When adding a fulfillment function, you must customize a copy of this file, located in **OracleComms_OSM_O2A_COM_SalesOrderFulfillment\resources\ComponentInteraction**, and call it from the automation in the SITask that processes the response coming from the Application Business Connector Service (ABCS) for the fulfillment system. The response can be either a response EBM or an OrderFalloutNotification.

You may need to customize the **local:getEbmFromResponse** function to extract the response EBM, depending on the fulfillment function.

The function **local:getCFSystem** extracts the system name from the value of the componentKey for the executable order component. Due to the four orchestration stages defined in the orchestration sequence in the OracleComms_OSM_O2A_COM_Base cartridge, the componentKey takes the format:

```
FunctionName.SystemName.Granularity.FunctionSignificantUpdates
```

An example of a component key for an executable FulfillBillingFunction Order Component targeted at the BRM-VoIP billing system using ServiceBundleGranularity processing granularity, with a base line ID of the service bundle line of 3138373233393233393433332373635 is:

```
FulfillBillingFunction.BRM-VOIP.ServiceBundleGranularity.313837323339323339343333
2373635/ServiceBundleGranularity.FulfillBillingSignificantUpdates
```

Example 5–20 is a sample XQuery code fragment from **OracleComms_OSM_O2A_COM_Base/resources/FalloutHandling/TargetMapping.xquery**.

Example 5–20 Target System Map XQuery Code Fragment

```
(:
 : Function to return the Target system name that the given ActiveInteractionId is
associate to.
 :)
declare function osmmappip:getCFSystem(
    $orderData as element()?,
    $activeInteractionId as xs:string) as xs:string?
{
    (: First use the $activeInteractionId to locate the FulfillmentComponent under
_root/FulfillmentOrderManagement :)
    let $ffmOrdMgr := $orderData/osm:Data/oms:_root/oms:FulfillmentOrderManagement
    let $fulfillmentComponent :=
$ffmOrdMgr/oms:FulfillmentComponent[oms:FulfillmentOrder/oms:ActiveInteractionId/t
ext()=$activeInteractionId]
    return
        if (fn:exists($fulfillmentComponent))
        then
        (
            (: FulfillmentComponent found, get the componentKey :)
            let $componentKey := $fulfillmentComponent/oms:componentKey/text()
            return
                substring-before(substring-after($componentKey, "."), ".")
        )
        else ()
};
```

Failure During Revision

During the OSM fulfillment process, an order may fail due to various reasons like insufficient data, incorrect data and so on. To correct the failure, you may have to revise the failed order. In OSM, failure may occur even while revising the failed orders. With the existing functionality of Oracle Communications Order to Cash Integration Pack for Oracle Communications Order and Service Management, the following events happen when fallout occurs during revision:

- Initially, an update customer order status message, with Order Header status code (**FAILED**) and description "Order will be aborted due to failure during revision, manual intervention is required", is sent to Siebel CRM. This message contains the corresponding OrderLine status code/descriptions.
- The base Central Order Management/Service Order Management orders are put into **Aborted** state in OSM, followed by another update customer order status message to Siebel CRM, with Order Header status code (**FAILED** and description "Order is aborted". This message does not include any OrderLine status information.
- After the failure is resolved manually, Siebel CRM can resend the same customer order to OSM with the correct data.

When there are service order management and Provisioning systems involved in the customer order processing, the following events happen:

- The aborting request is propagated to all service order management systems and then all provisioning systems abort all associated service orders.
- The statuses of all associated service order management and provisioning AbortOrder requests are propagated back to central order management.
- While the order is in AbortInProgress state, incoming revisions are blocked.
- The statuses of the downstream order aborting operations are stored in the central order management or service order management order for reference.

It is mandatory that a fault thrown from a provisioning system must contain the service order state using an **AlternateObjectKey** element of the sales-order Identification element, where the ID element should have the attribute schemeID="SERVICE_ORDER_STATE". Otherwise, the fault is not treated as a fault during revision.

Since the **AIA_CreateProvisioningOrderQueue**, **AIA_CreateProvisioningOrderResponseQueue**, **ProcessProvisioningOrderEBM** and **UpdateFulfillmentOrderEBM** are reused for Abort Order Propagation,

- A new value **ABORT** in the **ProcessProvisioningOrderEBM** for the FulfillmentModeCode is introduced.
- For the **AIA_CreateProvisioningOrderResponseQueue**, a JMS message property, **CGBUPIPCFFALLOUT**, is introduced as: "CGBUPIPCFFALLOUT IS NULL" "CGBUPIPCFFALLOUT LIKE 'ABORT%'" for **AbortProvisioningOrderResponse**. So it is mandatory that this JMS message property is not stripped off or changed.

Adding a New Fulfillment Function for a New Service Offering

This procedure describes how to add a new fulfillment function for a new service offering. For more information about performing the actions in this procedure in Oracle Communications Design Studio, see the information about adding a new fulfillment function in the section on extending component cartridges in the Design Studio Modeling OSM Orchestration Help.

To add a new fulfillment function:

1. Create a new OSM project to host the new fulfillment function.
2. In the Order and Service Management Project editor **Properties** tab, deselect the **Standalone** check box.

   This allows the cartridge to be referenced in the composite cartridge as part of the solution (rather than as a standalone cartridge).
3. Delete the Order entity.
4. Create a base task for the new function from which all other new tasks will be extended.
5. Create any other tasks required by the new function by extending them from the base task created in the previous step.
6. Create a process that will execute when fulfilling the new function. You can create an entirely new process or have the new process extend an existing process.
7. Create the new fulfillment function that either extends from COM_ FulfillmentFunction or its extended fulfillment function, and specify keys for the order data at the following XPath location:
   - ./componentKey for order data/ControlData/Functions/*YourFunctionName*
   - ./orderItemRef/LineId for order data/ControlData/Functions/*YourFunctionName*/orderItem

   For more information about adding a new fulfillment function, see the information about adding new functional order components in the Design Studio Modeling OSM Orchestration Help.
8. If new fulfillment states are to be introduced for the new fulfillment function, add external fulfillment states for the new fulfillment function in the form of *State-YourFunctionName_Milestone*. For example, IN_ PROGRESS-*YourFunctionName*_START.
9. Optionally create a new fulfillment system for the fulfillment function.

   When you introduce a new fulfillment function, you may often also require a new fulfillment provider.
10. Create a new decomposition rule (with the COM_SalesOrderLine order item) that maps from the fulfillment function to the fulfillment provider.

    For example, DecompSyncCustomer_To_BRM-VoIP in the OracleComms_OSM_ O2A_TypicalTopology_Sample cartridge is a decomposition rule that maps SyncCustomer fulfillment function to the BRM-VoIP fulfillment provider.
11. Add the following cartridge to the Dependency tab for the new cartridge you created for the new fulfillment function:
    - OracleComms_OSM_O2A_COM_Base

    If there are any other cartridges that the new cartridge depends on, add them to the Dependency tab.
12. Add the cartridge you created for the new fulfillment function, to the Dependency tab for the Order-To-Activate composite cartridge.
13. Create composite cartridge views in the cartridge you created for the new fulfillment function.

- Create a composite cartridge view that adds data to the sales order creation task COM_SalesOrderFulfillment_CreationTask for the new fulfillment function. This composite cartridge view should extend from the base task of the new fulfillment function.
- Create a composite cartridge view that adds data to the sales order query tasks such as COM_SalesOrder_StateChangeView and COM_SalesOrder_AggregatedOLMView for the new fulfillment function.

14. Add the data for the new tasks you created to the composite cartridge views.
15. Create a task data contribution to extend the existing sales order creation task with the following information:
    - Order = COM_SalesOrderFulfillment
    - Process = COM_SalesOrderFulfillmentOrchestrationProcess
    - Task = COM_SalesOrderFulfillment_CreationTask
    - Composite Cartridge View = *YourCompositeCartridgeViewForCreationTask*
16. Create a query task data contribution to extend the existing sales order query tasks with the following fields:
    - Order = COM_SalesOrderFulfillment
    - Role = COM_SalesOrder_AggregatedOLM_Role, COM_SalesOrder_StateChange_Roles
    - Query Task = COM_SalesOrder_AggregatedOLMView, COM_SalesOrder_StateChangeView
    - Composite Cartridge View = *YourCompositeCartridgeViewForQueryTask*
17. In the **resources/SolutionConfig** folder of the Order-to-Activate composite cartridge such as OracleComms_OSM_O2A_COMSOM_TypicalSolution:
    - Add a new <Component> entry to the ComponentExtensionPointMap.xml file for the new fulfillment function. For each applicable fulfillment function extension point, create an XQuery file based on an existing fulfillment function extension point XQuery file, for example, SyncCustomerComplete_Event.xquery.
    - Add a new entry of a query task to the ComponentQueryViewMap.xml file for the new fulfillment function.
    - If new milestones are to be introduced for the new fulfillment function:
      - add a new <MilestoneMap> entry to the ComponentMilestoneMap.xml file for the new fulfillment function
      - add a new <StatusItemContext> entry for each new milestone to the OrderItemStatusContextMap.xml file
    - If fallout simulation is needed for the new fulfillment function, add a new <FaultModeMap> entry to the FaultModeMap.xml file.
    - Add a new <StatusMap> entry to the OrderStateMap.xml file for the new fulfillment function per system type and fulfillment mode.
    - If a new fulfillment provider is added, add a new <targetSystem> entry for each new fulfillment provider instance to the TargetSystemMap.xml file.
    - For each automated task in the process of the new fulfillment function, add a new <TaskExitStatusMap> entry to the TaskExitStatusMap.xml file.

18. Package and deploy the Order-To-Activate composite cartridge.

Adding a New Fulfillment Provider

Oracle AIA has logical identifiers for fulfillment providers (for example, fulfillment instances). There are naming conventions that must correspond to your Oracle AIA deployment, for example, fulfillment system type and fulfillment system code. Currently the logical identifiers and fulfillment system type and application are defined in XML Document **TargetSystemMap.xml**, which is deployed with the Order-to-Activate composite cartridge (such as OracleComms_OSM_O2A_COMSOM_TypicalSolution). You modify this file when restructuring the fulfillment topology definition, for example when you add more billing system instances. The Sender IDs and Target IDs in the EBM messages must match the logical identifiers for the system instances configured in the Oracle AIA deployment. The following is the summary for fulfillment functions for Oracle AIA:

- Naming convention used for OSM central order management instances: OSMCFS_01, OSMCFS_02, and so on
- Naming convention used for OSM service order management instances: OSMPROV_01, OSMPROV_02, and so on
- Naming convention used for Billing and Revenue Management instances BRM_01, BRM_02, and so on
- Naming convention used for Siebel CRM instances: SEBL_01, SEBL_02, and so on

When adding a fulfillment provider, such as a billing system instance, you must customize a copy of the following files to map the Studio entity name of the system entity to the target ID:

- *O2A_CompositeCartridge***resources\SolutionConfig\TargetSystemMap.xml**

  For example,

  OracleComms_OSM_O2A_COM_CSO_Solution\resources\SolutionConfig\TargetSystemMap.xml contains all fulfillment providers and their logical identifiers used in the Order-to-Activate cartridges in the Typical topology.

You can name the Studio entity using the Oracle AIA naming convention such as BRM_01, BRM_02, to simplify the fulfillment system mapping to be a direct mapping.

The following procedure describes how to add a new fulfillment provider. For more information about performing the actions in this procedure in Design Studio, see the Design Studio Modeling OSM Orchestration Help.

To add a new fulfillment provider:

1. In the topology cartridge such as the OracleComms_OSM_O2A_TypicalTopology_Sample cartridge, add a new order component specification that extends COM_FulfillmentSystem with COM_SalesOrderFulfillment namespace to represent the new fulfillment provider, and ensure the **Order Component Executable** check box is deselected.
2. Open the decomposition rule in the form of *Topology_*DetermineProcessingGranularity_For_*FulfillmentFunction* in the OracleComms_OSM_O2A_COM_CSO_Topology, OracleComms_OSM_O2A_TypicalTopology, or OracleComms_OSM_O2A_SimpleTopology_Sample cartridge depending on the solution option and topology.

For example, Typical_DetermineProcessingGranularity_For_SyncCustomer is the decomposition rule for SyncCustomerFunction fulfillment function in the Typical topology.

3. In the Decomposition Rule editor **Source/Target** tab, select the desired processing granularity under COM_FulfillmentGranularity in the **Target Order Components** section.

   See "Configuring a New Processing Granularity Rule" for more information on creating a new processing granularity.

4. (Optional) Add or Change decomposition condition in the Decomposition Rule editor **Conditions** tab.

5. If you are using the Calculate Service Order option in your Order-to-Activate cartridges and you want to add a new fulfillment provider that makes use of the **Calculate Service Order** transformation sequence, create an order component that references the **Calculate Service Order** provider function. You can use an existing provisioning order component as an example of the correct way to model this.

6. Package and deploy the Order-to-Activate composite cartridge.

Adding a New Fulfillment Mode

This procedure describes how to add a new fulfillment mode. For more information about performing the actions in this procedure in Design Studio, see the information about adding a new fulfillment mode in the section on extending component cartridges in the Design Studio Modeling OSM Orchestration Help.

To add a new fulfillment mode:

1. Create a new Order and Service Management project to host the new fulfillment mode.

2. In the Order and Service Management Project editor **Properties** tab, deselect the **Standalone** check box.

   This allows the cartridge to be referenced in the composite cartridge as part of the solution (rather than as a standalone cartridge with no dependencies).

3. Add a new fulfillment mode with COM_SalesOrderFulfillment namespace for central order management or SOM_ProvisionOrderFulfillment namespace for service order management.

4. Modify the order recognition rule's XQuery to recognize the new fulfillment mode.

   a. Copy the existing **OracleComms_OSM_O2A_COM_Base/resources/OrderRecognitionRule/AIAOrderRecognition.xquery** to the resources folder of the new OSM cartridge created in step 1, if the new fulfillment mode is for central order management. For service order management, copy the existing **OracleComms_OSM_O2A_SOM_Base/resources/OrderRecognitionRule/ProvisionOrderRecognition.xquery** to the resources folder of the new OSM cartridge.

   b. Modify the XQuery to recognize the new fulfillment mode.

   c. Create an XML Catalog rewrite rule to override the order recognition rule XQuery.

      For example, for central order management's AIAOrderRecognition.xquery

      ```
      <rewriteURI
      ```

```
uriStartString="http://xmlns.oracle.com/communications/ordermanagement/o2ac
ombase/order_recognition/AIAOrderRecognition.xquery"
rewritePrefix="osmmodel:///<New_OSM_Cartridge>/1.0.0.0.0/resources
/AIAOrderRecognition.xquery"/>
```

5. Add the new fulfillment mode to the base fulfillment pattern or to any applicable fulfillment patterns.
6. Model the orchestration plan for the new fulfillment mode for all of the affected fulfillment patterns.
7. In the Fulfillment State Map editor, create fulfillment state mappings for the new fulfillment mode. The fulfillment state map to edit is one of the following:
   - For the calculate service order option:

     In the OracleComms_OSM_O2A_COM_CSO_FulfillmentStateMap cartridge, edit the COM_CSO_FulfillmentStateMap entity.
   - For the solution option without calculate service order:

     In the OracleComms_OSM_O2A_FulfillmentPatternMap cartridge, edit the COM_FulfillmentStateMap entity.
8. Add the cartridge you created for the new fulfillment mode, to the Dependency tab for the Order-to-Activate composite cartridge.
9. Open the Ant view and select **Add Buildfiles** to add **SolutionConfig.xml** in the Order-to-Activate composite cartridge, for example OracleComms_OSM_O2A_COMSOM_TypicalSolution.
10. In the Ant view, expand the Order-to-Activate composite cartridge and double-click **config_Metadata_And_ModelVariable** to ensure that the new fulfillment mode is visible in the Order-to-Activate composite cartridge.
11. Package and deploy the Order-to-Activate composite cartridge.

Adding a New Product Specification

This section provides information about adding a new product specification.

Mapping Product Specifications to Order-to-Activate Sample Fulfillment Patterns

There are two types of product specifications in Design Studio: conceptual model products and OSM product specifications. You do not create a new product specification in OSM, although OSM supports existing product specifications that were created in OSM. Instead, to create a new product, create a new conceptual model Product entity, which will then be mapped to a conceptual model fulfillment pattern, which is realized into an OSM fulfillment pattern. For more information about conceptual model products, see *Design Studio Concepts*.

Product specification entities can either be imported from a product catalog, such as Oracle Product Hub, or manually created in Design Studio. If you import products, the result will be new conceptual model products, not new OSM products as it was in earlier releases of Design Studio and OSM.

For more information about mapping product specifications to fulfillment patterns in OSM, see the XQuery appendix in *OSM Concepts*.

Each product specification can be mapped to an Order-to-Activate sample fulfillment pattern. Design Studio generates a mapping file in the resources folder of the OracleComms_OSM_O2A_FulfillmentPatternMap_Sample cartridge. In Oracle AIA, a field called fulfillment item code (FIC) is used to specify the product specification name.

Note: In communications industry customer orders in AIAEBM format, 'fulfillment item code' is a unique identifier that maps an order line item subject to a Studio recognized fulfillment pattern entity. By default, this is populated with the product specification name for product specification mapping.

Alternatively you can use any combination of attribute values on the order template to drive the mapping of order lines to fulfillment patterns. The value 'item class name' determines some business classification of a product and can be used for product specification mapping. It becomes significant when you adopt a methodology that aligns commercial product specifications with fulfillment commercial services (fulfillment patterns).

When new FulfillmentItemCodes are introduced, you must ensure a mapping of the new FulfillmentItemCode to a fulfillment pattern that exists in the **OracleComms_OSM_O2A_FulfillmentPatternMap_Sample/resources/productClassMapping/productClassMapping.xml** file so that product specification validation will succeed.

Creating a New Product

This section contains information about creating a new conceptual model product to work with the Order-to-Activate cartridges. For more information about creating a product in Design Studio, see the information about adding a new product specification in the section on extending component cartridges in the Design Studio Modeling OSM Orchestration Help.

A new product is created in the same way as any new conceptual model product in Design Studio, with the following specifics:

1. If you create a new product that requires a new fulfillment pattern, add a new conceptual model fulfillment pattern and then add a new OSM fulfillment pattern that realizes the new conceptual model fulfillment pattern and that extends from the base fulfillment pattern **BaseProductSpec**.

Note: If you create a new conceptual model cartridge to contain your new conceptual model entities, you must add that cartridge in the Common Model Entity Container field in an appropriate cartridge. If you are using the Calculate Service Order solution option, add the cartridge to the OracleComms_OSM_O2A_COM_CSO_Model_Container cartridge for a central order management (or combined central order management and service order management) environment, or to OracleComms_OSM_O2A_SOM_CSO_ModelContainer for an environment that contains only service order management. If you are using the solution option without Calculate Service Order, add the cartridge to the recognition cartridge in your environment or create a new OSM component cartridge to contain the entries.

