## Contents

**Preface**

- Audience ................................................................................................................................................ xi
- Downloading Oracle Communications Documentation ................................................................. xi
- Documentation Accessibility .............................................................................................................. xi

**1 Order and Service Management Overview**

- About OSM ........................................................................................................................................ 1-1
- How OSM Fulfills an Order ................................................................................................................. 1-2
- About Orders ...................................................................................................................................... 1-4
- About Central Order Management and Service Order Management ............................................. 1-4
- About Managing Orders ..................................................................................................................... 1-9
- About XPath and XQuery .................................................................................................................. 1-9

**2 About Orders**

- About Orders ....................................................................................................................................... 2-1
  - About OSM Orders .......................................................................................................................... 2-1
  - About the Actions that OSM Can Perform ...................................................................................... 2-2
  - About Modeling Order Specifications ........................................................................................... 2-3
  - What an Order Contains .................................................................................................................. 2-4
  - Re-Using an Order Specification .................................................................................................... 2-4
- About Modeling Order Data .............................................................................................................. 2-4
  - About the Order Template .............................................................................................................. 2-5
    - About Order Template Control Data .......................................................................................... 2-6
    - About Status Data in the Order Template .................................................................................. 2-7
  - About the Data Dictionary ............................................................................................................... 2-9
    - About Importing Incoming Customer Order Data into the Data Dictionary ............................ 2-9
- About the Creation Task .................................................................................................................... 2-10
- About the Default Process ................................................................................................................. 2-11
- About Specifying the Order Priority ................................................................................................. 2-14
- About Order Rules ............................................................................................................................. 2-15
- About Setting Permissions for Orders ............................................................................................... 2-16
- Specifying Which Data to Display in the OSM Web Clients ............................................................ 2-16
  - About Query Tasks for OSM Clients ............................................................................................... 2-17
- Specifying Which Data to Provide to Automation Plug-ins ............................................................. 2-18
  - About Query Tasks for Order Automation Plug-ins .................................................................... 2-18
3 About Receiving and Creating OSM Orders

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>About Receiving and Creating Orders in OSM</td>
<td>3-1</td>
</tr>
<tr>
<td>How Incoming Customer Orders are Structured</td>
<td>3-2</td>
</tr>
<tr>
<td>Understanding Order Recognition</td>
<td>3-4</td>
</tr>
<tr>
<td>About Recognition Rule Relevancy</td>
<td>3-4</td>
</tr>
<tr>
<td>Recognition Rule Errors</td>
<td>3-4</td>
</tr>
<tr>
<td>Creating a Catch-All Recognition Rule</td>
<td>3-4</td>
</tr>
<tr>
<td>Understanding Order Validation</td>
<td>3-5</td>
</tr>
<tr>
<td>Understanding Order Transformation</td>
<td>3-5</td>
</tr>
<tr>
<td>Adding the Input Message to the Order Template</td>
<td>3-6</td>
</tr>
<tr>
<td>Modeling Recognition Rules</td>
<td>3-6</td>
</tr>
</tbody>
</table>

4 Understanding Orchestration

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>About the Conceptual Model and OSM</td>
<td>4-1</td>
</tr>
<tr>
<td>Overview of Orchestration</td>
<td>4-2</td>
</tr>
<tr>
<td>About Orchestration Plans</td>
<td>4-10</td>
</tr>
<tr>
<td>About Order Items</td>
<td>4-13</td>
</tr>
<tr>
<td>About Creating Order Items from Customer Order Line Item Node-sets</td>
<td>4-14</td>
</tr>
<tr>
<td>About Order Item Properties</td>
<td>4-15</td>
</tr>
<tr>
<td>About Order Item Hierarchy</td>
<td>4-19</td>
</tr>
<tr>
<td>About Using the Distributed Order Template</td>
<td>4-20</td>
</tr>
<tr>
<td>About Mapping Order Items to Fulfillment Patterns</td>
<td>4-20</td>
</tr>
<tr>
<td>About Mapping Product Specifications</td>
<td>4-20</td>
</tr>
<tr>
<td>About Order Item Control Data</td>
<td>4-21</td>
</tr>
<tr>
<td>About Decomposition</td>
<td>4-22</td>
</tr>
<tr>
<td>About Component IDs and Component Keys</td>
<td>4-23</td>
</tr>
<tr>
<td>About the Decomposition of Order Items to Function Order Components</td>
<td>4-24</td>
</tr>
<tr>
<td>About Assigning Order Items to Fulfillment Pattern Function Components</td>
<td>4-24</td>
</tr>
<tr>
<td>About the Function Components Stage</td>
<td>4-25</td>
</tr>
<tr>
<td>About Order Component Control Data</td>
<td>4-26</td>
</tr>
<tr>
<td>About Fulfillment Pattern Conditions for Including Order Items</td>
<td>4-26</td>
</tr>
<tr>
<td>Summary of Order Item to Function Components Decomposition</td>
<td>4-27</td>
</tr>
<tr>
<td>About the Decomposition of Function to Target System Components</td>
<td>4-27</td>
</tr>
<tr>
<td>About Decomposition Rules from Function Components to Target Systems</td>
<td>4-27</td>
</tr>
<tr>
<td>About Decomposition Rule Conditions for Choosing a Target System</td>
<td>4-28</td>
</tr>
<tr>
<td>About the Target Systems Stage</td>
<td>4-29</td>
</tr>
<tr>
<td>Summary of Configuring Target System Components Decomposition</td>
<td>4-29</td>
</tr>
<tr>
<td>About the Decomposition of Target System to Granularity Components</td>
<td>4-30</td>
</tr>
<tr>
<td>About Decomposition Rules from Target System to Granularity Components</td>
<td>4-30</td>
</tr>
<tr>
<td>About Customized Component IDs to Separate Bundled Components</td>
<td>4-30</td>
</tr>
<tr>
<td>About the Granularity Components Stage</td>
<td>4-31</td>
</tr>
<tr>
<td>Summary of Configuring Granularity Components Decomposition</td>
<td>4-31</td>
</tr>
<tr>
<td>About Associated Order Items</td>
<td>4-31</td>
</tr>
<tr>
<td>About Modeling Fulfillment Modes</td>
<td>4-33</td>
</tr>
<tr>
<td>About Dependencies</td>
<td>4-35</td>
</tr>
<tr>
<td>About Intra-Order Dependencies</td>
<td>4-37</td>
</tr>
<tr>
<td>Modeling an Order Item Dependency</td>
<td>4-37</td>
</tr>
</tbody>
</table>
5 About Tasks and Processes

About Tasks and Processes ......................................................................................................... 5-1
About Tasks .................................................................................................................................. 5-2
   About Task Data ...................................................................................................................... 5-4
      Generating Multiple Task Instances from a Multi-Instance Field ........................................ 5-5
   About Extending Tasks ........................................................................................................... 5-6
   About Task States .................................................................................................................... 5-6
      Using Task States to Manage Orchestration Dependencies ................................................ 5-6
   About Task Statuses ................................................................................................................ 5-7
      Using Task Statuses and Statuses to Trigger Event Notifications ....................................... 5-9
      Task Statuses and Constraint Behavior Violation Severity Levels ...................................... 5-9
      Using Task Statuses to Control Process Transitions ........................................................... 5-9
   Specifying the Expected Task Duration .................................................................................. 5-10
   Specifying the Task Priority ................................................................................................... 5-10
   Managing Tasks During Amendment Processing and Task Compensation .......................... 5-11
   Controlling Who Can Process a Task ...................................................................................... 5-12
   Specifying How to Assign Tasks to OSM Users .................................................................... 5-12
      Deploying a Custom Task Algorithm using the OSM Cartridge Management Tool ........ 5-12
   About Automated Tasks ......................................................................................................... 5-15
      Understanding Automation Plug-ins .................................................................................. 5-16
   Understanding Activation Tasks .............................................................................................. 5-17
      About Service Action Request Mapping ............................................................................ 5-18
      About Service Action Response Mapping ......................................................................... 5-18
      About State and Status Transition Mapping ....................................................................... 5-19
      About Activation Tasks and Amendment Processing ......................................................... 5-20

Understanding Processes ........................................................................................................ 5-20

   About Workflow Processes and Workstream Processes ..................................................... 5-21
   About Modeling Processes ..................................................................................................... 5-22
   About Process Properties ....................................................................................................... 5-22
   About Process Activities ......................................................................................................... 5-23
   Using the System Date in Delays ......................................................................................... 5-24
   About Flows ............................................................................................................................ 5-25
   About Subprocesses ............................................................................................................... 5-26
   Understanding Parallel Process Flows ................................................................................... 5-28
11 About Notifications

About Notifications ........................................................................................................ 11-1
  About Notification Priority ......................................................................................... 11-1
  About Sending Notifications in Email ......................................................................... 11-2
  About Using Order Rules in Notifications .................................................................. 11-2
  About Using Notifications to Run Automation Plug-Ins ............................................... 11-3
  About Configuring Notifications ................................................................................ 11-4
  Strategies for Using Notifications ............................................................................... 11-4

About Jeopardy Notifications ....................................................................................... 11-5
  About Jeopardy Notification Conditions ..................................................................... 11-6
    Specifying Jeopardy Notification Conditions for an Order ........................................ 11-6
    Specifying Jeopardy Notification Conditions for a Task ........................................... 11-6
  Example of a Jeopardy Notification ............................................................................ 11-7
## About Event Notifications
- About Using Task Transitions to Trigger Event Notifications
- About Using Task States and Rules to Trigger Event Notifications
- About Using Task States to Trigger Automated Event Notifications
- About Using Order Milestones to Trigger Event Notifications
- About Using Order Data Changes to Trigger Notifications

## Summary of Notification Functionality

---

## 12 Order Fallout Management

### Understanding Order Fallout
- Managing Fallout Generated from a Failure in a Downstream System
- Managing Fallout Generated from a Failure in a Network or System Resource
- Managing Fallout Generated from Failure When an Order Is Created in OSM
- Managing Fallout Generated from Failure in Run-Time OSM Execution
- About the Order States Used for Fallout Management
- Example of Discovering Fallout

### About Configuring Fallout
- About Configuring Fallout Compensation in Tasks
- About Configuring Fallout Processes and Tasks

### Managing Fallout in the OSM Web Clients
- Managing Fallout in the OSM Order Management Web Client
- Managing Fallout in the Task Web Client

---

## A XQuery Examples

### General XQuery Information
- About Creating XQuery Expressions with Design Studio
- OSM XQuery Functions
- Referencing Items from a Distributed Order Template in XQuery Expressions

### Order Recognition Rule XQuery Expressions
- About Recognition Rule XQuery Expressions
- About Validation Rule XQuery Expressions
- About Order Priority XQuery Expressions
- About Order Reference XQuery Expressions
- About Order Data Rule XQuery Expressions

### Decomposition XQuery Expressions
- About Orchestration Sequence XQuery Expressions
- About Order Sequence Order Item Selector XQuery Expressions
- About Order Sequence Fulfillment Mode XQuery Expressions
- About Order Item Specification XQuery Expressions
- About Order Item Specification Order Item Property XQuery Expressions
- About XQuery Expressions for Mapping Product Specifications and Fulfillment Patterns
- About Order Item Specification Order Item Hierarchy XQuery Expressions
- About Order Item Specification Condition XQuery Expressions
- About Fulfillment Pattern Order Component XQuery Expressions
- About Fulfillment Pattern Order Component Condition XQuery Expressions
Preface

This guide provides conceptual information about Oracle Communications Order and Service Management (OSM).

Audience

This guide is intended for:

- Business domain experts who make decisions about the order fulfillment process.
- Order management personnel who need to know how OSM works and how orders are processed.
- Developers who extend OSM to interface with external systems.

Downloading Oracle Communications Documentation

OSM documentation and additional Oracle documentation (such as database and WebLogic Server documentation) is available from the Oracle Technology Network Web site:

http://docs.oracle.com

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at


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.
This chapter provides an overview of Oracle Communications Order and Service Management (OSM).

About OSM

OSM coordinates the order fulfillment functions required to complete a customer order created in a customer relationship management (CRM) system, or other order-source system. As an order management system, OSM receives and recognizes customer orders and coordinates the actions to fulfill the order across provisioning, shipping, inventory, billing, and other fulfillment systems. As such, OSM occupies a central place in your order management solution.

The basic steps for order fulfillment are:

1. The CRM system captures the order data; for example, a customer’s order for a telephone service.

   **Note:** OSM is not an order capture system. It does not collect order information directly from customers.

2. The CRM system sends the customer order to OSM in a Web services operation. A customer order is typically based on a product that is sold to a customer, which might require interactions with multiple systems to complete. For example, a customer might purchase a product that combines a telephone service and a DSL service. OSM orchestrates all of the actions that need to be performed by external systems; for example, billing, shipping, and activation systems. (See "How OSM Fulfills an Order" for a detailed explanation of order fulfillment.)

3. As OSM fulfills the order, OSM informs the CRM system of the status of the order. OSM returns the data necessary to complete the order; for example, the customer’s new telephone number.

4. After all of the fulfillment actions are completed, OSM communicates the status to the CRM system and closes the order in OSM.

While orders are being fulfilled, you can use the OSM Web clients to monitor and manage the orders as they are fulfilled. You can automate many of the tasks needed to complete an order, or you can use the Web clients to manually complete tasks.

OSM fulfills orders specifically to support your service offerings. Before you can use OSM to fulfill your orders, you need to use Oracle Communications Design Studio to model how your orders need to be fulfilled. For example, if you sell a DSL service, you
model your DSL order to include the data necessary to activate the DSL service on the network.

To implement and use OSM, you follow this process:

1. Define your business requirements; for example, the products and services you offer.
2. Plan how to implement the fulfillment requirements for those products and services. For example:
   - Which systems (activation, inventory, billing) does OSM need to communicate with?
   - What data is needed to activate a service?
   - Which tasks need to be performed manually, and which can be performed automatically?
   - How are changes to an order handled?
3. Model the orders in Design Studio and test the order execution.
4. Implement the order models in your production system.
5. Monitor and manage in-flight orders by using the OSM Web clients. An in-flight order is an order that is currently being processed by OSM.
6. As your business changes, redefine, test, and implement changes to how orders are fulfilled.

How OSM Fulfills an Order

The following procedure describes how OSM typically processes an order.

1. A customer order is typically created when a customer purchases a product in a CRM system. Before submitting the order to OSM, the CRM system usually performs validations, such as validating customer information from its customer database. For some orders, the order may require technical qualification, such as validating that the network has enough capacity to offer the purchased products.
2. OSM receives the order from the CRM system. The order is submitted to OSM by using the CreateOrder Web services operation.
   
   When received by OSM, the order is called a customer order. The customer order specifies all the data and tasks required to fulfill the order. For example, it might specify that the customer needs a telephone and a telephone number. These requirements are specified in order line items in the customer order.
3. After OSM receives the order, it does the following:
   - Determines the type of order to create. For example, you can model different types of orders based on the services that are being fulfilled. You can also model different order types based on the fulfillment mode of the order. For example, you might fulfill an order in the Qualify fulfillment mode to ensure that the order can be fulfilled, and then fulfill the order in the Delivery fulfillment mode to complete the order and deliver the services. To determine the type of order to create, you create recognition rules.
   - Validates the order data. This step determines if the order syntax is correct. If it is not, OSM sends an error message back to the CRM system.
1. Transforms the order data. This step translates the order into the internal OSM order format. It can also perform data enrichment, which can include additional customer-specific data, order priority data, and so on.

2. Creates the OSM order. This step creates the order in the internal OSM format. See "About Receiving and Creating OSM Orders" for more information.

4. OSM generates an orchestration plan for the order. The orchestration plan specifies the fulfillment actions required to fulfill the order; (for example, add ADSL service). It manages the sequence of those actions and manages dependencies between them.

To create the orchestration plan, OSM reads the requirements defined in each order line item in the customer order and identifies the processes and tasks to fulfill them. For example:

- OSM determines which fulfillment systems need to be involved; for example, a billing system and a service activation system.

- OSM determines which tasks need to be performed, and in which order; for example, initiate payment from the billing system, find a telephone number, and send data to the activation system.

A unique orchestration plan is generated for each order, based on the contents of the order.

An orchestration plan includes the following:

- **Order items.** Order items are individual products, services, and offers that need to be fulfilled as part of an order. Each item includes the action required to implement it: Add, Suspend, Delete, and so on. For example, a new order might add a wireless router.

- **Order components.** Order components are groupings of order items that can be processed together, such as a group of order items that need to be fulfilled by the same fulfillment system and share the same processing granularity. For example, to implement a broadband service, a group of order items to activate the service can be grouped in one component, and a group of order items to ship a modem can be grouped in another component. The process of organizing order items into order components is called **decomposition**.

- **Dependencies.** Dependencies are relationships in which a condition related to one item must be satisfied before the other item can be completed. For example, the order items related to VoIP provisioning are dependent on the order items for DSL provisioning. These dependencies determine the sequence in which order components are processed.

---

**Note:** An order can be created without recognition rules and without an orchestration plan. This is common when the order has a limited set of tasks that do not have dependencies; for example, an order that only manages service activation.

See "Understanding Orchestration" for more information about orchestration plans.

5. OSM runs the orchestration plan. Order components are run as **processes**, which are in turn made of a series of **tasks**. You can use the OSM Web clients to monitor automated tasks and to perform manual tasks.
As the order progresses, OSM communicates with the originating CRM or order-source system to provide information about the status of the order. OSM can aggregate notifications of task completion events to present a real time, unified view of the order to the originating system and to the OSM Web clients.

OSM manages changes to the order by using revision orders. For example, a customer might order Bronze level DSL service and later change the order to Gold level service. When a revision order is received, OSM analyzes it to determine what data has changed and what compensation must be performed due to the change. See "Managing Changes to Orders" for more information.

If an error occurs during order fulfillment; (for example, if a resource that has been assigned is not available on the network), OSM manages order fallout. You can use both the Order Management Web client and the Task Web client to search for and resolve fallout. OSM can also be configured to return data to CRM systems such as Siebel for the creation of trouble tickets. See "Order Fallout Management" for more information.

6. When all order components for the order are complete, OSM closes the order and sends the status to the originating system.

About Orders

An order describes the products or services that need to be fulfilled. The contents of an order can include:

- Information about the order. For example:
  - The type of order, such as a request for a new service or a change to an existing service.
  - Order creation date.
  - Expected completion date.
- Information about the customer; for example, name and address.
- Information about the services being requested; for example, telephone number, bandwidth, or DSLAM port.
- The order components, order items, processes, tasks, and dependencies that are required to fulfill the order.
- Status information. For example:
  - If the order is still in flight, or if it has completed.
  - State of the tasks that need to be performed.
- Tracking information; for example, remarks, notifications, and order history.

See "About Orders" for more information.

About Central Order Management and Service Order Management

When fulfilling orders, OSM can perform two primary roles:

- Central order management
- Service order management

OSM in the central order management role receives customer orders from one or more order-source systems. OSM creates an order, and manages the fulfillment of the order
across other enterprise systems including billing systems, shipping systems, and service fulfillment systems.

The central order management role is also responsible for receiving status information from the fulfillment systems and providing an aggregated status back to the order-source systems. The central order management role is sometimes called central fulfillment.

OSM in the service order management role is typically a part of a dedicated service fulfillment system, working with inventory and activation systems to fulfill services in one or more network domains. OSM in its service order management role typically receives a service order that manages a limited part of the overall order fulfillment. A service order is typically sent by OSM in its central order management role. OSM service order management can orchestrate and manage the fulfillment of the services and resources for the order. It typically works in conjunction with an inventory system to track and allocate resources (assign-and-design) and an activation system to configure the network devices and applications. The service order management role is sometimes called provisioning or local fulfillment.

All OSM functionality; (for example, orchestration) can be used in both of the roles. However, the order processing performed by OSM in the central order management role typically uses orchestration more, because of the need to manage relationships between multiple systems. Orders sent to a service order management system often do not require an orchestration plan because the tasks in the order can be run as a static process by OSM.

As an example, an order might be processed as follows:

1. OSM in its central order management role receives a customer order for a broadband service. Included in the order are requirements for billing, shipping, and provisioning.
2. OSM generates an orchestration plan, which runs the various fulfillment processes needed to fulfill the order.
3. To provision the order, OSM uses an automated task to create a separate service order, which is sent to another instance of OSM functioning in the service order management role.
4. OSM in its service order management role receives the service order and processes the provisioning task. It sends the status back to the OSM instance running in the central order management role.

Service order management typically handles specific provisioning tasks that do not require orchestration, but you can use orchestration with service order management. Figure 1–1 shows two scenarios. In the first scenario, central order management handles provisioning for a fixed-line service and a DSL service separately, and it is therefore able to send service orders directly to OSM in the service order management role. In the second scenario, the fixed-line service and the DSL service are sent simultaneously to service order management. Service order management uses an orchestration plan to send the provisioning requests to separate fulfillment systems.
To take advantage of the separation between customer-facing configuration and network-facing configuration, a typical OSM system architecture uses multiple instances of OSM in both roles. For example:

- You can run one instance of OSM in the central order management role. This instance receives orders from the CRM system, manages the entire fulfillment of the order, and communication with other systems.
- You can run one or more instances of OSM in the service order management role. Each of these instances can communicate with a specific service fulfillment stack; for example, a DSL provisioning system or a PSTN provisioning system.

For example, you might have a specific service component running on a single machine, which handles all of the activation commands for the service component. You can create an instance of OSM in the service order management role on that machine. This instance would deploy an OSM cartridge configured exclusively for provisioning the service component. OSM in the central order management role would send provisioning orders to that system, and the provisioning orders would return the status.

Figure 1–2 shows a typical service provider environment. This figure shows how central order management communicates with multiple systems, including service order management systems.
Central order management applies to order handling and problem handling in the Customer Relationship Management layer.

Service order management applies to service configuration and activation in the Service Management and Operations layer.

Figure 1–3 shows how the two roles apply to the Business Process Framework. The Business Process Framework (eTOM) model is used in the communications, information, and entertainment industries. The central order management and service order management roles can be differentiated in the Operations section of the Business Process Framework.

- Central order management applies to order handling and problem handling in the Customer Relationship Management layer.
- Service order management applies to service configuration and activation in the Service Management and Operations layer.
By running OSM in two roles, central order management and service order management, you can decouple your customer-facing product configuration from the network-facing service configuration and simplify how you manage and maintain your overall system. For example:

- Changes to the products that you sell to customers can be managed by changing how orders are processed by central order management. For example, you might change how products are organized and presented to customers. That can change how products are bundled in an order but not how the services are provisioned on the network.

- Changes to how you implement your services on the network can be managed by changing how orders are processed by service order management. For example, you might make equipment-level changes to your network that require changes to how OSM works in the service order management role but do not affect OSM in the role of central order management. In this case, how the services are activated does not affect how they are presented to customers.

In general, system management can be more flexible if central order management and service order management run on different systems:

- Central order management systems are sometimes managed by a different team than service order management.
Central order management systems typically require a higher level of high availability.

Product upgrade and patching requirements can be handled separately.

**About Managing Orders**

When you manage orders, you typically perform two basic activities:

- **Complete manual tasks.** Tasks can be run automatically or manually. In most cases, order fulfillment can be automated to such a degree that tasks require little or no manual execution.

  To complete manual tasks, you are assigned tasks in the OSM Task Web client. To help you manage your assigned tasks, you can add comments to the order, attach documents, display the history of the order, and receive notifications that alert you to events that occur in the system and to at-risk orders or tasks.

- **Manage the order progress and status.** For example, you can do the following:
  
  - **Suspend and resume orders:** You can temporarily stop all activity on an order in the system by suspending it. When you resume the order, the suspended tasks are resumed.
  
  - **Cancel and resubmit orders:** When you cancel an order, OSM stops all activity on that order and rolls back the tasks that have been completed. After the order is canceled, you can delete it or resubmit it from the upstream order-source system.
  
  - **Manage fallout:** You can identify errors that occur during order fulfillment, notify the appropriate individuals or systems, and take corrective measures.
  
  - **Delete orders:** To delete orders, you use the `orderPurge` command. See *OSM System Administrator’s Guide* for information. You cannot delete orders by using the OSM Web clients.

    A user who has privileges to create orders manually may delete orders before the order has begun processing. To avoid synchronization issues with orders in upstream systems, orders should not be deleted manually.

You can also run reports to get information about the overall order processing load. You can run the following summary reports:

- Pending Orders
- Order Volume
- Completed Order Statistics
- Completed Tasks Statistics

You can also use the OSM Reporting Interface to generate reports. See *OSM Reporting Interface Guide* for more information.


**About XPath and XQuery**

OSM makes extensive use of the XPath and XQuery languages to find, filter, and transform data. Data sources that can be queried include:

- Incoming customer orders
- Data internal to OSM created from customer order data
- Data from external sources retrieved during order processing

XQuery expressions can be included in Design Studio entities or referenced in separate files.

A typical XQuery expression includes:

- The prolog: The prolog can contain default namespace declarations, namespace declarations, schemas imports, module imports, variable declarations, function declarations, and option declarations.
- The body: The body is one XQuery expression that can contain a sequence of one or more expressions separated by commas such as a (for, let, where, order by, and return) FLWOR expression.

For more information about XQuery, see the W3 Web site:

http://www.w3.org/TR/xquery/
This chapter provides conceptual information about Oracle Communications Order and Service Management (OSM) orders.

Before reading this chapter, read "Order and Service Management Overview" to find out about basic OSM concepts.

### About Orders

An order in OSM contains all the data necessary to fulfill the products and services requested by an incoming customer order.

When a customer order is captured in a CRM or other order-source system, it includes data such as the customer’s name and contact information, customer billing information, the products that the customer is ordering, and the requested date of delivery. A subset of that information is included in the customer order that is sent to OSM; for example, the customer information and the order line items that specify the service actions that need to be performed.

After the order is created in OSM, the order includes the data needed for processing the order, as well as information that specifies how to complete the order; for example, the default process to run and the order life-cycle policy. See "What an Order Contains" for more information.

### About OSM Orders

This section introduces the terminology used in the OSM documentation when describing orders:

- **Customer order**: An order request received by OSM to obtain a product or products, typically generated by the CRM or some other order-source system. OSM converts the customer order to OSM format after which it is referred to as an order. Sometimes called an in-bound order or a sales order.

- **Order**: An order in the OSM format. You model orders by creating order specifications in Design Studio.

- **Service order**: An order received by an OSM instance acting in the service order management role. A service order is sometimes called a **provisioning order**.

- **Revision order**: An order that modifies a previously submitted order that is still being processed. For example, a customer may want to switch to a higher level of service before an order is completed. The system can process revision orders until the original order reaches its point of no return. A revision order is sometimes called a supplemental order.
About Orders

- **Follow-on order**: An order that is submitted to modify a completed order. Follow-on orders are not processed until their order-item dependencies on the in-flight orders allow them to proceed. Follow-on orders are also used for sequencing orders.

Orders that are submitted to OSM typically have a specific purpose that is defined as an **order action**. This information is usually included in the order header to indicate if the order adds, deletes, or moves a service. For example, the following line from an incoming customer order specifies that the order adds services:

```
<im:typeOrder>Add</im:typeOrder>
```

In addition to orders that manage services in different ways, you can create orders for specific order-management purposes. For example:

- An order that is created to manage fallout handling.
- An order that communicates with a single external system to provision and activate a specific service. For example, to manage a certain configuration, you might create two order types:
  - An order that is processed by OSM in the central order management role, which handles all of the fulfillment functions.
  - A service order that is created by OSM in the central order management role and is sent to an instance of OSM acting in the service order management role. This order type would manage the fulfillment requirements specific to the provisioning system.

You typically model a different order type when the structure or order data is different from any existing order type, or when there are specific and different fulfillment requirements.

You can use multiple order specifications to create multiple **order types**. Each order specification that you create defines a different order type. See “About Modeling Order Specifications” for more information. In addition, you can use inheritance to manage common configurations between orders. See “Re-Using an Order Specification” for more information.

### About the Actions that OSM Can Perform

Each order line item in an incoming customer order that OSM receives specifies an action to perform. Order line item actions are typically one of the following:

- Add a product or service.
- Change an existing product or service.
- Delete a product or service.
- Update attributes of a product or service.
- Cancel an existing product or service.
- Move a product or service.
- Suspend or resume a product or service.

An order can contain a mix of actions for different products or services. For example, an existing customer might request to add some new products, change some existing products, and remove other products. These can all be included on the same order. See "About Order Items" for more information.
About Modeling Order Specifications

Each order type uses a different order specification. When you model order specifications, you can define the following:

- **The order data.** The data an order can contain is defined in the order template. (See "About the Order Template" for more information.) The order data is initially populated by the creation task. The creation task is used to create an order instance and define its required data. The creation task is required in all orders. See "About Modeling Order Data" and "About the Creation Task" for more information.

- **The default process** that is run when the order is started. See "About the Default Process" for more information.

- **The order life-cycle policy.** Every order type you create must be associated with an order life-cycle policy. The life-cycle policy defines the states that an order can be in, (such as In Progress or Canceled), the rules governing the transitions between those states, and who is authorized to initiate those transitions. For example, you can specify that an order can be transitioned to the Suspended state only when it is in the In Progress state, and only by OSM users of a designated role. See "About OSM Order Life-Cycle Management" for more information.

- **The order priority range.** An order priority value is used by OSM at run time to determine which orders should be given more processing resources when the system is under maximum load. See "About Specifying the Order Priority" for more information.

- **Order rules.** The rules in an order control how various actions take place; for example, when to trigger a jeopardy notification and how delays in the order process should be handled. See "About Order Rules" for more information.

- **Order fallout definitions.** Order fallout definitions enable you to identify specific order data that can cause fallout, and to use order change management to compensate for the error and proceed with processing the order.

  For example, it might be common for a task that activates a port to return an error that the port is already in use. The fallout definition can identify the port ID as the data that needs correcting. This allows OSM to undo the resource assignment task in the inventory system, so the task can be redone and the port ID corrected. The order can then resume processing with the corrected data.

  See "Order Fallout Management" for more information.

- **Order-based behaviors.** You can use behaviors to manipulate data and to control how data is displayed in the Task Web client. For example, you can validate data, specify the contents of a list, calculate values, or create tooltips for fields. See "About Behaviors" for more information.

- **Notifications** to send when specified events occur, when the order is in jeopardy, or when specific order data has changed. Users can display notifications on the Task Web client Notifications page or receive them in email. You can use notifications with automation plug-ins to send messages to other systems or perform other business logic. See "About Notifications" for more information.

- **Order permissions.** Order permissions control the actions that workgroups can perform on orders. See "About Setting Permissions for Orders" for more information.
What an Order Contains

At run time, an order includes the data needed for service fulfillment, as well as information about how to process the order. An order includes the following:

- The **order data**. The data an order can contain is defined in the **order template**. The order template also includes control data. Control data is used by OSM to create the orchestration plan and includes order item data and the structure of the function order components, which represent the first level of decomposition. See "About Modeling Order Data" for more information.

- The **orchestration plan**. The orchestration plan includes the order components, order items, the dependencies between them, and the order in which order items need to be processed.

  You do not specify an orchestration plan when you create an order specification. You define the default process, which, for orchestration orders, is an orchestration process. See "About the Default Process" and "Understanding Orchestration" for more information.

  You can display the orchestration plan, and the order components and order items included in it, in the Order Management Web client.

- The tasks run by the order. You can display information about tasks in the Task Web client. You can also display historical information about the tasks.

Re-Using an Order Specification

When you create a new order model in Oracle Communications Design Studio, you can base the order on an existing order. When you extend an order specification, the extended specification inherits all of the data, tasks, rules, and behaviors of the base specification. You can add new data and behaviors to define unique order specifications and functionality. When you modify a base order specification, the order specifications extended from it are also modified. This means that you can make changes in one place, in the base specification, and those changes apply to the orders that are extended from the base specification.

For example, you might have three order specifications that share a common set of data. You can create a base order that includes configurations common to all three orders. You can then add configurations to each of the three order specifications for the data that is unique to each order specification.

When defining an order specification that is inherited from a base order specification, you cannot edit the inherited order data. For example, you cannot remove or rename data elements inherited from the base order specification. To implement changes to the inherited data, you must edit the data in the base order specification. Design Studio automatically implements those changes among all of the extended order specifications.

About Modeling Order Data

When you model the data in an order, you specify the data that the order must include to fulfill the service. For example, in an order for a telephone service, the order must include telephone number data.

The data elements that you can use in an order are defined in the Design Studio **Data Dictionary**. When you define order data, you can use data elements that already exist in the Data Dictionary data schemas, or you can create new data elements and add them to the Data Dictionary. See "About the Data Dictionary" for more information.
About the Order Template

The data model defined in an order specification is called the order template. An order template is the part of an order specification that defines the order data that OSM uses to process and fulfill an order. For example, the order template defines the data required for order items as well as the data required in an order header.

Figure 2–1 shows an order template.

Figure 2–1  Order Template in an Order Specification

OSM uses the order template when processing the order. For example:

- OSM adds the input message to the order template automatically. See "Adding the Input Message to the Order Template" for more information.
- You can use data in the order template to manage orders; for example, you can create order keys used by amendment processing. See "About Order Keys" for more information.
- You can specify which data in the order template should be considered for amendment processing (data significance). See "About Data Significance" for more information.
- You can assign behaviors to data in the order template. See "About Behaviors" for more information.

The data in the order template defines the data that must be present when the order is created and the data that is generated during order processing. Design Studio generates the order-level order template by aggregating the order template definitions for the order item specifications and order components with any data defined at the order level.

In the data dictionary, you can model the same data element in one or more locations, and assign different type definitions for the elements, such as string or integer, and so on. For example, you might have a data dictionary that contains two instances of a data element called EmployeeID: one defined as a string (defined by the employee’s name and a two-digit number), the other defined as an integer (defined by a 6-digit number). Although you can do this in the data dictionary, you cannot have the same data instance with different type definitions in the order template.
To avoid such data element conflicts, you can rename the first instance of the parameter after you import it into an order template using the refactoring function which allows you to rename an imported parameter at the order template level without changing the data dictionary instance from which it is derived. This creates an alias for the imported data element and you can then import the second instance of the data element without any data conflict errors. See Modeling OSM Processes Help for more information about renaming data elements in the order template.

**About Order Template Control Data**

The order template includes control data. **Control data** is used by OSM to generate the orchestration plan. Control data is used only for orchestration.

There are typically two areas of the order control data:

- ControlData/OrderItem provides the data and structure of order items received in the incoming customer order. **Figure 2–2** shows order item data in the order template control data.

  ![Figure 2–2 Order Item Data in the Order Template](image)

- ControlData/Functions stores the structure of the function order components generated by the first level of decomposition. **Figure 2–3** shows function components represented in the order template. The types of functions (BillingFunction, MarketingFunction, and so on) represent the function-level order components.
You manually model the order control data of order items in Design Studio. Control data for function order components is automatically generated by Design Studio. See the Design Studio Help for information on how control data is modeled and generated.

**About Status Data in the Order Template**

You can configure the order template to hold status data returned from external systems. Figure 2–4 shows an order template structure that holds status data.
You can also store status data in the order item data and in the function data. Figure 2–5 shows a structure for storing status data. In this example:

- The **LineID** data element provides a reference to the order line item in the incoming customer order.
- The **SystemInteraction** data element stores data about status events; for example, a status code, description, and timestamp.

**Figure 2–5  Status Data in Order Item**

![Diagram of Status Data in Order Item]

Figure 2–6 shows a structure for storing status data for functions. In this example:

- The **componentKey** data element provides a reference to the order component instance.
- The **Response** data element stores the message from the external system, as well as the timestamp, description, and status code.

**Figure 2–6  Status Data in Functions**

![Diagram of Status Data in Functions]
About the Data Dictionary

Before OSM can receive an order from an order-source system, you need to create the OSM Data Dictionary.

The Data Dictionary is the repository of data elements used in Design Studio. The Data Dictionary defines data types and structures that can be used within OSM orders. For example, you can define a simple type that represents an IP address or a phone number, or more complex types representing addresses, product attributes and so on.

Data elements in a Data Dictionary are used as building blocks of an OSM order. The data elements within a Data Dictionary project can be referenced by other projects in a workspace.

Design Studio automatically creates a Data Dictionary when you create an OSM cartridge project. You can use this default Data Dictionary or create multiple data schemas to add data elements or structure within the same project.

Each data schema can include a set of data relevant to a particular type of service. For example, a data schema for mobile services could include mobile-related data such as IMSI and MSISDN.

About Importing Incoming Customer Order Data into the Data Dictionary

To import the Data Dictionary for the data received in orders, you import the XSD file for that incoming customer order into OSM. The elements in the XSD file are loaded into the Data Dictionary as OSM data elements. Example 2–1 shows part of an XSD file that could be used for importing customer data.

Example 2–1 Elements in Input Message XSD File

```xml
;element name="order" type="im:OrderType"/>
<element maxOccurs="1" minOccurs="1" name="numSalesOrder" type="string">
</element>
<element maxOccurs="1" minOccurs="1" name="typeOrder">
</element>
```

For each data element, you specify attributes about the data element; for example, the data type and display name. Figure 2–7 shows the configuration for a requestedDeliveryDate data element.
Child XML elements are imported as child data elements. The Path field shows the parent data elements. In this example, the parent data element of requestedDeliveryDate is OrderItem.

In addition to the order data, the Data Dictionary contains information about the data structure of each incoming customer order. For example, it contains information about the hierarchy of sales item lines, which can consist of offers, bundles, products, and so on. This data structure information can be used to manage the data when it is passed between different fulfillment systems.

**About the Creation Task**

When you define an order specification in Design Studio, you must model a creation task. The creation task is a required task. It specifies the required and optional data to be present when the order is created.

The creation task data is used as follows:

- The creation task defines the data that must be present when the order is created.
- When an order is canceled, the order is returned to the creation task.

If an order includes an orchestration plan, the Cancelled state is the final state. The order cannot be resumed. If the order does not have an orchestration plan, the canceled order is returned to the creation task for the order, and can be re-submitted to be processed again.
When performing compensation, OSM compares the creation task data of the base order with the creation task data of the revision order. The creation task differs from other tasks as follows:

- It is not modeled explicitly as part of a process, but is identified in the order specification.
- When an order manager is manually editing an order at the creation task, the order has not been submitted to a process and has had no work completed. The order manager submits the order and at that point the default process is started and the order enters the first task in the process. Prior to submitting the order from the creation task, an OSM user with appropriate privileges may delete the order. Accordingly, the creation task has two task states, submit and delete.

**Tip:** When modeling a creation task, create a manual task, even if the order is intended to be processed automatically. Using manual tasks as creation tasks ensures that task behaviors are supported at run time if you manually create an order. This can be useful for testing purposes.

When an order is created, some data must be populated to the creation task data. To populate the data, you use a transformation rule, defined in a recognition rule. See "Understanding Order Transformation" for more information.

### About the Default Process

For orders that require an orchestration plan for fulfillment, (called orchestration orders), the default process is an orchestration process. For orders that do not use orchestration, the default process is a workflow process or workstream process. See "About Workflow Processes and Workstream Processes" for more information.

When an orchestration order is submitted to OSM, the following occurs:

1. OSM processes the order by running the orchestration process that is specified in the order specification.
2. The orchestration process specifies the orchestration sequence to use, which in turn specifies the first orchestration stage, which starts the orchestration process.
3. When the orchestration plan is complete, OSM runs the executable order components in the order specified in the orchestration plan. The order is based on dependencies between the order components and order items.
4. When the last task in the order completes, the order transitions to the Completed state.

Orchestration orders are typically used by OSM in the central order management role, where multiple fulfillment systems need to be managed and there are dependencies between the fulfillment actions.

**Figure 2–8** shows the process flow for an orchestration order.
See "Understanding Orchestration" for more information.

For orders that do not require an orchestration plan for fulfillment, (called **process-based orders**), the default process is an OSM process, which includes tasks such as Activate_DSLAM. When a process-based order is submitted to OSM for processing, the following occurs:

1. OSM starts the process that is defined as the default process.
2. The default process can start subprocesses that run sequentially or in parallel.
3. After the last task has completed, the order transitions to the Completed state.

**Figure 2–9** shows the process flow for a process-based order.

**Figure 2–9  Process-Based Order**

See "About Tasks and Processes" for more information.

It is common for an order to be fulfilled by both orchestration orders and process-based orders. For example:

1. OSM receives an orchestration order, which generates the orchestration plan, and begins running the executable order components.
2. One of the executable order components runs a process that spawns a separate, process-based order. The order is sent to a separate OSM instance that is configured to interact with a provisioning system.
3. The OSM instance configured for provisioning accepts the order, processes it, and returns the status to the originating order.

**Figure 2–10** shows an orchestration order running a process-based order.
You assign the default process in the order specification. You specify an orchestration process the same way that you specify any other process. Figure 2–11 shows a default process in an order specification.

Figure 2–11  Default Orchestration Process
About Specifying the Order Priority

OSM uses order priority to determine which orders should be given more OSM system resources when the system is under heavy load. This ensures that orders that have higher priority are not starved for resources by lower priority orders.

Order priority does not prevent all lower priority orders from completing until all higher priority orders have completed. OSM is a multi-threaded system and processes as many orders as possible concurrently. You can use follow-on orders to manage inter-order dependencies.

You can specify two values to set the order priority:

- The order priority in the recognition rule that specifies which order specification to use.
- The order priority range in the order specification.

The order priority in the recognition rule defines the priority of the order in relation to other order types. The default order priority is 5. You can enter a number between 0 and 9, inclusive, or you can include an XQuery expression that sets the order priority based on data in the incoming customer order. For example, the XQuery shown in Example 2–2 retrieves the order priority (as a number) from the FulfillmentPriorityCode data element:

```xquery
declare namespace fulfillord="http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/SalesOrder/V2";
```

The order priority is typically set on the order submitted to OSM from the order-source system, and it is mapped to the OSM priority when transforming the order. An order’s priority also can be modified programmatically or manually by using the Task Web client.

---

**Important:** Because OSM is typically one of several systems involved in fulfilling orders, order priority must be supported in all systems and middleware for it to be the most effective.

---

The order priority range specifies the acceptable range of numeric priority (between 0 and 9) that orders of a single type may use. For example, this could allow you to configure a fixed-line order type with a lower range (0 to 4) and a mobile order type with a higher priority range (5 to 9), ensuring that mobile orders are prioritized higher than fixed-line orders.

You create an order priority range by specifying a minimum and maximum priority for the order. OSM rounds priority values up or down to ensure they conform to the order priority range. For example, if you specify a priority range of 5 to 7 and an order is created with a priority of less than 5, the system assumes the intent is to provide the lowest priority allowed for the order, and the priority value of the order is set to 5. Similarly, if a priority higher than 7 is provided for another order of the same type, the system assumes the intent is to provide the highest priority allowed for the order, and the priority value of the order is set to 7.

Table 2–1 shows examples of how the order priority is set by using the order priority from the recognition rule, and the order priority range from the order specification.
You can set the order priority range in the Design Studio Order editor Details tab.

The order priority value is also considered when an order’s tasks are run, so that automated tasks are run according to order priority. This requires that Java Messaging Service (JMS) message priority settings are configured for the JMS queues.

You can change the order priority of an in-flight order by using the Order Management Web client. You can specify permissions for which roles can change the priority. See the discussion of changing order priority in *OSM Order Management Web Client User’s Guide*.

### About Order Rules

Order rules control how various actions take place; for example, when to trigger a jeopardy notification and how delays in the order process should be handled. Rules are used in process flow decisions, conditional transitions, subprocess logic, delay activities, jeopardies, and events.

OSM evaluates order rules by comparing data to data, or data to a fixed value. **Figure 2–12** shows an order rule in the Design Studio Order editor Rules tab. This rule identifies residential customers in a specific city. This is an example of a rule that might be used to send a fallout notification to a regional fallout manager.

### Table 2–1 Order Priority Examples

<table>
<thead>
<tr>
<th>Order Priority Range</th>
<th>Recognition Rule</th>
<th>Recognition Rule</th>
<th>Recognition Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Order Priority 1</td>
<td>Order Priority 5</td>
<td>Order Priority 9</td>
</tr>
<tr>
<td>Order Priority Range 1 - 3</td>
<td>Priority = 1</td>
<td>Priority = 3</td>
<td>Priority = 3</td>
</tr>
<tr>
<td>Order Priority Range 3 - 5</td>
<td>Priority = 3</td>
<td>Priority = 5</td>
<td>Priority = 5</td>
</tr>
<tr>
<td>Order Priority Range 5 - 9</td>
<td>Priority = 5</td>
<td>Priority = 5</td>
<td>Priority = 9</td>
</tr>
</tbody>
</table>

*You can set the order priority range in the Design Studio Order editor Details tab.*

*The order priority value is also considered when an order’s tasks are run, so that automated tasks are run according to order priority. This requires that Java Messaging Service (JMS) message priority settings are configured for the JMS queues.*

*You can change the order priority of an in-flight order by using the Order Management Web client. You can specify permissions for which roles can change the priority. See the discussion of changing order priority in *OSM Order Management Web Client User’s Guide*.*

---

*Figure 2–12 Example of an Order Rule Defined in Design Studio*
About Setting Permissions for Orders

OSM Web client users are assigned roles, which you can use to manage who works on different types of orders, and different types of tasks. When you assign permissions to orders, you define the following for each role:

- You can specify if the OSM users belonging to the role can create the order in the Task Web client.
- You can specify the data that OSM users can see in the Task Web client Query filter for the associated order. To do so, you can define flexible headers in the Design Studio Order Permissions tab. Flexible headers allow specific fields to display in the Task Web client Query filter.

Flexible headers are typically used when there are one or more fields on an order that contain information that is the same for multiple orders and which can be used to query and find related orders. Examples of this are external reference numbers, customer numbers, and telephone numbers. Flexible headers can be used to allow order managers to query these data fields across orders in different cartridges if they have the same mnemonic path in their order templates. The Task Web Client query screen allows you to input search criteria once. It returns all orders that match the flexible header search values.

- You can specify which data OSM users can display in the OSM Web clients. See "About Query Tasks for OSM Clients" for more information.
- You can specify the orders that OSM users of the role can see, based on data in the order. Use the Order editor Permissions Filters subtab to limit the orders a role can view. For example, you specify that OSM users see only orders from a region or for a specific type of service.

Figure 2–13 shows conditions defined in the Design Studio Order editor permissions tab that allows OSM users in the role to see only orders from customers who have the 408 and 510 area codes.

![Figure 2–13 Order Filters Defined in Design Studio](image)

See "About OSM Roles" for more information.

Specifying Which Data to Display in the OSM Web Clients

As an order runs tasks, the data that is available at each task should be the minimum subset of order data necessary for the task to be performed. You can choose the data to display in the OSM Web clients using the following methods:

- Use task data to specify which data to display in the Task Web client for manual tasks.
Use **behaviors** to specify how OSM displays the task data within a manual task; for example, to hide or show task data or to make data read only. See "About Behaviors" for more information.

- Use **query tasks** to specify which data to display in the Order Management Web client **Summary** tab and **Data** tab. Query tasks are manual tasks that specify which data to display in the Task Web client when opening an order from a query result rather than from a task in the worklist. A query task is associated with a role that has permissions to view an order and should be limited to the subset of an order specification’s data that the particular role is allowed to view. See "About Query Tasks for OSM Clients" for more information.

### About Query Tasks for OSM Clients

Order management personnel can display orders in the Task Web client and in the Order Management Web client. You can specify which data is displayed by assigning query tasks to an order. The data that is specified in the query task data is the data that is displayed.

You can select any task as the query task. You can also create special tasks whose only function is to specify which data to display.

*Figure 2–14* shows the **Permissions** tab in the Design Studio Order Editor. The upper screen shows the permissions for the provisioning role, with the provisioning function task as the query task. For the billing role, the billing function task is assigned as the query task.

### Figure 2–14  Roles Assigned to Query Tasks

The Order Management Web client uses two types of views to display orders; a summary view in the **Summary** tab and a detailed view in the **Data** tab. When you model a query task, you can specify which of those views (either or both) to display the query task data in.
You can use multiple tasks as query tasks for an order. When you do so:

- For the summary view, all the data is displayed in the Order Management Web client **Summary** tab.
- For the detailed view, the data from the query tasks appears as options in the Order Management Web client **Data** tab **View** field; each option presents the OSM user with a different view, each containing a specific set of data.

You can use multiple query tasks in the Order Management Web client when using an orchestration cartridge. For process-based cartridges, only the default query task is available in the Order Management Web client. To display the query task in the Task Web client, select the **Default** check box, as shown in Figure 2–14.

In addition to defining the data that can be displayed, you can specify who can see it by using roles. Each role that is associated with an order can be assigned different query tasks. For example, if your order management personnel includes a role for billing specialists, you can create query tasks that show data specific to their activities.

### Specifying Which Data to Provide to Automation Plug-ins

The data that is available for each automation plug-in should be the minimum subset of order data necessary for the plug-in to be performed. You can choose the data to provide to automation plug-ins using the following methods:

- Use the **task data** contained in an automation task to specify which data to provide to an automation plug-in.
- Use **query tasks** to specify which data to provide to an automation plug-in associated to order notification, events, and jeopardies. A query task is a manual task that is associated with a role that has permissions to use some or all order data to run an automation plug-in. See “About Query Tasks for OSM Clients”.

### About Query Tasks for Order Automation Plug-ins

In automated tasks, the data that is available to automation plug-ins associated to automated task is already defined in the **Task Data** tab. However, automation plug-ins used with order notifications, events, and jeopardies do not have immediate access to this task data, and, as a result, must reference a manual task called a query task that defines the task data and behavior data available to the automation plug-in.

You can select any manual task as the query task. You can also create special tasks that are only used as query tasks. Their only function is to specify which data to provide to an automation plug-in.

**Figure 2–14** shows the **Permissions** tab in the Design Studio order editor. The upper screen shows the permissions for the provisioning role, with the provisioning function task as the query task. For the billing role, the billing function task is assigned as the query task.

To associate a query task with an automation plug-in, use the Default check box, as shown in **Figure 2–14**.

**Figure 2–15** shows an event notification with an automation plug-in that uses the **ProvisioningFunctionTask** query task that is defined as the default query task for the provisioning role. This role must be associated to the **Run as OSM user** that runs the automation plug-in as shown in the Properties **Details** tab. For more information about associating roles to OSM users, see the *OSM Administrator Application User’s Guide*.
Figure 2–15  Order Event Notification Automation Query Task
This chapter describes how Oracle Communications Order and Service Management (OSM) processes an order when it is received from an order-source system and how it creates an OSM order.

Before reading this chapter, read "Order and Service Management Overview" for more information.

About Receiving and Creating Orders in OSM

The typical process for receiving orders in OSM using the CreateOrder Web services operation is as follows:

1. The order data is captured in a CRM system; for example, as a Seibel order.
2. The CRM system sends the order to OSM by using the OSM CreateOrder Web services operation.

   The CreateOrder operation receives orders that are in the XML format of the order-source system, which is different from the OSM order format. The CreateOrder operation can recognize external order formats, and it uses a recognition rule to transform the requests to the OSM internal order format before creating the order.

   You can use Java Message Service (JMS) or HTTP or HTTPS to send orders to OSM. Use JMS on production systems, because it provides quality-of-service guarantees not available from HTTP or HTTPS. Use HTTP or HTTPS on development and test systems.

   **Note:** A single OSM instance can receive orders from multiple order-source systems.

3. After OSM has recognized and validated the incoming customer order, the OSM recognition rule calls the CreateOrderBySpecification Web services operation. This operation does the following:
   - Creates the order in OSM
   - Sets the order priority
   - Populates the data in the creation task

   The CreateOrderBySpecification operation receives orders from systems that can provide order requests in the OSM native XML format. The CreateOrderBySpecification operation references an order specification that you
define in Oracle Communications Design Studio, and the order details must conform to that order specification.

The CreateOrderBySpecification Web services operation is typically used for sending an order to an instance of OSM running in the service order management role. In that case, OSM creates the order and begins processing it.

4. If the order is created successfully, and the default order process is an orchestration process, OSM begins generating the orchestration plan. If the default process is a workflow process or a workstream process, OSM begins processing the tasks in the default process.

If OSM is unable to create the order by using the CreateOrderBySpecification operation, the in-bound order is handled in one of two ways:

- If the order type is not valid, a failed order is created with the in-bound order attached.
- If the order type and source are valid, the in-bound order is put on the JMS redelivery queue. OSM attempts to receive the order again, up to the receive limit configured for the queue. When that limit is reached, the failed message is moved to an error queue.

To receive and create orders, you need to do the following:

- Configure your order-source system to output orders in XML format.
- Do the following in Design Studio:
  - Populate the Data Dictionary with the data elements that the order needs. See "About Modeling Order Data" for more information.
  - Create recognition rules to recognize, validate, and transform the data. See "Understanding Order Recognition" for more information.
  - Create order specifications for the types of orders you need to create in OSM. See "About Modeling Order Specifications" for more information.

How Incoming Customer Orders are Structured

When received by OSM, the incoming customer order typically consist of these sections:

- The order header information, which contains information that is applicable to the entire order; for example, the sales order number, the order action (Add, Cancel, Delete, and so forth), and customer account information.
- Order line items, which include the products, services, and offers requested by the customer and the action to be performed on them (Add, Suspend, Delete, Move, and so on).