2. Specify the location of the external directory containing the fulfillment pattern. From the **Window** menu, select **Preferences**, then expand **Oracle Design Studio** in the Preferences navigation tree, then select **Order and Service Management Preferences**, and then select **Orchestration Preferences**. Enter the appropriate directory in the **Product Specification Mapping** field. For example, enter **OracleComms_OSM_O2A_COM_ FulfillmentPattern/resources/productSpecMapping** if you are using the calculate service order option or **OracleComms_OSM_O2A_FulfillmentPatternMap_ Sample/resources/productSpecMapping** if you are using the option without calculate service order.

Creating a New Fulfillment Pattern

This procedure describes how to add a new fulfillment pattern. For more information about performing the actions in this procedure in Design Studio, see the information about adding a new fulfillment pattern in the Design Studio Modeling OSM Orchestration Help.

To add a new fulfillment pattern:

1. Create a new Order and Service Management project to host the new fulfillment pattern or optionally use the existing fulfillment pattern sample cartridge to host the new fulfillment pattern.
2. In the Order and Service Management Project editor **Properties** tab, deselect the **Standalone** check box.

   This allows the cartridge to be referenced in the composite cartridge as part of the solution (rather than as a standalone cartridge with no dependencies).
3. Add a new fulfillment pattern that extends from the base fulfillment pattern **BaseProductSpec** or its extended fulfillment pattern.
4. Model the orchestration plan for the new fulfillment pattern such as indicating which fulfillment functions are decomposed for which fulfillment mode.
5. Include the new fulfillment pattern on the appropriate decomposition rules that map order line items from fulfillment functions to fulfillment providers. For example, DecompSyncCustomer_To_BRM-VoIP is a decomposition rule that maps order line items for SyncCustomerFunction to BRM-VoIP fulfillment provider.
6. If you added a new cartridge to host the new fulfillment pattern, add the cartridge to the Dependency tab for the Order-to-Activate composite cartridge.
7. Package and deploy the Order-to-Activate composite cartridge.

Customizing Mapping Rules

The following section describes customizing rules for mapping order items to Order-to-Activate sample fulfillment patterns.

The XQuery function **osmpip:getProductSpec()** defined in the OracleComms_OSM_ O2A_CommonUtility cartridge provides an API for mapping customer order items to Order-to-Activate sample fulfillment patterns.

The XQuery function **osmpip:getProductSpec()** is invoked from the XQuery in the **productSpec** order item property in the **COM_SalesOrderLine** order item specification in the OracleComms_OSM_O2A_COM_Base cartridge. By default, the direct mapping rules map the Fulfillment Item Code to a fulfillment pattern.

Customize Rules for Mapping Order Items to Fulfillment Patterns

Customize a copy of the **OracleComms_OSM_O2A_COM_Base/resources/OrderItemProperties/ProductSpec.xquery** file to customize the direct mapping rule, such as to pass as input the Fulfillment Item Code, to adopt a methodology that aligns the commercial product specifications with the fulfillment patterns when the Fulfillment Item Code is not populated.

Customize Rules for Mapping Order Items with Product Specifications to Fulfillment Patterns

Customize a copy of OracleComms_OSM_O2A_**CommonUtility/resources/ProductClassToProductSpec.xquery** file to enable the mapping of order items with new product specifications on the customer order to Order-to-Activate sample fulfillment patterns. The **ProductClassToProductSpec.xquery** file also provides access to common XQuery functions across the central order management cartridges.

Table 5–51 lists the functions found in **ProductClassToProductSpec.xquery**:

Table 5–51 Functions in ProductClassToProductSpec.xquery

Name	Function Interface	Description

getProductSpec	`declare function` **`osmpip:getProductSpec`**`(`
	`    $salesOrder as element(),`
	`    $salesOrderLine as element(),`
	`    $productClassName as xs:string*) as xs:string`
		This function returns the fulfillment pattern entity as a string based on the given Fulfillment Item Code of a customer order line. If the input value is not specified, or no direct mapping is found, the function uses mapping rules based on data provided on the customer order.
		The mapping rules, applicable when no direct mapping is found, should check (in the order listed) whether there is a mapping rule specific to:
		1. Subject of the order line item, if one exists (One of: Discount, Product, Service Bundle, Offer, SpecialRating, or Unknown).
		2. Subject Type (a short way to get couple of other attributes), if one exists.
		3. Unknown (as a last resort, use a default mapping rule that maps the Order Line Item to a Fulfillment Item Code of a special Unknown item action).
		The default mapping rules achieve the following mappings:
		■ Service bundle lines having no FulfillmentItemCode and identified by ServiceInstanceIndicator=true, are mapped to a fulfillment pattern of its child order lines.
		■ OFFER lines are mapped to fulfillment pattern 'NonService.Offer'.
		■ DISCOUNT, SPECIAL RATING, and BUNDLE lines having no FulfillmentItemCode are mapped to fulfillment pattern 'NonService.BillingItem'.
		■ When a fulfillment pattern cannot be determined, it is set to fulfillment pattern 'Service.Unknown'.

getDoublePlayPrimaryClassificationCode	`declare function` **`osmpip:getDoublePlayPrimaryClassificationCode`**`(`
	`    $orderline as element(),`
	`    $salesOrder as element()) as xs:string*`
		This function returns the classification code of a customer order line. The classification code is used by decomposition rule conditions and order line item dependencies.
		The classification of order line items is based on the fulfillment topology definition. For example, for order lines in which the fulfillment pattern itself is not sufficient to determine the billing provider (such as offer, discount, and bundle lines) order line items are classified into VoIP only, BroadBand only, or combination Broadband and VoIP, based on the demonstration Typical fulfillment topology definition to determine the appropriate billing provider. Offer and bundle lines go to as many different, unique billing providers as in its child lines in the customer order. Discount lines, if contained in a service bundle, follow the service bundle.

getBillingPattern	`declare function` **`osmpip:getBillingPattern`**`(`
	`    $orderline as element()) as xs:string`
		Return the billing pattern of the current customer order line.

Importing the New Product Specification

It is possible to query product specifications and transaction attributes into Design Studio directly from the Oracle Product Hub. Design Studio users use the existing Oracle AIA interface QueryProductClassAndAttributesSCECommsReqABCSImpl to import product specifications from both Siebel CRM and the Product Hub. When product specifications are queried using this interface, the interface API checks for Product Hub implementation in the solution stack, and if it is there, the product specifications will be imported to Design Studio from the Product Hub. Otherwise, the product specifications will be imported from Siebel CRM.

Import the new product specifications as described in the Design Studio Help. After importing the product specifications, follow this procedure:

1. Open the newly imported or modified conceptual model product.
2. Map the new or changed product to the appropriate conceptual model fulfillment pattern.

Changing Processing Granularity

This section provides information on changing the processing granularity for an order item.

To change processing granularity for a fulfillment function:

1. Open the decomposition rule in the form of *Topology_*DetermineProcessingGranularity_For_*FulfillmentFunction* in the OracleComms_OSM_O2A_COM_CSO_Topology, OracleComms_OSM_O2A_TypicalTopology, or OracleComms_OSM_O2A_SimpleTopology_Sample cartridge depending on the solution option and topology.

   For example, Typical_DetermineProcessingGranularity_For_SyncCustomer is the decomposition rule for SyncCustomerFunction fulfillment function if you are using the calculate service order option.

2. In the Decomposition Rule editor **Source/Target** tab, select the desired processing granularity under COM_FulfillmentGranularity in the **Target Order Components** section.

   See "Configuring a New Processing Granularity Rule" for more information on creating a new processing granularity.

3. (Optional) Add or Change decomposition condition in the Decomposition Rule editor **Conditions** tab.
4. Package and deploy the Order-to-Activate composite cartridge.

Configuring a New Processing Granularity Rule

Begin by creating a new order component specification in Design Studio that extends COM_FulfillmentGranularity and has the COM_SalesOrderFulfillment namespace and give it a name such as *YourCustomGranularity*.

On the **Component ID** tab of the newly created order component specification, specify an XQuery condition to return the ComponentId of an order line. Order lines with the same ComponentId value are grouped together for the processing of order lines a group at a time. To construct the XQuery, you can copy the XQuery expression from the **Component ID** tab of an existing processing granularity rule, such as BundleGranularity shown in Example 5–21, and in it replace BundleGranularity with *YourCustomGranularity*, and replace TypeCode=(BUNDLE) with the condition that identifies the parent of the group. If you require nested groups to make it on separate fulfillment requests, also replace **[fn:last()] with [1]**. There are multiple instances of index fn:last() in the XQuery expression; the ones to replace are highlighted below. Otherwise, nested groups are processed on the same fulfillment request.

Example 5–21 Customizable Granularity Configuration XQuery

```
(: Copyright (c) 2008, 2012, Oracle and/or its affiliates. All rights reserved. :)
```

```
import module namespace comqueryviewconstants =
"http://xmlns.oracle.com/communications/ordermanagement/o2acombase/comqueryviewcon
stants" at
"http://xmlns.oracle.com/communications/ordermanagement/o2acombase/constants/Query
ViewConstants.xquery";

declare namespace
osm="http://xmlns.oracle.com/communications/ordermanagement/model";
declare namespace prop="COM_SalesOrderFulfillment";
declare namespace osmfn =
"java:oracle.communications.ordermanagement.orchestration.generation.Orchestration
XQueryFunctions";

let $ancestors := osmfn:ancestors($comqueryviewconstants:COM_ORDER_ITEN_
SPEC,$comqueryviewconstants:COM_PARENTCHILD_HIER, $comqueryviewconstants:COM_
ORDER_NAMESPACE)
let $relatedItems := osmfn:ancestors($comqueryviewconstants:COM_ORDER_ITEN_
SPEC,$comqueryviewconstants:COM_RELATEDITEM_HIER,$comqueryviewconstants:COM_ORDER_
NAMESPACE)
return
    (: to locate the bundle that this order line should be included with on a
fulfillment request, first follow the RelatedSalesOrderLineId references (if any
exist) :)
    if (osm:properties/prop:RelatedSalesOrderLineId/text() != '' and
fn:exists($relatedItems))
    then
    (
        let $topmostRelatedItem := $relatedItems[fn:last()]
        let $topmostRelatedItemAncestors :=
osmfn:ancestors($topmostRelatedItem,$comqueryviewconstants:COM_ORDER_ITEN_
SPEC,$comqueryviewconstants:COM_PARENTCHILD_HIER,$comqueryviewconstants:COM_ORDER_
NAMESPACE)
        return
        if
(fn:exists($topmostRelatedItemAncestors[osm:properties/prop:TypeCode=('BUNDLE')]))
        then
        (
            (: for the topmost related order line, follow the ParentLineId
references to locate the outermost bundle. This will cause nested bundles to make
it on the same fulfillment request. :)

concat($topmostRelatedItemAncestors[osm:properties/prop:TypeCode=('BUNDLE')][fn:la
st()]/osm:properties/prop:BaseLineId/text(),'/BundleGranularity')
        )
        else
        (
            (: locate the root node, such that any other root node on the order
along with their related order items makes a separate fulfillment request :)

concat($topmostRelatedItemAncestors[fn:last()]/osm:properties/prop:BaseLineId/text
(),'/BundleGranularity')
        )
    )
    else
    (
        if (fn:exists($ancestors[osm:properties/prop:TypeCode=('BUNDLE')]))
        then
        (
            (: follow the ParentLineId references to locate the outermost bundle.
This will cause nested bundles to make it on the same fulfillment request. :)
concat($ancestors[osm:properties/prop:TypeCode=('BUNDLE')][fn:last()]/osm:properti
es/prop:BaseLineId/text(),'/BundleGranularity')
        )
        else
        (
            (: locate the root node, such that any other root node on the order
along with their related order items makes a separate fulfillment request :)

concat($ancestors[fn:last()]/osm:properties/prop:BaseLineId/text(),'/BundleGranula
rity')
        )
    )
```

Changing Fulfillment Function Dependencies

This procedure describes how to change fulfillment function dependencies. For more information about performing the actions in this procedure in Design Studio, see the information about the fulfillment pattern editor in the Design Studio Modeling OSM Orchestration Help.

To change fulfillment function dependencies:

1. Open the Fulfillment Pattern editor **Orchestration Plan** tab for the base fulfillment pattern or any applicable fulfillment pattern for which the dependencies between fulfillment functions needed to be changed.
2. Select the fulfillment mode, for which the dependencies between fulfillment functions need to be changed, in the **Fulfillment Mode** field.
3. Select the **Dependencies** tab and do one of the following:
   - In the **Dependencies** table, select the dependency and change the **From Order Component** or **To Order Component** to update the fulfillment function dependency.
   - In the **Dependencies** table, add a new dependency and specify the **From Order Component** and **To Order Component** for the new fulfillment function dependency.
4. Package and deploy the Order-to-Activate composite cartridge.

Setting a Point of No Return

A point of no return (PoNR) is a point during the orchestration process when revisions are no longer accepted and processed for an order. The Hard PoNR indicates that it is technically infeasible to amend the order.

A PoNR is realized when a condition is met on an order line item.

The seeded values in the Order-to-Activate cartridges are:

- A value of **NOT YET** indicates that the Soft PoNR has been reached for an order line.
- A value of **HARD** indicates that a Hard PoNR has been reached for an order line, which signifies that it is technically infeasible to revise the order beyond this point.

Each fulfillment pattern may have a PoNR set to HARD at a different fulfillment state in the fulfillment flow.

In the Order-to-Activate cartridge, the value of PoNR for each order line item is stored in ControlData/OrderItem/WorkLineItemData/RevisionPermissibleCode in the order data in the format [SOFT]NOT YET or [HARD]HARD.

A fulfillment state is set before and after each fulfillment function. There may be multiple fulfillment states during the progress of a fulfillment function, such as IN_PROGRESS-PROVISION_DESIGNED and COMPLETE-PROVISION_COMPLETE. In this case, the RevisionPermissibleCode value is returned in the fulfillment data updates from provisioning and is updated into the order data.

RevisionPermissibleCode must be updated at every fulfillment function transition because there is no guarantee on the conditions that cause fulfillment functions to be called in a fulfillment pattern. Update the order data as follows:

- Before every fulfillment function until PoNR is reached (if a PoNR is non-existent or null, set **RevisionPermissibleCode** to **NOT YET**)
- For every function transition after which the HARD PoNR is reached (if a PoNR is non-existent, null, or has the value **NOT YET**, set **RevisionPermissibleCode** to **HARD**)
- productSpec1: (NOT YET) FunctionA --> (NOT YET)--> FunctionB--> (HARD) --> FunctionC
- productSpec2: (NOT YET) FunctionA --> (HARD) --> FunctionB --> (HARD) --> FunctionC

A HARD PoNR is set at the FunctionC Start milestone for productSpec1, and at the FunctionB Start milestone for productSpec2. Assuming that FunctionB may be skipped if a conditional expression is not met, PoNR=HARD must be set between FunctionA and FunctionC: (NOT YET): FunctionA --> (HARD) --> FunctionC.

For fulfillment functions, these values are implemented in the order data updates in the automated tasks of the subprocess. For example:

Table 5–52 lists the various hard PoNRs for each fulfillment pattern.

Table 5–52 Hard Points of No Return by Fulfillment Pattern

Product Spec	Fulfillment Function	HARD PoNR is set	Fulfillment State
NonService.BillingInitatedItem	FulfillBillingFunction	Y	IN_PROGRESS-FULFILL_BILLING_START
NonService.BillingItem	FulfillBillingFunction	Y	IN_PROGRESS-FULFILL_BILLING_START
NonService.Offer	FulfillBillingFunction	Y	IN_PROGRESS-FULFILL_BILLING_START
Service.Broadband	ProvisionOrderFunction	Y	For solutions that are using the Calculate Service Order option: IN_PROGRESS-PROVISION_ISSUED For solutions that are not using the Calculate Service Order option: IN_PROGRESS-PROVISION_DESIGNED
Service.CPE.Broadband	ShipOrderFunction	Y	COMPLETE-SHIP_ORDER_SHIPPED
Service.CPE.VoIP	ShipOrderFunction	Y	COMPLETE-SHIP_ORDER_SHIPPED
Service.Install	InstallOrderFunction	Y	IN_PROGRESS-INSTALL_COMMITTED
Service.VoIP	FulfillBillingFunction	Y	IN_PROGRESS-FULFILL_BILLING_START

OSM enforces HARD PoNR in the order life-cycle policy by disallowing Submit Amendment if the PoNR value in the order data is found to be HARD (for a revised order line item) when a revision order arrives.

PoNR (SalesOrderLine/RevisionPermissibleCode) is included in status updates to Siebel CRM, which occur at every milestone data change. This enables Siebel CRM to enforce the rule that a revision cannot be submitted beyond HARD PoNR for an order line.

Modeling a PoNR

This section describes the steps to add a point of no return for a fulfillment pattern. For more information about performing the actions in this procedure in Design Studio, see the information about configuring points of no return in the Design Studio Modeling OSM Orchestration Help.

To model a point of no return:

1. In the Fulfillment Pattern editor **Orchestration Plan** tab, select the fulfillment mode for which a PoNR will be added for the fulfillment pattern, and also select a fulfillment function in **Order Components** section at which the PoNR will be set.
2. Once a fulfillment function is selected, add an entry to the **Point of No Return Values** box in the **Point of No Return** subtab. Either click **Select** and select an existing PoNR value, or click **Add** and add a new PoNR value. Then select a fulfillment state that will trigger this PoNR for the fulfillment flow of the fulfillment pattern.
3. If you added a new PoNR value rather than selecting an existing value, it will automatically be added as a hard PoNR. If you would like your new PoNR to be a soft PoNR, click the **Details** tab in the Fulfillment Pattern editor, select your new PoNR from the **Point of No Return Values** box and then deselect **Hard Point of No Return** in the **Details** subtab.

Note: Point-of-no-return enforcement can be disabled for testing purposes when a new fulfillment function is introduced, or when revision or order cancellation testing is performed. See "Controlling Point of No Return" for more information.

Configuring Fulfillment States

This section describes the steps for configuring fulfillment states. For more information about performing the actions in this procedure in Design Studio, see the information about configuring fulfillment states in the Design Studio Modeling OSM Orchestration Help.