The order line items include details about the services that the order must fulfill. They can include:
  - The bundles, products (and their product specifications), and services being ordered
  - Information about the services; for example, speed, storage size, and requested service date

Figure 3–1 shows part of a customer order, received from a CRM system.
How Incoming Customer Orders are Structured

Figure 3–1  Example of a Customer Order

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ord:CreateOrder
 xmlns:ord="http://communications/ordermanagement">
<im:order xmlns:im="http://xmlns.oracle.com/ImOrderMessage"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ImOrderMessage ../dataDictionary/ImOrderMessage.xsd">
<im:numSalesOrder>3PlayRevision_100</im:numSalesOrder>
<im:typeOrder>Add</im:typeOrder>
<im:customerAccount>
<im:salesOrderLineGroup>
<tl:OFFER -->
<im:salesOrderLine>
<tl:FIXED BUNDLE - BUNDLE -->
<im:salesOrderLine>
<tl:FIXED SERVICE - SERVICE BUNDLE -->
<im:salesOrderLine>
<tl:FIXED SERVICE CLASS -->
<im:salesOrderLine>
<tl:END FIXED SERVICE CLASS -->
</im:salesOrderLineGroup>
</im:customerAccount>
</im:order>
</ord:CreateOrder>
```

Figure 3–2 shows one of the order line items. Included in this order line item are the requested delivery date and the action to take (Add).

Figure 3–2  Example Order Line Item in a Customer Order

```xml
tl: Speed By Demand - PRODUCT -->
<im:salesOrderLine>
<im:lineld>16</im:lineld>
<im:parentLineReference>
<im:parentLineld>14</im:parentLineld>
<im:hierarchyName>default</im:hierarchyName>
</im:parentLineReference>
<im:rootParentLineld>13</im:rootParentLineld>
<im:promotionalSalesOrderLineReference>1</im:promotionalSalesOrderLineReference>
<im:serviceld></im:serviceld>
<im:requestDeliveryDate>2011-12-31T12:00:00</im:requestDeliveryDate>
</im:salesOrderLine>
<tl: END Speed By Demand - PRODUCT -->
```
The AIA Order-to Activate solution uses three sections, an AIA header, an order header, and order line items.

**Understanding Order Recognition**

Order recognition is the process of determining the type of an incoming customer order so it can be mapped to an order type in OSM. Recognition rules allow OSM to accept any input message structure.

During order recognition, OSM steps through a prioritized list of recognition rules to determine which rule applies to the in-bound order. Each recognition rule is associated with an order specification. The first rule that evaluates to true determines the order specification to use for the incoming customer order and which validations and transformations are required, if any. Rules are evaluated in an order based on a property called *relevancy*, which is defined as part of the recognition rule specification in Design Studio.

Order recognition rules use XQuery expressions to match incoming customer orders. See "About Recognition Rule XQuery Expressions" for information about creating order recognition XQuery expressions.

**About Recognition Rule Relevancy**

Relevancy determines the order in which the OSM server evaluates the order recognition rules at run time. OSM evaluates a rule with a higher relevance before it evaluates a rule with a lower relevance. For example, OSM evaluates a rule with a relevancy value of 2 before it evaluates a rule with a relevancy value of 1. At run time, OSM uses the first order recognition rule that evaluates to true.

**Recognition Rule Errors**

There are two possible errors during order recognition:

- A recognition rule fails to run; for example, because of bad syntax. Evaluation of other rules continues.

- The in-bound order is not recognized. If all recognition rules run and fail to find a match, then no OSM order can be created. This failure generates fallout, which you can view and manage as an order failure in the Order Management Web client.

To avoid this kind of failure, you can create a lowest-relevancy catch-all rule that recognizes any in-bound order and maps it to a default order specification. See "Creating a Catch-All Recognition Rule" for more information.

**Creating a Catch-All Recognition Rule**

An order that fails to be recognized by any recognition rule is rejected by OSM, and an error is returned by the Web services operation to the order-source system. To ensure that all orders are entered into OSM, create a catch-all recognition rule that accepts all incoming customer orders.

To configure this recognition rule:

- Set the relevancy to 0, and set the relevancy for all other recognition rules higher than 0, so they are processed first.

- Include the following recognition rule XQuery:

  ```xml
  fn:true()
  ```
Understanding Order Transformation

About Receiving and Creating OSM Orders

- Select the **Fail Order** check box, and enter a reason. For example:
  
  **No valid recognition rule found.**

  Using this lowest-level recognition rule, an invalid order is recognized and then fails during validation. It then transitions to the Failed state and is kept by OSM.

Understanding Order Validation

When an incoming customer order is recognized, OSM checks the order using a validation rule XQuery defined in the order recognition rule. The validation step ensures that the order is syntactically correct.

For example, a validation rule can determine that:

- All mandatory fields are populated
- Valid characters (numeric or alphanumerical) are used for fields
- The order has a valid status code, such as Open

See "About Validation Rule XQuery Expressions" for information about creating validation rule XQueries.

Understanding Order Transformation

Incoming customer orders might come from multiple order-source systems. Data is often represented differently in different systems; for example, telephone number formats might include varying numbers of digits. You can use transformation rules to normalize data and to make it usable in OSM and in external systems. For example, you might use transformation rules to add information to the customer order data or to transform the generic order items and their attributes into specific attributes of an order required in an external system.

In addition, you use transformation rules to add data and properties to an order when it is created. This includes:

- Creation task data. Creation task data is required for the order to be created. See "About the Creation Task" for information about the creation task. Most data required for the creation task is included in the incoming customer order. To retrieve that data, you use an XQuery transformation rule (see "About Order Data Rule XQuery Expressions").

  **Note:** An internal transformation rule always stores the raw XML input message in an XML data field as part of the order data. However, that data does not populate the fields in the creation task.

  You can use a transformation rule to modify data in the order. For example, you can concatenate the area code and phone number into a single data element.

  You can retrieve data from external systems if it does not exist on the incoming customer order. For example, the incoming customer order might have a customer address, but you need to add the geographic region to the order, which is not in the input data. You can use a Web services operation, or an SQL call to an external system, to look up the region, based on the customer’s address. You can then add the region code to the order.
The order priority. The order priority defines the processing priority of the order in relation to other orders in the system. If you do not define a priority, the OSM server sets the order priority to 5 (of 0 through 9). You can define data-driven XQuery rules to set the priority of incoming customer orders based on one or more property values. See "About Specifying the Order Priority" for more information.

The order reference number. The order reference number is derived from a specific order header field on an OSM order using an XQuery expression (see "About Order Reference XQuery Expressions"). OSM uses an order reference number as an identifier to external systems. Order reference numbers can be used as keys to correlate orders between systems.

The order reference number is an alphanumeric value supplied by the order-source system. It is usually unique, but it does not have to be unique. When OSM creates the order, OSM gives the order an OSM order ID. The original order reference number is stored as well, so the order reference number is associated with the OSM order ID.

You use XQuery expressions to define transformation rules. You can define the XQuery to use to transform the order in the Order Data Rule section of the Design Studio Recognition Rule editor Transformation tab.

At run time, the OSM server always runs all transformation rules, regardless of the failure of any transformation rule. Running all transformation rules ensures that the order is populated with all available data.

If a transformation rule fails, the order is populated with whatever data is available, and the order is placed in a Failed state with reasons corresponding to the type of transformation rule that failed:

- Could not set order priority.
- Could not set order header reference.
- Could not create order data.
- Could not store incoming message. Message stored as attachment.

Adding the Input Message to the Order Template

OSM can add the input message to the order template automatically. To do so, you specify the data structure of the incoming customer order in the order’s recognition rule. You can select from the Data Dictionary a previously imported XML structure, or a data element from a previously imported XML structure. At run time, when a recognition rule evaluates to true, OSM adds this input message to the order template (as an XML data type) to contain the incoming customer order.

Modeling Recognition Rules

When you define recognition rules in Design Studio, you configure the following attributes:

- The target order specification. The first recognition rule that evaluates to true creates the target order type.
- The specific version of an order if you want the recognition rule to recognize an order specification version other than the default version.
- The data structure of the incoming customer order. See "Adding the Input Message to the Order Template" for more information.
■ How to handle order failure. You can use the Fail Order check box to specify that the order fails when a recognition rule fails. If you disable this option, OSM continues to process other recognition rules.

■ The recognition rule relevancy. Each rule has a relevancy to determine the order in which the rule is evaluated. Rules with a higher relevancy are evaluated first: OSM runs a rule with a relevancy of 2 before it evaluates a rule with a relevancy of 1.

You can create the following recognition, validation, and transformation rules:

■ The recognition rule that recognizes the order. See "About Recognition Rule XQuery Expressions" for more information.

■ The validation rules that validate data. See "About Validation Rule XQuery Expressions" for more information.

■ The transformation rule that specifies the order priority. See "About Specifying the Order Priority" for more information.

■ The transformation rule that specifies the order reference number. See "About Order Reference XQuery Expressions" for more information.

■ The transformation rule that populates the creation task data. See "About Order Data Rule XQuery Expressions" for more information.
This chapter describes Oracle Communications Order and Service Management (OSM) orchestration.

Before you read this chapter, read "Order and Service Management Overview" for information about basic OSM concepts.

About the Conceptual Model and OSM

Conceptual models define the relationships between your commercial products, the services they represent, and the resources required to implement the services. They define how commercial products and technical services are related, and they enable you to associate the products that you sell with the technical services and resources that are required to fulfill orders.

Conceptual model items are not built into OSM cartridges or deployed to the OSM server directly, but are included in OSM by the realization process. Realization refers to converting the abstract entities in the conceptual model into actual instances in the OSM configuration. You can use this conceptual model metadata as part of your OSM run-time solution to help define order item to fulfillment pattern mappings, and to give you a representation of what you need to implement in OSM as part of your overall fulfillment solution.

See Design Studio Concepts for more information about conceptual model projects.

Oracle recommends you make full use of all the conceptual model features, though they are not all mandatory. Although you do not need to use all of the conceptual model features, there are some conceptual model entities that are required, depending on the OSM features you are using and the role that OSM is fulfilling in your solution.

If you are using the order transformation manager, see “The Order Transformation Manager and the Conceptual Model” for information about conceptual model entities that are required.

In addition to the conceptual model entities needed for the order transformation manager, some entities are required based on OSM’s role, regardless of whether the order transformation manager is used:

- If OSM is operating in the central order management role, you must define products and fulfillment patterns in the conceptual model.
- If OSM is operating in the service order management role, you must define customer facing services, action codes, and fulfillment patterns.
Overview of Orchestration

A single customer order typically includes multiple order line items that request multiple products and fulfillment actions. To process the order, some order line items need to be fulfilled before others; for example, you cannot configure a call-waiting service until the base telco service is provisioned. There are also multiple external systems that OSM must interact with. OSM uses orchestration to handle all of the fulfillment actions efficiently, taking into consideration all of the dependencies between the actions.

To manage orchestration, OSM creates a unique orchestration plan for each customer order. The orchestration plan specifies the fulfillment functions required to fulfill the order, manages the sequence of those functions, and manages dependencies between them.

To create the orchestration plan, OSM reads the requirements defined in each order line item of the customer order and identifies the processes and tasks to fulfill them. For example:

- OSM determines which fulfillment systems need to be involved; for example, a billing system and a service activation system.
- OSM determines which tasks need to be performed, and in which order; for example, initiate payment from the billing system, find a telephone number, and send data to the activation system.

The orchestration plan includes:

- **Order items.** OSM transforms order line items from the customer order into OSM order items that represent the individual products, services, and offers that need to be fulfilled. Each order item includes the action required to implement it: Add, Suspend, Delete, and so on. For example, a new order might add a wireless router; the order item created in OSM is **Add Wireless Router**.

- **Order components.** Order components are groupings of order items that can be processed together, such as a group of order items that need to be fulfilled by the same fulfillment system and share the same processing granularity. For example, to implement a broadband service, a group of order items to activate the service can be grouped in one component, and a group of order items to ship a modem can be grouped in another component.

- **Dependencies.** Dependencies are relationships in which a condition related to one order item must be satisfied before the other item can be completed. For example, the order items related to VoIP provisioning are dependent on the order items for DSL provisioning. These dependencies determine the sequence in which order components are processed.

There are more entities in an orchestration plan than order items and order components; such as orchestration sequences and orchestration stages, but the main purpose of the orchestration plan is to manage how order items are fulfilled.

You can view order components and their order items in the Order Management Web client. **Figure 4-1** shows a list of order components and their order items:
An orchestration plan is generated for each order, based on the contents of the order. The orchestration plan for an order specifies the following:

- How order items are grouped into order components for processing. For example, some order items in an order for VoIP service need to be fulfilled by billing systems, some by provisioning systems, some by activation systems, and some by a combination of systems.

- The dependencies between the individual actions. For example, the VoIP service must be activated before a VoIP phone is shipped.

In the following example, an orchestration plan is created to fulfill the following order items:

- One fixed phone line and a handset in Ontario
- A fixed phone line and a handset in Quebec

Billing has the following requirements:

- Billing must be performed by different billing systems in the two regions. This requires different target system order components to contain the order items for each region.

- Billing for the services must be processed separately from billing for the handsets. This is because the handset can be billed immediately, whereas the provisioning of the fixed-line service can take longer, which in turn delays the billing for the fixed-line service. To capture revenue for the handset as soon as possible, the order component granularity specifies two order components: an order component for billing the service and an order component for billing the handset.

The following example shows how order components and order items are decomposed to fulfill different services in different regions.

1. OSM first reads the incoming customer order and creates order items from the order line items contained in the order. Figure 4–2 shows that order line items for two fixed services and handsets are derived from the order. There are different regions defined for each service and handset (Ontario and Quebec).
2. OSM begins the orchestration process. The first step is to assign the order items to order components that are based on fulfillment functions. Figure 4–3 shows that the order items are organized into three function order components: Provisioning, Shipping, and Billing. The fixed-line services require provisioning, the handsets require shipping, and all order items require billing, so they are all included in the Billing order component.
3. The Billing order component must be decomposed further. Figure 4–4 shows two levels of decomposition:

- Order items for the Ontario and Quebec regions are decomposed into target system order components. This sends the billing fulfillment process to the correct region, Ontario or Quebec.
- For each region, the fixed-line service must be billed separately from the handset.
The order components specify how the order items are processed by the orchestration plan. They also specify how the order items are displayed in the Order Management Web client. In this example, an order manager can monitor the status of the different billing components to track the order item progress.

In Figure 4–4, the order components serve two primary functions:

- To organize order items during decomposition
- To run OSM processes

Not all order components run OSM processes. Those order components are used only for decomposition. In the example above, order components in the first and second stages of orchestration play that role. The order components that run OSM processes are called **executable order components**. The final stage of decomposition creates those order components.

To model how orchestration plans are generated, you model several OSM entities in Oracle Communications Design Studio.

- Orchestration processes. An orchestration process specifies which orchestration sequence to use for the order.
- Orchestration sequences. The orchestration sequence specifies the fulfillment mode (for example, Deliver or Qualify), the orchestration stages, and selects the customer order line item node-sets that OSM uses in orchestration.
- Order item specification. The order item specification includes the order item properties that are used for decomposition, including how to retrieve order items from order line items. Order item properties define data that is used for decomposition; for example, the fulfillment pattern.
- Order components. Order components specify how to organize order items in the decomposition process.
- Orchestration stages. Orchestration stages specify the order components to assign order items to.

**Figure 4–5** shows a generalized process flow for orchestration.
The following process flow shows how OSM uses the orchestration entities to create orchestration plans.

1. After receiving and validating an incoming customer order, OSM creates the order according to the order specification chosen by the recognition rule. At this point, the following has been accomplished:
   - The order has been populated with the creation task data.
   - OSM has used the order item specification to identify order items from the order line items in the incoming customer order.

2. The order specification includes a default process. For an orchestration order, the order specifies an orchestration process. (If no orchestration is required, you should define a non-orchestration OSM process. See "About Tasks and Processes" for more information.)

3. The orchestration process specifies an orchestration sequence.

4. The orchestration sequence specifies the following:
   - The order item specification to use for the order. The order item specification defines the order item properties that are used for decomposition and for displaying the order item in the Order Management Web client. See "About Order Items" for more information.
   - The order item selector that identifies the customer order line item node-sets to use as order items. See "About Creating Order Items from Customer Order Line Item Node-Sets" for more information.
   - The fulfillment mode that the order requires; for example, Deliver or Cancel. See "About Modeling Fulfillment Modes" for more information.
The orchestration stages that produce the order components. For example, the orchestration stages might be:

- **Produce function order components.** This stage organizes order items into order components based on the fulfillment functions required for each order item. Fulfillment functions are the activities that must be performed to process the item, such as billing, shipping, provisioning, and so on.

- **Produce target system order components.** This stage organizes order items into order components based on the target fulfillment systems required to perform the fulfillment functions. For example, this step might determine that certain items need to be fulfilled by a billing system called BRM_Residential and others by a BRM_Wholesale system.

- **Produce granularity order components.** This stage organizes order items that need to be processed together into order components. For example, you might need to fulfill billing requirements for mobile and fixed services. You can use different order components to process the billing requirements for those services separately.

Figure 4–6 shows how to define orchestration stages in Design Studio.

**Figure 4–6 Orchestration Sequence Specifies the Orchestration Stages**

![Orchestration Sequence: CentralOMOrchestrationSequence](image)

5. Each orchestration stage produces a set of order components. Figure 4–7 shows how an orchestration stage specifies order components.
6. Based on the default orchestration process, and the orchestration sequence and stages that are defined, OSM can start the process of assigning order items to order components. The first step is to find the fulfillment pattern used by each order item.

Each order item belongs to a product specification. A **product specification** is a group of related products that share common attributes. For example, the products Broadband Light, Broadband Medium, and Broadband Ultimate would all belong to the ServiceBroadBand product specification. OSM maps the product specification to a fulfillment pattern.

The **fulfillment pattern** manages the first stage of orchestration. It assigns order items to function order components in the first stage of orchestration. It also specifies the dependencies between the function order components. For example, the fulfillment pattern might specify to process function order components in this order:

a. ProvisioningFunction
b. BillingFunction
c. CollectionsFunction

The fulfillment pattern also specifies the fulfillment mode that the order items can be used for. See "About Mapping Order Items to Fulfillment Patterns" for more information.
Figure 4–8 shows sample order component dependencies. Provisioning must occur before billing, which must occur before marketing, customer updates (SyncCustomer), and collections.

**Figure 4–8 Dependency Relationships for Order Item Dependency**

7. After assigning order items to function components, OSM further decomposes the order into target system order components and granularity order components, following the specifications defined in the orchestration stages. See “About the Decomposition of Function to Target System Components” and “About the Decomposition of Target System to Granularity Components” for more information.

8. While decomposing the order, OSM finds dependencies between order components and generates an orchestration plan. Dependencies determine the order in which order components can be processed. See “About Decomposition” for more information.

9. After generating the orchestration plan, OSM runs it. Each executable order component runs a process. Each process includes the tasks that fulfill the order requirements.

**About Orchestration Plans**

An orchestration plan includes the order items, order components, and dependencies. An order-specific orchestration plan is generated for each order that requires orchestration.

---

**Note:** Many types of orders do not require an orchestration plan; for example, some service orders are created specifically for a simple service provisioning task and therefore require no dependencies.

The orchestration plan for an order specifies the following:

- How order items are grouped into order components for processing
- The dependencies between the order components

In the OSM Order Management Web client, you can view graphical representations of an order’s orchestration plan and dependencies. You can use this information as you model orders to validate that order decomposition and orchestration plan generation
is functioning as intended. The graphical representation shows exactly how an order is fulfilled.

The Order Management Web client provides a graphical representation of the orchestration plan in two views:

- Orchestration plan decomposition
- Orchestration plan order item dependencies

Figure 4–9 shows three orchestration stages, represented in three columns:

- Determine the fulfillment function
- Determine the fulfillment system
- Determine the processing granularity

---

**Note:** You can model any number of orchestration stages.

---

At each orchestration stage, the graph shows the order components created by that stage. The final column on the right shows the order components that are run as part of the orchestration plan. Each component includes a name, which is based on the orchestration stages. Components also list their included order items.

The inset in Figure 4–9 shows details for three executable order components, as displayed in the orchestration plan decomposition.
Figure 4–9 Decomposition Tree

Figure 4–10 shows the orchestration plan displayed in the Order Management Web client dependency graph. The dependency graph shows the executable order components which are the components shown in the final stage of the decomposition display. In this case, executable components are based on three orchestration stages corresponding to fulfillment function, fulfillment system, and processing granularity. The different colors represent fulfillment functions, such as InitiateBilling or FulfillBilling. The inset shows a detailed view of two order components. Even though the two fulfillment functions are targeted to the same system (BRM-VOIP), processing granularity rules defined for this order require that they take place as two separate actions.
Both of these representations are useful at design time and when debugging orchestration plans. For example, you can use the dependency graph to confirm that an order goes to all of the correct systems in the correct order. Use the decomposition tree to verify that decomposition happens as expected at a particular stage and that the order was decomposed into the correct components, each containing the correct order items.

About Order Items

Order items are individual products, services, and offers that need to be fulfilled as part of an order. Each item includes the action required to implement it: Add, Suspend, Delete, and so on. For example, a new order might add a wireless router; the order item created in OSM is Add Wireless Router.

When you model order items, you do not model every possible order item. Instead, you create an order item specification, which defines:

- The data that each order item can include
- The structure of the data; for example, the hierarchy between order items
- Data needed for orchestration

There must be one order item specification for each type of order received from the order-source system. When you model an order item specification, you can configure the following:

- **Order item properties.** Order item properties represent the data that is included in order items. See "About Order Item Properties" for more information.
- **Orchestration conditions.** Use orchestration conditions to customize how order items are added to order components. For example, you can use the region order item property to assign order items to different target system order components.
See “About the Decomposition of Function to Target System Components” for an example of how orchestration components are used.

- **Order item hierarchies.** You use order item hierarchies to model how parent and child items are identified. For example, you can use line IDs and parent line IDs. See “About Order Item Hierarchy” for more information.

- **Order template.** This data is the order item control data, which is used by OSM when generating an orchestration plan. You can also assign behaviors to order items. See ”About Order Item Control Data” for more information.

- **Order item dependencies.** Use order item dependencies to create inter-order dependences. See ”About Inter-Order Dependencies” for more information.

- **Permissions.** Use permissions to allow specific roles access to order item search queries in the Order Management Web client and to specify if the query returns summary data or detailed data. See ”About OSM Roles” for more information.

**About Creating Order Items from Customer Order Line Item Node-Sets**

To create order items from customer order line items, OSM needs to know what nodes in the incoming customer order include the data to use in order items. OSM creates orchestration control data from these nodes (see ”About Order Item Control Data”).

Example 4–1 shows the `<salesOrderLine>` node-set in an incoming customer order. You can specify these node-sets as order items by creating an XQuery expression in the Orchestration Sequence editor that returns every instance of `<salesOrderLine>` contained in the customer order (see ”About Order Item Specification Order Item Property XQuery Expressions”). These node-sets produce the **Broadband Bundle** and the **Mobile Bundle** order items. The elements in these node-sets can then be specified as order item properties in the order item specification.

**Example 4–1   The `<salesOrderLine>` element in an Incoming Customer Order**

```xml
<im:salesOrderLine>
  <im:lineId>13</im:lineId>
  <im:promotionalSalesOrderLineReference>1</im:promotionalSalesOrderLineReference>
  <im:serviceId></im:serviceId>
  <im:requestedDeliveryDate>2001-12-31T12:00:00</im:requestedDeliveryDate>
  <im:itemReference>
    <im:name>Broadband Bundle</im:name>
    <im:typeCode>BUNDLE</im:typeCode>
    <im:specificationGroup></im:specificationGroup>
  </im:itemReference>
</im:salesOrderLine>

<im:salesOrderLine>
  <im:lineId>14</im:lineId>
  <im:promotionalSalesOrderLineReference>2</im:promotionalSalesOrderLineReference>
  <im:serviceId></im:serviceId>
  <im:requestedDeliveryDate>2001-12-31T12:00:00</im:requestedDeliveryDate>
  <im:itemReference>
    <im:name>Mobile Bundle</im:name>
    <im:typeCode>BUNDLE</im:typeCode>
    <im:specificationGroup></im:specificationGroup>
  </im:itemReference>
</im:salesOrderLine>
```
About Order Item Properties

Prior to generating an orchestration plan, OSM processes each customer order line item in the incoming customer order and turns it into an order item. The order item properties define the data that is included from these order items using XQuery expressions.

Figure 4–11 shows an order item and order item properties shown in the Order Management Web client.

Figure 4–11 Order Item Displayed in the Order Management Web Client

Most order items properties must be created in Design Studio and associated to corresponding customer order element values using XQuery expressions (see "About Order Item Specification Order Item Property XQuery Expressions"). However, in some cases the order item property is not provided in the customer order. In this case, you must use an XQuery expression to derive the missing property value from the existing customer order element values (see "About Order Item Specification Order Item Property XQuery Expressions").

Example 4–2 shows an order line item. This order line item adds a Commercial Fixed Service order item. In the following example, notice that the items in bold correspond to the order item properties. However, there are order item properties, such as productSpec and region, that are not in the order line item. Instead, you specify to create those order item properties by using XQuery expressions in the order item specification.

Example 4–2 Order Line Item in an Incoming Customer Order

```
<im:salesOrderLine>
  <im:lineId>4</im:lineId>
  <im:parentLineReference>
    <im:parentLineId>3</im:parentLineId>
    <im:hierarchyName>default</im:hierarchyName>
  </im:parentLineReference>
  <im:rootParentLineId>2</im:rootParentLineId>
  <im:promotionalSalesOrderLineReference>1</im:promotionalSalesOrderLineReference>
  <im:serviceId>552131313131</im:serviceId>
  <im:requesteDeliveryDate>2001-12-31T12:00:00</im:requesteDeliveryDate>
  <im:serviceAddress>
    <im:locationType>Street</im:locationType>
    <im:nameLocation>OLoughlin</im:nameLocation>
    <im:number>48</im:number>
    <im:city>Toronto</im:city>
  </im:serviceAddress>
  <im:itemReference>
```
Figure 4–12 shows all of the order items derived from an order, including the order item shown in Example 4–2.

In Figure 4–12, notice that order items are hierarchical. For example, the Fixed Service order item shown in Example 4–2 is part of the Fixed Bundle order item. In addition, the Fixed Service order item includes three more order items: Commercial Fixed Service, Fixed Caller ID, and Fixed Call Waiting. When you model orchestration, you ensure that the hierarchy in the incoming customer order is duplicated in the OSM order items. See "About Order Item Hierarchy" for more information.

The order item specification defines the order item properties that are required for generating the orchestration plan and the data to display in the Order Management Web client. This typically includes the display name, product or product specification, line ID, requested delivery date, and so on. By contrast, the order item usually would not include supplementary account and customer details such as the street address or mailbox size. That type of data is defined in the task data for each task in the fulfillment data, and in the creation task data when the order is created.

**Important:** Order item properties do not represent all of the data in an order. For example, they do not define creation task data. That data is captured by transformation rules. Order item properties are a subset of the data and are used for orchestration.
Example 4–3 shows part of an order input file, and Figure 4–13 shows how the city field is mapped to the region order item property in Design Studio. In this example, the <city> element in the XML file is used in the order item property expression.

See “About the Decomposition of Function to Target System Components” for an example of how the region order item property is used in orchestration.

Example 4–3 Partial Customer Order Input File