To configure fulfillment states:

1. If a new fulfillment function is introduced, add external fulfillment states to represent status information sent to OSM by fulfillment systems in the Order Component Specification editor **External Fulfillment States** tab for that order component specification. The external fulfillment states should be in the form of *State-YourFunctionName_Milestone*. For example, IN_ PROGRESS-*YourFunctionName*_START, representing the starting of your fulfillment function.
2. In the Fulfillment State Map editor, create fulfillment state mappings for each new external fulfillment state for any applicable fulfillment mode for the COM_

SalesOrderLine order item. The fulfillment state map to edit is one of the following:

- For the calculate service order option:

  In the OracleComms_OSM_O2A_COM_CSO_FulfillmentStateMap_Sample cartridge, edit the COM_CSO_FulfillmentStateMap entity.

- For the solution option without calculate service order:

  In the OracleComms_OSM_O2A_FulfillmentPatternMap_Sample cartridge, edit the COM_FulfillmentStateMap entity.

3. In the Order Item Fulfillment State Composition Rule Set editor for COM_OrderItemStateCompositionRule (in the same cartridge as the fulfilment state map), modify existing composition rules for the BaseProductSpec fulfillment pattern and COM_SalesOrderLine order item if necessary.
4. In the Order Fulfillment State Composition Rule Set editor for COM_OrderStateCompositionRule (in the same cartridge as the fulfilment state map), modify existing composition rules for the COM_SalesOrderFulfillment order if necessary.
5. If a new fulfillment state is introduced in addition to the base fulfillment states (that is, OPEN, IN_PROGRESS, COMPLETE, FAILED and CANCELLED) defined in COM_FulfillmentStateMap, modify the XQuery implementation for any applicable fulfillment state extension points. See Table 5–53 for more information about fulfillment state extension points.
6. If a new fulfillment state is introduced to the solution, ensure that the state is mapped appropriately in the Order Lifecycle Manager entity in the OracleComms_OSM_O2A_COM_Base cartridge. If you need to add or change any mappings, you must first unseal the OracleComms_OSM_O2A_COM_Base cartridge. Oracle recommends resealing the cartridge after you have made your changes. See the information about the Order Lifecycle Manager entity in the Design Studio Modeling OSM Processes Help for more information about viewing and changing these mappings.
7. Package and deploy the Order-to-Activate composite cartridge.

External Fulfillment States

In Order-to-Activate cartridges, the following structure in the order template is required for processing fulfillment states to support revision of orders. The default location for this structure is in the root level of the order data. The data elements (as opposed to the structure elements) are indicated in bold below. These values should be populated by the task that handles the interaction with the external system.

```
OrderLifeCycleManagement
   OrderItemStatus
      BaseLineId
      LineType
      LineName
      OrderItemComponentStatus
         componentKey
         componentType
         systemType
         MilestoneStatusRecord
            StatusTimestamp
            ExternalFulfillmentStateCode
            MilestoneCode
```

```
ExecutionMode
componentId
compensationId
Status
   Code
   Description
```

See the information about modeling order template structures for fulfillment states in the *OSM Developer's Guide* for more information.

Fulfillment State Extension Point Interface

The Order-to-Activate cartridges use XQuery resources to perform functions including setting order item properties, mapping product specifications to fulfillment patterns, managing fulfillment function dependencies, and managing the order life cycle. One way to customize XQueries is to rewrite or add to the out-of-box XQuery module and use the XML catalog to enable URI reference mapping. Extension points are defined for both fulfillment functions and fulfillment states. This section contains information about the fulfillment state extension points. For information about the fulfillment function extension points, see "Fulfillment Function Extension Point Interface."

XML catalogs are system-wide entities, which means an XML Catalog specified in one cartridge will be used when processing requests for orders on other cartridges. With the use of solution cartridges, multiple solutions can be deployed to a single system and coexist with each other.

An XQuery extension script must be implemented in a standalone file. The file URI must be registered to the extension configuration.

Fulfillment State Extension Point Overview

Table 5–53 lists the XQuery extension points for fulfillment states in the Order-to-Activate cartridges.

Table 5–53 Fulfillment State Extension Points

Fulfillment State Extension Point	Description
ORDERITEM_FULFILLMENT_STATE_UPDATED	ORDERITEM_FULFILLMENT_STATE_UPDATED is triggered when the OSM fulfillment state engine finishes evaluating the composite fulfillment state for an order item. This extension point overrides the default evaluation of the order item composite fulfillment state modeled in COM_OrderItemStateCompositionRule to support an order with no order items (order items had been dropped during revision), and to support the completion of a cancellation order.
ORDER_FULFILLMENT_STATE_UPDATED	ORDER_FULFILLMENT_STATE_UPDATED is triggered when the OSM fulfillment state engine finishes evaluating the composite fulfillment state for an order. This extension point overrides the default evaluation of the order composite fulfillment state modeled in COM_OrderStateCompositionRule to support an order with no order items (order items had been dropped during revision), and to support the completion of a cancellation order.
ORDER_STATUS	ORDER_STATUS at the order level is triggered when the OSM fulfillment state engine finishes evaluating the composite fulfillment state for the order. This extension point provides order status to the upstream system. ORDER_STATUS at the order item level is triggered when the OSM fulfillment state engine finishes evaluating the composite fulfillment state for an order item. This extension point provides order item status to the upstream system.

Table 5–53 (Cont.) Fulfillment State Extension Points

Fulfillment State Extension Point	Description
ORDER_STATUSCONTEXT	ORDER_STATUSCONTEXT is triggered when the OSM fulfillment state engine finishes evaluation of the composite fulfillment state for the order. This extension point provides order status context to the upstream system.
ORDERITEM_MILESTONE	ORDERITEM_MILESTONE is triggered when the OSM fulfillment state engine finishes evaluating the composite fulfillment state for an order item. This extension point calculates the order item milestone, taking order cancellation into consideration.
ORDERITEM_STATUSCONTEXT	ORDERITEM_STATUSCONTEXT is triggered when the OSM fulfillment state engine finishes evaluating the composite fulfillment state for an order item. This extension point provides order item status context to the upstream system.
REPORT_ORDERITEM_STATUS	REPORT_ORDERITEM_STATUS is triggered when the OSM fulfillment state engine finishes calculating the composite fulfillment state for an order item. This extension point is not currently being used in the Order-to-Activate cartridges.
REPORT_ORDERITEM_ MILESTONE	REPORT_ORDERITEM_MILESTONE is triggered when the OSM fulfillment state engine finishes evaluating the composite fulfillment state for an order item. This extension point overrides the default milestone when handling a cancellation order.
REPORT_ORDERITEM_ STATUSCONTEXT	REPORT_ORDERITEM_STATUSCONTEXT is triggered when the OSM fulfillment state engine finishes calculating the composite fulfillment state for an order item. This extension point overrides the default evaluation of the order item composite fulfillment state modeled in COM_ OrderItemStateCompositionRule to support different order item status contexts for failed order items with different order fulfillment modes.

Fulfillment state extension points provide a means to handle additional context, such as fulfillment mode and order types (cancel order, revision order, etc), in Order-to-Activate cartridges. This enables you to alter the default behavior modeled in both order composition rules and order item composition rules.

For example, in the ORDER_FULFILLMENT_STATE_UPDATED fulfillment state extension point, the order fulfillment state is changed from what is configured in the COM_OrderStateCompositionRule for the following scenarios:

- If an order has no child order items (because existing lines were dropped in a revision order), the order fulfillment state should be CANCELLED (fulfillment mode CANCEL) or COMPLETED (fulfillment mode DELIVER) instead of PENDING.
- If an order is in progress, the order fulfillment state should be IN_PROGRESS regardless of whether all of the order items have been completed.
- If an order is completed, the order fulfillment state should be COMPLETE or, if the order’s fulfillment mode is CANCEL, CANCEL COMPLETE.

ORDERITEM_FULFILLMENT_STATE_UPDATED Extension Point

This section describes the XQuery script that implements the logic to handle the ORDERITEM_FULFILLMENT_STATE_UPDATED extension point. The extension point detects an orphaned order item and sets the fulfillment state value for the orphaned order item. For example, an order item might have no child order item and may not invoke any components (possibly because the order itself does not have any lines in its base order, or because existing lines or components have been removed due to an order amendment). The core fulfillment state engine will not update the order item's fulfillment state, and the order item fulfillment state will remain as PENDING (if no order line was present) or IN_PROGRESS (if an order line has been started but has been removed by an order amendment). In either case, the actual fulfillment state of this order depends on the type of operation (CANCEL or DELIVER). If it is CANCEL, this script changes the fulfillment state of the orphaned order item to CANCELLED. If the operation is DELIVER, the script changes the fulfillment state of the orphaned order item to COMPLETED.

Table 5–54 lists the input parameters for the extension point XQuery.

Table 5–54 ORDERITEM_FULFILLMENT_STATE_UPDATED Input Parameters

Name	Type	Scope	Description
$fulfillmentMode	xs:string	External variable	Fulfillment mode of the sales order (DELIVER, CANCEL, or TSQ)
$fulfillmentState	xs:string	External variable	Order item's current composite fulfillment state
$orderEventType	xs:string	External variable	The event type when this extension is triggered. ORDER_EVENT_UPDATE and ORDER_EVENT_COMPLETE. ORDER_ EVENT_UPDATE is set if this is triggered within the fulfillment function's Sub-process's automation task. ORDER_ EVENT_COMPLETE is set if this is triggered within OSM order complete event handler.
$orderOperationType	xs:string	External variable	ORDER_OPERATION_CANCEL is set if the Oracle AIA order is doing a cancel operation no matter the cancel is triggered from upstream or from an OSM web client, otherwise ORDER_OPERATION_ NORMAL is set.
$hasChildLines	xs:boolean	External variable	True to indicate this order item has children, otherwise false
$hasComponents	xs:boolean	External variable	True to indicate this order item is contained in fulfillment function, otherwise false
.	element()	Context node	The OrderLifeCycleManagement/OrderItemStatus XML fragment for the order item

Table 5–55 lists the return parameters for the extension point XQuery.

Table 5–55 ORDERITEM_FULFILLMENT_STATE_UPDATED Return Parameters

Output Parameter Type	Description
xs:string	Calculated fulfillment state to be set for the orphaned order item

Example 5–22 is a code fragment from **OracleComms_OSM_O2A_ Configuration/fulfillment-state-extension/OnOrderItemFulfillmentStateUpdated.xquery** that demonstrates the extension implementation.

Example 5–22 ORDERITEM_FULFILLMENT_STATE_UPDATED XQuery Code Fragment

```
import module namespace o2acomfulfillmentstate =
"http://xmlns.oracle.com/communications/ordermanagement/o2acom/fulfillmentstate/co
nstant" at
```

```
"http://xmlns.oracle.com/communications/ordermanagement/o2acom/fulfillmentstate/co
nstant/FulfillmentStateConstantModule.xquery";
import module namespace osmpiplog =
"http://xmlns.oracle.com/communications/ordermanagement/pip/omspiplog" at
"http://xmlns.oracle.com/communications/ordermanagement/pip/omspiplog/LogModule.xq
uery";
declare namespace oms="urn:com:metasolv:oms:xmlapi:1";
declare namespace fulfillmeneStateConstant =
"java:oracle.communications.ordermanagement.fulfillmentstatelifecycle.FulfillmeneS
tateConstant";

declare variable $fulfillmentMode as xs:string external;
declare variable $fulfillmentState as xs:string external;
declare variable $orderEventType as xs:string external;
declare variable $orderOperationType as xs:string external;
declare variable $hasChildLines as xs:string external;
declare variable $hasComponents as xs:string external;

declare variable $MODULE_NAME := "OnOrderItemFulfillmentStateUpdate";

declare variable $ORDER_EVENT_CANCELLED := fulfillmeneStateConstant:ORDER_EVENT_
CANCELLED();
declare variable $ORDER_EVENT_UPDATE := fulfillmeneStateConstant:ORDER_EVENT_
UPDATE();
declare variable $ORDER_EVENT_COMPLETE := fulfillmeneStateConstant:ORDER_EVENT_
COMPLETE();

declare variable $ORDER_OPERATION_NORMAL := fulfillmeneStateConstant:ORDER_
OPERATION_NORMAL();
declare variable $ORDER_OPERATION_CANCEL := fulfillmeneStateConstant:ORDER_
OPERATION_CANCEL();

declare variable $HAS_CHILDS := fulfillmeneStateConstant:HAS_CHILDS();
declare variable $HAS_COMPONENT := fulfillmeneStateConstant:HAS_COMPONENTS();

let $calculatedFulfillmentState :=
    if ($hasChildLines != $HAS_CHILDS and $hasComponents != $HAS_COMPONENT)
    then
    (
        (:
         : No current status can be detects from components or children lines,
meaning this line don't invoke any component
         : and also no children line exists, set the status to cancelled if this
is cancel order operation or set the status to complete if
         : this is not cancel operation
         :)
        if ($orderEventType = ($ORDER_EVENT_UPDATE, $ORDER_EVENT_COMPLETE))
        then
        (
            if ($orderOperationType = $ORDER_OPERATION_CANCEL)
            then $o2acomfulfillmentstate:CANCELLED_STATE
            else $o2acomfulfillmentstate:COMPLETE_STATE
        )
        else $o2acomfulfillmentstate:CANCELLED_STATE
    )
    else $fulfillmentState
return
    $calculatedFulfillmentState
```

ORDER_FULFILLMENT_STATE_UPDATED Extension Point

This section describes the XQuery script that implements the logic to handle the ORDER_FULFILLMENT_STATE_UPDATED extension point. This extension point overrides the default calculation result that is based on the COM_OrderStateCompositionRule defined in cartridge OracleComms_OSM_O2A_FulfillmentPatternMap_Sample. The COM_OrderStateCompositionRule only defines the basic aggregation rule that is based on children order item's fulfillment state and does not consider the current order's operation and event.

For example, if an order has no child order items, it maybe because the order itself does not have any base order items, or because the existing order items were dropped during revision. The server Fulfillment state engine may not update the order level fulfillment state or will calculate the order level fulfillment state as PENDING (If no line has started) or IN_PROGRESS (If line has been started but now get dropped). In either case, the actual fulfillment state of this order should depends on the type of operation (CANCEL or DELIVER) and if is CANCEL then this script will override it to CANCELLED or if it is DELIVER then the script will override it to COMPLETED.

Given another example where the fulfillment state engine calculates a CANCEL or COMPLETE state but the order is still in progress state (detected with $orderEventType=ORDER_EVENT_UPDATE), the override value in this case is IN_PROGRESS since the order is still in the middle of processing.

ORDER_EVENT_COMPLETE can be due to DELIVER COMPLETE or DELIVER COMPLETE of a CANCEL order. In this case, operation type is used to detect if it is a normal COMPLETE or CANCEL COMPLETE.

Table 5–56 lists the input parameters for the extension point XQuery.

Table 5–56 ORDER_FULFILLMENT_STATE_UPDATED Input Parameters

Name	Type	Scope	Description
$fulfillmentMode	xs:string	External variable	Fulfillment mode of the sales order (DELIVER, CANCEL, or TSQ)
$fulfillmentState	xs:string	External variable	Order item's current composite fulfillment state
$orderEventType	xs:string	External variable	The event type when this extension is triggered. ORDER_EVENT_UPDATE and ORDER_EVENT_COMPLETE. ORDER_EVENT_UPDATE is set if this is triggered within the fulfillment function's Sub-process's automation task. ORDER_EVENT_COMPLETE is set if this is triggered within OSM order complete event handler.
$orderOperationType	xs:string	External variable	ORDER_OPERATION_CANCEL is set if the Oracle AIA order is doing a cancel operation no matter the cancel is triggered from upstream or from an OSM web client, otherwise ORDER_OPERATION_NORMAL is set.
$hasChildLines	xs:boolean	External variable	True to indicate this order item has children, otherwise false
.	element()	Context node	The OrderLifeCycleManagement XML fragment

Table 5–57 lists the return parameters for the extension point XQuery.

Table 5–57   ORDER_FULFILLMENT_STATE_UPDATED Return Parameters

Output Parameter Type	Description
xs:string	Calculated fulfillment state for the order

Example 5–23 is a code fragment from **OracleComms_OSM_O2A_Configuration/fulfillment-state-extension/OnOrderFulfillmentStateUpdated.xquery** that demonstrates the extension implementation.

Example 5–23   ORDER_FULFILLMENT_STATE_UPDATED XQuery Code Fragment

```
import module namespace o2acomfulfillmentstate =
"http://xmlns.oracle.com/communications/ordermanagement/o2acom/fulfillmentstate/co
nstant" at
"http://xmlns.oracle.com/communications/ordermanagement/o2acom/fulfillmentstate/co
nstant/FulfillmentStateConstantModule.xquery";
declare namespace fulfillmeneStateConstant =
"java:oracle.communications.ordermanagement.fulfillmentstatelifecycle.FulfillmeneS
tateConstant";

declare variable $fulfillmentMode as xs:string external;
declare variable $fulfillmentState as xs:string external;
declare variable $fulfillmentState as xs:string external;
declare variable $orderOperationType as xs:string external;
declare variable $hasChildLines as xs:string external;

declare variable $ORDER_EVENT_CANCELLED := fulfillmeneStateConstant:ORDER_EVENT_
CANCELLED();
declare variable $ORDER_EVENT_UPDATE := fulfillmeneStateConstant:ORDER_EVENT_
UPDATE();
declare variable $ORDER_EVENT_COMPLETE := fulfillmeneStateConstant:ORDER_EVENT_
COMPLETE();
declare variable $ORDER_OPERATION_NORMAL := fulfillmeneStateConstant:ORDER_
OPERATION_NORMAL();
declare variable $ORDER_OPERATION_CANCEL := fulfillmeneStateConstant:ORDER_
OPERATION_CANCEL();
declare variable $HAS_CHILDS := fulfillmeneStateConstant:HAS_CHILDS();

let $calculatedFulfillmentState :=
    if ($hasChildLines != $HAS_CHILDS)
    then
    (
        (: No child lines detected :)
        if ($orderEventType = $ORDER_EVENT_UPDATE)
        then $o2acomfulfillmentstate:IN_PROGRESS_STATE
        else if ($orderEventType = $ORDER_EVENT_COMPLETE)
        then
        (
            (: This is an order complete event,
             : if this is a normal order completion, the status is complete.
             : if this is a cancel order completion, the status is canceled.
             :)
            if ($orderOperationType = $ORDER_OPERATION_NORMAL)
            then $o2acomfulfillmentstate:COMPLETE_STATE
            else $o2acomfulfillmentstate:CANCELLED_STATE
        )
        else $o2acomfulfillmentstate:CANCELLED_STATE
    )
    else if($fulfillmentState = ($o2acomfulfillmentstate:COMPLETE_STATE,
$o2acomfulfillmentstate:CANCELLED_STATE))
    then
    (
        (: Children line exist and all completed or cancelled
         : However if the event is not an order complete or cancelled even than
need
         : to switch the status back to in progress
         :)
        if ($orderEventType != $ORDER_EVENT_COMPLETE and $orderEventType !=
$ORDER_EVENT_CANCELLED)
        then $o2acomfulfillmentstate:IN_PROGRESS_STATE
        else $fulfillmentState
    )
    else
    (
        if ($orderEventType = $ORDER_EVENT_COMPLETE)
        then
        (
            if ($orderOperationType = $ORDER_OPERATION_CANCEL)
            then
            (
                (: An OSM order get cancelled by AIA or Admin then always set the
order status to cancelled :)
                $o2acomfulfillmentstate:CANCELLED_STATE
            )
            else (
                (: Only set status to complete if it is not failed.
                 : So OPEN and IN_PROGRESS will forced to COMPLETE but FAILED will
be remained.
                 :)
                if ($fulfillmentState != $o2acomfulfillmentstate:FAILED_STATE)
                then $o2acomfulfillmentstate:COMPLETE_STATE
                else $fulfillmentState
            )
        )
        else $fulfillmentState
    )
return
    $calculatedFulfillmentState
```

ORDER_STATUS Extension Point

This section describes the XQuery script that implements the logic to handle the ORDER_STATUS extension point. This extension point generates the upstream expected status value for the order or order item. The generated value is based on the current composite fulfillment state value of the order or order item.

The mapping between the composite fulfillment state and the upstream status is defined in **OracleComms_OSM_O2A_Configuration/solution-config/OrderStatusMap.xml**.

Table 5–58 lists the input parameters for the extension point XQuery.

Table 5–58 ORDER_STATUS Input Parameters

Name	Type	Scope	Description
$fulfillmentMode	xs:string	External variable	Fulfillment mode of the sales order (DELIVER, CANCEL, or TSQ)
$fulfillmentState	xs:string	External variable	Current composite fulfillment state of the order or order item
$componentType	xs:string	External variable	Should be set to **OrderLifecycleManagement**
$systemType	xs:string	External variable	Should be set to **CRM**
.	element()	Context node	**OrderLifeCycleManagement/OrderItemStatus** XML fragment if this is invoked for an order item or **OrderLifeCycleManagement** XML fragment if this is invoked for an order

Table 5–59 lists the return parameters for the extension point XQuery.

Table 5–59 ORDER_STATUS Return Parameters

Output Parameter Type	Description
xs:string	Calculated status value

Example 5–24 is a code fragment from **OracleComms_OSM_O2A_Configuration/fulfillment-state-extension/OrderStatus.xquery** that demonstrates the extension implementation.

Example 5–24 ORDER_STATUS XQuery Code Fragment

```
import module namespace statusctxmapmodule =
"http://xmlns.oracle.com/communications/ordermanagement/o2acom/fulfillmentstate/st
atusctxmapmodule" at
"http://xmlns.oracle.com/communications/ordermanagement/o2acom/fulfillmentstate/st
atusctxmapmodule/OrderAndOrderItemStatueContextModule.xquery";

declare namespace oms="urn:com:metasolv:oms:xmlapi:1";

declare namespace fulfillmeneStateConstant =
"java:oracle.communications.ordermanagement.fulfillmentstatelifecycle.FulfillmeneS
tateConstant";

declare variable $fulfillmentMode as xs:string external;
declare variable $fulfillmentState as xs:string external;
declare variable $componentType as xs:string external;
declare variable $systemType as xs:string external;
statusctxmapmodule:getOrderStatus($fulfillmentMode, $fulfillmentState,
$componentType, $systemType)
```

ORDER_STATUSCONTEXT Extension Point

This section describes the XQuery script that implements the logic to handle the ORDER_STATUSCONTEXT extension point. This extension point generates the upstream expected description value to the status for Order. The generated value is based on the current composite fulfillment state value of the Order.

The status context is defined in OracleComms_OSM_O2A_Configuration/solution-config/OrderStatusContextMap.xml.

Table 5–60 lists the input parameters for the extension point XQuery.

Table 5–60 ORDER_STATUSCONTEXT Input Parameters

Name	Type	Scope	Description
$fulfillmentMode	xs:string	External variable	Fulfillment mode of the sales order (DELIVER, CANCEL, or TSQ)
$fulfillmentState	xs:string	External variable	The order item's current composite fulfillment state
$orderEventType	xs:string	External variable	The event type when this extension is triggered. ORDER_EVENT_UPDATE and ORDER_EVENT_COMPLETE. ORDER_EVENT_UPDATE is set if this is triggered within the fulfillment function's Sub-process's automation task. ORDER_EVENT_COMPLETE is set if this is triggered within OSM order complete event handler.
$orderOperationType	xs:string	External variable	ORDER_OPERATION_CANCEL is set if the Oracle AIA order is doing a cancel operation no matter the cancel is triggered from upstream or from an OSM web client, otherwise ORDER_OPERATION_NORMAL is set.
$doubleFailure	xs:boolean	External variable	True to indicate this Order is in Fallout during a revision
.	element()	Context node	The OrderLifeCycleManagement XML fragment

Table 5–61 lists the return parameters for the extension point XQuery.

Table 5–61 ORDER_STATUSCONTEXT Return Parameters

Output Parameter Type	Description
xs:string	Calculated description of the current order status

Example 5–25 is a code fragment from **OracleComms_OSM_O2A_Configuration/fulfillment-state-extension/OrderStatusContext.xquery** that demonstrates the extension implementation.

Example 5–25 ORDER_STATUSCONTEXT XQuery Code Fragment

```
import module namespace statusctxmapmodule =
"http://xmlns.oracle.com/communications/ordermanagement/o2acom/fulfillmentstate/st
atusctxmapmodule" at
"http://xmlns.oracle.com/communications/ordermanagement/o2acom/fulfillmentstate/st
atusctxmapmodule/OrderAndOrderItemStatueContextModule.xquery";
import module namespace osmpiplog =
"http://xmlns.oracle.com/communications/ordermanagement/pip/omspiplog" at
"http://xmlns.oracle.com/communications/ordermanagement/pip/omspiplog/LogModule.xq
uery";
import module namespace o2acomfulfillmentstate =
"http://xmlns.oracle.com/communications/ordermanagement/o2acom/fulfillmentstate/co
nstant" at
"http://xmlns.oracle.com/communications/ordermanagement/o2acom/fulfillmentstate/co
nstant/FulfillmentStateConstantModule.xquery";
```

```
declare namespace oms="urn:com:metasolv:oms:xmlapi:1";

declare namespace fulfillmeneStateConstant =
"java:oracle.communications.ordermanagement.fulfillmentstatelifecycle.FulfillmeneS
tateConstant";

declare variable $fulfillmentMode as xs:string external;
declare variable $fulfillmentState as xs:string external;
declare variable $orderEventType as xs:string external;
declare variable $orderOperationType as xs:string external;
declare variable $doubleFailure as xs:string external;

declare variable $IS_DOUBLE_FAILURE := fulfillmeneStateConstant:DOUBLE_FAILURE_
TRUE();
declare variable $ORDER_OPERATION_CANCEL := fulfillmeneStateConstant:ORDER_
OPERATION_CANCEL();
declare variable $ORDER_EVENT_CANCELLED := fulfillmeneStateConstant:ORDER_EVENT_
CANCELLED();

declare variable $TRUE := "TRUE";
declare variable $FALSE := "FALSE";
declare variable $MODULE_NAME := "OrderStatusContext";

declare function local:hasOrderItemCancelled(
   $orderItemsFulfilmentState as element()) as xs:boolean
{

fn:exists($orderItemsFulfilmentState/oms:fulfillmentState[text()=$o2acomfulfillmen
tstate:CANCELLED_STATE])
};

declare function local:isAllOrderItemFailed(
   $orderItemsFulfilmentState as element()) as xs:boolean
{
   let $itemCount := fn:count($orderItemsFulfilmentState/oms:fulfillmentState)
   let $failCount :=
fn:count($orderItemsFulfilmentState/oms:fulfillmentState[text()=$o2acomfulfillment
state:FAILED_STATE])
   return
      if ($itemCount = $failCount)
      then fn:true()
      else fn:false()
};

let $orderItemsFulfilmentState := .
return
(
   let $statusStateInfo :=
      <oms:StatusStateInfo>
      {
         if ($fulfillmentState = $o2acomfulfillmentstate:COMPLETE_STATE)
         then
         (
            <oms:statusState>
            {
               if
(local:hasOrderItemCancelled($orderItemsFulfilmentState)=fn:true())
               then fulfillmeneStateConstant:COMPLETE_WITH_CANCELLED()
               else fulfillmeneStateConstant:COMPLETE_ALL_COMPLETE()
            }
            </oms:statusState>
         )
         else if ($fulfillmentState = $o2acomfulfillmentstate:FAILED_STATE)
         then
         (
            <oms:statusState>
            {
               if ($orderOperationType = $ORDER_OPERATION_CANCEL or $doubleFailure
= $IS_DOUBLE_FAILURE)
               then fulfillmeneStateConstant:FAILED_REVISION_FAILED()
               else
               (
                  if
(local:isAllOrderItemFailed($orderItemsFulfilmentState)=fn:true())
                  then fulfillmeneStateConstant:FAILED_ALL_FAILED()
                  else fulfillmeneStateConstant:FAILED_PARTIAL_FAILED()
               )
            }
            </oms:statusState>
         )
         else if ($fulfillmentState = $o2acomfulfillmentstate:CANCELLED_STATE)
         then
         (
            <oms:statusState>
            {
               if ($orderEventType = $ORDER_EVENT_CANCELLED)
               then fulfillmeneStateConstant:CANCELLED_BY_ADMIN()
               else fulfillmeneStateConstant:CANCELLED_BY_UPSTREAM()
            }
            </oms:statusState>,
            <oms:hasOrderItems>
            {
               if (fn:exists($orderItemsFulfilmentState/oms:fulfillmentState))
               then $TRUE
               else $FALSE
            }
            </oms:hasOrderItems>
         )
         else ()
      }
      </oms:StatusStateInfo>
   return
      statusctxmapmodule:getOrderStatusContext($fulfillmentMode,
$fulfillmentState, $statusStateInfo)
)
```

ORDERITEM_MILESTONE Extension Point

This section describes the XQuery script that implements the logic to handle the ORDERITEM_MILESTONE extension point. This extension point generates the upstream expected milestone value to the order item. The implementation for this script is to calculate the milestone value (expected by the upstream CRM system) based on the calculated fulfillment state value and the last reported milestone. For a fulfillment state value equal to CANCELLED, the milestone code is the last milestone code before the cancel was applied to this order item. For other fulfillment states, the milestone code is the current latest milestone code injected by Order-to-Activate or reported from external system.

Also if none of the components invoked by this order item has started then if fulfillment state value equals to CANCELLED, the milestone code is

$o2acomfulfillmentstate:NOTSTARTED_MILESTONE, and for all other fulfillment state value, the milestone code is $o2acomfulfillmentstate:NO_MILESTONE.

Table 5–62 lists the input parameters for the extension point XQuery.

Table 5–62 ORDERITEM_MILESTONE Input Parameters

Name	Type	Scope	Description
$fulfillmentMode	xs:string	External variable	Fulfillment mode of the sales order (DELIVER, CANCEL, or TSQ)
$fulfillmentState	xs:string	External variable	The order item's current composite fulfillment state.
$orderEventType	xs:string	External variable	The event type when this extension is triggered. ORDER_EVENT_UPDATE and ORDER_EVENT_COMPLETE. ORDER_EVENT_UPDATE is set if this is triggered within the fulfillment function's Sub-process's automation task. ORDER_EVENT_COMPLETE is set if this is triggered within OSM order complete event handler.
$orderOperationType	xs:string	External variable	ORDER_OPERATION_CANCEL is set if the Oracle AIA order is doing a cancel operation no matter the cancel is triggered from upstream or from an OSM web client, otherwise ORDER_OPERATION_NORMAL is set.
$milestoneCode	xs:string	External variable	The prior milestone code of the order item
$orderItemComponentfulfillmentState	xs:string	External variable	The most recent fulfillment state updated from the order component
.	element()	Context node	The OrderLifeCycleManagement XML fragment

Table 5–63 lists the return parameters for the extension point XQuery.

Table 5–63 ORDERITEM_MILESTONE Return Parameters

Output Parameter Type	Description
xs:string	Calculated milestone value for the current order item

Example 5–26 is a code fragment from **OracleComms_OSM_O2A_Configuration/fulfillment-state-extension/OrderItemMilestone.xquery** that demonstrates the extension implementation.

Example 5–26 ORDERITEM_MILESTONE XQuery Code Fragment

```
import module namespace o2acomfulfillmentstate =
"http://xmlns.oracle.com/communications/ordermanagement/o2acom/fulfillmentstate/co
nstant" at
"http://xmlns.oracle.com/communications/ordermanagement/o2acom/fulfillmentstate/co
nstant/FulfillmentStateConstantModule.xquery";
import module namespace osmpiplog =
"http://xmlns.oracle.com/communications/ordermanagement/pip/omspiplog" at
"http://xmlns.oracle.com/communications/ordermanagement/pip/omspiplog/LogModule.xq
uery";

declare namespace oms="urn:com:metasolv:oms:xmlapi:1";
declare variable $fulfillmentMode as xs:string external;
declare variable $fulfillmentState as xs:string external;
declare variable $orderEventType as xs:string external;
declare variable $orderOperationType as xs:string external;

declare variable $MODULE_NAME := "OrderItemMilestone";

(:
 : Current fulfillment state is not cancelled then calculate the latest milestone
of this line for this time being.
 :)
declare function local:getLatestMilestoneFromComponents(
    $orderItemStatus as element()?) as xs:string
{
    if (fn:exists($orderItemStatus/oms:OrderItemComponentStatus))
    then
    (
        let $lookupIndex :=
fn:max($orderItemStatus/oms:OrderItemComponentStatus/oms:MilestoneStatusRecord/oms
:Status/oms:Code/@index)
        let $latestMilestoneStatusRecord :=
$orderItemStatus/oms:OrderItemComponentStatus/oms:MilestoneStatusRecord[oms:Status
/oms:Code/@index=$lookupIndex]
        return
            $latestMilestoneStatusRecord/oms:MilestoneCode/text()
    )
    else $o2acomfulfillmentstate:NO_MILESTONE
};

declare function local:getLastMilestoneCodeBeforeCancel(
    $orderItemStatus as element()?) as xs:string
{
    if (fn:exists($orderItemStatus/oms:OrderItemComponentStatus))
    then
    (
        let $lookupIndex :=
fn:min($orderItemStatus/oms:OrderItemComponentStatus/oms:MilestoneStatusRecord/oms
:Status/oms:Code[text()=$o2acomfulfillmentstate:CANCELLED_STATE]/@index)
        let $cancelledMilestoneStatusRecord :=
$orderItemStatus/oms:OrderItemComponentStatus/oms:MilestoneStatusRecord[oms:Status
/oms:Code/@index=$lookupIndex]
        let $milestoneStatusRecordBeforeCancelled :=
$cancelledMilestoneStatusRecord/preceding-sibling::oms:MilestoneStatusRecord[1]
        return
            if (fn:exists($milestoneStatusRecordBeforeCancelled))
            then $milestoneStatusRecordBeforeCancelled/oms:MilestoneCode/text()
            else $o2acomfulfillmentstate:NOTSTARTED_MILESTONE
    )
    else $o2acomfulfillmentstate:NOTSTARTED_MILESTONE
};

let $orderItemStatus := .
return
    if ($fulfillmentState = $o2acomfulfillmentstate:CANCELLED_STATE)
    then local:getLastMilestoneCodeBeforeCancel($orderItemStatus)
    else local:getLatestMilestoneFromComponents($orderItemStatus)
```

ORDERITEM_STATUSCONTEXT Extension Point

This section describes the XQuery script that implements the logic to handle the ORDERITEM_STATUSCONTEXT extension point. This extension point generates the upstream expected description value to the status for order item. The generated value is based on the current composite fulfillment state value of the order item. The implementation for this script is to calculate the status context (Description of status) value (expected by the upstream CRM system) based on the calculated fulfillment state value and the current calculated milestone (The milestone code calculated by the XQuery registered to extension ORDERITEM_MILESTONE).

For fulfillment state value equals to FAILED, the status context is the error message map defined in OracleComms_OSM_O2A_
Configuration/solution-config/OrderMessageMap.xml.

For all other fulfillment state, the status context is the milestone code append with the string define in OracleComms_OSM_O2A_
Configuration/solution-config/OrderItemStatusContextMap.xml

Table 5–64 lists the input parameters for the extension point XQuery.

Table 5–64 ORDERITEM_STATUSCONTEXT Input Parameters

Name	Type	Scope	Description
$fulfillmentMode	xs:string	External variable	Fulfillment mode of the sales order (DELIVER, CANCEL, or TSQ)
$fulfillmentState	xs:string	External variable	The order item's current composite fulfillment state
$orderEventType	xs:string	External variable	The event type when this extension is triggered. ORDER_EVENT_UPDATE and ORDER_EVENT_COMPLETE. ORDER_ EVENT_UPDATE is set if this is triggered within the fulfillment function's Sub-process's automation task. ORDER_ EVENT_COMPLETE is set if this is triggered within OSM order complete event handler.
$orderOperationType	xs:string	External variable	ORDER_OPERATION_CANCEL is set if the Oracle AIA order is doing a cancel operation no matter the cancel is triggered from upstream or from an OSM web client, otherwise ORDER_OPERATION_ NORMAL is set.
$milestoneCode	xs:string	External variable	The current milestone code of the order item
.	element()	Context node	The OrderLifeCycleManagement XML fragment

Table 5–65 lists the return parameters for the extension point XQuery.

Table 5–65 ORDERITEM_STATUSCONTEXT Return Parameters

Output Parameter Type	Description
xs:string	Calculated description of the current order item status

Example 5–27 is a code fragment from **OracleComms_OSM_O2A_ Configuration/fulfillment-state-extension/OrderItemStatusContextForDeliver.xquery** that demonstrates the extension implementation.