```xml
<im:serviceAddress>
  <im:locationType>Street</im:locationType>
  <im:nameLocation>Jangadeiros</im:nameLocation>
  <im:number>48</im:number>
  <im:typeCompl>floor</im:typeCompl>
  <im:numCompl>6</im:numCompl>
  <im:district>Ipanema</im:district>
  <im:codeLocation>5000</im:codeLocation>
  <im:city>Rio de Janeiro</im:city>
  <im:state>RJ</im:state>
  <im:referencePoint>Gen. Osorio Square</im:referencePoint>
  <im:areaCode>22420-010</im:areaCode>
  <im:typeAddress>Building</im:typeAddress>
</im:serviceAddress>
```

Figure 4–13 Order Item Property Definition

A single order item specification is used for generating all of the order items that can be created for an order. This ensures a consistent order item structure. Therefore:

- Order item properties should not be specific to a product or service. The only product information you need to include is the product or product specification name, which is a generic value used for identifying the fulfillment pattern. By not applying order items to a specific product, you can use the order item specifications for multiple products, and you can support new products and services and multiple order entry systems.

- Order item properties should not be specific to any order entry system.
The properties you define for your order items will be different from those pictured in Figure 4–14. However, this selection provides a good example of the type of order properties that are commonly configured:

- **productSpec**: This property retrieves the product specification from the incoming customer order. OSM maps each order item to a fulfillment pattern based on the item’s product specification (defined in the order-source system). The fulfillment pattern specifies the order components in the first level of decomposition.

- **FulfillPatt**: This property stores the fulfillment pattern that the order item uses. This value is obtained by mapping the **productClass** value in a mapping file. See "About Mapping Order Items to Fulfillment Patterns" for more information.

- **lineId**: This is the line ID of the order line item in the incoming customer order. Each order line item in the incoming customer order has a unique line ID. This
property is used for determining the hierarchy of the order items. See "About the Decomposition of Target System to Granularity Components" for an example of how this property is used.

- **lineItemName**: This property is the display name used in OSM Web clients.
- **requestedDeliveryDate**: This property is the requested completion date for the order item.
- **parentLineId**: This property defines the parent of the order line item in the incoming customer order. This property is used for determining the hierarchy of the order items. See "About the Decomposition of Target System to Granularity Components" for an example of how this property is used.
- **region**: This property is an example of data that can be used to manage decomposition into target system order components. See "About the Decomposition of Function to Target System Components" for more information.
- **serviceId**: This property is used to display the service ID in the OSM Web clients.
- **lineItemPayload**: This property stores the entire incoming customer order in OSM as an XML file. This property is typically used in a development environment as an aid to modeling.

### About Order Item Hierarchy

Order items can be organized hierarchically based on the content of the original customer order. The hierarchy can include various types of order line items, such as offers, products, and bundles of products or services. For example, an order could include a Broadband-VoIP offer with a High Speed Internet bundle, an Internet Services service bundle, and a Wireless Router product item. OSM maintains the order line item hierarchy from the customer order in the order item hierarchy. Maintaining this hierarchy allows OSM to aggregate order item status from a hierarchy of order items.

Figure 4–15 shows an item hierarchy that reflects the structure of the original customer order.

**Figure 4–15 Item Hierarchy**

```plaintext
- On Top of the World Broadband-VoIP
  - High Speed Internet First Month Free Discount
- High Speed Internet Service
  - Dynex Modem
  - High Speed Internet Installation
  - High Speed Internet Activation
  - Wireless Rrotter
- Internet Services
  - Basic High Speed Internet - 1Mbps
- Internet email Service
  - Internet email
- Internet Media Service
  - Internet Video on Demand
  - Internet Content on Demand
  - Internet Secure Firewall
```

The hierarchy is defined in the `<lineID>` and `<parentLineId>` elements.
To define the order item properties that specify the hierarchy, you configure the order item hierarchy in the order item specification using an XQuery expression. See "About Order Item Specification Order Item Hierarchy XQuery Expressions" for more information.

**About Using the Distributed Order Template**

The distributed order template is a structure data type that is available only for order item specifications. It improves performance and also has the following benefits:

- Reduces order node conflicts: Without the distributed order template, data elements in the data dictionary that have the same name need to have the same definition (type, description, etc.) regardless of whether they appear in different structures in different places in the data dictionary. With the distributed order template, this is no longer necessary.

- Allows data changes without having to redeploy the entire solution: Without the distributed order template, any changes to the data defined for the order (including order item property updates) requires redeployment of all of the cartridges that reference the order item. With the distributed order template, if you change order item properties, you need to deploy only the cartridge containing the changed order item.

You decide whether to use the distributed order template by selecting the appropriate box in the order item creation wizard or in the Order Item Specification editor in Design Studio. For more information, see Modeling OSM Orchestration Help.

If you use the distributed order template, any references you make to order item data in XQuery expressions or automation must include a namespace. References to data in data change notifications and flexible headers do not need to change. For any order item that is not a transformed order item, the namespace will always be the namespace of the order item specification. Following is an example of an XQuery reference to the lineItemID property on the InputOrderItem order item with the namespace http://ex-input.com:

```
/ControlData/OrderItem[@type='(http://ex-input.com)InputOrderItem']/lineItemID
```

If you are using the order transformation manager, see "Using the Distributed Order Template with the Order Transformation Manager" for information about the namespace that will be used for transformed order items.

**About Mapping Order Items to Fulfillment Patterns**

The first orchestration stage assigns order items to function order components, by using fulfillment patterns. You need to model how to map order items to fulfillment patterns and implement the model using an XQuery expression (see "About XQuery Expressions for Mapping Product Specifications and Fulfillment Patterns" for more information about using XQuery to map product specifications to fulfillment patterns).

**About Modeling Product Specifications**

New product specifications should be imported (which will create conceptual model products) or created in the conceptual model. If you have an existing configuration, however, you can still use product specifications that were created in OSM. See Design Studio Concepts for more information about conceptual model products.

You can map multiple product specifications to one fulfillment pattern. This enables you to introduce new products in existing product specifications without needing to create new fulfillment patterns or fulfillment flows.
About Order Item Control Data

In addition to defining order item properties in the order item specification, you need to provide a storage area for the order item properties. You do so by adding control data to the order item specification Order Template tab. This definition is automatically added to the order’s order template. This makes it easier to track which entity is the master of the data and enables easier refactoring and maintenance of the overall order specification. Figure 4–16 shows the order item properties in the control data in an order template.

Figure 4–16 Order Item Properties in the Order Template

When you define the control data, note the following:

- The name used in the control data must exactly match the spelling and case of the order item property name.
- Ensure that the Data Dictionary properties are correct for the type of data; for example, string or number.
- Configure each data element as a multi-instance data element.
  - Minimum = 0
  - Maximum = Unbounded

Note: To define data properties, you edit the entry in the data schema, not in the order item specification.
An instance of **ControlData/OrderItem** is created for each data element returned by the order item selector from the orchestration sequence (see “About Creating Order Items from Customer Order Line Item Node-Sets”).

The **OracleComms_OSM_CommonDataDictionary** model project contains predefined base data elements for control data. Oracle recommends that you use the data schema of this model project to add the **ControlData/OrderItem** structure to the order item specification **Order Template** tab.

---

**About Decomposition**

Decomposition is the process of organizing order components into increasingly granular order components.

There are typically three stages of decomposition, which decompose order items into these types of order components:

- Function order components
- Target system order components
- Granularity order components

Figure 4–17 shows a simplified decomposition process. In this example, the order includes two order items, adding a mobile service and adding a fixed service. The decomposition proceeds as follows:

1. In the first stage of decomposition, the order items are assigned to function order components. There are two function order components: provisioning and billing. Both of the order items require provisioning and billing, so they are both included in the provisioning and billing function order components.

   Order items are assigned to function order components by using a fulfillment pattern. See "About Mapping Order Items to Fulfillment Patterns" for more information.

2. In the next stage of decomposition, the order items are assigned to target system function components. The mobile and fixed services require different provisioning systems, so there are two separate target system components for provisioning. Both order items, however, can be processed by the same billing system, so they are both included in the same billing target system order component. See "About the Decomposition of Function to Target System Components" for more information.

3. In the final stage of decomposition, no further decomposition is required for provisioning; the order items are processed by their target systems. For billing, however, the mobile service billing and the fixed service billing need to be processed separately. In this stage of decomposition, therefore, the orchestration plan generates two bundle granularity order components: one for the mobile service billing and one for the fixed service billing. See "About the Decomposition of Target System to Granularity Components" for more information.
Order items are often included in more than one order component. This is because order item fulfillment often requires multiple functions. For example, the top-level order item Create Mobile Service would be included in any order component that included the child order items belonging to Create Mobile Service.

Order components are usually modeled by extending order component specifications in Design Studio. For example, you can create a base order component for all function types and extend it for individual function types such as billing or collections.

### About Component IDs and Component Keys

Each order component has a component ID and a component key. The component ID is specified at design time. The component key is generated for each instance of the order component at run time. You can use customized order keys when assigning order items to order components.

By default, the component ID is the name of the component; for example, Billing Function or Billing System. The component key is a concatenation of the names of the components in the orchestration stages. For example, if the component IDs are modeled as:

- Billing Function
- Billing System
- Bundle

The order keys generated at run time are:
You can customize how order keys are generated. See "About the Decomposition of Target System to Granularity Components" for an example of how to use customized order keys to generate granularity components. Figure 4–18 shows order component keys displayed in the Order Management Web client, in the Orchestration Plan page’s Dependencies tab.

Figure 4–18 Order Component Keys Displayed in the Order Management Web Client

About the Decomposition of Order Items to Function Order Components

The following sections describe the decomposition of order items to function order components.

About Assigning Order Items to Fulfillment Pattern Function Components

The first step in decomposition is to assign order items to function components. To do so, OSM uses the product specification to find the fulfillment pattern that the order item uses. (See “About Mapping Order Items to Fulfillment Patterns” for more information.) The fulfillment pattern defines the order components to add the order item to.

Figure 4–19 shows the function order components selected in the Service.Broadband fulfillment pattern. In this case, order items that use this fulfillment pattern need all of the functions; billing, collections, provisioning, and so on.
Figure 4–19  Function Order Components Selected for a Service Fulfillment Pattern

Figure 4–20  How to Use a Base Specification to Define Function Components

Base Specification

Service Specification  Non-Service Specification

About the Function Components Stage
In addition to using the fulfillment pattern to assign order items to function components, you model an orchestration stage, which specifies to create the function order components to create. Figure 4–21 shows the function order components created at the DetermineFulfillmentFunction orchestration stage.
**About Order Component Control Data**

When OSM creates the order items and order components, it produces a set of control data. The control data provides information OSM requires to fulfill the order. OSM uses the control data to track the status of the entire order and to track the status of the individual order items. During fulfillment, order component transactions update this control data with system interaction responses.

Design Studio automatically generates control data for function order components provided that orchestration entities are preconfigured correctly and you use the `OracleComms_OSM_CommonDataDictionary` model project. If you do not use the `OracleComms_OSM_CommonDataDictionary` model project, you must manually model order component control data. See the Design Studio Help for information on how order component control data is automatically generated or how to manually model it.

See "About Order Template Control Data" for more information on adding function order components to the order control data.

**About Fulfillment Pattern Conditions for Including Order Items**

You can use conditions to add order items to an order component only when the XQuery for the condition evaluates to true. For example, you might include an order item based on an XQuery that checks the action code (Add or None). This is useful in the case of an update to a service that changes some features while leaving other features unchanged. See "About Order Item Specification Condition XQuery Expressions" for more information.
Summary of Order Item to Function Components Decomposition
To summarize this example, to model the decomposition from an order item to a function component, you model the following:

- The fulfillment pattern order item property so that order items can be mapped to fulfillment pattern function components
- Any XQuery expressions that evaluate conditions to include or exclude order items
- The Order control data for orchestration
- The orchestration stage that produces the function components

About the Decomposition of Function to Target System Components
The following sections describe the decomposition of order items from functional components to target systems.

About Decomposition Rules from Function Components to Target Systems
After the order items have been assigned to function order components, they need to be further decomposed into target system order components. To do so, you use decomposition rules.

A decomposition rule specifies a source order component and a target order component. Figure 4–22 shows a decomposition rule from the billing function component to the billing target system component.
You can use decomposition rules to decompose order items from one function component to multiple target system components. Figure 4–23 shows the source and target order components for two decomposition rules:

- Provision to DSL Provisioning System - Region1
- Provision to DSL Provisioning System - Region2

These two decomposition rules decompose the order items in the ProvisioningFunction order component into two target system order components based on Region 1 and Region 2.
Each of the decomposition rules uses decomposition conditions to specify which target system to use for a particular order. The target system is selected if the XQuery expression associated with the condition evaluate to true. In this example, the XQuery expression uses the value of the `region` order item property to make this evaluation. If the value of region is Toronto, then OSM selects the condition and target system for Region 1. If the value of region is New York, then OSM selects the condition and target system for Region 2. See "About Order Item Specification Condition XQuery Expressions" for more information about creating an XQuery condition expression that can be used for with a decomposition rule.

**About the Target Systems Stage**

In addition to creating the decomposition rules that define the source and target components, you need to create an orchestration stage that produces the target system order components.

**Summary of Configuring Target System Components Decomposition**

To summarize, to configure how order items are decomposed from a function order component to a target system order component, you do the following:
- Define an orchestration stage to produce the target system order components.
- Create dependency rules to specify the source order components and target order components.
- If a function order component decomposes order items to more than one target system order component, create decomposition conditions. Decomposition conditions depend on data specific to the order items, so decomposition rules typically use XQuery expressions to retrieve the data that is used for evaluating the condition.

About the Decomposition of Target System to Granularity Components

The following sections describe the decomposition of order items from target system components to granularity components.

About Decomposition Rules from Target System to Granularity Components

After order items have been decomposed into target system order components, the next step is to decompose them into the granularity order components.

Some examples of the granularity requirements are:

- A billing system might require the entire order in the message to calculate discounts.
- A billing system might require separate bundles for mobile billing and fixed billing, to handle different completion times (fixed billing typically has more dependencies and takes longer).

The following example shows how to decompose target system order components items into bundle granularity components. You configure the following:

- Create a decomposition rule, which decomposes the target system order component into bundle granularity components.
- Create customized component IDs that are used to create separate order components for each bundle. (See "About Customized Component IDs to Separate Bundled Components" for information.)

About Customized Component IDs to Separate Bundled Components

You create the customized order component by editing the bundle order component specification.

In this example, you need to configure a decomposition rule and a bundle granularity order component specification to ensure that order items for a fixed service and a broadband service are decomposed into separate bundle granularity components, based on their customized component IDs. The customized component IDs result in separate instances of bundle order components, with separate component keys. This allows OSM to process the order components for the fixed service and the broadband service separately.

If you had not created customized component IDs, the component key of both order items would be BillingFunction.BillingSystem.Bundle. The order items would be processed together in the same order component.

This customization also ensures that the component ID is the same for order items within the same granularity (for example, a bundle) but not for order items at a higher granularity.
In addition, you may want to group order items into custom component IDs based on order item requested delivery date. For example, you might want an order component to process all order items with a requested delivery date that falls within the first two days of when an order start, and another order component for the next two days. You can further combine these grouping by requested delivery date within order item hierarchy groupings.

See "About Component Specification Custom Component ID XQuery Expressions" for more information about configuring custom order component hierarchies using XQuery.

**About the Granularity Components Stage**

In addition to creating the decomposition rules that define the source and target components, you need to create an orchestration stage that produces the granularity order components.

**Summary of Configuring Granularity Components Decomposition**

To summarize this example, to model the decomposition from a target system order component to a bundle order component, you model the following:

- The decomposition rule, which decomposes the target system order component into bundle granularity components
- The orchestration stage that produces the bundle order component
- The order item hierarchy that the XQuery `ancestors` function uses in the order item specification
- The XQuery for the customized order component in the bundle order component specification

**About Associated Order Items**

Sometimes, you need to assign order items to order components that would not be assigned by their fulfillment pattern. This is often the case when an interaction with an external system requires a specific context for an order item.

For example, a billing system might need to process billing-related order items in the context of a bundle, to manage the relationships between balances, discounts, and so on. Billing charges are often order line items, such as an installation service, that are included in the order outside of the service billing bundle hierarchy. However, they might need to be associated with the billing bundle to ensure that the charge is made against the correct service. In that case, you can associate the billing charges with a bundle order component.

By contrast, billing order items might be sent to the billing system in the context of a whole order. In that case, you do not need to associate the order items to a bundle, because they are already in context.

*Figure 4–24 shows the associated order items, displayed with (assoc) in the orchestration plan.*
You model order item associations in fulfillment patterns. Figure 4–25 shows an order item association modeled for the Bundle order component in the Service.Mobile fulfillment pattern.

There are two ways to associate order items:

- Fulfillment pattern: This is the default entry. It associates order items by fulfillment pattern, which is the normal orchestration method.
Property correlation: This associates order items by using order item properties. See "About Associating Order Items Using Property Correlations XQuery Expressions" for more information.

About Modeling Fulfillment Modes

The fulfillment mode is the overall purpose of the order. For example:

- Deliver a service.
- Qualify a service before delivering it. This ensures that a service can be fulfilled before attempting to fulfill it.
- Cancel an entire order.

Every incoming customer order can specify a fulfillment mode. OSM can use the fulfillment mode as part of the orchestration process. For example, if OSM receives two identical incoming customer orders with different fulfillment mode order item properties, it generates a different orchestration plan for each order. The two plans include different executable order components with different dependencies among order items.

Fulfillment modes are configured in the following places:

- Fulfillment mode entities: These entities include no data other than a name. They provide the means to assign fulfillment modes to other entities, such as orchestration sequences and fulfillment patterns.

- Orchestration sequences define a single fulfillment mode using an XQuery expression based on a customer order attribute (see "About Order Sequence Fulfillment Mode XQuery Expressions").

- Fulfillment patterns list the fulfillment modes that the associated order items can be used with.

Figure 4–26 shows the fulfillment modes defined in a fulfillment pattern. Any order item that uses this fulfillment pattern can be processed in either the Cancel or Deliver fulfillment mode.
When a fulfillment pattern includes multiple fulfillment modes, you can model a different set of order components and dependencies for each fulfillment mode. Figure 4–27 shows the order components specific to the Deliver fulfillment mode.
About Dependencies

An orchestration plan is based on two main factors: decomposition, which derives the order components, and dependencies, which dictate when the order components are allowed to run.

Dependencies are relationships in which a condition related to one order item must be satisfied before another item can be processed successfully. For example, a piece of equipment must be shipped to a location before the action to install it at that location can be taken.

You can model the following types of dependencies:

- A dependency that requires the completion of one type of fulfillment function for an order item before starting another type of fulfillment function for the same order item. For example, for a single order item, you can specify to provision the order item before you bill for it.

- A dependency that requires the completion of one type of fulfillment function for one type of order item before starting the same fulfillment function for another order item. For example, you must provision DSL order items before you can provision VoIP order items.
A dependency that requires the completion of one order item before starting another order item based on order item properties.

You typically create dependencies between order items in the same order (intra-order dependencies). However, you can also create dependencies between order items in different orders (inter-order dependencies). For example, the order items in a follow-on order for VoIP provisioning might depend on the execution of the order items in the original order for DSL provisioning. See "About Inter-Order Dependencies" for more information.

A dependency requires two order components: the waiting order item and the blocking order item. The blocking order item is the order item that must complete before the waiting order item is started.

Dependencies can be based on several different factors, including:

- Completion status. For example, the blocking order item must be complete before the waiting order item can start. For example, you can specify to start billing only after provisioning has completed.

- Actual and relative date and time. For example, you may want an order component that contains order items for an installation to start two days after the completion of the order component that contains the order items for shipping the equipment.

- Data change. For example, you can specify that shipping must wait until a specified order item property in the blocking order item has a specified value.

Order items can have combinations of dependencies. For example, an order item for an installation can depend on a combination of a completion status dependency (item successfully shipped) and date dependency (wait two days after shipment to schedule installation).

---

**Note:** You can manage dependencies during amendment processing; for example, when you submit a revision order. See "About Compensation and Orchestration" for more information.

---

Although dependencies exist logically between order items, they are managed by order components. In other words, if any item in a component has a dependency, the component as a whole cannot be started until the dependency is resolved. In the Order Management Web client, order items include dependency IDs to indicate items whose dependencies are managed together. See Order Management Web Client User’s Guide for more information.

You can model dependencies in two ways in Design Studio:

- As order item dependencies. These dependencies are modeled as part of fulfillment patterns. Most dependencies are modeled in this manner.

- As orchestration dependencies. These dependencies are modeled outside of fulfillment patterns. While not as common as those modeled in fulfillment patterns, orchestration dependencies are useful in specific circumstances; for example, if you need to define a generic dependency or want to model one without having to modify a fulfillment pattern. See "About Modeling Orchestration Dependencies" for more information.

Figure 4–28 shows order items displayed in the Order Management Web client. In this example, the billing order items for a fixed service can start immediately because they
have no dependencies. The billing order items for high-speed Internet must wait until the provisioning order items have completed.

Figure 4–28 Dependencies Displayed in the Order Management Web Client

About Intra-Order Dependencies

The following sections describe dependencies that can be created within the same order.

Modeling an Order Item Dependency

The simplest form of dependency is an order item dependency, configured in a fulfillment pattern. This type of dependency is based on function order components; for example, the billing order component cannot start until the provisioning function has completed. Figure 4–29 shows an order item dependency in Design Studio.
Figure 4–29 Order Item Dependency in Design Studio

Figure 4–30 shows the dependency relationships shown in Figure 4–29. Note the two layers of dependency: billing is dependent on provisioning, and everything else is dependent on billing.

Figure 4–30 Dependency Relationships for Order Item Dependency
In addition to defining the function order components, you need to define the conditions that govern the dependency. The default condition is to wait until the final task associated with the order item has completed. Figure 4–31 shows a wait condition defined in Design Studio. In this case, the waiting order item must wait until the blocking order item task has reached the Completed state. See "About Order Item Dependency Wait Conditions" for more information.

**Figure 4–31 Wait Condition in Design Studio**

![Wait Condition in Design Studio](image)

**About Order Item Dependency Wait Conditions**

Dependency wait conditions specify the condition that the blocking order item must be in before the waiting order item can start. For example, the default wait condition is to start the waiting order item when the last task associated with the blocking order item reaches the Completed state.

You specify wait conditions in fulfillment patterns and orchestration dependencies. You can set different wait conditions for each dependency. The wait conditions can be:

- The task state of the final task associated with the blocking order item
- A change in the data for a specified field. See "About Order Item Dependency Wait Conditions Based on Data Changes" for more information.
- A specified duration after the task state or data change condition has been met. You can specify a value in months, weeks, days, hours, or minutes, or you can specify an XQuery expression to determine the delay (see "About Wait Delay Duration XQuery Expressions"). For example, you can specify to start the waiting order item two days after the blocking order item has completed.
- A specific date and time based on the result of an XQuery expression (see "About Wait Delay Date and Time XQuery Expressions"). For example, you can specify to start the To Component order component on a date specified in an order item property.

Figure 4–32 shows the wait condition options in a fulfillment pattern in Design Studio. The orchestration dependency wait condition options are identical.
About Order Item Dependency Wait Conditions Based on Data Changes

You can base a dependency on a change to data. The data must be included in an order item property, and it must be in the task data of the task associated with the blocking order item.

To configure the dependency, you define the following:

- The order item property that is evaluated. Any change to the data in the order item property triggers an evaluation of the data to determine if it matches the conditions required for the dependency.
- An XQuery expression that evaluates the data retrieved from the blocking order item. The expression returns true or false; if true, the dependency has been met.

Figure 4–33 shows a data change dependency in Design Studio.

In Figure 4–33:
- The Order Item field specifies the order item specification to use.
The order item property that the dependency is based on is **milestone**.

The **Relative Path** field (not used in this example) is an optional field you can use to specify a child data element in the order item properties.

- The XQuery expression evaluates the data in the **milestone** property to determine if the dependency has been met (see “About Order Data Change Wait Condition XQuery Expressions” for more information).

### Modeling a Fulfillment Pattern Dependency

You can define dependencies across different order items by basing the dependency on the fulfillment patterns of the order items. For example, you can create a dependency that specifies to provision fixed services only after broadband services have been provisioned.

**Figure 4–34** shows a dependency based on fulfillment pattern. In this example, the dependency requires that fixed services be provisioned before broadband services. To configure this type of dependency, you edit the fulfillment pattern of the waiting order item. In the fulfillment pattern, you provide a list of waiting and blocking order components.

**Figure 4–34  Dependency Based on Fulfillment Pattern**

![Fulfillment Pattern: Service.Mobile](image)

**Figure 4–35** shows the dependency relationships shown in **Figure 4–34**. Note that fixed provisioning is the blocker for broadband provisioning and for fixed billing.
Figure 4–35  Dependency Relationships for Fulfillment Pattern Dependency

Modeling an Order Item Property Correlation Dependency
Using properties correlation is the most flexible way to configure dependencies. You use this method to create a dependency on two different order items that share the same order item property. As with other dependencies, you specify a blocking component (the From Component field) and a waiting component (the To Component field), but you also enter an XQuery expression to select the order item property that order items in the To Component field must share with order items in the From Component field (see "About Order Item Dependency Property Correlation XQuery Expressions" for more information).

About Inferred Dependencies
OSM is able to create dependencies at run time by inferring dependencies. For example, you might create this series of dependencies:

Provisioning - Billing - Marketing

If the order item has no billing function, there is an inferred dependency between Provisioning and Marketing, even though you have not modeled that dependency. Provisioning must complete before Marketing can start.

Inferred dependencies mean that whenever A is dependent on B and B is dependent on C, A is dependent on C. This avoids the need to model every dependency that might be possible.

Figure 4–36 shows a sample dependency configuration. Figure 4–37 shows the run-time view of the same configuration when there is no billing function. In this case, the Order Management Web client shows dependencies from provisioning to marketing, synchronize customer, and collections.
Inferred dependencies are supported within a fulfillment pattern, but they are not supported across fulfillment patterns. For example, in Figure 4–35, OSM does not infer a dependency from ProvisioningFunction(Service.Fixed) to BillingFunction(Service.Broadband). You must specifically model that dependency.

**About Inter-Order Dependencies**

An inter-order dependency is a dependency between order items in different orders. You typically configure this type of dependency to manage changes to an order when that order has passed the point of no return and cannot be amended. However, you can also use inter-order dependencies for other purposes, such as managing fulfillment functions on different systems, load balancing, and so on.

When using inter-order dependencies, the blocking order is the base order, and the waiting order is a follow-on order. A typical scenario is:
1. A customer has ordered a broadband service.

2. The next day, while the order is still in-flight but past the point of no return, the customer requests a change to the service bandwidth.

3. Because a revision to the base order cannot be submitted, the customer service representative creates a follow-on order.

4. The follow-on order is submitted to OSM; however, it does not begin processing until the base order has completed.

Here are some important points to know about inter-order dependencies:

- Inter-order dependencies are based on order items. After the base order completes the blocking order item, the follow-on order can start, even though the base order is still in-flight.

- Inter-order dependencies are sometimes used to manage technical dependencies when a specific fulfillment requirement cannot be handled by a revision. However, they can also be based on business reasons, when it is simpler or more efficient to use a follow-on order than to model revisions.

- A follow-on order does not perform amendment processing on the base order. A follow-on order can be used to add, modify, or cancel services, similar to any order. The key feature is that a follow-on order has a dependency on another order.

You must model the inter-order dependencies into both the base order and the follow-on order.

- The follow-on order must be able to find the base order and be able to recognize if the blocking order item has completed.

- The base order must contain a reference to allow the follow-on order to find it.

To configure an inter-order dependency, you use the Order Item Dependencies tab. The configuration typically consists of the name of the dependency and its XQuery or data instance (see "About Order Item Inter-Order Dependency XQuery Expressions" for more information about inter order item XQuery expressions).

You can create inter-order dependencies that involve order item hierarchies. For example, you can specify that the blocking order item include all of the order items in its hierarchy. To do so, select Use For Child Completion Dependency when specifying an order item hierarchy (see Figure 4-38). For more information about order item hierarchies, see "About Order Item Hierarchy").
About Modeling Orchestration Dependencies

You use orchestration dependencies to create dependencies between order components that are not based on fulfillment patterns. For example, if you need to define a generic dependency or want to model one without having to modify a fulfillment pattern, you can use an orchestration dependency specification.

As with dependencies defined in fulfillment patterns, you can specify wait conditions and the type of order item dependency (for example, order item, fulfillment pattern, and property correlation).

About Fulfillment States

You use fulfillment states to provide an aggregated status of orders and order items. Fulfillment state functionality, which is an optional part of the orchestration configuration, maps the statuses received from external systems into normalized statuses for the order components. Then, you can use fulfillment states to compose these normalized external states into individual statuses for order items, and ultimately to a single status per order. These statuses can be sent to upstream systems, like a CRM system, and can also be seen in the Order Management Web client. For example, if some order components succeeded and some failed, you could use fulfillment states to send a single status of PART_COMPLETE to the CRM system.

In addition to using fulfillment states to provide order item and order processing state information to upstream systems, you can also use them to restrict processing of order amendments from the upstream system. This functionality is provided by point-of-no-return processing, which is based on fulfillment states. See "About Point of No Return" for more information.

OSM can send requests to many different systems for the same order item or order. Each request may have one or many responses. Fulfillment states allow you to send meaningful status values to an upstream system, which would not be able to parse all of the individual statuses returned by the external systems. You can use fulfillment states to manage system complexity in the following ways:
Different external fulfillment systems can return different statuses which indicate the same thing. For example, one system might return OK to indicate success, while another returns Complete. These responses are known as **external fulfillment states**. You can use **fulfillment state mappings** to map both of these responses to the same value, for example Succeed. The result in OSM is referred to as a **mapped fulfillment state**.

The same external fulfillment system can send back multiple messages for the same request, sending status information before the request is complete. For example, the same request might send back Accepted, Started, and Processing before sending back a final status of Complete. You can use fulfillment state mapping to ensure that only the relevant statuses cause a change to the mapped fulfillment state. For example, you might map Accepted, Started and Processing all to the mapped fulfillment state of In_Progress.

Statuses are returned for each order component; that is, for each fulfillment request sent by OSM. An individual order item can be associated with many order components, and so the same order item will have multiple mapped fulfillment states. You can use **order item fulfillment state composition rule sets** (and if you are using the order transformation manager, **transformed order item composition rule sets**) to compose all of those states into a **composite fulfillment state** for the whole order item.

Order items can be hierarchical. You can use order item fulfillment state composition rule sets to produce a composite fulfillment state for an order item, regardless of whether its children are other order items or order components. Fulfillment state composition rule sets use the hierarchies defined in the order item specification to determine parent and child order items. If a composition hierarchy is defined for the order item, OSM will use this hierarchy to determine parent/child relationships. If a composition hierarchy is not defined, parent/child relationships for order items will not be used.

If you are using the order transformation manager, you can define transformed order item composition rule sets, which allow you to define conditions for using transformed order items to compute the fulfillment state of their original order items. For more information about the order transformation manager, see "About the Order Transformation Manager."

Orders can contain many order items. You can use **order fulfillment state composition rule sets** to produce a composite fulfillment state for an entire order.

*Figure 4–39* depicts a fulfillment state mapping scenario.
In the figure, OSM receives the external fulfillment states (for example, Completed and Success) and uses fulfillment state maps and composition rules to translate those states into a value that the CRM system uses, such as SuccessfulCompletion. In some cases, the external fulfillment state that indicates a completed order may not be obvious. For example, if the task sent to a billing system is supposed to start the billing, the billing system might indicate a completed order by sending the status Started.

Figure 4–40 is a more detailed depiction of fulfillment state processing for a small part of a sample implementation. It shows the way multiple external responses can be translated into a single fulfillment state for the order.
At run time, OSM maps the external fulfillment states to mapped fulfillment states on an order item. Order item fulfillment states are composed using the immediate children of the order item, and order fulfillment states are composed using the root-level order items.

Whenever one of the input fulfillment states for an order item changes, the fulfillment state of that order item (and all of its parents, including the order) is recalculated. For example, if the mapped fulfillment state of "leaf" order item A changes, the composite fulfillment state of order item A is recalculated. If the composite fulfillment state for order item A changes and it has a parent, order item B, order item B's fulfillment state is recalculated as well. If the composite fulfillment state of order item A does not change, the fulfillment state for order item B is not recalculated.

In the figure:

1. The external billing system sends a status of OK, which is used directly as the external fulfillment state for OrderComponent_Billing.

2. The external fulfillment state of OK is mapped to Complete for both the order items that are fulfilled by that order component (OrderItem_VoIP and OrderItem_Mobile) using the fulfillment state mappings.

3. The activation system has sent a complex message indicating the statuses of different parts of the fulfillment request. That message is translated by the custom code in the automation to the external fulfillment state of MOBILE_FAIL.
4. The fulfillment state mappings are configured to map MOBILE_FAIL for this order component to mean that OrderItem_Mobile has failed and OrderItem_VoIP has succeeded.

5. The fulfillment state composition rules for the OrderItem_VoIP order item then look at the mapped fulfillment states for OrderItem_VoIP for each order component (OrderComponent_Billing and OrderComponent_Activation) that fulfills that order item. Because the mapped fulfillment states for both of the order components are Complete, the composite fulfillment state for the order item is also set to Completed.

6. The fulfillment state composition rules for the OrderItem_Mobile order item then look at the mapped fulfillment states for OrderItem_Mobile for each order component (OrderComponent_Billing and OrderComponent_Activation) that fulfills that order item. Because the mapped fulfillment states for one of the order items is Complete and for the other order item is Fail, the composite fulfillment state for the order item is set to Failed.

7. The fulfillment state composition rules for the order then take the composite fulfillment state of the highest-level parent order items to determine the fulfillment state of the order. In many cases, the failure of any part of an order might be configured as a failure of the order as a whole. However in this example, fulfillment states have been configured that, because part of the order (VoIP) is ready for customer use, the composite fulfillment state is set to Part_Success.

Defining Fulfillment States

Fulfillment states are configured in Design Studio. At a high level, configuration of fulfillment state management has the following main steps:

1. Define external fulfillment states for order components: Create a list of values for the order component that matches the statuses returned by the external systems or automations. An external fulfillment state is available on the order component where it is defined and on any order component that extends that order component. See "Modeling External Fulfillment States" for more information.

2. Create and configure fulfillment state maps: Create one or more lists of values for the common fulfillment states and create mappings to translate external fulfillment states into mapped fulfillment states. Common fulfillment states are used as mapped fulfillment states and as composite fulfillment states. Fulfillment state mappings provide the evaluation and normalization of the external system’s states into mapped fulfillment states. Common fulfillment states and fulfillment state mappings are available for the entire workspace. See "Modeling Fulfillment State Maps" for more information.

3. Create and configure order item fulfillment state composition rule sets, transformed order item fulfillment state composition rule sets if you are using the order transformation manager, and order fulfillment state composition rule sets: Create the composition rule sets to determine the fulfillment state of an order or order item from the fulfillment state of its child items. Composition rule sets are based on the order item and order hierarchy, and compose fulfillment states into composite fulfillment states that reflect the state of entire order items or orders. See "Modeling Fulfillment State Composition Rule Sets" for more information.

The external fulfillment states, order item fulfillment states, and order fulfillment are stored in the ControlData for the order. See "Modeling Order Template Structures for Fulfillment States" in OSM Developer’s Guide for more information. Mapped fulfillment states are not stored on the order.
**Modeling External Fulfillment States**

External fulfillment states consist of a list of responses expected by an order component and any order components that extend the order component. Once an external fulfillment state is defined, it can be used in a fulfillment state mapping.

**Modeling Fulfillment State Maps**

You use fulfillment state maps to configure common fulfillment states and fulfillment state mappings. Fulfillment state mappings are the entities that contain the actual mapping information, and fulfillment state maps are containers for the information. Functionally, it does not matter whether you have one or many fulfillment state maps. Each common fulfillment state is available to all of the fulfillment state mappings, regardless of which fulfillment state map it is configured in. This means that each common fulfillment state needs to be unique in the workspace. There are optional default common fulfillment states that can be used. Please see Modeling OSM Orchestration Help for more information about the default states.

Common fulfillment states have two functions:

- They are used as the result of the fulfillment state mappings. When they are used this way, they are referred to as mapped fulfillment states.
- They are used as the result of the composition rules. When they are used this way, they are referred to as composite fulfillment states. If these fulfillment states are to be sent to an upstream system, you configure these values to match what the upstream system expects. (For more information about composition rules, see "Modeling Fulfillment State Composition Rule Sets").

Common fulfillment states, used as either mapped or composite fulfillment states, are configured in a single list in the **States** tab of the Fulfillment State Map editor. You do not need to assign the common fulfillment state as either a mapped fulfillment state or a composite fulfillment state when you configure it. The same common fulfillment state can be used for both purposes at the same time. Figure 4–41 shows the common fulfillment states configured in a fulfillment state map.

**Figure 4–41 Detail from Fulfillment State Map Editor States Tab**

<table>
<thead>
<tr>
<th>Description</th>
<th>OsmCentralOMExample-FulfillmentStateMap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Namespace</td>
<td><a href="http://oracle.communications.ordermanagement.unsupported.centralom">http://oracle.communications.ordermanagement.unsupported.centralom</a></td>
</tr>
</tbody>
</table>

After the fulfillment states have been created, you create the mappings in the **Mappings** tab of the Fulfillment State Map editor.

A fulfillment state mapping maps an external fulfillment state to a common fulfillment state. When defining a fulfillment state mapping, you must define when that
particular mapping will be used. Each mapping must specify a single fulfillment pattern, order item, and orchestration sequence, with a single set of orchestration stage and order component combinations. There may be a large number of mappings because wild cards cannot be used.

These criteria are defined in Design Studio and should be specified in the order given. Some of the entries later on the list cannot be set until the earlier ones have been entered.

1. **Fulfillment pattern**: The fulfillment pattern value restricts the fulfillment state mapping to apply only to order components defined on orchestration plans associated with the specified fulfillment pattern. For example, the fulfillment state mappings might be very different between mobile and IP services.

2. **Order item**: The selected value restricts the fulfillment state mapping to apply only to order components responsible for processing the specified order item.

3. **Orchestration sequence**: The available orchestration sequences are those related to the specified order item. The selected value restricts the orchestration stages to which the mapping can apply.

4. **Orchestration stage**: One or more orchestration stages must be specified for the mapping. Any of the orchestration stages in the orchestration sequence can be specified. Use only one orchestration stage per mapping, if possible. Using only one orchestration stage facilitates maintenance of the solution because your decomposition rules may change over time.

5. **Order component**: One order component must be specified for each specified orchestration stage.

You can further restrict the application of the mapping by specifying any of the following:

- **Fulfillment mode**: If specified, the fulfillment mode value, combined with the fulfillment state mapping’s fulfillment pattern value, determines the orchestration plan to which the fulfillment state mapping applies. The fulfillment state mapping is evaluated for order components associated only with the identified orchestration plan. The fulfillment state mapping returned for an item with Cancel fulfillment mode could be very different than that for an item with Deliver fulfillment mode.

- **Properties/property value** combinations: After the order item is selected, one or more order item property value criteria values may be specified. The set of order item properties available for selection are those properties that are defined on the fulfillment state mapping’s selected order item specification. For example, you might have a property called LineType and have different mappings based on whether the value was VoIP Phone or Softphone, as displayed in Figure 4–42.
About Fulfillment States

**Current Fulfillment State:** If a current fulfillment state is specified, the fulfillment state mapping is evaluated only for those order components where the current fulfillment state of the item on the component matches the specified value. This current fulfillment state is taken from the list of common fulfillment states, meaning that it is the target fulfillment state of another fulfillment state mapping or the result of composition rules. You might use this to set a mapped fulfillment state of Failed if that is the current state; if the current state is In_Progress, the new state might be Complete.

**Modeling Fulfillment State Composition Rule Sets**

Orders contain one or more order items. Order items can in turn be fulfilled by one or more order components and also contain other order items using the order item hierarchy. See "About Order Item Hierarchy" for more information.

There is a fulfillment state assigned to the order and order item as a whole that takes into account all of the fulfillment states of its immediate children. This is referred to as a composite fulfillment state.

Fulfillment state composition rules for the order item are defined in order item fulfillment state composition rule sets. These rules aggregate the mapped fulfillment states for any order components that fulfill the order item and also the fulfillment states of any child order items of the order item.

Fulfillment state composition rules for the order are defined in order composition rule sets. These rule sets aggregate the composite fulfillment states of the root-level order items.

The configuration processes for order fulfillment state composition rule sets and order item fulfillment state rule sets are similar.

A fulfillment state composition rule set contains rules, which in turn contain conditions, as shown in Figure 4–43.
You use composition rules to specify the fulfillment state for the order or order item when all of the conditions are met (logical AND). If there are separate situations that can result in the same fulfillment state (logical OR), create separate rules that evaluate to the same fulfillment state.

For example, say that you have one condition that specifies that all of the input fulfillment states must be FAILED, and another condition that specifies that all of the input fulfillment states must be CANCELLED. Both of these conditions should result in a fulfillment state of NOT_DONE. You also have another condition that allows a mixture of FAILED and CANCELLED states that should result in a fulfillment state of CHECK_STATUS. In this case you would need three separate rules. The last condition requires its own rule because it results in a different fulfillment state. The other two conditions each require their own separate rule because it would never be possible for both of those conditions to be met at the same time.

The fulfillment state condition based on the input fulfillment states is the same for both order item composition rule sets and order composition rule sets. It allows the inclusion (or exclusion) of one or more fulfillment states according to whether any, all, or none of the input fulfillment states are in a selected list of fulfillment states.
The fulfillment states selected in the condition are constrained by a conjunction that must be true for the condition to evaluate to true. The available conjunctions are:

- **Any**: The condition requires at least one of the input fulfillment states to match one of the selected fulfillment states.
- **All**: The condition requires all of the input fulfillment states to match the selected fulfillment states.
- **None**: The condition requires that none of the input fulfillment states match any of the selected fulfillment states.

The list of fulfillment states that can be assigned as mapped fulfillment states and the list that can be assigned as composite fulfillment states is the same list. The common fulfillment states created in the Fulfillment State Map editor **States** tab apply to both the mapped and composite fulfillment states. Therefore, when you are generating a composite fulfillment state, the list of fulfillment states that you can choose in this condition is the list of common fulfillment states. (See "Modeling Fulfillment State Maps" for more information about this list.)

### Order Item Fulfillment State Composition Rule Sets

In addition to the fulfillment state conditions discussed above, in order item fulfillment state composition rule sets you can set order item property values that must be present for the composition rule to evaluate to true. If both Any/All/None and property values are defined, both must be true for the composition rule to evaluate to true.

### Order Fulfillment State Composition Rule Sets

In addition to the common fulfillment state-related criteria discussed above, in order fulfillment state composition rule sets you can also specify an XQuery expression that must evaluate to true for the condition as a whole to evaluate to true. For example:

```
/GetOrder.Response/_root/OrderHeader/AccountIdentifier > 0
```
This XQuery expression provides the same functionality available to XQuery expressions exposed elsewhere in Design Studio, including access to order data, access to behavior instances, and external configuration.
About Tasks and Processes

This chapter describes how Oracle Communications Order and Service Management (OSM) uses processes and tasks to fulfill order requirements.

Before reading this chapter, read "Order and Service Management Overview" for more information.

About Tasks and Processes

A **task** is a specific activity that must be carried out to complete the order; for example, if an order needs to verify that an ADSL service was activated, you might model a task named Verify ADSL Service. Tasks can be manual or automated. Manual tasks must be processed by an order manager, using the Task Web client. Automated tasks run automatically with no manual intervention. See "About Tasks" for more information.

A **process** is a sequence of tasks. A process includes tasks, subprocesses, the sequence in which tasks are run, and ways to control how the tasks are run; such as rules and delays. Processes allow you to break down the work required to execute and fulfill an order into functional tasks, which can be distributed to various systems and people to be completed in a controlled manner. See "Understanding Processes" for more information.

**Figure 5–1** shows a process in Oracle Communications Design Studio. The process adds a DSL service. Each box represents a task in the process. Automated tasks include an arrow in the icon.
About Tasks

The process shown in Figure 5–1 includes the following tasks:

1. An automated task verifies that an ADSL service is available.
2. A manual task assigns a port. If the port is not available, another manual task runs that adds capacity. The process then returns to the task for assigning a port.
3. An automated task activates the service.
4. A manual task sends a customer survey.
5. After the customer survey is sent, a manual task verifies the order. The process is complete.

About Tasks

A task is a specific activity that must be carried out to complete the order. For example, to complete an order for a telephone service, the following tasks might be required:

- Find a telephone number.
- Assign a telephone number.
- Ship a telephone device.
There are two types of tasks: automated and manual.

- **Automated tasks** require no manual intervention. Automated tasks are implemented using automation plug-ins. Automated tasks are used to handle internal interactions with external fulfillment systems, such as billing systems, shipping systems, activation systems, and other fulfillment systems. OSM processes typically include more automated tasks than manual tasks.

  In Figure 5–1, the automated tasks are used for verifying and activating a DSL service; for example, the automated task Activate DSLAM is used to interact with an activation system to activate the ADSL service.

  When you create an automated task, you must also configure an automation plug-in to perform the operation. For example, you could define a task called Verify Address. An automation plug-in can be configured to send order data to a third-party address verification system whenever an order reaches the Verify Address task. The third party returns an address verification to OSM, completing the task. Design Studio provides several built-in plug-ins and a custom template to develop your own plug-in. See “About Automated Tasks” for more information.

- **Manual tasks** must be run from the Task Web client. These tasks involve manually entering or reviewing information. Manual tasks typically include tasks that require decision-making, when there are multiple choices for how to proceed with order processing. Fallout management typically uses manual tasks. In Figure 5–1, the manual tasks include sending a customer survey and verifying the order at completion.

  To run manual tasks by using the OSM Task Web client, an order manager works from a list of manual tasks called a worklist. Tasks are assigned to the worklist by assigning tasks to workgroups. To complete a task, an order manager typically enters data or reviews the data, and clicks a button to indicate that the task is complete.

  Figure 5–2 shows manual tasks displayed in the Task Web Client.

![Figure 5–2 Manual Tasks Displayed in the Task Web Client](image)

Automated and manual tasks share many of the same modeling activities, such as defining task data. Some configuration steps, however, are specific to each task type. For example, you model behaviors for manual tasks only.

Most tasks perform actions that fulfill order data; for example, activating a service. You can also use tasks for other purposes. For example:

- Each order includes a creation task. This task includes all of the data required for the order when it is created. See “About the Creation Task” for more information.

- You can use query tasks to specify which data is displayed in the OSM Web clients. See “About Query Tasks for OSM Clients” for more information.

- An activation task is a type of automated task, designed specifically to interact with the Oracle Communications ASAP product and the Oracle Communications IP Service Activator product. You can include these tasks in your process flows to
activate services in your network by using ASAP or by using IP Service Activator. See “Understanding Activation Tasks” for more information.

**Tip:** You can insert manual tasks in a process that function as breakpoints for debugging. This allows you to control a process when you test it.

### About Task Data

Each task includes a set of data, which you specify when modeling the task. Figure 5–3 shows an example of task data that might be included in a task for assigning a port.

![Task Data in Design Studio](image)

The data included in a task is data relevant to the function of the task. Table 5–1 shows some example tasks and the task data they include.

<table>
<thead>
<tr>
<th>Task</th>
<th>Task Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add capacity</td>
<td>Bandwidth</td>
</tr>
<tr>
<td>Send customer survey</td>
<td>Name, phone number, address</td>
</tr>
<tr>
<td>Query task (to display data in the</td>
<td>Name, phone number, bandwidth, port ID</td>
</tr>
<tr>
<td>Task Web client)</td>
<td></td>
</tr>
</tbody>
</table>

When you model a task, you assign it to an order. The available task data is limited to the data that the order requires. At run time, task data can be entered by an OSM user, provided on an incoming order, or provided from a previous task in the order.

Figure 5–4 shows task data defined in a task in Design Studio and how the data is displayed in an order in the Task Web client.
When modeling orders, it is common to include the entire XML representation of the order in the order data as an XML data type. If you include the XML data, consider defining smaller XML elements for storing sections of a sales order rather than including a single XML data type that contains the entire sales order. This allows you to map only the parts of the order that are needed for each task. Including the XML representation is typically done only in the modeling process as an aid to development.

In addition to defining the data included in each task, you can use behaviors in manual tasks to manipulate many aspects of how the data is displayed, formatted, and validated. For example, you can specify if data is read-only, or you can modify the value of the data in a task. See “About Behaviors” for more information.

**Generating Multiple Task Instances from a Multi-Instance Field**

Some tasks require multiple task instances to complete. For example, you might need to create three task instances to retrieve three different address fields. To accomplish this, you designate a field as a pivot data element for the task. When OSM runs the task at run time, the system generates a separate task instance for each separate instance of the pivot data element in the order. The system creates as many instances of the task as there are instances of the data field or data structure, up to the maximum number defined for the field. This feature works for a structure of data also. For example, if the address is a structure called Address, with nested elements of Street,
City, and Postal Code, the system generates an instance of a task for each instance of the structure. The data that is visible to the task instance will be restricted to data structure that it is for, and that task will not have visibility to the other instances of the data.

---

**Note:** OSM compensation processing does not support task pivot data elements.

---

### About Extending Tasks

You can create a new task by extending from an existing task. The new task inherits all of the data, tasks, rules, and behaviors of the base task from which it was extended. Changing something on the base task is reflected in all tasks extending from it.

For example, if you have multiple tasks that all require the same data subset, you can create a base task that contains this data, then extend from this task to create as many new tasks as necessary. You can add new data and behaviors to each of the new tasks to create unique task and behavior functionality. Extending tasks can significantly reduce duplication and maintenance.

### About Task States

A task **state** determines the condition of a task in a process. Every task in OSM has a set of states that reflect the lifecycle of the task. The minimum states required for a task to be completed are Received, Accepted, and Completed.

Changing the state of a task changes how it can be worked on; for example, changing the state to Assigned restricts who can work on the task. Table 5–2 shows the predefined task states. All of them are mandatory.

**Table 5–2 Task States**

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Received</td>
<td>The task has been received by a workgroup and is waiting to be accepted.</td>
</tr>
<tr>
<td>Assigned</td>
<td>The task has been assigned to a specific OSM user. Tasks that are in the assigned state cannot be worked on by other users. Users with assignment privileges can re-assign tasks if needed.</td>
</tr>
<tr>
<td>Accepted</td>
<td>An order manager has accepted the task and is working on it. A task can be accepted by a user by explicitly changing the state of the task to Accepted or by editing the order. Tasks that are in the Accepted state cannot be worked on by other users unless the state is returned to the Received state.</td>
</tr>
<tr>
<td>Completed</td>
<td>The task has been completed by a user or an automation plug-in. A task that has been completed no longer appears in a user's worklist.</td>
</tr>
</tbody>
</table>

The Received, Assigned, and Accepted task states are the only task states provided by OSM. However, you can create your own task states. For example, you can define different types of suspended task states; for example, Waiting for Client Confirmation.

### Using Task States to Manage Orchestration Dependencies

You can use task states when defining orchestration dependencies. For example, you can specify to wait until a task has reached a specified state before an order component can be processed. See "About Dependencies" for more information.
About Task Statuses

A task status represents the result of the task, and determines how a task can transition to a next step in the process. For example, if a task can either succeed or fail, the task status can be either success or failure, and each might transition to a different task in the process.

The statuses that you define in Design Studio appear as task transition options in an OSM Web client. Figure 5-5 shows part of the Task Web client. The list showing Delete and Submit shows the possible statuses for the task. Task statuses can also be displayed in the Task Web client as buttons; for example, the Back and Next buttons.

Figure 5-5  Task Statuses in the Task Web Client

Figure 5-6 shows part of a process defined in Design Studio. The Assign Port task has two possible statuses: Port Available and Port Unavailable. If the port is available, the status is changed to Port Available. If not, the status is changed to Port Unavailable.

If the status is changed to Port Unavailable, the process transitions to the Add Capacity task. This task has two possible statuses: Next and Failure. If the capacity cannot be added, the task status is changed to Failure, and the process ends. If the capacity can be added, the task status is changed to Next and the process transitions back to the Assign Port task.
The default statuses for a manual task are:

- Back
- Cancel
- Finish
- Next

The default statuses for an automated task are:

- Failure
- Success

The default statuses for activation tasks are:

- Success
- Failed
- Update Order Failed

You can also select from additional predefined statuses:

- Delete
- False
- Rollback
- Submit
- Failed
- True

You can also define your own task statuses.

**Using Task States and Statuses to Trigger Event Notifications**
You can use task states and task statuses to trigger event notifications. For example, changing to the Failure status can trigger a notification to a fallout specialist. See "About Event Notifications" for more information.

**Task Statuses and Constraint Behavior Violation Severity Levels**
You can use task statuses in combination with Constraint behaviors to specify the conditions under which a process can make a transition to the next task or activity in the process.

You use Constraint behaviors to validate order data. For example, you can validate that a telephone number has 10 digits or ensure that a numeric value is between 0 and 100.

Constraint behaviors include a **Display as** violation severity level and a message to be displayed in the Task Web client when a constraint behavior violation occurs. When Save is clicked in the Task Web client Order editor, the save action taken depends on the constraint behavior violation severity level.

**Table 5–3  Constraint Behavior Actions**

<table>
<thead>
<tr>
<th>Constraint behavior violation severity levels, from highest severity to lowest</th>
<th>Message display:</th>
<th>When Save is clicked:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Critical</td>
<td>OSM displays the message in bold red text, with the label &quot;ERROR&quot;.</td>
<td>The data is not saved.</td>
</tr>
<tr>
<td>Error</td>
<td>OSM displays the message in red text, with the label &quot;ERROR&quot;.</td>
<td>The data is saved.</td>
</tr>
<tr>
<td>Warning</td>
<td>OSM displays the message in yellow text, with the label &quot;WARNING&quot;.</td>
<td>The data is saved.</td>
</tr>
<tr>
<td>Valid</td>
<td>OSM displays the message in green text, with the label &quot;INFO&quot;.</td>
<td>The data is saved.</td>
</tr>
</tbody>
</table>

**Using Task Statuses to Control Process Transitions**
You can use task status Constraint values to determine how Constraint behavior violation severity return values affect whether or not a process can make a transition to the next task or activity. Task status Constraint values include:

- Critical
- Error
- Warning
- None
- Valid
The task status Constraint value represents the highest allowable Constraint behavior violation value with which the task transition will be allowed to occur. When Update is clicked, in the Task Web client Order editor, the transition action taken depends on the task status Constraint severity value in conjunction with the Constraint behavior violation severity level, if any.

For example, if the task status Constraint value is set to Error, then Error is the highest allowable Constraint behavior violation value with which the task can be transitioned. The task is not allowed to transition if a Constraint behavior violation of Critical occurs, but is allowed if an Error, a Warning, or a Valid Constraint violation occurs.

The following table explains whether task transition is allowed for all combinations of Constraint behavior violation severities and task status Constraint values.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Critical</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Error</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Warning</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Valid</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>None</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

### Specifying the Expected Task Duration

You can specify the expected length of time to complete a task. This information can be used to trigger jeopardy notifications and for reporting. See ”About Notifications” for more information. This information is also used by OSM to calculate the order component duration.

You can specify the length of time in weeks, days, hours, minutes, and seconds. The default is one day.

You can also calculate the duration based on your workgroup calendars. If you have more than one workgroup with different calendars all responsible for the same task, the calculation is based on the first available workgroup that has access to the task. This ensures that a the task only exceeds it's duration based on the workgroup calendar time.

For example, there might be a task with an expected duration of two hours, and the workgroup that processes the task only works 9 AM - 5 PM Monday to Friday as indicated on their workgroup calendar. If such a task is received at 4 PM on Friday, then the expected duration of the task will expire at 10 AM Monday, as there was only two hours of the workgroup calendar time that had elapsed (4-5 PM Friday, then 9-10 AM Monday). This ensures that notifications and jeopardies are triggered appropriately.


### Specifying the Task Priority

Task priority is the same as the order priority unless a priority offset is defined. Priority of orders and their tasks becomes effective when the system is under heavy
load, ensuring that high priority orders and tasks are not starved of resources by lower priority orders and tasks.

You define the task priority as an offset from the priority of the order itself. This specifies the priority of the task in relation to other tasks in the order.

For example, if the order is created at priority 6, and this task is assigned a priority offset of -2, then this task would run at priority 4 while tasks in the order with no offset would run at priority 6. Similarly, you could assign a task a priority offset of +2, which would mean that the task would run at a slightly higher priority than other tasks in the order.

See "About Specifying the Order Priority" for more information.

Managing Tasks During Amendment Processing and Task Compensation

See "Managing Changes to Orders" for information about amendment processing.

You can define how amendment processing affects each task; for example, if it should be redone or undone. To model amendment processing, you specify if a task should be re-evaluated and, if so, what should be done (redo or undo then do).

Tasks must be re-evaluated when:

- An order amendment changes data that is flagged as significant.
- The task is affected by compensation performed on another task. For example, an amendment changes the data for Task A, which when redone changes the output data that is used as input into the data for Task B, requiring Task B to be redone.

If a task must be re-evaluated, you can specify the following actions:

- **Do Nothing**: If nothing must be done, you can specify to skip the task. You typically choose this option if the change to the order is handled by a similar task downstream in the process or if there is no functional need to compensate for the task; for example, if there is a task to send a marketing brochure, this may not have any compensation necessary.
- **Redo**: You can specify to redo the task (this is the default). Redoing performs Undo and Do operations in a single operation. This option is recommended, because it performs the fewest number of operations.
- **Undo then do**: You can specify to undo and then redo the task. This option undoes the task and all successor tasks, rolls back all order changes, then performs the Do operation again. This is useful if you need to rollback all subsequent order changes and re-perform the task from the beginning.
- You can specify to run a task manually or automatically during compensation. For example, you can specify that an automated task can only be redone or undone manually. This is useful for gradually introducing amendment processing into a solution.

You can also specify how to handle the task if it is no longer needed. A task can no longer be needed in two cases:

- When the order is canceled.
- When the branch on which the task is located becomes obsolete. A branch becomes obsolete when the redo processing causes a task or subprocess to exit with a different completion status and start a new branch.

In both scenarios, the system rolls back the order changes. For each task, you can specify that OSM either roll back the task automatically or run a manual undo task.
The manual undo task must be completed in the Task Web client to allow OSM to roll back the canceled or obsolete task.

**Controlling Who Can Process a Task**

You can assign roles to each task and to each processing type that can be performed on a task; for example, Do, Redo, and Undo. For example, you can restrict basic order processing personnel from redoing and undoing tasks and allow those operations only for fallout specialists. Roles are also used for managing jeopardy notifications for tasks.

See the discussion of roles and permissions in *OSM System Administrator’s Guide* for more information.

**Specifying How to Assign Tasks to OSM Users**

There are two approaches to assigning tasks to users in OSM:

- A work offer approach, which is by role, where tasks are associated to a role and users performing that role may select tasks from their worklist to work on them.
- A work assign approach, where a task assignment algorithm is used to specifically assign each task to a user performing the role.

For each manual task, you can specify how it is assigned to an OSM user for completion. You can use the following methods:

- **Round robin** assignment automatically assigns tasks to users in a workgroup alphabetically by user name.
- **Load balancing** assignment automatically assigns users in a workgroup to balance the workload across users, based on the number of tasks assigned to each user. The user with the least number of tasks is assigned the task.

You can also create custom automatic assignment methods. For example, you might specify that the first task received is the first one assigned or that the last task received is the first one assigned.

Before you can use a custom task assignment algorithm in an OSM cartridge, you must develop then deploy custom task assignment algorithm Java code. The OSM cartridge management tool (CMT) is an Ant based application that you must use to deploy a custom task assignment algorithm to the OSM WebLogic Server.

OSM provides the `OSM_home/SDK/Samples/TaskAssignment/code/CustomizedTaskAssignment.java` sample task assignment algorithm as part of the selectable **SDK Samples** OSM installation component as a reference to help develop custom task assignment algorithms. This sample implements the com.mslv.oms.behavior and the com.mslv.oms.behavior.taskassignment OSM API packages and the java.util Java SE package. For more information about the classes in the OSM packages extract the `OSM_home/SDK/oms7.2.x-javadocs.zip` OSM Java docs (where x is the software build numbers). For more information about the java.util Java SE package, see the Java SE documentation.

**Deploying a Custom Task Algorithm using the OSM Cartridge Management Tool**

Before you can use OSM CMT to deploy a custom task assignment algorithm, ensure that:

- You can access and reference a WebLogic Server and ADF installation home directory from the OSM CMT build files. See *OSM Installation Guide* for version information.
You must download and install Ant. See OSM Installation Guide for version information.

You install the SDK Tools and the SDK Samples components using the OSM installer. You do not need to install the other options. See OSM Installation Guide for more information about using the OSM installer.

You have created a custom task assignment algorithm. See the OSM_home/SDK/Samples/TaskAssignment/code/CustomizedTaskAssignment.java reference sample for more information about creating a custom task assignment algorithm.

To deploy a custom task algorithm to an OSM server using OSM CMT:

1. From a Windows command prompt or a UNIX terminal, go to WLS_home/server/lib (where WLS_home is the location of the base directory for the WebLogic Server core files).

2. Create a WebLogic client wlfullclient.jar file that OSM CMT uses to communicate with the OSM WebLogic server:
   
   ```
   java -jar wljarbuilder.jar
   ```

3. Copy the following files required by OSM CMT to the Ant_home/lib folder (where Ant_home is the location of the Ant installation base directory).
   - WLS_home/server/lib/weblogic.jar
   - WLS_home/server/lib/wlfullclient.jar
   - MW_home/modules/com.bea.core.descriptor.wl_1.2.0.0.jar (where MW_home is the location where the Oracle Middleware products were installed.)
   - OSM_home/SDK/deploytool.jar
   - OSM_home/SDK/Automation/automationdeploy_bin/automation_plugins.jar
   - OSM_home/SDK/Automation/automationdeploy_bin/xmlparserv2.jar
   - OSM_home/SDK/Automation/automationdeploy_bin/commons-logging-1.1.jar
   - OSM_home/SDK/Automation/automationdeploy_bin/log4j-1.2.13.jar

4. Set the following environment variables and add them to the command shell’s path:
   - ANT_HOME: The base directory of the Ant installation.
   - JAVA_HOME: The base directory of the JDK installation.

   For example, for a UNIX or Linux Bash shell:
   ```
  ANT_HOME=/home/user1/Middleware/modules/org.apache.ant_1.7.1
JAVA_HOME=/usr/bin/local/jdk170_51
PATH= $ANT_HOME/bin:$JAVA_HOME/bin:$PATH
export ANT_HOME JAVA_HOME PATH
   ```

   For example, for a Windows command prompt:
   ```
   set ANT_HOME=c:\path\to\oracle\home\Middleware\modules\org.apache.ant_1.7.1
set JAVA_HOME=c:\path\to\oracle\home\Middleware\jdk170_51
set PATH=%ANT_HOME%\bin;%JAVA_HOME%\bin;%PATH%
   ```

5. Open the OSM_home/SDK/Samples/config/samples.properties file.
6. Set the following variables:
   - Set `osm.root.dir` to the OSM installation base directory.
   - Set `oracle.home` to the Oracle Middleware products base directory.
     For example, for a UNIX or Linux Bash shell:
     
     ```
     /home/oracle/Oracle
     ```
     
     For example, for a Windows command prompt:
     
     ```
     C:/Oracle
     ```

7. Copy the custom task assignment algorithm file you created to `OSM_home/SDK/Samples/TaskAssignment/code`.

8. Open the `OSM_home/SDK/Samples/TaskAssignment/code/build.properties` file.

9. Set the following variables:
   - Set `weblogic.url` to the WebLogic Administration Server URL. The format is:
     
     ```
     t3://ip_address:port
     ```
     
     where:
     - `ip_address` is the IP address for the WebLogic Administration Server.
     - `port` is the port number for the WebLogic Administration Server.
   - Set `weblogic.domain.server` to the name of the WebLogic Administration Server.
   - Set `weblogic.username` to the WebLogic Administration Server user name.
   - Set `webLogicLib` to the path to the `WLS_home/server/lib` folder.
   - Set `ejbname` to the Enterprise Java Bean (EJB) name for the task assignment behavior.
   - Set `ejbclass` to the class name for the task assignment behavior.
   - Set `jndiname` to the Java Naming and Directory Interface (JNDI) bind name for task assignment behavior.
   - Set `targetfile` to the deploy target file name for a target file that does not contain a suffix like `.ear` or `.jar`.

---

**Note:** `ejbname`, `ejbclass`, `jndiname`, and `targetfile` are preconfigured to deploy the `OSM_home/SDK/Samples/TaskAssignment/code/CustomizedTaskAssignment.java` sample task assignment algorithm. Replace these default values with those for the custom task assignment algorithm.

---

10. Create and deploy Design Studio cartridge that includes a manual task that you want to associate to the custom task assignment algorithm. You can associate the custom task assignment algorithm in the `Details` tab of the manual task using the `Assignment Algorithm` and `JNDI Name` fields. See the Design Studio Help for more information.
11. From the Windows command prompt or UNIX shell, type:

```plaintext
ant
```

The Ant script begins to run.

12. When the ant script reaches **Input WebLogic Password for user weblogic...**, enter the WebLogic Administration Server password.

The ant tool compiles, assembles, and deploys the custom task assignment algorithm to the OSM WebLogic Server.

---

**Note:** You can also individually compile, assemble, deploy, or undeploy using the following Ant commands:

- `ant compile`
- `ant assemble`
- `ant deploy`
- `ant undeploy`

---

**About Automated Tasks**

You can create an automated task to connect to a database, transform some data, or communicate with an external system. Most tasks are automated.

Automated tasks and automated notifications trigger automation plug-ins. An **automated task** triggers a specific automation plug-in when the task transitions to the **Received** state. This occurs when an instance of the task is created in OSM. An automated task receives a message internally from OSM, and the information contained in the internal message is made available to the automation plug-in. The automation plug-in uses that information to perform custom logic, update OSM, or send a message to an external system. If a message is sent to an external system, the plug-in is capable of receiving a message externally from the system with which it initiated communication.

When you create an automated task, you must also configure at least one automation plug-in to perform the operation. An automated task might have a single automation plug-in associated with it (for example, to interrogate the task data, perform some calculation, and update the order data), or it might have multiple automation plug-ins associated with it (one to send information to an external system, one to receive replies from the external system, and another to perform some calculation, update the order, and transition the task).

When referring to an automation, the following meanings can apply:

- The automation plug-in code that you create and associate with an automation task in Design Studio.
- The instance of an automation plug-in that the OSM run-time server creates in response to an event that triggers an automation. OSM creates and reuses such instances as required when processing automated tasks. OSM maintains these plug-in instances even if the instance is no longer required and only creates additional plug-in instances when the current pool of instances are insufficient to
handle the number of incoming orders. OSM only destroys automation plug-in instances in the following scenarios:

- When you shut down the OSM server, OSM destroys all plug-in instances.
- When you undeploy a cartridge, OSM destroys all plug-in instances associated to the undeployed cartridges.
- When OSM detects an error condition in the instance, OSM destroys the instances.

Automated tasks are supported by the following plug-ins:

- **XSLT Plug-in.** A plug-in that supports an XSLT script that generates an outbound message and processes an in-bound message.
- **Email Plug-in.** A plug-in that sends notifications to external systems through email.
- **JDBC Plug-in.** A plug-in that uses Java Database Connectivity (JDBC) to retrieve or update data in the database.
- **XQuery Plug-in.** A plug-in that runs XQuery expressions to generate an outbound message and process an in-bound message.

You can create custom plug-ins. See the Design Studio online Help and OSM Developer’s Guide for more information.

**Understanding Automation Plug-ins**

There are two types of predefined automation plug-ins: automator and sender.

- Automator plug-ins receive information from OSM or an external system and perform some business logic. They can also be configured to update the order data.
- Sender plug-ins send information from OSM to an external system. They perform business logic and can update orders. Additionally, they can produce outgoing JMS or XML messages to be delivered to an external system.

Both plug-in types can be implemented using XSLT or XQuery.

An automated task can be associated with either a single automation plug-in or with multiple automation plug-ins. For example, you might model multiple automation plug-ins for an automated task to communicate with external systems. These could include:

- A sender plug-in to send task data to an external system for processing.
- An automator plug-in to receive the reply from the external system and update the order data.
- A sender plug-in to send the external system a message to begin processing.
- An automator plug-in to receive processing messages from the external system, update the order, and transition the task.

Although automation concurrency is normally highly beneficial in order processing, in some cases, you may need to restrict the number of automation plug-ins that OSM can create at one time to avoid situations where excessive automation concurrency creates problems. For example, situations may occur where the design-time logic allows for two or more automated tasks to run plug-ins at the same time, but at run-time, these automation plug-ins must process sequentially (in no particular order) to avoid possible race conditions or excessive message retries. These kinds of situations can occur in automation plug-ins triggered internally, but most often in automation
plug-in exchanges between OSM and external systems where there is a limitation in
the external system. Another possible scenario is when OSM sends multiple messages
to the same system that returns multiple different fulfillment state updates that should
be processed sequentially (also in no particular order). You can limit the number of
automation plug-ins that OSM runs at one time using the order automation concurrence control (OACC) function. Using OACC, you can limit some or all
automation plug-in instances for all orders of a specific type, for all orders supported
by a specific cartridge or cartridge and version, or for all orders processed in an OSM
server. For more information, see *OSM System Administrator’s Guide*.

Understanding Activation Tasks

**Important:** Before you can model Activation tasks in Design Studio,
you must install the Design Studio for Order and Service Management
Integration feature. This feature includes the Design Studio for
Activation feature for integrating with ASAP and IP Service Activator.
To model activation tasks, you must also install the Design Studio for
Activation feature.

An activation task is a type of automated task that provides an integration between
OSM and ASAP or OSM and IP Service Activator. By using activation tasks, you can
model a process flow that includes one or more tasks that activate services in a
network.

**Note:** Activation tasks include many of the same properties as
automated tasks; for example, you can assign permissions, define the
task data, and configure notifications. However, you also configure
activation-specific data elements, such as how to map data sent to and
received from ASAP or how to map data sent to and received from IP
Service Activator.

The interaction between OSM and ASAP or OSM and IP Service Activator is
established through a Web request and response, which you configure by mapping
OSM task data to ASAP parameters or to IP Service Activator parameters.

1. OSM transforms order data into an operations support system through Java
   (OSS/J) message or a Web Service message and sends it to ASAP or to IP Service
   Activator. To model this, you configure service action request mapping, to map
   OSM data to ASAP data or to map OSM data to IP Service Activator data. See
   "About Service Action Request Mapping" for more information.

2. ASAP or IP Service Activator receives the data, activates the service, and returns a
   success or failure status to OSM. To allow OSM to handle the returned data, you
   model service action response mapping. See "About Service Action Response
   Mapping" for more information.

Other elements specific to activation tasks are:

- You can configure state and status transitions for completion events and
  exceptions returned by ASAP or IP Service Activator.

- You can configure how to handle amendment processing with activation tasks.
If you are sending JMS OSS/J messages, Oracle recommends that you configure JMS store and forward (SAF) queues to manage the connection to ASAP or to manage the connection to IP Service Activator.

If you are sending Web Service messages, Oracle recommends that you configure Web Service SAF queues to manage the connection to ASAP or to manage the connection to IP Service Activator.

**About Service Action Request Mapping**
You send fulfillment data to ASAP or to IP Service Activator as a service action request. To model a service action request, you map OSM header data (information that applies to the customer or to all order line items on the order) and OSM task data to the following service order activation data:

- **Activation order header**: Information that applies to the entire work order.
- **Service action**: Information that is required to activate a service.
- **Global parameters**: Information that you define once and which applies to multiple service actions.

*Figure 5–7* shows service action request modeling in Design Studio.

**Figure 5–7  Service Action Request Modeling in Design Studio**

**About Service Action Response Mapping**
After ASAP or IP Service Activator activates a service, it returns information to OSM. You create data structures in OSM to contain the response information returned from ASAP or IP Service Activator. For each event and exception returned by ASAP or IP Service Activator, you select the ASAP or IP Service Activator data that you want to retain, then identify the OSM data structure to which that data is added. When ASAP or IP Service Activator returns an event or exception, OSM updates the order data with the ASAP or IP Service Activator data that you specified.
Tip: The amount of response data from ASAP or IP Service Activator can be very large, though the data that is needed might be small. Parsing large amounts of ASAP or IP Service Activator response data can affect OSM performance. If you notice a reduction in OSM performance due to large amounts of ASAP or IP Service Activator response data, you can specify a condition on specific parameters to limit the ASAP or IP Service Activator response data.

Figure 5-8 shows service action response modeling in Design Studio.

**Figure 5-8 Service Action Response Modeling in Design Studio**

---

**About State and Status Transition Mapping**

You can configure state and status transitions to manage completion events (for example, activation complete) and errors returned by ASAP or returned by IP Service Activator. You can define multiple transitions to model different scenarios for variations in the data received from ASAP or received from IP Service Activator. For example, if an ASAP parameter or IP Service Activator parameter returns the value DSL, you may want the task to transition to a DSL task; when the same parameter returns the value VOIP, you want the task to transition to a different task.

You can define state transitions for user-defined states only; you cannot define transitions for system states, such as Received, Accepted, and Completed. At run time, OSM evaluates the conditions in the order and stops evaluating when a condition evaluates to true. Completion events and errors must include a default transition in case all specified conditions fail.
About Activation Tasks and Amendment Processing
You can configure how to manage an activation task if the associated order undergoes amendment processing. The options are:
- Intervene manually.
- Do not perform any revision/amendment.
- Have OSM redo the activation task, using the previously defined request mapping.
- Have OSM redo the task, using different request mapping.

Understanding Processes

A process is a sequence of tasks and subprocesses that run consecutively or concurrently to fulfill all or part of an order. Any number of processes can be defined, consisting of any number or combination of manual and automated tasks. **Subprocesses** are processes that are launched from another process, as opposed to being launched from an order.

Figure 5–9 illustrates a simple process and its tasks, as shown in Design Studio:

*Figure 5–9  Example of an OSM Process*

This process manages the fulfillment of a request for an ADSL service:

1. OSM runs the process. The first task, **Verify ADSL Service**, is an automated task that verifies that the ADSL service exists. For example, the task might run a Web
services operation that reads a database to determine if the service is available at the specified address.

---

**Note:** This example is a simplified example. A more complete process would have a failure path from this task, typically a fallout task to manage the order if the service is not available at that address.

---

2. After verifying that the service is available, the process branches to two tasks that are independent, and can run in parallel:
   
a. The Ship ModemSelf-Install Pkg task sends a shipping order to the hardware provider.

   b. The Assign Port task looks up a port in the inventory system and assigns it.

   If the port is available, the next task is Activate DSLAM. However, if the port is not available, the process transitions to the Add Capacity task, and then back to the Assign Port task.

As the tasks are processed, they are displayed in the Task Web client. Figure 5–10 shows two parallel tasks displayed in the Task Web client. They can be completed separately.

![Figure 5–10 Parallel Tasks Displayed in the Task Web Client](image)

3. After the Assign Port task is finished, the Activate DSLAM task can run. This task contains an OSM integration with ASAP, IP Service Activator, or any third-party activation system to activate the DSLAM.

   The Assign Port task is dependent on the completion of both the Ship Modem self-install Pkg task and the Activate DSLAM task. Therefore, even if the Ship Modem self-install Pkg task completes, the Activate DSLAM task cannot start until the Assign Port task is finished.

4. When the activation is complete, the next two tasks send the customer survey and require that an OSM user verifies the order to ensure it is complete. After these two manual tasks are completed, the order is complete.

Any of the tasks in this process could be configured as automated tasks. For example, the Assign Port task could be an automated task if there was an integration with the inventory system, and the inventory system was able to return the port number for the service.

For a detailed example of a process, see *OSM Developer’s Guide*.

---

**About Workflow Processes and Workstream Processes**

A workflow process is a standard OSM process. A workstream process is a special type of process that is run differently from a workflow process in the Task Web client.
With a workflow process, the Task Web client displays the worklist after it completes each task. This is because a workflow process is intended to distribute work among different users in different workgroups. The next task in the process might be handled by a different user.

With a workstream process, OSM displays the order editor page for the next task automatically without first returning you to the worklist. Next and Back buttons allow you to navigate through the sequence.

A workstream process can include manual and automated tasks. If an automated task occurs, OSM processes it and displays a message indicating that processing is taking place. While automated processing is occurring in the workstream, you can return to the worklist to work on other tasks. The automated task in the workstream will continue to progress to completion. This is useful if the automated task in the workstream takes some time.

After the automated task finishes, and the next task becomes available, any user in the workgroup can pick up the workstream from that point. When the final task in a workstream completes, OSM returns you to the worklist. OSM automatically displays the order editor page for the next manual task in the workstream to the user.

About Modeling Processes

Processes have a single entry point and one or more exit points. When you create the process structure, you must place the tasks in the order in which the process is to complete them.

In addition to running tasks and subprocesses, you can control how a process runs; for example, specify to delay processing a task or redirect the process to another process.

Note: Redirecting a process is useful for backward compatibility, but is not recommended. An order that has been redirected cannot use amendment processing.

About Process Properties

You can specify the following properties to processes:

- The expected duration. This information can be used for jeopardy notifications and for reporting.
- If you want this process to appear in the Process History - Summary Table window in the Task Web client.
- If the process is a workstream process. See "About Workflow Processes and Workstream Processes" for more information.

Figure 5–11 shows process Start properties in Design Studio.
Understanding Processes

**About Process Activities**

You use process activities to design how the process runs. Figure 5–12 shows the Activities options in Design Studio. The example process includes a timer delay between the two tasks.

**Figure 5–12 Process Activities Options in Design Studio**

In addition to the tasks and subprocesses that the process runs, you can control the process by using the following:

- **Rules**
- **Timer delays**
- **Event delays**
- **Joins**
- **Ends**
- **Redirects**

**Rules** evaluate a condition and then specify the next step in the process. For example, a rule task might evaluate the data that describes the geographic region of the order and branch the process appropriately. Rule tasks perform as follows:

- They typically read and evaluate data to determine what to do.
- They always evaluate to true or false.
They are always run automatically, with no manual involvement.

**Timer delays** delay the process until a rule evaluates to true. Timer delays perform as follows:

- The rule is evaluated at specified timed intervals.
- The data evaluated in the rule must be data that is included in the order.
- The rule always evaluates to true or false.
- The delay is always run automatically, with no manual involvement.

**Event delays** delay the process until a rule evaluates to true. Event delays perform as follows:

- The rule is evaluated only when the data specified in the rule changes.
- The data evaluated in the rule must be data that is included in the order.
- The rule always evaluates to true or false.
- The delay is always run by OSM, with no manual involvement.

**Joins** combine a set of flows into a single flow. (Process flows define the sequence of tasks that the process performs. See "About Flows" for more information.) The unified flow can join flows based on all transitions completing or any one transition completing (by selecting All or selecting Any). Selecting Any will create one instance of the flow for each incoming transition.

**Ends** stop the process from continuing.

**Redirects** redirect the process to another task in the same process or to a different process.

---

**Note:** Timer and event delays are not used during amendment processing.

---

**Using the System Date in Delays**

You can create a rule that uses the system data as part of a condition. For example, you can create a rule used in a delay that delays a task transition until the system date is at least the value of a particular order data element of the **dateTime** data type. Figure 5–13 shows a rule that triggers when the system date is at least the value of the date when a particular poll is run.
About Flows

Process flows define the sequence of tasks that the process performs. You can design flows for specific scenarios, including:

- A flow that ends in a successful process completion (Success) or a process failure (Failure).
- Flows for various activities, such as Cancel, Next, and Back.

Figure 5–14 shows how flows appear in a process in Design Studio. In this figure, flows are labeled with the task status; for example, route_to_osm.

Figure 5–14 Flows in a Process

You can control flows in the following ways:

- You can use an order rule to apply conditions that must be met before the flow can continue.
- You can ensure that the system verifies that mandatory fields are present when a task completes. (This option is not available for tasks with a Rollback status.)
- You can specify a reporting status to display in an OSM Web client. This status is tracked in the Web client’s OSM history.

Figure 5–15 shows flow properties in Design Studio.
Figure 5–15  Flow Properties

<table>
<thead>
<tr>
<th>General</th>
<th>routeToProvisioningTask to activationOrderAdslRegion2Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Events</td>
<td>Property</td>
</tr>
<tr>
<td></td>
<td>Condition</td>
</tr>
<tr>
<td></td>
<td>From</td>
</tr>
<tr>
<td></td>
<td>Mandatory Check</td>
</tr>
<tr>
<td></td>
<td>Reporting Status</td>
</tr>
<tr>
<td></td>
<td>Status</td>
</tr>
<tr>
<td></td>
<td>To</td>
</tr>
</tbody>
</table>

About Subprocesses

A process that starts from another process is called a subprocess.

Figure 5–16 shows a process (Process A) that has two branches. In one branch, a task completes successfully and the process ends. In the other branch, the task fails, which starts a subprocess. When the subprocess ends successfully, the calling task is completed and Process A completes.
When you model subprocesses, you specify the following properties:

- If you want the associated tasks to appear in the Process History window in the Task Web client.

- The pivot data element on which OSM spawns individual subprocess instances. For example, if you have a subprocess that creates an email address for every person in a list, you might select the Person data element as the pivot data element, so the subprocess spawns an instance for each person. See "Generating Multiple Task Instances from a Multi-Instance Field" for more information.

- How to display the associated tasks in the Task Web client. For example, you can display them sequentially, sorted, or unsorted.

- The process to run, based on rules.

- How the subprocess handles exceptions. For example, you might have a process called create_vpn. Within that process, there is a subprocess called validate_address. The subprocess validate_address can throw an exception when an address is invalid. Using the exception mapping functionality, you can instruct the parent process and subprocesses to take specific actions when the subprocesses throw exceptions. Exception mapping enables you to indicate whether the parent process create_vpn should terminate all of the invoked instances, terminate only the offending instance, or ignore the exception altogether.
Understanding Parallel Process Flows

There are two ways to model parallel processes:

- **Subprocesses branching from a task.** This allows multiple tasks to run within the same time frame. Parallel flows can be rejoined at an appropriate point if needed. Typically, there are no dependencies defined between parallel flows, but whether these tasks actually run simultaneously depends on the order data, how order tasks are fulfilled, and other factors.

- **Subprocesses running from a pivot data element.** Multi-instance subprocesses are subprocesses that can be instantiated multiple times. When a subprocess has a pivot data element defined, multiple instances of the subprocess, running in parallel, are created. For example, if the pivot data element for a subprocess is defined as `interested_party`, and an order contains three instances of interested_party, each containing a different person’s name and contact information, OSM creates three separate instances of the subprocess, one for each set of data.

When planning your order specifications, give careful consideration to which data you make available to each parallel process. Excessive and unnecessary data can have negative impacts on performance, and on usability if manual tasks are involved. Also, make sure to flag data as non-significant if the data is not needed for revision orders. By default, OSM assumes that all data is significant.

**About Amendments and Multi-Instance Subprocesses**

An amendment to an order on which some of the data affecting a multi-instance subprocess has changed can cause *all* subprocess instances to be redone, instead of only directly affected subprocesses to be redone. This can result in unneeded processing for the subprocesses with no data changes.

In amendment processing with multi-instance subprocesses, it is important to contain compensation to only the subprocess instances that require compensation. This is achieved by specifying a key. You specify a key in the **Key** subtab on the **Order Template Node** editor for the data element specified as the pivot data element of the subprocess in the order template. When a key is specified for a subprocess, OSM maps the revised data to the current data using the key field and redoes only the subprocess that was affected.
This chapter describes how to use minimum processing duration, life-cycle policies, and order events to manage the life cycle of Oracle Communications Order and Service Management (OSM) orders.

Before reading this chapter, read "Order and Service Management Overview" to learn about basic OSM concepts.

About Managing the Order Life Cycle

The order life cycle controls when the order starts, and how the order transitions between order states; for example, the conditions that allow an order to be amended.

To manage the order life cycle, you can do the following:

- You can configure an order to complete as close as possible to the requested delivery date. To do so, you need to specify the expected duration of the order components, which is how long you expect the order processing to take. See "About OSM Order Fulfillment Timeline" for more information.

- You can configure OSM order states and transitions to control order behavior within OSM. For example, an OSM user cannot work on new tasks of an order while it is in the Suspended state. You can also use state transitions to specify a point after which revisions are not allowed on an order, a point of no return. The allowable order states and transactions for an order are defined by using an order life-cycle policy. See "About Managing Order States and Transitions" for more information.

- You can configure an order to publish events at specified milestones. See "Order Life-Cycle Events" for more information.

About OSM Order Fulfillment Timeline

OSM can process orders at different times. In many cases, a customer wants an order to be completed as soon as possible, in which case OSM can start processing the order immediately. However, in some cases, the start date of an order should be delayed until a future date. For example:

- A customer might request that a new VoIP service be added at the beginning of the next month, when their current service expires.

- A customer might request the disconnect of an existing service at the end of the current month.

In addition, there may be groups of order items within an order that need to be fulfilled at different times. For example, an order might contain three services, such as
internet, IPTV, and VoIP. The internet and IPTV services might have an immediate requested delivery date, but the VoIP service might only be required at the end of the month, after the customer’s current phone service plan has expired. In this case, you can enable OSM to calculate a time to start fulfilling the VoIP service at a future date that would allow the service to be activated by the requested delivery date: at the end of the month. For more information about enabling calculated start dates, see “About Calculated Order Component Start Dates”.

Different groups of order items may have orchestration dependencies configured that have an impact on when a service gets fulfilled. For example, the internet service might be required before you can activate an IPTV or VoIP service. These dependency scenarios are fixed and take precedence over honoring requested delivery dates. In other words, OSM will only honor a requested delivery date for a service if there is enough time to fulfill that service given the time it takes to perform the fulfillment tasks and any dependencies that might exist between one service and another. In such a scenario, the order completion date will be later than the delivery date requested by the customer.

To accurately calculate when an order should start so that it can meet a requested delivery date, you must determine how long it takes to perform certain tasks contained in the order and you must know when a customer wants a service.

**Important:**  Orders must have an orchestration plan to be able to calculate the order completion date.

When viewing an entire order in the Order Management Web client Summary tab General area, you see the following fields:

- **Order Creation Date:** The date when the order is created in OSM.
- **Expected Order Start Date:** The date when the order is expected to start being processed.
- **Expected Order Completion Date:** The date when the order is expected to be completed.
- **Requested Order Delivery Date:** The date by which the customer requests the order be delivered.
- **Expected Order Duration:** The amount of time the order is expected to take to complete processing.

These fields are used in, or derived from, an orchestration plan algorithm. This algorithm, at its highest level, uses the **Order Creation Date** (for orders that start immediately) or the **Expected Order Start Date** (for future dated orders) in conjunction with the **Expected Order Duration** to determine whether there is enough time to achieve the **Requested Order Delivery** date. If there is enough time, then the **Expected Order Completion Date** is the same date as the **Requested Order Delivery Date**. If there is not enough time, then the **Expected Order Completion Date** is later than the **Requested Order Delivery Date**.

When viewing a specific order item in the Order Management Web client Summary tab General area, you see the following fields:

- **Expected Order Component Start Date:** The date when the order component that processes the order item is expected to start.
- **Expected Order Item Start Date**: The date when the order item is expected to start. This is always the same date as the first Order Component Start Date to start processing the order item.

- **Expected Order Item Completion Date**: The date when the order item is expected to be complete. In some scenarios, an order item may require processing from more than one order component. For example, one order component may provision the service while another performs the billing function. And so the order item completion date must take into account the total time it takes to complete these two order components.

The following sections describe the design-time and run-time elements that you must model so that the orchestration plan algorithm can generate an order fulfillment timeline.

### About Order Item Requested Delivery Date and Order Components

The following sections describe order item requested delivery dates and aspects of order components that relate to how OSM attempts to honor these requested delivery dates.

### How OSM Decomposes and Processes Order Items in Order Components

OSM calculates the order component start dates based on the requested delivery date for order line items in customer orders. This requested delivery date order line item value must be mapped to an order item specification `requestedDeliveryDate` order item property in Oracle Communications Design Studio.

You can model the decomposition of order items into order components that typically share the same function, are destined for the same fulfillment system, and share the same processing granularity. The entity that ultimately processes order items is an executable order component that is linked to a process that contains a sequence of manual and automated tasks that fulfill every order item in the order component.

For example, a group of six order items might be gathered in an executable component that is linked to a process that contains an automated task that generates and sends a service request to an activation system. The service request that the automated task builds would contain all the information from the six order items that the activation system requires to activate services that correspond to the order items in the network.