Example 5–27 ORDERITEM_STATUSCONTEXT XQuery Code Fragment

```
import module namespace statusctxmapmodule =
"http://xmlns.oracle.com/communications/ordermanagement/o2acom/fulfillmentstate/st
atusctxmapmodule" at
"http://xmlns.oracle.com/communications/ordermanagement/o2acom/fulfillmentstate/st
atusctxmapmodule/OrderAndOrderItemStatueContextModule.xquery";
import module namespace o2acomfulfillmentstate =
"http://xmlns.oracle.com/communications/ordermanagement/o2acom/fulfillmentstate/co
nstant" at
"http://xmlns.oracle.com/communications/ordermanagement/o2acom/fulfillmentstate/co
nstant/FulfillmentStateConstantModule.xquery";
import module namespace osmpiplog =
"http://xmlns.oracle.com/communications/ordermanagement/pip/omspiplog" at
"http://xmlns.oracle.com/communications/ordermanagement/pip/omspiplog/LogModule.xq
uery";

declare namespace oms="urn:com:metasolv:oms:xmlapi:1";

declare variable $fulfillmentMode as xs:string external;
declare variable $fulfillmentState as xs:string external;
declare variable $orderEventType as xs:string external;
declare variable $orderOperationType as xs:string external;
declare variable $milestoneCode as xs:string external;

declare variable $MODULE_NAME := "OrderItemStatusContextForDeliver";

(:
 : For DELIVER, CANCEL fulfilllmentMode order, only return the last reported
lifecycle if that is failure.
 :)
declare function local:getLatestFailedMilestoneStatusRecordsFromComponent(
   $orderItemComponentStatus as element()) as element()?
{
   let $lookupIndex :=
fn:max($orderItemComponentStatus/oms:MilestoneStatusRecord/oms:ExternalFulfillment
StateCode/@index)
   let $latestMilestoneStatusRecord :=
$orderItemComponentStatus/oms:MilestoneStatusRecord[oms:ExternalFulfillmentStateCo
de/@index=$lookupIndex]
   return
      if
($latestMilestoneStatusRecord/oms:ExternalFulfillmentStateCode/text()=$o2acomfulfi
llmentstate:FAILED_STATE)
      then
      (
         <oms:OrderItemComponentStatus>
            <oms:componentKey>{ $orderItemComponentStatus/oms:componentKey/text()
}</oms:componentKey>
            <oms:componentType>{
$orderItemComponentStatus/oms:componentType/text() }</oms:componentType>
            <oms:systemType>{ $orderItemComponentStatus/oms:systemType/text()
}</oms:systemType>
            {
               $latestMilestoneStatusRecord
            }
        </oms:OrderItemComponentStatus>
     )
     else ()
};
```

```
(:
 : Concatenate all translated error message from all components that is currently
failed.
 :)
declare function local:getStatusContextForFailedFulfillmentState(
   $orderItemStatus as element()) as xs:string
{
   if (fn:exists($orderItemStatus/oms:OrderItemComponentStatus))
   then
   (
      let $allOrderItemComponentStatus :=
$orderItemStatus/oms:OrderItemComponentStatus
      let $allFailedComponent :=
         <oms:AllFailedComponents>
         {
            for $orderItemComponentStatus in $allOrderItemComponentStatus
            return
local:getLatestFailedMilestoneStatusRecordsFromComponent($orderItemComponentStatus
)
         }
         </oms:AllFailedComponents>
      let $allFailedStatusContext :=
         <oms:AllFailedStatusContext>
         {
            for $failedComponent in
$allFailedComponent/oms:OrderItemComponentStatus
            let $failedRecord := $failedComponent/oms:MilestoneStatusRecord
            return
               if ($failedRecord/oms:Status/oms:Description/text()!="")
               then
               (
                   let $errorMsg :=
statusctxmapmodule:translateErrorMessage($failedComponent/oms:componentType/text()
, $failedComponent/oms:systemType/text(),
$failedRecord/oms:Status/oms:Description/text())
                   return
                      <oms:context>{
fn:concat($failedRecord/oms:MilestoneCode/text(),": ", $errorMsg)}</oms:context>
                    )
               else ()
        }
        </oms:AllFailedStatusContext>
     return
        if (fn:exists($allFailedStatusContext/oms:context))
        then fn:string-join($allFailedStatusContext/oms:context/text(), ", ")
        else ""
  )
  else ""
};

let $orderItemStatus := .
return
   if ($fulfillmentState = $o2acomfulfillmentstate:FAILED_STATE)
   then local:getStatusContextForFailedFulfillmentState($orderItemStatus)
   else statusctxmapmodule:getOrderItemStatusContext($fulfillmentMode,
$fulfillmentState, $milestoneCode)
```

REPORT_ORDERITEM_STATUS Extension Point

This section describes the XQuery script that implements the logic to handle the REPORT_ORDERITEM_STATUS extension point. This extension point enhances the upstream expected status value of the given order item. The enhanced value will then be updated to order item status field.

Table 5–66 lists the input parameters for the extension point XQuery.

Table 5–66 REPORT_ORDERITEM_STATUS Input Parameters

Name	Type	Scope	Description
$fulfillmentMode	xs:string	External variable	Fulfillment mode of the sales order (DELIVER, CANCEL, or TSQ)
$fulfillmentState	xs:string	External variable	The order item's current composite fulfillment state
$orderEventType	xs:string	External variable	The event type when this extension is triggered. ORDER_EVENT_UPDATE and ORDER_EVENT_COMPLETE. ORDER_EVENT_UPDATE is set if this is triggered within the fulfillment function's Sub-process's automation task. ORDER_EVENT_COMPLETE is set if this is triggered within OSM order complete event handler.
$orderOperationType	xs:string	External variable	ORDER_OPERATION_CANCEL is set if the Oracle AIA order is doing a cancel operation no matter the cancel is triggered from upstream or from an OSM web client, otherwise ORDER_OPERATION_NORMAL is set.
$orderItem	Java Object	External variable	oracle.communications.ordermanagement.fulfillmentstatelifecycle.OrderItemFulfillmentStateLifecycle
.	element()	Context node	The OrderLifeCycleManagement XML fragment

Table 5–67 lists the return parameters for the extension point XQuery.

Table 5–67 REPORT_ORDERITEM_STATUS Return Parameters

Output Parameter Type	Description
xs:string	The decorated status value of this order item.

Note: This extension point is not used in current Order-to-Activate implementation.

REPORT_ORDERITEM_MILESTONE Extension Point

This section describes the XQuery script that implements the logic to handle the REPORT_ORDERITEM_MILESTONE extension point. This extension point enhances the upstream expected milestone value of the given order item. The enhanced value will then be updated to order item milestone field.

The implementation for this script is to override the milestone value which is generated by the XQuery registered to extension ODERITEM_MILESTONE. Due to

Oracle AIA requirements, if the order item's fulfillment state is cancelled then the milestone value set to upstream must be empty.

Table 5–68 lists the input parameters for the extension point XQuery.

Table 5–68 REPORT_ORDERITEM_MILESTONE Input Parameters

Name	Type	Scope	Description
$fulfillmentMode	xs:string	External variable	Fulfillment mode of the sales order (DELIVER, CANCEL, or TSQ)
$fulfillmentState	xs:string	External variable	The order item's current composite fulfillment state
$orderEventType	xs:string	External variable	The event type when this extension is triggered. ORDER_EVENT_UPDATE and ORDER_EVENT_COMPLETE. ORDER_EVENT_UPDATE is set if this is triggered within the fulfillment function's Sub-process's automation task. ORDER_EVENT_COMPLETE is set if this is triggered within OSM order complete event handler.
$orderOperationType	xs:string	External variable	ORDER_OPERATION_CANCEL is set if the Oracle AIA order is doing a cancel operation no matter the cancel is triggered from upstream or from an OSM web client, otherwise ORDER_OPERATION_NORMAL is set.
$orderItem	Java Object	External variable	oracle.communications.ordermanagement.fulfillmentstatelifecycle.OrderItemFulfillmentStateLifecycle
.	element()	Context node	The OrderLifeCycleManagement XML fragment

Table 5–69 lists the return parameters for the extension point XQuery.

Table 5–69 REPORT_ORDERITEM_MILESTONE Return Parameters

Output Parameter Type	Description
xs:string	The decorated milestone value of this order item.

Example 5–28 is a code fragment from **OracleComms_OSM_O2A_Configuration/fulfillment-state-extension/OnReportMilestoneDeliver.xquery** that demonstrates the extension implementation.

Example 5–28 REPORT_ORDERITEM_MILESTONE XQuery Code Fragment

```
import module namespace o2acomfulfillmentstate =
"http://xmlns.oracle.com/communications/ordermanagement/o2acom/fulfillmentstate/co
nstant" at
"http://xmlns.oracle.com/communications/ordermanagement/o2acom/fulfillmentstate/co
nstant/FulfillmentStateConstantModule.xquery";

declare namespace orderitem =
"java:oracle.communications.ordermanagement.fulfillmentstatelifecycle.OrderItemFul
fillmentStateLifecycle";

declare variable $fulfillmentMode as xs:string external;
declare variable $fulfillmentState as xs:string external;
declare variable $orderEventType as xs:string external;
declare variable $orderOperationType as xs:string external;
declare variable $orderItem external;

let $fulfillmentState := orderitem:getFulfillmentState($orderItem)
return
    if ($fulfillmentState = $o2acomfulfillmentstate:CANCELLED_STATE)
    then ""
    else orderitem:getLatestMilestoneCode($orderItem)
```

REPORT_ORDERITEM_STATUSCONTEXT Extension Point

This section describes the XQuery script that implements the logic to handle the REPORT_ORDERITEM_STATUSCONTEXT extension point. This extension point enhances the upstream expected status context (Description) value of the given order item. The decorated value will then be updated to order item status context field.

The implementation for this script is to decorate the status context value which is generated by the XQuery registered to extension ORDERITEM_STATUSCONTEXT.

For DELIVER and CANCEL, if an order item's fulfillment state is FAILED and if the failure is not by the order item itself (Not fail due to its invoking component) then populate a message to indicate the failure is caused by its children, otherwise concatenate the milestone value and the status context value.

Table 5–70 lists the input parameters for the extension point XQuery.

Table 5–70 REPORT_ORDERITEM_STATUSCONTEXT Input Parameters

Name	Type	Scope	Description
$fulfillmentMode	xs:string	External variable	Fulfillment mode of the sales order (DELIVER, CANCEL, or TSQ)
$fulfillmentState	xs:string	External variable	The order item's current composite fulfillment state
$orderEventType	xs:string	External variable	The event type when this extension is triggered. ORDER_EVENT_UPDATE and ORDER_EVENT_COMPLETE. ORDER_EVENT_UPDATE is set if this is triggered within the fulfillment function's Sub-process's automation task. ORDER_EVENT_COMPLETE is set if this is triggered within OSM order complete event handler.
$orderOperationType	xs:string	External variable	ORDER_OPERATION_CANCEL is set if the Oracle AIA order is doing a cancel operation no matter the cancel is triggered from upstream or from an OSM web client, otherwise ORDER_OPERATION_NORMAL is set.
$orderItem	Java Object	External variable	oracle.communications.ordermanagement.fulfillmentstatelifecycle.OrderItemFulfillmentStateLifecycle
.	element()	Context node	The OrderLifeCycleManagement XML fragment

Table 5–71 lists the return parameters for the extension point XQuery.

Table 5–71 REPORT_ORDERITEM_STATUSCONTEXT Return Parameters

Output Parameter Type	Description
xs:string	The decorated status context value of this order item

Example 5–29 is a code fragment from **OracleComms_OSM_O2A_Configuration/fulfillment-state-extension/OnReportStatusContextForDeliver.xquery** that demonstrates the extension implementation.

Example 5–29 REPORT_ORDERITEM_STATUSCONTEXT XQuery Code Fragment

```
import module namespace o2acomfulfillmentstate =
"http://xmlns.oracle.com/communications/ordermanagement/o2acom/fulfillmentstate/co
nstant" at
"http://xmlns.oracle.com/communications/ordermanagement/o2acom/fulfillmentstate/co
nstant/FulfillmentStateConstantModule.xquery";

declare namespace orderItem =
"java:oracle.communications.ordermanagement.fulfillmentstatelifecycle.OrderItemFul
fillmentStateLifecycle";

declare variable $fulfillmentMode as xs:string external;
declare variable $fulfillmentState as xs:string external;
declare variable $orderEventType as xs:string external;
declare variable $orderOperationType as xs:string external;
declare variable $orderItem external;

let $fulfillmentState := orderItem:getFulfillmentState($orderItem)
let $milestone := orderItem:getLatestMilestoneCode($orderItem)
let $statusContext := orderItem:getStatusContext($orderItem)
return
    if ($fulfillmentState = ($o2acomfulfillmentstate:FAILED_STATE,
$o2acomfulfillmentstate:CANCELLED_STATE))
    then $statusContext
    else
    (
        if (fn:exists($milestone) and fn:exists($statusContext) and $milestone !=
"" and $statusContext != "")
        then fn:concat($milestone, ": ", $statusContext)
        else ""
    )
```

Adding a New Service for the Calculate Service Order Solution Option

This procedure describes how to add a new service such as broadband, VoIP, or TV to an Order-to-Activate solution cartridge when you are using the calculate service order option. In this example, the new service would work with new Design Studio elements such as a new product specification, fulfillment pattern, and fulfillment provider. The procedure uses techniques and resources discussed throughout this chapter. For more information about performing the actions in this procedure in Design Studio, see the information about adding a new service in the section on extending component cartridges in the Design Studio Modeling OSM Orchestration Help.

To add a new service using Design Studio:

1. In the workspace containing the central order management cartridges, create a new model project to contain the conceptual model entities for your new service.

Note: If you create a new conceptual model cartridge to contain your new conceptual model entities, you must add that cartridge in the Common Model Entity Container field in an appropriate cartridge. For the Calculate Service Order solution option, add the cartridge to the OracleComms_OSM_O2A_COM_CSO_Model_Container cartridge for a central order management (or combined central order management and service order management) environment, or to OracleComms_OSM_O2A_SOM_CSO_ModelContainer for an environment that contains only service order management.

2. Configure new conceptual model entities as appropriate for the service you are adding. This may include adding a new domain or fulfillment pattern or both, and will include adding a new customer facing service (with a component to create a new named relationship) and adding a new product. For more information about the conceptual model, see *Design Studio Concepts*.
3. Map your conceptual model product to the appropriate conceptual model fulfillment pattern.
4. In the workspace containing the central order management cartridges, create a new Order and Service Management project to host the new service.
5. In the Order and Service Management Project editor **Properties** tab, deselect the **Standalone** check box.

   This allows the cartridge to be referenced in the composite cartridge as part of the solution, rather than as a standalone cartridge with no dependencies.
6. If the new service will communicate with one or more new external systems, create new fulfillment providers for the new systems.

   See "Adding a New Fulfillment Provider" for more information about adding a fulfillment provider.
7. If a new fulfillment provider is introduced for the new service, add decomposition rules for the new service in the topology cartridge, for example OracleComms_OSM_O2A_COM_CSO_Topology.

   Alternatively, modify existing decomposition rules such as, for example, Decomp*FulfillmentFunction*_To_*FulfillmentProvider* for the new service in the topology cartridge.
8. In the **resources/SolutionConfig** folder of the Order-to-Activate composite cartridge, for example OracleComms_OSM_O2A_COM_CSO_Topology, make the following modifications:

   - If new milestones are to be introduced for the new service for new fulfillment provider, add a new <MilestoneMap> element to the **ComponentMilestoneMap.xml** file for the new service without the PONR portion.

     For example:

     ```
     <oms:MilestoneMap systemType="BRM" systemName="*" execMode="do redo amend_
     do">
        <oms:ComponentMilestone>COMPONENT-COMPLETE</oms:ComponentMilestone>
        <oms:Milestone>SYNC CUSTOMER COMPLETE</oms:Milestone>
     </oms:MilestoneMap>
     ```

- If a new fulfillment provider is added for the new service, add a new <targetSystem> element for each new fulfillment provider instance to the **TargetSystemMap.xml** file.

9. If you are using order transformation manager for your new service, do the following:

   a. If you added a new conceptual model domain for your service, add a transformation manager for the domain.

   b. If you want to use a different transformation sequence than the one provided with the Order-to-Activate cartridges, add and configure that transformation sequence.

   c. Create a mapping rule and create mappings for your new named relationship.

10. Create order item parameter bindings from your new conceptual model product to the appropriate order item.

11. Add a new OSM fulfillment pattern that extends from the base specification **BaseProductSpec** (or its extended fulfillment pattern) to represent the new service in central order management. Ensure that your new fulfillment pattern realizes the appropriate conceptual model fulfillment pattern.

    See "Creating a New Fulfillment Pattern" for more information about adding a fulfillment pattern.

12. Specify the location of the external directory containing the fulfillment pattern. From the Window menu, select **Preferences**, then expand **Oracle Design Studio** in the Preferences navigation tree, then select **Order and Service Management Preferences**, and then select **Orchestration Preferences**. Enter the *YourCartridge***/resources/productSpecMapping** directory in the **Product Specification Mapping** field.

13. Add the new central order management cartridge you created for the new service and other cartridges on which it has dependencies to the Dependency tab for the Order-to-Activate composite cartridge, for example OracleComms_OSM_O2A_COM_CSO_Solution.

14. In the workspace containing the service order management cartridges, create new conceptual model entities for the service order management services. These should include:

    - All of the entities created for central order management: If central order management and service order management are in separate workspaces, you may want to export the cartridge containing the new central order management entities from the central order management workspace and import it into the service order management workspace.
    - Additional entities for service order management: These may include resource facing services, resources, and technical actions.

    For more information about the conceptual model, see *Design Studio Concepts*

15. In the workspace containing the service order management cartridges, in the OracleComms_OSM_O2A_SOM_CSO_FulfillmentPattern cartridge, add a new fulfillment pattern that extends from the base specification **SOM_Service.Provision** (or its extended fulfillment pattern) to represent the new service in service order management.

    See "Creating a New Fulfillment Pattern" for more information about adding a fulfillment pattern.

16. Add decomposition rules for the new service in the OracleComms_OSM_O2A_SOM_CSO_Topology cartridge for service order management.

    Alternatively, modify existing decomposition rules such as DecompSomProvisionOrder_To_*FulfillmentProvider* for the new service in the topology cartridge.

17. If the new service will communicate with one or more new external systems, create a new provisioning cartridge similar to OracleComms_OSM_O2A_SOM_CSO_Email_Mapping to provision the service.

18. Add the cartridge you created for the new service and other cartridges on which it has dependencies to the Dependency tab for the Order-to-Activate composite cartridge.

19. (Optional) In the appropriate workspace, add a new role for your service if desired. If you add a new role, you must add an entry to the **userConfig.xml** file for your composite cartridge by doing the following:

    a. Open the Package Explorer view and expand the package for your solution cartridge, such as OracleComms_OSM_O2A_COM_CSO_Solution.

    b. Expand the **userConfig** folder and open the **userConfig.xml** file.

    c. Select the **Source** tab and scroll to the end. Before the closing the </userConfig> element, add an entry like the following:

    ```
    <workgroup name="YourRoleName">
       <user>oms-automation</user>
       <user>osmlf</user>
    </workgroup>
    ```

20. Ensure that the appropriate orders, fulfillment patterns, and recognition rules are included in the manifest for the central order management solution cartridge.

21. Ensure that the appropriate orders, fulfillment patterns, and recognition rules are included in the manifest for the service order management solution cartridge.

22. Package and deploy the Order-to-Activate composite cartridges for both central order management and service order management.

Adding a New Service for the Service Option Without Calculate Service Order

This procedure describes how to add a new service such as broadband, VoIP, or TV to an Order-to-Activate solution cartridge when you are not using the calculate service order option. In this example, the new service would work with new Design Studio elements such as a new product specification, fulfillment pattern, and fulfillment provider. The procedure uses techniques and resources discussed throughout this chapter. For more information about performing the actions in this procedure in Design Studio, see the information about adding a new service in the section on extending component cartridges in the Design Studio Modeling OSM Orchestration Help.

To add a new service using Design Studio:

1. In the workspace containing the central order management cartridges, create a new model project to contain the conceptual model entities for your new service.

Note: If you create a new conceptual model cartridge to contain your new conceptual model entities, you must add that cartridge in the Common Model Entity Container field in an appropriate cartridge. For the solution option without Calculate Service Order, add the cartridge to the recognition cartridge in your environment or create a new OSM component cartridge to contain the entries.

---

2. Configure new conceptual model entities as appropriate for the service you are adding. This may include adding a new domain or fulfillment pattern or both, and will include adding a new customer facing service (with a component to create a new named relationship) and adding a new product. For more information about the conceptual model, see *Design Studio Concepts*.
3. Map your conceptual model product to the appropriate conceptual model fulfillment pattern.
4. In the workspace containing the central order management cartridges, create a new Order and Service Management project to host the new service.
5. In the Order and Service Management Project editor **Properties** tab, deselect the **Standalone** check box.

   This allows the cartridge to be referenced in the composite cartridge as part of the solution, rather than as a standalone cartridge with no dependencies.
6. If the new service will communicate with one or more new external systems, create new fulfillment providers for the new systems.

   See "Adding a New Fulfillment Provider" for more information about adding a fulfillment provider.
7. If a new fulfillment provider is introduced for the new service, add decomposition rules for the new service in the topology cartridge, for example, OracleComms_OSM_O2A_TypicalTopology_Sample cartridge for central order management.

   Alternatively, modify existing decomposition rules such as, for example, Decomp*FulfillmentFunction*_To_*FulfillmentProvider* for the new service in the topology cartridge.
8. In the **resources/SolutionConfig** folder of the Order-to-Activate composite cartridge, for example OracleComms_OSM_O2A_COMSOM_TypicalSolution, make the following modifications:

   - If new milestones are to be introduced for the new service for new fulfillment provider, add a new <MilestoneMap> entry to the ComponentMilestoneMap.xml file for the new service without the PONR portion.

     For example:

     ```
             <oms:MilestoneMap systemType="BRM" systemName="*" execMode="do redo
     amend_do">

     <oms:ComponentMilestone>COMPONENT-COMPLETE</oms:ComponentMilestone>
                 <oms:Milestone>SYNC CUSTOMER COMPLETE</oms:Milestone>
             </oms:MilestoneMap>
     ```

   - If a new fulfillment provider is added for the new service, add a new <targetSystem> entry for each new fulfillment provider instance to the TargetSystemMap.xml file.

9. Create order item parameter bindings from your new conceptual model product to the appropriate order item.
10. Add a new OSM fulfillment pattern that extends from the base specification **BaseProductSpec** (or its extended fulfillment pattern) to represent the new service in central order management. Ensure that your new fulfillment pattern realizes the appropriate conceptual model fulfillment pattern.

    See "Creating a New Fulfillment Pattern" for more information about adding a fulfillment pattern.
11. Specify the location of the external directory containing the fulfillment pattern. From the Window menu, select **Preferences**, then expand **Oracle Design Studio** in the Preferences navigation tree, then select **Order and Service Management Preferences**, and then select **Orchestration Preferences**. Enter the appropriate directory, for example, **OracleComms_OSM_O2A_FulfillmentPatternMap_Sample/resources/productSpecMapping**, in the **Product Specification Mapping** field.
12. Add the new central order management cartridge you created for the new service and other cartridges on which it has dependencies to the Dependency tab for the Order-to-Activate composite cartridge, for example OracleComms_OSM_O2A_COMSOM_TypicalSolution.
13. In the workspace containing the service order management cartridges, create new conceptual model entities for the service order management services. These should include:
    - All of the entities created for central order management: If central order management and service order management are in separate workspaces, you may want to export the cartridge containing the new central order management entities from the central order management workspace and import it into the service order management workspace.
    - Additional entities for service order management: These may include resource facing services, resources, and technical actions.

    For more information about the conceptual model, see *Design Studio Concepts*
14. In the workspace containing the service order management cartridges, in the fulfillment pattern cartridge for service order management (for example OracleComms_OSM_O2A_SomBBVoIP_FP_NP_Sample), add a new fulfillment pattern that extends from the base specification **SOM_Service.Provision** (or its extended fulfillment pattern) to represent the new service in service order management.

    See "Creating a New Fulfillment Pattern" for more information about adding a fulfillment pattern.
15. Add decomposition rules for the new service in the OracleComms_OSM_O2A_SomBBVoIPFulfillmentPattern_Sample cartridge for service order management.

    Alternatively, modify existing decomposition rules such as DecompSomProvisionOrder_To_*FulfillmentProvider* for the new service in the topology cartridge.
16. If the new service will communicate with one or more new external systems, create a new provisioning cartridge similar to OracleComms_OSM_O2A_SomProvisionVoIP_Sample to provision the service.

17. Add the cartridge you created for the new service and other cartridges on which it has dependencies to the Dependency tab for the Order-to-Activate composite cartridge.

18. (Optional) In the appropriate workspace, add a new role for your service if desired. If you add a new role, you must add an entry to the **userConfig.xml** file for your composite cartridge by doing the following:

    a. Open the Package Explorer view and expand the package for your solution cartridge, such as OracleComms_OSM_O2A_COMSOM_TypicalSolution.

    b. Expand the **userConfig** folder and open the **userConfig.xml** file.

    c. Select the **Source** tab and scroll to the end. Before the closing </userConfig> element, add an entry like the following:

    ```
    <workgroup name="YourRoleName">
       <user>oms-automation</user>
       <user>osmlf</user>
    </workgroup>
    ```

19. Ensure that the appropriate orders, fulfillment patterns, and recognition rules are included in the manifest for the central order management solution cartridge.

20. Ensure that the appropriate orders, fulfillment patterns, and recognition rules are included in the manifest for the service order management solution cartridge.

21. Package and deploy the Order-to-Activate composite cartridges for both central order management and service order management.

Customizing Service Order Management

The Order-to-Activate cartridges use XQuery resources to perform functions in service order management including formatting request messages, processing faults, tracking external system interactions, and processing external fulfillment states. One way to customize XQueries is to rewrite or add to the out-of-box XQuery module and use the XML catalog to enable URI reference mapping. Extension points are defined for both service order management and central order management. This section contains information about the service order management extension points. For information about the central order management extension points, see "Fulfillment Function Extension Point Interface" and "Fulfillment State Extension Point Interface."

XML catalogs are system-wide entities, which means an XML Catalog specified in one cartridge will be used when processing requests for orders on other cartridges. With the use of solution cartridges, multiple solutions can be deployed to a single system and coexist with each other.

An XQuery extension script must be implemented in a standalone file. The file URI must be registered to the extension configuration.

Service Order Management Extension Point Overview

Table 5–72 lists the XQuery extension points for the service order management Order-to-Activate cartridges.

Table 5–72 Service Order Management Extension Points

Service Order Management Extension Point	Description
SOM-CREATE-SOAP-REQUEST	SOM-CREATE-SOAP-REQUEST at the order level is triggered when the OSM fulfillment state engine finishes evaluating the composite fulfillment state for the order. This extension point provides order status to the upstream system. ORDER_STATUS at the order item level is triggered when the OSM fulfillment state engine finishes evaluating the composite fulfillment state for an order item. This extension point provides order item status to the upstream system.
SOM-DETECT-FAULT	SOM-DETECT-FAULT is triggered when the OSM fulfillment state engine finishes evaluation of the composite fulfillment state for the order. This extension point provides order status context to the upstream system.
SOM-GET-FAULT-DATA	SOM-GET-FAULT-DATA is triggered when the OSM fulfillment state engine finishes evaluating the composite fulfillment state for an order item. This extension point provides order item status context to the upstream system.
SOM-CHECK-IS-LAST-RESPONSE	SOM-CHECK-IS-LAST-RESPONSE is triggered when the OSM fulfillment state engine finishes evaluating the composite fulfillment state for an order item. This extension point calculates the order item milestone, taking order cancellation into consideration.
SOM-GET-UPDATE-DATA	SOM-GET-UPDATE-DATA is triggered when the OSM fulfillment state engine finishes evaluating the composite fulfillment state for an order. This extension point overrides the default evaluation of the order composite fulfillment state modeled in COM_OrderStateCompositionRule to support an order with no order items (order items had been dropped during revision), and to support the completion of a cancellation order.
SOM-GET-EXTERNAL-FULFILLMENT-STATE	SOM-GET-EXTERNAL-FULFILLMENT-STATE is triggered when the OSM fulfillment state engine finishes evaluating the composite fulfillment state for an order item. This extension point overrides the default evaluation of the order item composite fulfillment state modeled in COM_OrderItemStateCompositionRule to support an order with no order items (order items had been dropped during revision), and to support the completion of a cancellation order.
SOM-GET-EXTERNAL-FULFILLMENT-STATE-AT-FALLOUT	SOM-GET-EXTERNAL-FULFILLMENT-STATE-AT-FALLOUT is triggered when the OSM fulfillment state engine finishes calculating the composite fulfillment state for an order item. This extension point is not currently being used in the Order-to-Activate cartridges.
SOM-GET-NEW-CORRELATION-ID	SOM-GET-NEW-CORRELATION-ID is triggered when the OSM fulfillment state engine finishes evaluating the composite fulfillment state for an order item. This extension point overrides the default milestone when handling a cancellation order.

When you customize service order management, you can create an ExtensionPointMap entry for each applicable extension point (such as creating a SOAP request) in the **resources\SolutionConfig\SomComponenExtensionPointMap.xml** of the Order-to-Activate composite cartridge. You must create a separate XQuery file for each extension point.

SOM-CREATE-SOAP-REQUEST Extension Point

This section describes the XQuery script that implements the logic to handle the SOM-CREATE-SOAP-REQUEST extension point.

Table 5–73 lists the input parameters for the extension point XQuery.

Table 5–73 SOM-CREATE-SOAP-REQUEST Input Parameters

Name	Type	Scope	Description
$log	Java Object	External variable	Java type org.apache.commons.logging.Log Logging level related to server log
$context	Java Object	External variable	Java type com.mslv.oms.automation.OrderContext Context of the request
$executionMode	xs:string	External variable	Task execution mode
$breakpointDebugControl	element()	External variable	Debug control XML fragment
$componentName	xs:string	External variable	Fulfillment function name
$taskName	xs:string	External variable	Task name

Table 5–74 lists the return parameters for the extension point XQuery.

Table 5–74 SOM-CREATE-SOAP-REQUEST Return Parameters

Output Parameter Type	Description
element()*	Return XML message/payload to be sent to the external system. The external system properties (for example, JMS) must be set by the extension point.

SOM-DETECT-FAULT Extension Point

This section describes the XQuery script that implements the logic to handle the SOM-DETECT-FAULT extension point.

Table 5–75 lists the input parameters for the extension point XQuery.

Table 5–75 SOM-DETECT-FAULT Input Parameters

Name	Type	Scope	Description
$log	Java Object	External variable	Java type org.apache.commons.logging.Log Logging level related to server log
$taskInputData	element()	External variable	Task data XML fragment
$executionMode	xs:string	External variable	Task execution mode
$breakpointDebugControl	element()	External variable	Debug control XML fragment
$componentName	xs:string	External variable	Fulfillment function name
$taskName	xs:string	External variable	Task name
.	Context node	Context node	Response message XML fragment

Table 5–76 lists the return parameters for the extension point XQuery.

Table 5–76 SOM-DETECT-FAULT Return Parameters

Output Parameter Type	Description
element()	Return an XML fragment in the following format if a fault is detected: `<oms:FaultCnt><oms:Fault></oms:Fault></oms:FaultCnt>` Return an XML fragment in the following format if a fault is not detected: `<oms:FaultCnt></oms:FaultCnt>`

SOM-GET-FAULT-DATA Extension Point

This section describes the XQuery script that implements the logic to handle the SOM-GET-FAULT-DATA extension point.

Table 5–77 lists the input parameters for the extension point XQuery.

Table 5–77 SOM-GET-FAULT-DATA Input Parameters

Name	Type	Scope	Description
$log	Java Object	External variable	Java type org.apache.commons.logging.Log Logging level related to server log
$taskInputData	element()	External variable	Task data XML fragment
$executionMode	xs:string	External variable	Task execution mode
$breakpointDebugControl	element()	External variable	Debug control XML fragment
$componentName	xs:string	External variable	Fulfillment function name
$taskName	xs:string	External variable	Task name
.	Context node	Context node	Response message XML fragment

Table 5–78 lists the return parameters for the extension point XQuery.

Table 5–78 SOM-GET-FAULT-DATA Return Parameters

Output Parameter Type	Description
element()	Extract, transform, and return fault data from the response message into OrderDataUpdate.

SOM-CHECK-IS-LAST-RESPONSE Extension Point

This section describes the XQuery script that implements the logic to handle the SOM-CHECK-IS-LAST-RESPONSE extension point.

Table 5–79 lists the input parameters for the extension point XQuery.

Table 5–79 SOM-CHECK-IS-LAST-RESPONSE Input Parameters

Name	Type	Scope	Description
$log	Java Object	External variable	Java type org.apache.commons.logging.Log Logging level related to server log
$taskInputData	element()	External variable	Task data XML fragment
$executionMode	xs:string	External variable	Task execution mode

Table 5–79 (Cont.) SOM-CHECK-IS-LAST-RESPONSE Input Parameters

Name	Type	Scope	Description
$breakpointDebugControl	element()	External variable	Debug control XML fragment
$componentName	xs:string	External variable	Fulfillment function name
$taskName	xs:string	External variable	Task name
.	Context node	Context node	Response message XML fragment

Table 5–80 lists the return parameters for the extension point XQuery.

Table 5–80 SOM-CHECK-IS-LAST-RESPONSE Return Parameters

Output Parameter Type	Description
element()	Return an XML fragment in the following format if this is the last response: `<oms:IsLastResponseCnt><oms:LastResponse/></oms:IsLastResponseCnt>` Return an XML fragment in the following format if this is not the last response: `<oms:IsLastResponseCnt/>`

SOM-GET-UPDATE-DATA Extension Point

This section describes the XQuery script that implements the logic to handle the SOM-GET-UPDATE-DATA extension point.

Table 5–81 lists the input parameters for the extension point XQuery.

Table 5–81 SOM-GET-UPDATE-DATA Input Parameters

Name	Type	Scope	Description
$log	Java Object	External variable	Java type org.apache.commons.logging.Log Logging level related to server log
$taskInputData	element()	External variable	Task data XML fragment
$executionMode	xs:string	External variable	Task execution mode
$breakpointDebugControl	element()	External variable	Debug control XML fragment
$componentName	xs:string	External variable	Fulfillment function name
$taskName	xs:string	External variable	Task name
.	Context node	Context node	Response message XML fragment

Table 5–82 lists the return parameters for the extension point XQuery.

Table 5–82 SOM-GET-UPDATE-DATA Return Parameters

Output Parameter Type	Description
element()	Extract and transform order data from the response message into OrderDataUpdate.

SOM-GET-EXTERNAL-FULFILLMENT-STATE Extension Point

This section describes the XQuery script that implements the logic to handle the SOM-GET-EXTERNAL-FULFILLMENT-STATE extension point.

Table 5–83 lists the input parameters for the extension point XQuery.

Table 5–83 SOM-GET-EXTERNAL-FULFILLMENT-STATE Input Parameters

Name	Type	Scope	Description
$log	Java Object	External variable	Java type org.apache.commons.logging.Log Logging level related to server log
$taskInputData	element()	External variable	Task data XML fragment
$executionMode	xs:string	External variable	Task execution mode
$breakpointDebugControl	element()	External variable	Debug control XML fragment
$componentName	xs:string	External variable	Fulfillment function name
$taskName	xs:string	External variable	Task name
.	Context node	Context node	Response message XML fragment

Table 5–84 lists the return parameters for the extension point XQuery.

Table 5–84 SOM-GET-EXTERNAL-FULFILLMENT-STATE Return Parameters

Output Parameter Type	Description

element()	Return an XML fragment in the following format to pass back the state value that applies to all order items:
	`<oms:ExternalFulfillmentStateCnt>`
	    `<oms:ComponentFulfillmentState>$StateValue`
	    `</oms:ComponentFulfillmentState>`
	`</oms:ExternalFulfillmentStateCnt>`
	Return an XML fragment in the following format to pass back the state value for individual order items:
	`<oms:ExternalFulfillmentStateCnt>`
	    `<oms:lineItem index="123456">`
	        `<oms:baseLineId>baseLineId</oms:baseLineId>`
	        `<oms:milestoneCode>Milestone</oms:milestoneCode>`
	        `<oms:statusCode>Status</oms:statusCode>`
	        `<oms:statusContext>Description</oms:statusContext>`
	        `<oms:lineType>LINE_TYPE_NONCSO</oms:lineType>`
	    `</oms:lineItem>`
	`</oms:ExternalFulfillmentStateCnt>`

SOM-GET-EXTERNAL-FULFILLMENT-STATE-AT-FALLOUT Extension Point

This section describes the XQuery script that implements the logic to handle the SOM-GET-EXTERNAL-FULFILLMENT-STATE-AT-FALLOUT extension point.

Table 5–85 lists the input parameters for the extension point XQuery.

Table 5–85 SOM-GET-EXTERNAL-FULFILLMENT-STATE-AT-FALLOUT Input Parameters

Name	Type	Scope	Description
$log	Java Object	External variable	Java type org.apache.commons.logging.Log Logging level related to server log
$taskInputData	element()	External variable	Task data XML fragment
$executionMode	xs:string	External variable	Task execution mode
$breakpointDebugControl	element()	External variable	Debug control XML fragment
$componentName	xs:string	External variable	Fulfillment function name
$taskName	xs:string	External variable	Task name
.	Context node	Context node	Response message XML fragment

Table 5–86 lists the return parameters for the extension point XQuery.