When OSM has determined the order component start date, all order items in the order component begin processing immediately (regardless of their requested delivery date). Although this can mean that some order items might be delivered early, it ensures that no order items are delivered late.

### About Grouping Order Items in Order Components by Date Range

If order items belong to the same function and go to the same fulfillment system need to be fulfilled on substantially different dates, you can model different order components in Design Studio that execute at different stages or within the same stage, but that have different start dates.

In addition, OSM provides Java functions that can be used along with order item hierarchies to further delineate and group order component IDs based on order item requested delivery date. For more information about creating custom component IDs using Java function, see "About Component Specification Custom Component ID XQuery Expressions".
Modeling Order Component Minimum Processing Duration

When you model orders in Design Studio, you need to provide OSM with enough information to be able to meet the order item requested delivery dates with as much accuracy as possible. To do so, you specify a minimum processing duration value that defines how long it typically takes to fulfill all order item within an executable order component. You can model this value at the order component level (see Figure 6–1) or at the fulfillment pattern order component level (see Figure 6–2). OSM always uses the larger of the two values. This duration should take into account the total duration of any manual or automated tasks involved in completing the process. For example, if you know that it takes one week to ship a telephone, you specify one week for the minimum processing duration for an order component that is used for shipping a telephone.

You can specify a different minimum processing duration for each fulfillment mode in the fulfillment pattern. For example, the Deliver fulfillment mode can have a different duration than the Cancel fulfillment mode.

Figure 6–1 shows the duration defined for an order component.

![Figure 6–1 Processing Duration Defined for an Order Component](image)

Figure 6–2 shows the processing duration assigned to an order component when it is used in a fulfillment pattern Order Components tab, Selected Order Components sub-tab.

**Note:** The Duration tab parallel to the Order Components tab and Dependencies tab in Figure 6–2 is no longer used. This tab still appears in Design Studio to support OSM cartridges that target pre-OSM 7.2.2 servers.
The minimum processing duration of an order may vary greatly depending on a number of factors:

- The kinds of products or services. Orders for mobile services typically have a very short processing duration, whereas a complex business-to-business order might take weeks.
- What must be done to fulfill the actions on the product or service, such as shipping or installation work.
- Any dependencies within and between the products and services. For example, PSTN provisioning must complete before ADSL provisioning starts.

Because a single order can have multiple values for the minimum processing duration, defined in multiple order components and at the order level, OSM compares all of them (if they are defined) to find the longest processing duration for the order:

1. OSM compares the two possible values of the minimum duration for an order component:
   - The duration specified in the order component itself.
   - The duration assigned to the order component in its fulfillment pattern.

   OSM uses the larger of the two values as the order component minimum processing duration.
2. OSM adds the calculated durations for all of the order components in the order. OSM takes into consideration dependencies between order components. For example, if the order component that provisions a service depends on the order component that processes billing, the minimum processing duration for both components must be used.

3. OSM calculates the order duration based on the expected order completion date minus the start date.

About Minimum Processing Duration Inheritance in Fulfillment Patterns
For the minimum processing duration that is assigned to an order component by a fulfillment pattern, the minimum processing duration for the order component is inherited in fulfillment patterns extended from the parent fulfillment pattern. For example:

1. In the BaseFulfillPatt fulfillment pattern, the BillingFunction order component is assigned a duration of 2 days.

2. The Service.Fixed fulfillment pattern is extended from the BaseFulfillPatt fulfillment pattern. Therefore, if you do not specify a duration for the BillingFunction order component in the Service.Fixed fulfillment pattern, it inherits the duration of 2 days from the BaseFulfillPatt fulfillment pattern.

Figure 6–3 shows how the duration is inherited from a parent fulfillment pattern.
About Minimum Processing Duration Expressions

In addition to specifying a fixed amount of time as the duration, you can use an XQuery expression. The following expression returns a duration of three hours:

\[ \text{PT3H0M0S} \]

You typically use a duration expression if you have an external system that keeps track of processing duration and the load levels of systems. You can write a duration expression that uses this information dynamically. For example, the calculation can take into account peek activity periods.

Calculating the Earliest Order Component Start Date (Order Start Date)

The first order component to start processing can contain one or more order items. OSM uses the order item with the earliest requested delivery date to calculate the order component start date. If there were only one level of order component decomposition in the orchestration plan and there were no dependencies between order components, OSM would calculate the order component start date by taking the
earliest order item requested delivery date and subtracting the configured minimum processing duration for the order component. This calculated start date would also be the order start date.

In the scenario, the following order component start dates are possible:

- If the component start date (also the order start date) is in the future, OSM does not start the order component until the future date. In the Order Management Web client, you would see:
  - The expected order start date would be later than the order creation date.
  - The order component Expected Start Date would be the same as the expected order start date.
  - The expected order item start date for all order items in the order component would be the same as the order component Expected Start Date.
  - The expected order completion date and the requested order delivery date would be identical.

- If the component start date is in the past, OSM starts the order immediately. In the Order Management Web client, you would see:
  - The order component Expected Start Date would be the same as the expected order start date.
  - The expected order item start date for all order items in the order component would be the same as the order component Expected Start Date.
  - The requested order completion date would be before the expected order delivery date.

- If no minimum processing duration was configured for the order component, then the order component would start on the same day as the requested delivery date, assuming that day was a future date. In the Order Management Web client, you would see:
  - The expected order start date would be later than the order creation date.
  - The order component Expected Start Date would be the same as the expected order start date.
  - The expected order item start date for all order items in the order component would be the same as the order component Expected Start Date.
  - The requested order completion date would be on the same date as the expected order delivery date.

- If the order item contained no value for the requested delivery date property, then OSM starts the order immediately.

**About Calculated Order Component Start Dates**

The first order component in an order and any initial order component that does not depend on another order component always uses a calculated start date based on order item requested delivery date values. If the order items do not have values for the requested delivery date, then the order begins processing immediately.

Dependent order items start in the following ways:

- Any dependent order components start immediately after the first or initial order component completes and all dependencies are resolved. This is the default behavior for order components.
You can enable calculated start dates for dependent order components by selecting the **Use Calculated Start Date** check box in the Order Component Specification Details tab. Dependent order components use the calculated start date based on the earliest order item requested delivery date in the order component, minus the order component duration. This functionality depends on the `calculatedStartDate` and `duration` control data parameters included in the `OracleComms_OSM_CommonDataDictionary` model project. See *OSM Developer's Guide* for more information about modeling control data. See "Modeling Order Component Dependencies and Requested Delivery Dates" for more information about configuring dependent order component calculated start dates.

For a three stage orchestration cartridge with function, system, and granularity components, you can enable calculated start dates at the function level if you wanted all components related to that function to use a calculated start date. Or you can enable calculated start dates at the system level. In this second scenario, one function might decompose to more than one system level component and a calculated start date might only be required for one of them.

**Modeling Order Component Dependencies and Requested Delivery Dates**

An OSM orchestration cartridge can have several order components with dependencies configured between them. OSM always honors any order component dependency wait condition before starting a new order component. You can configure dependent order components to start immediately after the blocking order component is complete and all dependencies have been met, or you can use the calculated start date. See "About Calculated Order Component Start Dates" for more information.

This scenario assumes that the dependency between the order component order items are between different order items. For example, order item 1 is only processed by order component A (the blocking order component) and order item 2, which is dependent on order item 1, is only processed by order component B (the waiting order component).

The following dependent calculated order component start date scenarios are possible:

- If the component start date is in the future, and the blocking order component is complete with all dependencies met, then OSM does not start the order component until the calculated start date arrives.
- If the component start date is calculated to a date before the blocking order component is complete and all blocking order component dependencies are met, then OSM ignores the calculated start date. The order component begins immediately after the blocking order component completes and all dependencies are resolved.
- If the order item contained no value for the requested delivery date parameter, then OSM starts the order immediately.

**Modeling Order Items Processed by Multiple Dependent Order Components**

If OSM processes an order item in more than one executable order component, and there is a dependency between these executable order components, then OSM calculates the order component start dates for the first order component by subtracting the duration from the longest chain of order component durations involved in processing the order item from the earliest order item requested delivery date. This ensures that all order components can be delivered by the requested delivery date. All dependent order components in this scenario would start immediately after the previous order component was resolved. For example, if order item 1 is processed by
order component A, B, and C, and B and C depend on A, then the order component start date for A would be the requested delivery date for order item 1 minus the duration of either order components B or C (whichever was longer) and A. Or, if B was dependent on A, and C was dependent on B, then OSM would subtract the total duration of A, B, and C from the requested order delivery date of order item 1 to determine the start date for order component A.

Revisions of Future-Dated Orders

You can submit revision orders to future-dated orders. The revision order can have a different requested delivery date than the base order or the same requested delivery date. In either case, OSM re-calculates the start date for the revision order based on its requested delivery date and on the minimum processing durations of the revised order components.

Note: Future-dated orders that cancel a future-dated base order are special cases. In this situation, the base order is canceled immediately, regardless of the requested delivery dates.

You can submit a future-dated revision order for an order that has already started processing. Only order components that have not started can have new calculated start dates applied. The new requested delivery date will trigger a compensation only if the order item specification requestedDeliveryDate order item property is marked as significant. Any task compensation required (for example, in previous completed order components) also happens immediately.

As a result of changing a significant order item requested delivery date, OSM calculates a new orchestration plan. Order components that have compensation tasks set with undo, redo, or amenddo compensation strategies are executed based on the dependency graph of the revised base order orchestration plan. The order item requested delivery date modification may change the calculated start date of the order component that is processing the order item and, by extension, may also change the expected order completion date.

Examples of Calculating the Expected Start Date

The following examples show scenarios for calculating the expected start date for an order and order components.

Example 1: Calculating Start Dates for Order Components with No Dependencies

In this example:

- A billing function order component has a duration of 2 days and processes order item 1 with a requested delivery date of January 3rd.
- A provisioning function order component has a duration of 3 days and processes order item 2 with a requested delivery date of January 5th.
- There are no dependencies between order components.

The start date for each order component is calculated as follows:

1. The calculated start date for the Billing order component is calculated using the following logic:
   - Order item 1 requested delivery date January 3th
- Minus Billing order component duration 2 days
- The Billing order component start date is January 1st.

Because there are no dependencies between the order components, OSM calculates the start date for each order component separately.

2. The calculated start date for the Provisioning order component is calculated using the following logic:
   - Order item 1 requested delivery date January 5th
   - Minus Provisioning order component duration 3 days
   - The Provisioning order component start date is January 2nd.

**Example 2: Calculating Start Dates for Order Components with Dependencies**

OSM always uses the final set of order components for in an orchestration plan to determine the start date for the order component. A final order component has no successor order components. For example, Figure 6–4 shows the order component processing flow for three order items. Order components C and E are final order components.

![Figure 6–4 Order Component and Order Item Processing Flow](image)

OSM calculates start dates for each order component starting with the requested delivery date of the final order components minus the order duration and any dependency condition wait delay duration. In this example:

- **Order component C** processes order item 1 and 2. Order item 1 has a requested delivery date of January 8, while order item 2 has a requested delivery date of January 10. OSM always uses the earliest requested delivery date to calculate the start date for the order component, which means the January 8 date is used. Because **order component C** is configured with a duration of 2 days, then **order component C** starts on January 6th.

- **Order component E** processes order item 3 that has a requested delivery date of January 18. Because **order component E** is configured with a duration of 2 days, then **order component E** would start on January 16th.
OSM calculates the start date of order component B by subtracting the configured duration for order component B (2 days) minus the start date for order component C (January 6th) resulting in a start date for order component B of January 4th.

OSM uses order component C instead of order component E to calculate the start date for order component B because order component C is a final order component with an order item that has the earliest requested delivery date. OSM does this to ensure that all order items being processed by an order component are not started late, even though they may start early. In other words, those order items being processed in order component B complete earlier than order component E needs them, but those order items destined for order component C complete with sufficient time for order component C to meet order item 1’s requested delivery date of January 8th.

Finally, OSM calculates the start dates for order components A and D. Order component A has a configured duration of 3 days minus the start date for order component B (January 4th) resulting in a start date of January 1st. Order component D has a configured duration of 2 days resulting in a start date of January 2nd.

The order start date is the earliest of all starting order components. In this example, the earliest order component start date is January 1st for order component A.

About Managing Order States and Transitions

Every order has an order state, which indicates the current condition of the order; for example, In Progress, Amending, or Completed. These OSM order states control the progress of the order. For example, an OSM user cannot work on tasks while the order is in the Suspended state, and an order in the Aborted state cannot be restarted.

A typical order uses the following states:

1. An order is created in the Not Started state.
2. When processing begins on the order, the state transitions to the In Progress state.
3. When the order is complete, it transitions to the Completed state.

Changes from one order state to another order state are called transitions. Each order state has a set of allowable transitions. For example, when an order is completed, it transitions from the In Progress state to the Completed state.

Transitions are controlled by transactions. A transaction is an action taken by the OSM system. For example, the Suspend Order transaction performs the following actions:

- Stops all processing on the order
- Transitions the order to the Suspended state

Most transactions perform transitions that change the state of the order. However, some transactions do not perform a transition to another state. For example, the Update Order transaction can make changes to an order without changing the order’s state.
About Managing Order States and Transitions

About Modeling Life-Cycle Policies

You can customize how an order transitions from one state to another by customizing the order’s life-cycle policy. Every order type that you model must be associated with an order life-cycle policy. Customizing an order life-cycle policy enables you to control the following:

- You can specify conditions that need to be met before an order can transition from one state to another. A common example is specifying the point of no return for revision orders, which controls the transition from the In Progress state to the Submit Amendment and Process Amendment states. See "About Modeling Transition Conditions" for more information.

- You can specify a grace period that allows the order to complete processing tasks before transitioning to another order state. See "About Modeling Transition Grace Periods" for more information.

- You can specify the roles that are allowed to perform a transaction. See "About Modeling Transition Permissions" for more information.

- You can specify the error message and severity if a transition condition is not met. See Design Studio online Help.

OSM allows any number of order life-cycle policies to be configured. You can create a custom policy for each order type or one general policy that is applied to many order types. The default order life cycle contains the minimum set of order state and transaction combinations assigned to all roles defined in the system.

About Modeling Transition Conditions

Transition conditions are Boolean expressions that specify if a transition from one state to another is allowed. For example, for the Submit Amendment state, you can prevent the Process Amendment transition from occurring until a condition is true.

Figure 6–5 shows the life-cycle policy for the Process Amendment transition. In this case, it returns true, so it is always allowed to transition.

Figure 6–5  Order Life-Cycle Policy for the Process Amendment Transition

A common scenario for configuring permissions is when you set the point of no return for amendment processing. See "About Point of No Return" for more information.

When specifying conditions, the minimum set of required order states is Not Started, In Progress, and Completed. The life cycle must allow an order to transition to those states.
OSM uses more transactions than those shown in Design Studio. Design Studio shows only the transactions that you can assign permissions on and set conditions for. For example, the Complete Task transaction can transition an order to the Completed state, but that transaction cannot be customized.

**About Modeling Transition Grace Periods**

The transition grace period specifies the amount of time that OSM should wait before transitioning the order. For example, if a Suspend Order transaction is run on an In Progress order, a grace period can allow the order processing to reach a definitive state for all currently executing tasks before transitioning to the Suspended state. In this case, OSM waits until all active tasks are in the Received, Completed, or user-defined Suspended task state. (An active task is a task that is in the Accepted state.) If the grace period expires before all tasks move out of the Accepted task state, OSM transitions the order state.

During the grace period, the target order state header in the Task Web client displays the order state the order is transitioning to. The target order state is populated only when an order is in grace period. *Figure 6–6* shows the target order displayed in the Task Web client.

*Figure 6–6  Target Order Displayed in the Task Web Client*

You can specify a grace period for certain transactions, such as Suspend Order, Process Amendment, Cancel Order, and Fail Order. For other transactions, a grace period is unnecessary or not permitted, such as for Submit Amendment, Update Order, and Abort Order.

You can define the following grace period parameters:

- The length of time to wait (minimum and maximum, or indefinite)
- How often to generate a jeopardy event during the grace period

*Figure 6–7* shows how you can customize the order life cycle in Design Studio. In this figure, the Cancel Order exit transaction for the In Progress order state is selected. A grace period for transitioning to an order cancellation is set for a minimum of one day, and a maximum of five days. A jeopardy event is triggered every hour.
About Modeling Transition Permissions
You can specify the roles that are allowed to perform a transaction. For example, while an order is in the In Progress state, your customer service role might need to perform the Update Order and Cancel Order transactions, whereas your fallout specialist role might perform only the Raise Exception transaction.

OSM Order States and Transactions
OSM includes a standard set of order states and transactions. You cannot add states or transactions, but you can control how the order transitions between states.

Table 6–1 shows the OSM order states.
### Table 6–1 Order States

<table>
<thead>
<tr>
<th>Order State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aborted</td>
<td>The order has been permanently stopped. This is a final state. An order in the aborted state cannot transition to another order state. See &quot;About the Aborted Order State&quot; for more information.</td>
</tr>
<tr>
<td>Amending</td>
<td>The order is being amended. OSM identifies which tasks are affected by the amended data and compensates the order as necessary. See &quot;About the Amending Order State&quot; for more information.</td>
</tr>
<tr>
<td>Cancelled</td>
<td>The order has been canceled. All tasks have been undone back to the creation task. If an order includes an orchestration plan, the Cancelled state is the final state. The order cannot be resumed. If the order does not have an orchestration plan, the canceled order is returned to the creation task for the order. The order can be re-submitted to be run again. See &quot;About the Cancelled Order State&quot; for more information.</td>
</tr>
<tr>
<td>Cancelling</td>
<td>The order is being canceled. At least one task is running to perform amendment processing for the cancellation. While the order is in the Cancelling state, OSM undoes all completed tasks to return the order to the creation task. When OSM is finished, the order transitions to the Cancelled state. See &quot;About the Cancelling Order State&quot; for more information.</td>
</tr>
<tr>
<td>Completed</td>
<td>The order has been fulfilled. There are no tasks running and processing is complete. This is a final state. An order in the Completed state cannot transition to another order state. See &quot;About the Completed Order State&quot; for more information.</td>
</tr>
<tr>
<td>Failed</td>
<td>The order has failed during processing. The Failed state is not a final state; the order can be resumed when the problem is fixed. See &quot;About the Failed Order State&quot; for more information.</td>
</tr>
<tr>
<td>In Progress</td>
<td>The order is actively running. Future-dated orders have an In Progress state while they wait for dependencies to be resolved. See &quot;About the In Progress Order State&quot; for more information.</td>
</tr>
<tr>
<td>Not Started</td>
<td>The order has been created but has not started. There are no tasks running. See &quot;About the Not Started Order State&quot; for more information.</td>
</tr>
<tr>
<td>Suspended</td>
<td>The order has been suspended and all processing on the order in OSM has been halted. No task can be updated or transitioned while the order is in this state. See &quot;About the Suspended Order State&quot; for more information.</td>
</tr>
<tr>
<td>Waiting for Revision</td>
<td>The order is waiting for a revision. This state is common following compensation to an order for fallout, when the order is awaiting a revision from the order-source system to correct something that caused a failure in the originally submitted order. See &quot;About the Waiting for Revision Order State&quot; for more information.</td>
</tr>
</tbody>
</table>

Table 6–2 shows transactions that are included in the order life-cycle policy and the operations they perform.
### Table 6–2 Order State Transactions

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Description</th>
</tr>
</thead>
</table>
| Abort Order         | Immediately and permanently stops order processing. Transitions the order state to Aborted.  
                      | In the Order Management Web client and the Task Web client, Abort Order transactions are identified as “Terminate Order”.                  |
| Cancel Order        | Transitions the order to the Cancelling state. After OSM performs the necessary tasks to cancel the order, the order transitions to the Cancelled state.  
                      | In Design Studio, you can specify a grace period to wait for all accepted tasks to complete before transitioning the order.               |
| Complete Task       | Completes a task and allows the transition to the next task. Completing the last active task implicitly transitions the order to a Completed state.  
                      | This transaction is not configurable in the life-cycle policy.                                                                                  |
| Copy Order          | Copies an order; for example, when you create an order in the Task Web client by copying an order. This transaction does not change the order state. It is not configurable. |
| Create Order        | Creates an instance of an order.  
                      | The transaction starts the order in either the Not Started state or the In Progress state.  
                      | This transaction is not a configurable transaction in the OSM life-cycle policy. Permissions for creating an order are not set in the life-cycle policy. Instead you assign an order creation permission to roles and assign permissions on the orders. |
| Delete Order        | Removes an order from the system.  
                      | To delete orders, use the orderPurge command. See OSM System Administrator’s Guide for more information. If the order does not have an orchestration plan, you can delete an order using the Task Web client when the order is at the creation task. |
| Fail Order          | Transitions the order to the Failed state. Processing on the order is stopped.  
                      | In Design Studio, you can specify a grace period to wait for all accepted tasks to complete before transitioning the order.               |
| Fallout Order       | Compensates an existing order based on error data identified during provisioning.  
                      | This transaction is not configurable in the life-cycle policy.                                                                                  |
| Manage Order Fallout| Transitions the order to the state it had before it failed. Processing on the order resumes.  
                      | This transaction enables Task Web client users to locate orders with errors that require manual intervention, analyze the order to determine the cause of the errors, and take the corrective action to correct errors allowing the order to continue to process normally. |
| Process Amendment   | Transitions the order to the Amending state. This transaction is always preceded by the Submit Amendment transaction. See “About the Amending Order State” for more information.  
                      | In Design Studio, you can specify a grace period to wait for all accepted tasks to complete before transitioning the order. |
Figure 6–8 shows OSM order states and transactions.

- The transactions shown are those that perform transitions between order states. Some transactions, such as Update Order, do not always perform a transition.

- In this figure, a Resume Order transaction is shown from the Cancelled state to the In Progress state. This transaction is only possible for orders that do not have an orchestration plan. If the order has an orchestration plan, the Cancelled state is a final state and cannot be resumed.

- Some order state transitions are performed internally by OSM, not by running a transaction.

- The transition from Not Started to Completed occurs when an order is submitted for a revision to an in-flight order. In this case, all that the revision order must do is submit an amendment. When the revision order is processed, the Submit Amendment transaction places the revision order in the amendment queue. After doing so, the revision order itself requires no further processing because it

---

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raise Exception</td>