Table 5–86 SOM-GET-EXTERNAL-FULFILLMENT-STATE-AT-FALLOUT Return Parameters

Output Parameter Type	Description

element()	Return an XML fragment in the following format to pass back the state value that applies to all order items:
	`<oms:ExternalFulfillmentStateCnt>`
	    `<oms:ComponentFulfillmentState>$StateValue`
	    `</oms:ComponentFulfillmentState>`
	`</oms:ExternalFulfillmentStateCnt>`
	Return an XML fragment in the following format to pass back the state value for individual order items:
	`<oms:ExternalFulfillmentStateCnt>`
	    `<oms:lineItem index="123456">`
	        `<oms:baseLineId>baseLineId</oms:baseLineId>`
	        `<oms:milestoneCode>Milestone</oms:milestoneCode>`
	        `<oms:statusCode>Status</oms:statusCode>`
	        `<oms:statusContext>Description</oms:statusContext>`
	        `<oms:lineType>LINE_TYPE_NONCSO</oms:lineType>`
	    `</oms:lineItem>`
	`</oms:ExternalFulfillmentStateCnt>`

SOM-GET-NEW-CORRELATION-ID Extension Point

This section describes the XQuery script that implements the logic to handle the SOM-GET-NEW-CORRELATION-ID extension point.

Table 5–87 lists the input parameters for the extension point XQuery.

Table 5–87 SOM-GET-NEW-CORRELATION-ID Input Parameters

Name	Type	Scope	Description
$log	Java Object	External variable	Java type org.apache.commons.logging.Log Logging level related to server log
$taskInputData	element()	External variable	Task data XML fragment
$executionMode	xs:string	External variable	Task execution mode
$breakpointDebugControl	element()	External variable	Debug control XML fragment
$componentName	xs:string	External variable	Fulfillment function name
$taskName	xs:string	External variable	Task name
.	Context node	Context node	Response message XML fragment

Table 5–88 lists the return parameters for the extension point XQuery.

Table 5–88 SOM-GET-NEW-CORRELATION-ID Return Parameters

Output Parameter Type	Description
element()	Return an XML fragment in the following format to pass back the ID: `<oms:NewCorrelationIdCnt>$correlationId</<oms:NewCorrelationIdCnt>`

Extending XQuery Modules

This section contains general information about extending XQuery modules.

If it is necessary to extend XQuery modules that reside in a sealed cartridge, you must make a copy of the XQuery file and extend it to include custom business logic using the XML catalog.

To extend an XQuery module:

1. In Design Studio, from the **Window** menu, select **Show View** and then select **Package Explorer**.
2. Copy the XQuery file that you want to extend and modify the copy to include custom logic. It's recommended to put the copy of the XQuery files in the Order-to-Activate composite cartridge.
3. In the Project Explorer view, open the Order-to-Activate composite cartridge and navigate to the **xmlCatalogs/core** directory.
4. Open the file **catalog.xml**.
5. Update the **catalog.xml** file by adding an entry to override the XQuery implementation. The new entry should look like this:

```
<rewriteURI
uriStartString="http://xmlns.oracle.com/communications/ordermanagement/pip/<pat
h>/<XQuery>.xquery" rewritePrefix="osmmodel:///OracleComms_OSM_O2A_COMSOM_
TypicalSolution/1.0.0.0.0/resources/<path>/<XQuery>.xquery"/>
```

Note: This entry must appear on a single line in the file.

Sending Enriched Data to the CRM System

Data that was not originally supplied in the order from the CRM system must often be sent up to Siebel CRM from a downstream system; this data is referred to as *enriched data*. Enriched data can be an update to an order header value or an added attribute to a line item that was originally supplied in the order from the CRM system. Enriched data cannot include new line items that were not part of the original order from Siebel CRM (only modifications to existing line items).

In the Order-to-Activate cartridges, downstream systems can send enriched data up to the CRM system using the **SpecificationGroup** area of each order line item. **SpecificationGroup** is a structure that contains multiple substructures of the order specification. The **Specification.Name** and **Specification.Value** parameters store the enriched data.

The demonstration cartridges OracleComms_OSM_O2A_SomProvisionBroadband_Sample and OracleComms_OSM_O2A_SomProvisionVoIP_Sample provide an example of sending enriched data to the CRM system. These cartridges must populate the EBM type **ProcessProvisioningOrderUpdateEBM** and send that data to the OracleComms_OSM_O2A_SOM_Base cartridge. The OracleComms_OSM_O2A_SOM_Base cartridge in turn creates another EBM type **ProcessFulfillmentOrderUpdateEBM** that contains the **SpecificationGroup** structure and sends that data to the OracleComms_OSM_O2A_COM_Base cartridge. The OracleComms_OSM_O2A_COM_Base cartridge in turn creates the EBM type **UpdateSalesOrderEBM** and sends that data to the CRM system. The **SpecificationGroup** structure is defined in all of these EBM types.

For example, when the Provisioning task completes in the OracleComms_OSM_O2A_SomProvisionBroadband_Sample cartridge, the service ID is populated and propagated back to the CRM system, which can now use the service ID to track the asset.

Considerations When Integrating with Oracle AIA

The following points refer to EBO attributes that use domain value maps in Oracle AIA, and how to extend the list of seeded values in the OSM cartridges. You update the validation rule in the OracleComms_OSM_O2A_COM_Base cartridge (either reduce the validation level or include the new values) and describe the extensibility of each such EBO attribute on a case-by-case basis.

Consider the following factors when integrating OSM with Oracle AIA:

- XML tags - some fields are key fields with enumerated values. These values are hard coded in the cartridge so they have to match. These are documented in the data dictionary in the cartridge itself.
- The EBO attribute values can be extended - you can add your own values, but they have to line up in the data dictionary in the cartridge, in the customized rules in the cartridge, for instance, where you choose a billing instance based on a value such as BUSINESS vs. RESIDENTIAL. The value would have to come from ABCS correctly to match. Other custom rules could be created that switch on this value.
- Consumers of OSM generated EBMs such as OSM in its service order management role, and ABCS should not make use of the attributes schemeID, and schemeAgencyID in order identifications, order references, line identifications, and line references. The following are couple of examples:

```
<corecom:BusinessComponentID schemeID="SALESORDER_ID" schemeAgencyID="COMMON">
<corecom:BusinessComponentID schemeID="SALESORDER_LINEID"
```

```
schemeAgencyID="COMMON">
```

- Queue names have to agree if you add another billing function - you would not have to add more queues if you were not adding more fulfillment functions. But if you do, the queue names have to agree with the ABCS.

Table 5–89 lists the summary of JNDI names for WebLogic JMS Queues for system interactions included in the Order-to-Activate cartridges. In support of system interactions, OSM central order management and OSM service order management communicate with the ABCS for the fulfillment systems such as Siebel CRM ABCS and Oracle Communications Billing and Revenue Management (BRM) ABCS through posting JMS messages to the queues given below.

In the WebLogic Server Administration console, queues are found by navigating to: **Home /JMS Modules /oms_jms_module**.

Note: There must be an alignment of JNDI names between OSM and the ABCSs that communicate with it. You must be aware of this alignment if you add additional queues for new fulfillment functions.

Table 5–89 WebLogic JNDI Request/Response queues

System interaction request/response	JNDI name	In-bound/Out-bound
AIA Customer Order support	oracle/communications/ordermanagement/WebServiceQueue	Siebel CRM ABCS to OSM central order management
AIA Service Order support	oracle/communications/ordermanagement/WebServiceQueue	In-bound to OSM service order management
SyncCustomer request	oracle/communications/ordermanagement/WebServiceCreateCustomerQueue	OSM central order management to BRM ABCS
SyncCustomer response	oracle/communications/ordermanagement/WebServiceCreateCustomerResponseQueue	BRM ABCS to OSM central order management
InitiateBilling request	oracle/communications/ordermanagement/WebServiceCreateBillingOrderQueue	OSM central order management to BRM ABCS
InitiateBilling response	oracle/communications/ordermanagement/WebServiceCreateBillingOrderResponseQueue	BRM ABCS to OSM central order management
FulfillBilling request	oracle/communications/ordermanagement/WebServiceCreateBillingOrderQueue	OSM central order management to BRM ABCS
FulfillBilling response	oracle/communications/ordermanagement/WebServiceCreateBillingOrderResponseQueue	BRM ABCS to OSM central order management
ProvisionOrder request	oracle/communications/ordermanagement/WebServiceCreateProvisioningOrderQueue	OSM central order management to Oracle AIA destined for OSM service order management
ProvisionOrder response (ProcessFulfillmentOrderUpdate)	oracle/communications/ordermanagement/WebServiceUpdateFulfillmentOrderQueue	OSM service order management to Oracle AIA destined for OSM central order management
CancelProvisioningOrder request	oracle/communications/ordermanagement/WebServiceCancelProvisioningOrderQueue	OSM central order management to OSM service order management

Table 5–89 (Cont.) WebLogic JNDI Request/Response queues

System interaction request/response	JNDI name	In-bound/Out-bound
UpdateSalesOrder	oracle/communications/ordermanagement/WebServiceUpdateSalesOrderQueue	OSM central order management to Siebel CRM ABCS
CreateTroubleTicket request	oracle/communications/ordermanagement/CreateTroubleTicketRequestQueue	OSM central order management to Siebel CRM ABCS
CreateTroubleTicket response	oracle/communications/ordermanagement/CreateTroubleTicketResponseQueue	Siebel CRM ABCS to OSM central order management
UpdateTroubleTicket request	oracle/communications/ordermanagement/UpdateTroubleTicketRequestQueue	OSM central order management to Siebel CRM ABCS
CreateErrorFault	oracle/communications/ordermanagement/CreateErrorFaultQueue	OSM service order management to Oracle AIA
Fallout for service order management response	oracle/communications/ordermanagement/WebServiceFalloutLFResponseQueue	OSM service order management to central order management
Abort order response	oracle/communications/ordermanagement/LFAbortOrderPropagationRespQueue	OSM Provisioning to service order management

Security Considerations When Communicating with External Systems

Whenever you are integrating OSM with an external system via JMS, make sure that you have set up security settings on the JMS module in WebLogic Server. If you use the supplied JMS module, **oms_jms_module**, security settings have already been set up. However, if you use a different JMS module, you must set up appropriate security on it.

6 Performing Order-to-Activate Cartridge Operations

This chapter describes operational procedures that may be needed for the Order-to-Activate cartridges for Oracle Communications Order and Service Management (OSM).

XQuery Transformation Logging

To enable or disable logging for XQuery transformations, use the **log4jAdmin** page. See the section on configuring severity levels in the Monitoring and Managing OSM chapter in *OSM System Administrator's Guide* for more information about using the **log4jAdmin** page. The following information will help you when looking at the generic information about using **log4jAdmin** that is located there:

- If you want to set the logging level temporarily, first run an order that calls the XQuery transformation that you want to log. This ensures that the EJB that runs it is loaded. Otherwise you may not find the appropriate logger on the log4jAdmin page.
- To find the logger for the XQuery transformation that you want to change, look at the name formats below and look up the relevant information in Oracle Communications Design Studio.
  - If the automation is an external event receiver, the logger will be listed with a name like this:

    ```
    /automation/plugin/external/SolutionCartridgeName/SolutionCartridgeVersion/
    TaskOrEventName.EJBName
    ```

    For example:

    ```
    /automation/plugin/external/OracleComms_OSM_O2A_COMSOM_
    TypicalSolution/2.0.1.2.0/FulfillBillingSITask.FulfillBillingResponseBean
    ```

  - If the automation is an internal event receiver, the logger will be listed with a name like this:

    ```
    /automation/plugin/internal/AutomationType/SolutionCartridgeName/SolutionCa
    rtridgeVersion/AutomationEntityName/CompensationMode
    ```

    where /*CompensationMode* is only included for task automations.

    For example:

    ```
    /automation/plugin/internal/task/OracleComms_OSM_O2A_COMSOM_
    TypicalSolution/2.0.1.2.0/FulfillBillingEntryPointTask/do_redo_undo
    ```

Troubleshooting Order-to-Activate Cartridges

The following procedures can help you in troubleshooting issues with the Order-to-Activate cartridges.

Updating the JMS Redelivery Configuration Settings

When the Order-to-Activate cartridges are installed, the **Redelivery Delay Override** and **Redelivery Limit** WebLogic parameters are set during installation to 7000ms and 10, respectively. However, different values may be more effective for your OSM environment depending on your usage of the system.

If you encounter timing-related issues for message delivery on JMS queues, there are a number of WebLogic settings that you can modify to resolve the issue. These values are set on every JMS queue through the WebLogic Service Console. From **Home**, select **JMS Modules**, and then select **oms_jms_module** to modify the following settings:

- **Redelivery Delay Override**: Delay in milliseconds before rolled back or recovered messages are redelivered. This value overrides the Redelivery Delay setting.
- **Redelivery Limit**: The number of times to attempt to redeliver a message.

To find the best values for these parameters, start with initial values less than 7000ms for the **Redelivery Delay Override** and 10 for the **Redelivery Limit** and increase them slightly until no occurrences of errors are observed. The actual values you finalize on will depend on your particular implementation of OSM. See the Oracle WebLogic documentation for complete details on these parameters.

Setting Cartridge Breakpoints

There are process flows in the cartridge with a manual task between each automated task. With certain input data in the customer order, it causes it to go through the automation and stop at a particular manual task.

The Order-to-Activate cartridges have been instrumented with control points in the process flows so that a tester can control the process flow before or after functions, examine data anywhere in the flow, do revision testing, and do point-of-no-return testing. The flows are automated, but can be instructed to stop at a manual task before or after normal automated tasks.

The Siebel Customer Relationship Management (Siebel CRM) sales order number is used to control the flows by prefixing the number with format [AIATest.Task#.Target#]. Only one breakpoint can be set.

Table 6–1 lists the task numbers and names.

Table 6–1 Task Number and Task Name

Task #	Task Name
0	Any
1	Before SyncCustomer Task
2	Before InitiateBilling Task
3	Before FulfillBilling Task
7	After SyncCustomer Task
8	After InitiateBilling Task
9	After FulfillBilling Task

Table 6–1 (Cont.) Task Number and Task Name

Task #	Task Name
10	After Provisioning Request Sent Task
11	After Provision Response Received Task
31	Before Provision InternetServiceBundle Task
32	Before Provision InternetMediaServiceBundle Task
33	Before Provision InternetEmailServiceBundle Task
34	Before Provision CPEEntryPointTask
41	Before Provision VoIPServiceBundle Task
42	Before Provision VoIPCPE Task

Table 6–2 lists the OSM and Oracle Application Integration Architecture (Oracle AIA) fulfillment systems.

Table 6–2 OSM and Oracle AIA Targets

Target#	OSM Fulfillment System Name	AIA Logical Fulfillment System ID
0	Any	Any
1	BRM-ALL	BRM_01
2	BRM-VoIP	BRM_02
4	BRM-REZBDB	BRM_03
4	BRM-BIZBDB	BRM_04
7	Provisioning-ALL	OSMPROV_01
8	Provisioning-VoIP	OSMPROV_02
9	Provisioning-VoIP	OSMPROV_02
10	Provisioning-BRD	OSMPROV_03

It is also possible to disable the PoNR per-component level by setting the order key prefix as [PONRControl.Component#].

Table 6–3 lists the OSM component levels.

Table 6–3 OSM Component levels

Component #	OSM Component Name	Example
0	Any	[PONRControl.0]
1	SyncCustomerFunction	[PONRControl.1]
2	InitiateBillingFunction	[PONRControl.2]
3	ProvisionOrderFunction	[PONRControl.3]
4	ShipOrderFunction	[PONRControl.4]
5	InstallOrderFunction	[PONRControl.5]
6	FulfillBillingFunction	[PONRControl.6]
7	InstallOrderFunction ShipOrderFunction	[PONRControl.7]

Table 6–3 (Cont.) OSM Component levels

Component #	OSM Component Name	Example
8	ProvisionOrderFunction, InstallOrderFunction, ShipOrderFunction	[PONRControl.8]

7

Prior Versions of Order-to-Activate Cartridges

This chapter provides information about prior versions of Oracle Communications Order and Service Management (OSM) Order-to-Activate cartridges. It contains information about updating prior versions of the Order-to-Activate cartridges to work with newer versions of OSM and describes the changes that were made in recent versions of the cartridges.

Updating Prior Versions of the Cartridges to Work with Newer Versions of OSM

It is possible to update prior versions of the Order-to-Activate cartridges to work with newer versions of OSM. To update Order-to-Activate cartridges to work with OSM 7.2 and earlier, see *OSM Cartridge Guide for Oracle Application Integration Architecture* for the version of OSM you want to use. To update Order-to-Activate cartridges to work with OSM 7.2.2 or later, see *OSM Installation Guide* for the version of OSM you want to use.

Changes from Order-to-Activate 2.1.1 to Version 2.1.2

This section provides a high-level description of the changes between Order-to-Activate 2.1.1 cartridges and Order-to-Activate 2.1.2 cartridges.

Removed Support for Asset Processing

The changes that were made in version 2.1.1 to the Order-to-Activate cartridges to support managing assets on orders that come from Oracle Configure, Price, and Quote Cloud (Oracle CPQ Cloud) have been removed. The corresponding functionality is no longer available in the OSM product. Oracle recommends that you use corresponding functionality in Oracle Configure, Price, Quote (CPQ) Cloud for your hybrid cloud solution.

Processes Changed to Use a Single Task for Sending and Receiving

Starting in the Order-to-Activate cartridges version 2.1.2, system interaction processes have been modified to use a single task to handle both sending a message to an external system and receiving the response messages. The purpose of this change is to greatly simplify the process design by allowing the same task to handle all of the normal and exception processing for the message, and complements functionality introduced in OSM 7.3.

Central Order Management Fulfillment Functions

Following is a summary of the processing for the central order management fulfillment functions.

- **SyncCustomerFunction**: The process for this function is **SyncCustomerSubProcess**. The tasks are:
  - **SyncCustomerEntryPointTask**: This task sets up the data for the system interaction task, determines whether there are any open trouble tickets for the current instance of the function, and if so closes them. It also provides a breakpoint.
  - **SyncCustomerSITask**: This task creates and sends a request to the billing system. If the billing system has an error, it notifies fallout management directly, and then fallout management sends a notification to this task, which causes the state of the task to enter fallout mode and change its state to WaitForFalloutRecovery. You can manually transition the task from this state to the resolveFailAndRetry state to return to normal mode and retry the task. Alternatively, an amendment can be received to retry the task automatically. The task also provides a breakpoint if a successful response ifs received.
- **InitiateBillingFunction**: The process for this function is **InitiateBillingSubProcess**. The tasks are:
  - **InitiateBillingEntryPointTask**: This task works like SyncCustomerEntryPointTask.
  - **InitiateBillingSITask**: This task works like SyncCustomerSITask.
- **FulfillBillingFunction**: The process for this function is **FulfillBillingSubProcess**. The tasks are:
  - **FulfullBillingStartTask**: This task provides the opportunity for an extra breakpoint for this process.
  - **InitiateBillingEntryPointTask**: This task works like SyncCustomerEntryTask.
  - **InitiateBillingSITask**: This task works like SyncCustomerSITask.
- **ProvisionOrderFunction**: The process for this function is **ProvisionOrderSubProcess**. The tasks are:
  - **ProvisionOrderEntryPointTask**: This task works like SyncCustomerEntryPointTask.
  - **ProvisionOrderSITask**: This task works like SyncCustomerSITask, except that it sends the request to the service order management system rather than the billing system.

Service Order Management Fulfillment Functions

Following is a summary of the processing for the service order management fulfillment functions. These fulfillment functions are available only with the Calculate Service Order solution option.

- **DesignServiceFunction**: The process for this function is **DesignServiceProcess**. The tasks are:
  - **CaptureBITask**: This task creates and sends the capture business interaction request to the inventory system. If the inventory system has an error, it returns a failure response directly to the task, which causes the state of the task to enter fallout mode. You can manually retry the task by changing the state to received, or you can transition the task to WaitForFalloutRecovery, which

causes a failure message to be sent to the central order management system. You can manually transition the task from this state to the resolveFailAndRetry state to return to normal mode and retry the task. Alternatively, an amendment can be received to retry the task automatically. It also provides a breakpoint if a successful response ifs received.

  - **ProcessBITask**: This task creates and sends the process business interaction request to the inventory system. If the inventory system has an error, it returns a failure response directly to the task, which causes the state of the task to enter fallout mode. You can manually retry the task by changing the state to received, or you can transition the task to WaitForFalloutRecovery, which causes a failure message to be sent to the central order management system. You can manually transition the task from this state to the resolveFailAndRetry state to return to normal mode and retry the task. Or you can transition the task to clearFallout to ignore the error and complete the task successfully. Alternatively, an amendment can be received to retry the task automatically. The task also provides a breakpoint if a successful response ifs received.
  - **ApproveBITask**: This task works like ApproveBITask, except that it sends the approve business interaction request.

- **PlanDeliveryFunction**: The process for this function is **PlanDeliveryProcess**. The tasks are:
  - **IssueBITask**: This task works like ApproveBITask, except that it sends the approve business interaction request.
  - **CalculateTechnicalActionsTask**: This task works like ApproveBITask, except that it sends the calculate technical actions request.
- **DeliverOrderFunction**: The process for this function is **DeliverOrderProcess**. The tasks are:
  - **CreateTechnicalOrderTask**: This task works like ApproveBITask, except that it sends the create technical order request to the technical order management system.
- **CompleteProvisioningFunction**: The process for this function is **CompleteProvisioningProcess**. The tasks are:
  - **CompleteBITask**: This task works like ApproveBITask, except that it sends the complete business interaction request.

Changes from Order-to-Activate 2.1.0.2 to Version 2.1.1

This section provides a high-level description of the changes between Order-to-Activate 2.1.0.2 cartridges and Order-to-Activate 2.1.1 cartridges.

Support for Asset Processing

Changes have been made to the Order-to-Activate cartridges to support managing assets on orders that come from Oracle Configure, Price, and Quote Cloud (Oracle CPQ Cloud). During installation of the Order-to-Activate cartridges, you can select whether to enable asset processing using the standard OSM Customer Asset Manager, an external asset manager, or both. If you decide to use an external asset manager, you can configure the format of the message.

Support for Order Lifecycle Management User Interface

The Order-to-Activate cartridges have been updated to include an Order Lifecycle Manager entity in support of the new OSM Order Lifecycle Management user interface.

OSM order components previously included a "Minimum Processing Duration." Now, instead of one duration, there are three: Optimistic, Most Likely, and Pessimistic Processing Durations. These processing durations drive information in the Order Lifecycle Manager user interface. Since the order components for the Order -to-Activate cartridges are in sealed cartridges, changing these durations in Order Component Specification editor is not recommended. An XML file is provided to allow changing these durations without needing to unseal the cartridge. See "Changing Durations for Order Components" for more information on changing durations.

Support for Processing States

Processing states were added in OSM 7.3 at both the order item level and the order component order item level. The use of processing states provides better visibility into fulfillment progress, including warnings and errors that occur during the processing of order items. The Order-to-Activate 2.1.1 cartridges incorporate this new OSM feature.

Changes from Order-to-Activate 2.1.0.1 Cartridges to Version 2.1.0.2

This section provides a high-level description of the changes between Order-to-Activate 2.1.0.1 cartridges and Order-to-Activate 2.1.0.2 cartridges.

Changes to Fulfillment Function Extension Points

This section describes the changes to the fulfillment function extension points for the Order-to-Activate 2.1.0.2 cartridges.

New Extension Points

The following extension points were added in version 2.1.0.2:

- CREATE-EBM-CUSTOM
- CREATE-EBM-ALL-ORDERITEMS
- CREATE-EBM-ORDERITEM
- CREATE-EBM-ORDERITEM-CUSTOM
- CREATE-EBM-PRIORORDERITEM
- CREATE-EBM-PRIORORDERITEM-CUSTOM

Extension Points Added to the Billing Components

XQuery files for the following extension points have been added in the OracleComms_OSM_O2A_COM_Billing cartridge for the SyncCustomerFunction, InitiateBillingFunction, and FulfillBillingFunction billing functions:

- CREATE-EBM-ALL-ORDERITEMS
- CREATE-EBM-CUSTOM
- CREATE-EBM-ORDERITEM-CUSTOM
- CREATE-EBM-ORDERITEM (execution mode: *do*)

- CREATE-EBM-ORDERITEM (execution mode: *redo*)
- CREATE-EBM-ORDERITEM (execution mode: *undo*)
- CREATE-EBM-PRIORORDERITEM-CUSTOM
- CREATE-EBM-PRIORORDERITEM
- COMPONENT-RESPONSE-UPDATE

For more information, see "OracleComms_OSM_O2A_COM_Billing."

Changes to Action Code Mappings

Some action code mappings for solutions using the Calculate Service Order solution option were changed to support the Oracle Communications Rapid Offer Design and Order Delivery (RODOD) and Oracle Communications Rapid Service Design and Order Delivery (RSDOD) solutions. The changes are summarized in the following table:

Table 7–1 Changes to Action Code Mappings for Order-to-Activate 2.1.0.2

Upstream Action Code	Action Code Prior to Version 2.1.0.2	Action Code in Version 2.1.0.2 and later
Move-Delete	Move-Delete	[Action is no longer mapped]
Move-Add	Move-Add	Move
Update	Update	Change

New XML-type Parameter Added to Contain Custom Order Item Properties

A new XML-type parameter is available in the COM_Sales_OrderFulfillment order template. It allows you to add custom order item properties to your order template without unsealing any cartridges. For more information, see "Adding Custom Order Item Properties."

Changes from Order-to-Activate 2.1.0 Cartridges to Version 2.1.0.1

This section provides a high-level description of the changes between Order-to-Activate 2.1.0 cartridges and Order-to-Activate 2.1.0.1 cartridges.

New Silent Installation Option

A new option to install and deploy the Order-to-Activate cartridges in an OSM environment without running the interactive installers has been added. See "Performing a Silent Installation of the Order-to-Activate Cartridges" for more information.

Changes from Order-to-Activate 2.0.1 Cartridges to Version 2.1.0

This section provides a high-level description of the changes between Order-to-Activate 2.0.1 cartridges and Order-to-Activate 2.1.0 cartridges.

Support for Calculate Service Order

The order transformation manager was introduced in OSM 7.2.4 to provide a mechanism to transform order items from transform incoming order items into different order items. The Order-to-Activate 2.1.0 cartridges provide the option to use this feature.

Two Solution Options: With and Without Calculate Service Order

The Order-to-Activate 2.1.0 cartridges provide two distinct solution options: the calculate service order solution option and the solution option without calculate service order. Although most new Order-to-Activate users will want the increased functionality of the calculate service order solution option, having the solution option without calculate service order enables existing customers to access the new functionality of Order-to-Activate 2.1.0 without using the order transformation manager if desired.

> **Note:** Oracle Communications Order and Service Management Order Transformation Manager, available in OSM 7.2.4 and later, is a pre-requisite for using the calculate service order functionality in the Order-to-Activate cartridges.

The same Order-to-Activate cartridge installer contains both the cartridges for the calculate service order solution option and the solution option without calculate service order. At a high level, the two options are:

- The calculate service order solution option incudes:
  - Central order management: This includes cartridges with support for the calculate service order implementation of the OSM order transformation manager. It also includes enhancements like large order support and sharing groups.
  - Service order management: This includes new service order management cartridges designed to work with the central order management cartridges that use calculate service order. For more information, see "New Service Order Management Cartridges for the Calculate Service Order Solution Option."
- The solution option without calculate service order includes:
  - Central order management: This includes cartridges that do not have support for calculate service order, but do include other enhancements like large order support and sharing groups
  - Service order management: These cartridges are functionally the same as the service order management cartridges in the Order-to-Activate 2.0.1.

For more information about the solution options, see "Order-to-Activate Cartridge Solution Options."

The Calculate Service Order Solution Option

The calculate service order solution option uses the OSM order transformation manager feature. It is driven by a transformation sequence that is configurable with any number of transformation stages. Each sequence uses original order items as input, executes simple transformation logic for each stage, and creates transformed order items as output.

Productized Order-to-Activate cartridges are enhanced to support transformation of customer order items to service order items using the order transformation manager. The function that transforms customer order lines to service order lines is referred to as calculate service order. Calculate service order is a stable and domain-agnostic function that requires few changes when new products or services are introduced, which reduces the time to market for new product or service introduction. In the Order-to-Activate 2.1.0 cartridges, the calculate service order function is added in the central order management layer.

New Service Order Management Cartridges for the Calculate Service Order Solution Option

New service order management cartridges are added to work with the central order management calculate service order functionality. Domain-agnostic service order management cartridges contain features to support different types of orders such as new orders, revision orders, and change orders. It also has a framework to support fallout and point-of-no-return processing. It has emulators for inventory and technical order management (typically used for activation), so that service order management cartridges can be tested independently of external systems.

For more information about the service order management cartridges for the calculate service order option, see "Service Order Management Cartridges for the Calculate Service Order Solution Option."

Inclusion of Conceptual Model Projects

Conceptual model projects were introduced in Oracle Communications Design Studio 7.2.4 to help you define the relationships between your commercial products, the services that they represent, and the resources that are required to implement the services. Both of the solution options include conceptual model projects.

For more information about the conceptual model, see *Design Studio Concepts*. For more information about the conceptual model projects included with the Order-to-Activate cartridges, see "Conceptual Model Projects."

Large Order Support

OSM 7.2.4.1 introduced functionality to support the processing of orders containing thousands of lines. The Order-to-Activate 2.1.0 cartridges have incorporated this functionality in both of the solution options (with and without calculate service order).

Support for Sharing Groups

The Sharing Groups feature allows discounts, resources and charges like free minutes to be shared across multiple accounts. For example, if a group owner is sharing free minutes, a member is charged for usage, and then discount credits are applied to the member’s account and free minutes are deducted from the group owner.

This feature is introduced in the Oracle Communications Rapid Offer Design and Order Delivery (RODOD) solution, and the central order management cartridges for both of the solution options (with and without calculate service order) have been enhanced to support this feature. For more information about this feature, see the information about promotion groups in *Oracle Application Integration Architecture Oracle Communications Order to Cash Integration Pack Implementation Guide for Siebel CRM, Oracle Communications Order and Service Management, and Oracle Communications Billing and Revenue Management*, Release 11.4.

Changes from Order-to-Activate 7.2 Cartridges to Version 2.0.1

This section provides a high-level description of the changes between Order-to-Activate 7.2 cartridges and Order-to-Activate 2.0.1 cartridges.

Release Number Changes and Packaging Changes

This release of the Order-to-Activate cartridges contains changes to the way releases are numbered and changes to the way the cartridges are packaged.

Through the OSM 7.2 release, the Order-to-Activate cartridges were released at the same time as the OSM software, and the release numbers for OSM and Order-to-Activate were the same. Now however, Oracle has decided to separate the OSM and Order-to-Activate releases. The Order-to-Activate releases are now being aligned toward the Oracle Application Integration Architecture (Oracle AIA) releases. Because of these changes, the Order-to-Activate cartridges are being given their own release numbers. Order-to-Activate 2.0.1 is the first in the new version number series. Release numbers for the older versions of the Order-to-Activate cartridges are not being updated.

Order-to-Activate cartridges are also separate from the OSM software on the Oracle Software Delivery website, and patches for the Order-to-Activate cartridges will be released separately from OSM patches. For more information, see *Cartridges for Oracle Application Integration Architecture Release Notes, Release 2.0.1*.

Support for Multiple Price Lists

Previously, the productized integration supported only one default price list, so price list information was not included on the order. In Order-to-Activate 2.0.1, the price list has been added to the order so that multiple price lists can be supported.

Price list information is passed from Oracle AIA to the Order-to-Activate cartridges as part of the ProcessSalesOrderFulfillmentEBM message. the Order-to-Activate cartridges then populate the order item into the order template. When interacting with the billing system, the Order-to-Activate cartridges generate a ProcessFulfillmentOrderBillingEBM, which includes the price list information base on the Oracle AIA EBM schema.

The price list information is populated into the following structure in OSM:

```
/ControlData/Functions/FunctionName/orderItemRef/orderItem/BaseLineItemData/SalesO
rderSchedule/PriceListReference
```

Support for Importing Product Classes Directly from Oracle Product Hub

It is possible to query product classes and transaction attributes into Design Studio directly from the Oracle Product Hub. Design Studio users use the existing Oracle AIA interface QueryProductClassAndAttributesSCECommsReqABCSImpl to import product classes from both Siebel Customer Relationship Management (Siebel CRM) and the Product Hub. When product classes are queried using this interface, the interface API checks for Product Hub implementation in the Oracle Communications Order to Cash implementation, and if it is there, the product classes are imported to Design Studio from Product Hub. If Product Hub is not present in the Order to Cash implementation, the product classes are imported into Design Studio from Siebel CRM.

Changes from Order-to-Activate 7.0.3 Cartridges to Version 7.2

This section provides a high-level description of the changes between Order-to-Activate 7.0.3 cartridges and Order-to-Activate 7.2 cartridges.

Cartridge Re-Factoring Overview

In version 7.2, the Order-to-Activate cartridges have been reorganized to make use of cartridge extensibility. The changes include:

- The cartridges have been renamed with the prefix **OracleComms_OSM_O2A_**.
- Composite cartridges have been introduced.
- The base cartridges for central order management and service order management have been re-factored into multiple cartridges per fulfillment system.
- Cartridges that ordinarily should not be modified have been sealed. Cartridges that can be modified are not sealed and have the suffix **_Sample**.
- Fulfillment states have been implemented.
- The order template for a function is now constructed from **OracleComms_OSM_CommonDataDictionary** and the local data dictionary **OracleComms_OSM_O2A_COM_***Function* according to general recommendations for working with the common data dictionary.
- Composite cartridge views have been created to add task data to **COM_SalesOrderFulfillment_CreationTask**, **COM_SalesOrder_StateChangeView**, and **COM_SalesOrder_AggregatedOLMView**.
- The order component **GetCommunicationsServiceConfigurationDetails** has been removed from service order management.

Cartridge Mapping Between Order-to-Activate 7.0.3 and Order-to-Activate 7.2

Table 7–2 shows the functional mapping between the Order-to-Activate 7.0.3 cartridges and the Order-to-Activate 7.2 cartridges. See "Cartridge Overview" for descriptions of the Order-to-Activate 7.2 cartridges.

Table 7–2 7.0.3-to-7.2 Order-to-Activate Cartridge Mapping

Order-to-Activate 7.0.3 Cartridge	Order-to-Activate 7.2 Cartridge
[No equivalent]	OracleComms_OSM_CommonDataDictionary
OracleCgbuOsmAIAInstallation	OracleComms_OSM_O2A_Install
OracleCgbuAIAComponentsDataDictionaryPIP	OracleComms_OSM_O2A_AIAEBMDataDictionary
OracleCgbuCommonDataDictionaryPIP	OracleComms_OSM_O2A_CommonUtility
OracleCommSystemAdminOrders	OracleComms_OSM_O2A_SystemAdmin
OracleCgbuControlMap	OracleComms_OSM_O2A_ControlMap
OracleCgbuSIFalloutPIP	[Merged into OracleComms_OSM_O2A_COM_Base]
CommunicationsSalesOrderFulfillmentPIP	OracleComms_OSM_O2A_COM_Base OracleComms_OSM_O2A_COM_SalesOrderFulfillment OracleComms_OSM_O2A_COM_Shipping_Sample OracleComms_OSM_O2A_COM_Billing OracleComms_OSM_O2A_COM_Provisioning OracleComms_OSM_O2A_COM_Install_Sample
OracleCgbuProvisioningFallout	[Merged into OracleComms_OSM_O2A_SOM_Base]

Table 7–2 (Cont.) 7.0.3-to-7.2 Order-to-Activate Cartridge Mapping

Order-to-Activate 7.0.3 Cartridge	Order-to-Activate 7.2 Cartridge
CommunicationsProvisioningOrderFulfillmentPIP	OracleComms_OSM_O2A_SOM_Base OracleComms_OSM_O2A_SOM_Provisioning OracleComms_OSM_O2A_SomBBVoIPFulfillmentPattern_Sample
OracleCgbuCommunicationsORPFalloutPIP	OracleComms_OSM_O2A_RecognitionFallout
[Drawn from various base cartridges]	OracleComms_OSM_O2A_COMSOM_Recognition_Sample
[Drawn from various base cartridges]	OracleComms_OSM_O2A_COM_Recognition_Sample
[Drawn from various base cartridges]	OracleComms_OSM_O2A_SOM_Recognition_Sample
OracleCgbuDoublePlayProductMap	OracleComms_OSM_O2A_FulfillmentPatternMap_Sample
DoublePlayProductSpecificationNile	OracleComms_OSM_O2A_BBVoIPFulfillmentPatternNileFlow_Sample
DoublePlayProductSpecificationDanube	OracleComms_OSM_O2A_BBVoIPFulfillmentPatternDanubeFlow_Sample
OracleCgbuDoublePlayProductSpecNileTdDcn	OracleComms_OSM_O2A_BBVoIPFulfillmentPatternNileFlowDcn_Sample
TypicalSalesOrderFulfillment	OracleComms_OSM_O2A_TypicalTopology_Sample
SimpleSalesOrderFulfillment	OracleComms_OSM_O2A_SimpleTopology_Sample
OracleCgbuTypicalSalesOrderFulfillment	[Cartridge has been removed, but functionality is duplicated in OracleComms_OSM_O2A_ControlMap]
BroadbandServicesProvisioning	OracleComms_OSM_O2A_SomProvisionBroadband_Sample
VoIPServiceProvisioning	OracleComms_OSM_O2A_SomProvisionVoIP_Sample
SalesOrderSubmission	OracleComms_OSM_O2A_SalesOrders_Sample