<td>Raises an exception. The system can be configured to initiate fallout compensation with this transaction. In this situation the order transitions to the Amending state while it compensates tasks. From the Amending state, it can transition to the Failed, In Progress, or Waiting for Revision states. For backward compatibility this transaction can also trigger a preconfigured exception process. Exception processes are incompatible with OSM's built-in compensation. An order for which an exception process is triggered cannot have compensation applied for revisions, cancellations, or fallout. In this case, the order remains in the In Progress state. In Design Studio, you can specify a grace period to wait for all accepted tasks to complete before transitioning the order.</td>
</tr>
<tr>
<td>Resume Order</td>
<td>Transitions the order to the In Progress state, typically from the Suspended state.</td>
</tr>
<tr>
<td>Submit Amendment</td>
<td>Submits an amendment but does not change the order state. This transaction is followed by the Process Amendment transaction, which changes the order state to Amending. See “About the Amending Order State” for more information.</td>
</tr>
<tr>
<td>Suspend Order</td>
<td>Transitions the order to the Suspended order state. Processing on the order halts. In Design Studio, you can specify a grace period to wait for all accepted tasks to complete before transitioning the order.</td>
</tr>
</tbody>
</table>
| Update Order        | Allows changes to order data, remarks, and attachments outside the context of a task. The Update Orders can add new data elements, delete existing data elements, or change existing data element. Update Orders can be sent from locations such as:  
  - The Task Web client.  
  - Automation plug-in XSLT or XQuery automators.  
  - Web Services or XML API requests.  
  In most situations, Update Order does not allow the state of the task to change; for example, when updating an order that is in the Aborted state. When an order is in the Not Started state or the Cancelled state, the Update Order transaction can start or resume the order by running the creation task. To use Update Order to start or resume an order, you need to use the startOrder flag in the Update Order transaction, in an automation plug-in, a Web services operation, or through the Task Web client. You cannot specify to start or resume an order by configuring the order life-cycle policy in Design Studio. |
compensation happens to the base order, so the revision order is transitioned directly to the Completed state automatically by OSM, without going to the In Progress state.

**Note:** Because the transition from Not Started to Completed for revision orders is required by OSM and is performed by the system, you cannot define permissions or conditions for it. Therefore, it is not shown as a transaction from the Not Started state in Design Studio.
About Order State Categories

Order states can be categorized by the overall condition of the order that they apply to; for example, if the order is open, closed, or running:

- **Open - Not Running**
About Managing Order States and Transitions

- Not Started
- Suspended
- Waiting for Revision
- Canceled
- Failed

■ Open - Running
  - In Progress

■ Open - Running - Compensating
  - Amending
  - Cancelling

■ Closed
  - Completed
  - Aborted

Figure 6–9 shows the order state categories.

**Figure 6–9  Order State Categories**

Common Order State Transitions

A typical order processing scenario uses the following order states:

1. The order is submitted to the Not Started state and transitions to the In Progress state. The order remains in the In Progress state while processing occurs.
2. When the last task has completed, the order transitions to the Completed state.

Figure 6–10 shows the states, state categories, and transactions for a basic order processing flow.

**Figure 6–10 Simple Order Processing Flow**

The process for revising an order uses the following order states:

1. The base order is submitted and transitions to the In Progress state.
2. The revision order is submitted and transitions to the In Progress state.
3. The base order transitions to the Amending state.
4. The revision order, after it has amended the base order, transitions to the Completed state.
5. After processing the amendment, the base order returns to the In Progress state.
6. When the last task has completed, the base order transitions to the Completed state.

Figure 6–11 shows the order states used for a revision order.

**Figure 6–11 Order States Used When Processing a Revision Order**
A follow-on order uses the following order states:

1. The base order is submitted and transitions to the In Progress state.
2. The follow-on order is submitted and transitions to the In Progress state, but it must wait until an order item in the base order completes before it can be processed.
3. The order item in the base order completes. The base order can continue processing, or it can complete and transition to the Completed state.
4. Since the order item in the base order has completed, the dependency has been met and the follow-on order begins processing.
5. When the last task in the follow-on order has completed, it transitions to the Completed state.

A future-dated order uses the following order states:

1. The order is submitted, but OSM determines that there is a future start date. The order transitions to the Not Started state.
2. When the order start date is reached, the order transitions to the In Progress order state.
3. When the last task has completed, the order transitions to the Completed order state.

Optional, Mandatory, and Prohibited Transactions

Transactions for each order state can be optional, mandatory, or prohibited. Optional transactions can either be allowed or prohibited based on conditions and permissions defined in the order life-cycle policy.
### Table 6-3 OSM Order Transactions

<table>
<thead>
<tr>
<th>Order State</th>
<th>Mandatory Transactions</th>
<th>Prohibited Transactions</th>
<th>Optional Transactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aborted</td>
<td>None</td>
<td>- Abort Order&lt;br&gt;- Cancel Order&lt;br&gt;- Complete Task&lt;br&gt;- Fail Order&lt;br&gt;- Manage Order Fallout&lt;br&gt;- Process Amendment&lt;br&gt;- Raise Exception&lt;br&gt;- Resume Order&lt;br&gt;- Submit Amendment&lt;br&gt;- Suspend Order</td>
<td>- Delete Order&lt;br&gt;- Update Order</td>
</tr>
<tr>
<td>Amending</td>
<td>Process Amendment</td>
<td>- Cancel Order&lt;br&gt;- Complete Task&lt;br&gt;- Delete Order&lt;br&gt;- Fail Order&lt;br&gt;- Manage Order Fallout&lt;br&gt;- Process Amendment&lt;br&gt;- Raise Exception&lt;br&gt;- Resume Order&lt;br&gt;- Update Order</td>
<td>- Abort Order&lt;br&gt;- Submit Amendment&lt;br&gt;- Suspend Order</td>
</tr>
<tr>
<td>Canceled</td>
<td>None</td>
<td>- Cancel Order&lt;br&gt;- Complete Task&lt;br&gt;- Fail Order&lt;br&gt;- Manage Order Fallout&lt;br&gt;- Process Amendment&lt;br&gt;- Raise Exception&lt;br&gt;- Resume Order&lt;br&gt;- Submit Amendment&lt;br&gt;- Suspend Order</td>
<td>- Abort Order&lt;br&gt;- Delete Order&lt;br&gt;- Update Order</td>
</tr>
<tr>
<td>Canceling</td>
<td>None</td>
<td>- Cancel Order&lt;br&gt;- Complete Task&lt;br&gt;- Delete Order&lt;br&gt;- Fail Order&lt;br&gt;- Process Amendment&lt;br&gt;- Raise Exception&lt;br&gt;- Resume Order&lt;br&gt;- Submit Amendment&lt;br&gt;- Update Order</td>
<td>- Abort Order&lt;br&gt;- Suspend Order</td>
</tr>
</tbody>
</table>

About Managing Order States and Transitions

6-24 OSM Concepts
### Table 6–3  (Cont.) OSM Order Transactions

<table>
<thead>
<tr>
<th>Order State</th>
<th>Mandatory Transactions</th>
<th>Prohibited Transactions</th>
<th>Optional Transactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Completed</td>
<td>None</td>
<td>■ Abort Order</td>
<td>■ Abort Order</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Cancel Order</td>
<td>■ Delete Order</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Complete Task</td>
<td>■ Update Order</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Fail Order</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Process Amendment</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Raise Exception</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Resume Order</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Submit Amendment</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Suspend Order</td>
<td></td>
</tr>
<tr>
<td>Failed</td>
<td>None</td>
<td>■ Complete Task</td>
<td>■ Abort Order</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Delete Order</td>
<td>■ Cancel Order</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Fail Order</td>
<td>■ Manage Order Fallout</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Process Amendment</td>
<td>■ Submit Amendment</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Raise Exception</td>
<td>■ Update Order</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Resume Order</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Suspend Order</td>
<td></td>
</tr>
<tr>
<td>In Progress</td>
<td>Complete Task</td>
<td>■ Delete Order</td>
<td>■ Abort Order</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Manage Order Fallout</td>
<td>■ Cancel Order</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Resume Order</td>
<td>■ Fail Order</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>■ Process Amendment</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>■ Raise Exception</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>■ Submit Amendment</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>■ Suspend Order</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>■ Update Order</td>
</tr>
</tbody>
</table>
About Managing Order States and Transitions

Table 6–3 (Cont.) OSM Order Transactions

<table>
<thead>
<tr>
<th>Order State</th>
<th>Mandatory Transactions</th>
<th>Prohibited Transactions</th>
<th>Optional Transactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not Started</td>
<td>Complete Task</td>
<td>- Cancel Order</td>
<td>- Abort Order</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Manage Order Fallout</td>
<td>- Delete Order</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Process Amendment</td>
<td>- Fail Order</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Raise Exception</td>
<td>- Suspend Order</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Resume Order</td>
<td>- Update Order</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Submit Amendment</td>
<td></td>
</tr>
<tr>
<td>Suspended</td>
<td>None</td>
<td>- Complete Task</td>
<td>- Abort Order</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Delete Order</td>
<td>- Cancel Order</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Manage Order Fallout</td>
<td>- Fail Order</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Process Amendment</td>
<td>- Resume Order</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Raise Exception</td>
<td>- Submit Amendment</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Suspend Order</td>
<td>- Update Order</td>
</tr>
<tr>
<td>Waiting for Revision</td>
<td>None</td>
<td>- Complete Task</td>
<td>- Abort Order</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Delete Order</td>
<td>- Cancel Order</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Manage Order Fallout</td>
<td>- Fail Order</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Process Amendment</td>
<td>- Resume Order</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Raise Exception</td>
<td>- Submit Amendment</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Suspend Order</td>
<td>- Update Order</td>
</tr>
</tbody>
</table>

About the Aborted Order State

An order can be transitioned to the Aborted order state when an unrecoverable error or condition has stopped the processing for the order and the order cannot return to a valid processing state through a revision or fallout management activity within OSM. It can be considered a last resort to prevent any further execution of an order.

An order can be terminated manually from the Order Management Web client or from the Task Web client. (In the Web clients, the command **Terminate Order** moves the order to the Aborted order state.) You can also transition to the Aborted order state programmatically by using the OSM Web Services API or by using an automated task.

The Aborted order state is a final state; the order has been permanently stopped. An order in the Aborted state cannot transition to another state.

Terminated orders may require manual intervention in an OSM Web client to compensate for tasks that have completed or that were in the process of completing. For example, you may be required to release port assignments, delete accounts in billing systems, and so forth.

The entrance transaction to the Aborted order state is Abort Order. This transaction can be run from all order states except the Completed order state.

The exit transaction from the Aborted state is Delete Order, which removes the order from the OSM system.

The Update Order transaction is used when the order is updated manually, outside of the order processing.

**Figure 6–12** shows the order states that can transition to or from the Aborted order state.
About the Amending Order State

An order in the Amending state is undergoing compensation. See "Managing Changes to Orders" for more information.

The transactions that cause an order to move to the Amending state are the Submit Amendment transaction (as a result of a revision order) and the Raise Exception transaction (as a result of fallout for which compensation is needed). The order can be amended from the following order states:

- In Progress
- Failed
- Suspended
- Waiting for Revision

To transition an order to the Amending state, OSM uses two transactions: Submit Amendment and Process Amendment. These transactions work together to ensure that the order is in a condition that can be amended and that the amendment is allowed.

Each revision to an order uses the Submit Amendment transaction to place the amendment in a queue. The Submit Amendment transaction does not change the
order state. Instead, it makes sure that the order is ready to be amended and that there are no life-cycle rules that prevent the order from being amended until a condition is met. For example, although an order in the Suspended state can receive amendments from the Submit Amendment transaction, the order must be resumed before it can process the amendments.

When the order is able to process the amendment, the Process Amendment transaction is run on the latest amendment in the queue, and the transition is made to the Amending state. Not every order in the queue is processed:

- A revision for the same order might have been received while the order is queued. In that case, the later revision is used instead.
- Restrictions in the life-cycle policy might prevent an amendment from being processed by the Process Amendment transaction.

Unless multiple revisions are common and frequent, the order state transition to Amending will happen almost immediately after the Submit Amendment transaction.

The configurable exit transactions for the Amending state are:

- Submit Amendment: An order can process a Submit Amendment transaction while the order is in the Amending state. This can occur because additional revision orders can be submitted while the order is in the Amending state. In this case, the Submit Amendment transaction adds the amendment to the amendment queue.
- Suspend Order: Transitions to the Suspended state.
- Abort Order: Transitions to the Aborted state.

An order can transition from the Amending state to the In Progress state, but there is no transaction involved. This transition is handled internally by OSM.

An order can transition from the Amending state to the Waiting for Revision state. However, there is no transaction required to transition from the Amending state to the Waiting for Revision state. This transition happens when fallout occurs, and OSM has found that the fallout is caused by the submitted order. In that case, OSM cannot use further compensation (redo/undo) to fix the problem. Instead, OSM waits for a revision to be submitted from the upstream order-source system to fix the problem.

**Figure 6–13** shows the order states that can transition to or from the Amending order state.
About the Cancelled Order State

When an order is in the Cancelled state, all tasks have been undone back to the creation task.

The actions allowed when an order is in the Cancelled state are different depending on if the order has an orchestration plan:

- If an order has an orchestration plan, the Cancelled state is the final state. The order cannot be resumed.
- If the order does not have an orchestration plan, the order can be resumed at the In Progress state, either by manually opening the order at the creation task and submitting it or by programmatically transitioning the order state using the OSM APIs.

The transaction that causes the Cancelled state is the same Cancel Order transaction that was used for canceling the order.

If the order includes an orchestration plan, the configurable exit transactions are:

- Update Order: Allows the order data to be changed but does not transition the order to another order state.
- Abort Order: Transitions to the Aborted state.
- Delete Order: Removes the order from the OSM system.

If the order does not have an orchestration plan, the configurable exit transactions are:

- Resume Order: Transitions to the In Progress state.
- Update Order: Allows the order data to be changed. This transaction can also transition the order to the In Progress state if the `startOrder` option is used. See
About Managing Order States and Transitions

Discussion of the Update Order transaction in Table 6–2, "Order State Transactions" for more information.

- Abort Order: Transitions to the Aborted state.
- Delete Order: Removes the order from the OSM system.

**Important:** When resumed after being canceled, the order begins again at the beginning of the execution; it is not resumed at the point in the execution it was in when canceled.

Figure 6–14 shows the order states that can transition to or from the Cancelled order state if the order has an orchestration plan.

**Figure 6–14 Order States that Can Transition to or from the Cancelled Order State if the Order Has an Orchestration Plan**

![Diagram showing order states and transitions]

Figure 6–15 shows the order states that can transition to or from the Cancelled order state if the order does not have an orchestration plan.
About the Cancelling Order State

When an order is in the Cancelling state, at least one live task is running in a cancellation compensation mode. OSM undoes all completed tasks to return the order to the creation task. When OSM has finished, the order transitions to the Cancelled state.

The entrance transaction for the Cancelling order state is the Cancel Order transaction. An order can be canceled from the following order states:

- In Progress
- Suspended
- Waiting for Revision
- Failed

The configurable exit transactions for the Cancelling order state are:

- Suspend Order: Transitions to the Suspended state.
- Abort Order: Transitions to the Aborted state.

*Figure 6–16* shows the order states that can transition to or from the Cancelling order state.
About the Completed Order State

The order has been fulfilled. There are no live tasks and processing is complete. This is a final state. An order in the Completed state cannot transition to another state.

The entrance transaction for the Completed state is the Complete Task transaction. It transitions from the In Progress state.

The Complete Task transaction is used internally whenever the last task is completed in the order, which is determined automatically by OSM. Therefore the Complete Task transaction is not shown as part of the life-cycle policy in Design Studio.

The transition from the Not Started state to the Completed state is specific to revision orders. When a revision order that has been submitted and accepted transitions to the Completed state directly, because the compensation for the revision happens on the base order being revised.

The configurable exit transactions for the Completed order state are:

- Delete Order: Removes the order from the OSM system.
- Update Order: Allows the order data to be added, changed, or deleted but does not transition the order to another order state.

Figure 6–17 shows the order states that can transition to or from the Completed order state.
About the Failed Order State

If an order is in the Failed state, the order failed during fulfillment, after the order was submitted by the order-source system.

The entrance transaction for the Failed order state is the Fail Order transaction. An order can transition to the Failed state from the following states:

- Not Started
- In Progress
- Suspended
- Waiting for Revision

The configurable exit transactions for the Failed order state are:

- Manage Order Fallout: Transitions back to the state that the order was in when the Fail Order transaction occurred. For example, if the order was in the Not Started state and then failed, the Manage Order Fallout transaction returns the order to the Not Started state. It can exit to the following states:
  - Not Started
  - In Progress
  - Waiting for Revision
- Suspend Order: Transitions to the Suspended state.
- Update Order: Allows the order data to be added, changed, or deleted but does not transition the order to a different order state.
- Submit Amendment/Process Amendment: Submits an amendment and is followed by the Process Amendment transaction and transitions the order to the Amending state.
- Cancel Order: Transitions to the Cancelling state.
- Abort Order: Transitions to the Aborted state.

Figure 6–18 shows the order states that can transition to or from the Failed order state.
About the In Progress Order State

An order in the In Progress state is actively running. Future-dated orchestration orders have an In Progress state while they wait for dependencies to be resolved.

The entrance transactions for the In Progress state are:

- **Update Order:** Transitions from the Not Started state or Cancelled state when the `startOrder` option is used. Programmatic creation of an order typically begins the execution of the order, transitioning it to the In Progress order state when the `startOrder` option is set to `true` on the CreateOrder or CreateOrderBySpecification OSM Web services operation. See the discussion of the Update Order transaction in Table 6–2, "Order State Transactions" for more information.

- **Resume Order:** Transitions from the following states:
  - Suspended
  - Waiting for Revision
  - Cancelled
Manage Order Fallout: Transitions from the Failed state.

An order can transition from the Amending state to the In Progress state, but there is no transaction involved. This transition is handled internally by OSM.

The exit transactions for the In Progress order state are:

- Update Order: Allows the order data to be added, changed, or deleted.
- Submit Amendment/Process Amendment: Submits an amendment (typically from an external CRM system) and is followed by the Process Amendment transition. Transitions to the Amending state.
- Suspend Order: Transitions to the Suspended state.
- Cancel Order: Transitions to the Cancelling state.
- Fail Order: Transitions to the Failed state.
- Abort Order: Transitions to the Aborted state.
- Raise Exception: The Raise Exception transaction is a special type of transaction from the In Progress state. For order fallout scenarios, the Raise Exception transaction can transition the order to the Amending state to perform compensation for the error. However, for backward compatibility with orders that use process exceptions, the Raise Exception transactions starts an exception handling process, but the order remains in the In Progress state. See the discussion of the Raise Exception transaction in Table 6–2, "Order State Transactions" for more information.
- Complete Task: Transitions from the In Progress state, but only when the last task in the order is completed. This transaction is also used internally whenever a task is completed in the order. It is not shown in the life cycle display in Design Studio.

Figure 6–19 shows the order states that can transition to or from the In Progress order state.
About the Not Started Order State

When an order is in the Not Started state, the order has been created but has not started. There are no live tasks other than the creation task.

The entrance transactions for the Not Started state are:

- Resume Order: Transitions from the Suspended state if the order was in the Not Started state when it was Suspended.
- Manage Order Fallout: Transitions from the Failed state if the order was in the Not Started state when the Fail Order transaction occurred.

The exit transactions for the Not Started state are:

- Update Order: Allows the order data to be added, changed, or deleted. Can also transition the order to the In Progress state if the startOrder option is used. See the discussion of the Update Order transaction in Table 6-2, ”Order State Transactions” for more information.
• Suspend Order: Transitions to the Suspended state.
• Fail Order: Transitions to the Failed state.
• Abort Order: Transitions to the Aborted state.
• Submit/Process Amendment: Transitions to the Completed state. This transition is specific to revision orders. When a revision order is submitted, if it is accepted it transitions to the Completed order state directly, because the compensation for the revision happens on the base order being revised.
• Delete Order: Removes the order from the OSM system.

Figure 6–20 shows the order states that can transition to or from the Not Started order state.

Figure 6–20  Order States that Can Transition to or from the Not Started Order State

About the Suspended Order State

In the Suspended state, all processing on the order has been halted. No task can be updated or transitioned.

The only entrance transaction for the Suspended state is the Suspend Order transaction. Orders can be suspended from the following order states:
- Not Started
- Failed
- Canceling
- In Progress
- Amending

The exit transactions for the Suspended order state are:

- Resume Order: Transitions the order to the state that it was in when it was suspended.
- Submit Amendment: Submits an amendment (typically from an external CRM system) to the amendment queue. Typically, the Submit Amendment transaction is followed by the Process Amendment transaction, which transitions the order to the Amending state. However, an order in the Suspended state must be resumed with the Resume Order transaction before amendments can be processed. After the order is resumed, the Process Amendment transaction is run on the latest amendment in the queue and the order transitions to the Amending state.
- Update Order: Allows the order data to be added, changed, or deleted.
- Cancel Order: Transitions to the Cancelling state.
- Fail Order: Transitions to the Failed state.
- Abort Order: Transitions to the Aborted state.

Figure 6–21 shows the order states that can transition to or from the Suspended order state.
About the Waiting for Revision Order State

This state is common following compensation to an order for fallout, when the order is awaiting a revision from the order-source system to correct something that caused a failure in the originally submitted order.

The entrance transaction for the Waiting for Revision order state is the Manage Order Fallout transaction, which runs from the Failed state.

An order can transition from the Amending state to the Waiting for Revision state. However, there is no transaction required to transition from the Amending order state to the Waiting for Revision order state. This internal transition is triggered by the Raise Exception transaction and it happens when fallout occurs and OSM has found that the fallout is generated by the submitted order instead of by a task in the process. Therefore, OSM cannot use compensation (redo/undo) to fix the problem. Instead, OSM waits for a revision to be submitted from upstream to fix the problem.

The exit transactions for the Waiting for Revision order state are:

- Submit Amendment/Process Amendment: Submits an amendment (typically from an external CRM system) and is followed by the Process Amendment transition. Transitions to the Amending state.
- Update Order: Allows the order data to be added, changed, or deleted.
- Resume Order: Transitions to the In Progress State
- Cancel Order: Transitions to the Cancelling state.
- Fail Order: Transitions to the Failed state.
- Abort Order: Transitions to the Aborted state.

Figure 6–22 shows the order states that can transition to or from the Waiting for Revision order state.

Figure 6–22  Order States that Can Transition to or from the Waiting for Revision Order State

About Deleting Orders
You cannot use either of the OSM Web clients or any Web services operation to delete orders from the OSM system. Instead, use the `orderPurge` command. See OSM System Administrator’s Guide for more information.

Order Life-Cycle Events
You can configure orders to publish events when any of the following occurs:
- The order is created.
- The order is removed.
- The order state changes.
- Amendment processing starts.
- Amendment processing completes.
- Amendment processing is abandoned.

Order life-cycle events are published as Java Message Service (JMS) messages containing order identification and state information. You can configure which
life-cycle events you want to be generated for an order type in the Design Studio Order editor Amendable tab.
Managing Changes to Orders

This chapter describes how Oracle Communications Order and Service Management (OSM) manages changes to in-flight orders.

About Managing Changes to Orders

Any order that is not in a closed state (Completed or Aborted) is an in-flight order. An in-flight order still has the potential for further work to be performed on it.

There are many scenarios that require changes to an in-flight order. The most common scenarios are:

- **Revisions to an in-flight order.** For example, a customer who has just submitted an order might decide a short time later to change to a higher level of service. In this case, the CRM submits an order that includes the revised requirements. This order is known as the base order. OSM creates a new version of the base order, called a revision order (also called a supplemental order). This order contains all the relevant data for the order, including the updated requirements. OSM compares the data in the revision order with the data in the base order and makes the changes as required. This allows the base order to continue processing with, and compensating for, the customer’s new order requirements provided by the revision order. The customer does not have to wait for the base order to be completed or canceled before changing it.

A revision order can be used to correct a failed order.

**Note:** A single revision order can make multiple changes to an order.

- **Canceling an in-flight order.** In this case, tasks run on external systems, such as billing and shipping systems, need to be undone. Order cancellation is normally triggered by the originating CRM system, typically using the OSM CancelOrder Web services operation.

**Note:** Order cancellation can also be run from the OSM Web clients. However, if order processing has been configured such that the upstream system initiates cancellation, canceling an order by using the Web clients might cause discrepancies in the order life cycle as it is tracked in the upstream system.

- **Error in an in-flight order.** In this case, a user or an automation plug-in can raise an exception, trigger fallout, and make changes to compensate for errors that
caused the error. See "Order Fallout Management" for more information.

Using revision orders is the most efficient way to manage changes made to in-flight orders. OSM automatically detects the revisions that must be made and changes the orchestration plan as necessary. No manual work is required to find changes that need to be made. The revision order changes the base order in OSM, so only one order needs to be managed, even when there are multiple revisions to the same order.

When you model orders and tasks, you can control the amendment processing that is allowed for the order. For example:

- If the order is allowed to be amended
- At which point in the order processing the order is no longer allowed to be amended (the point of no return)
- Who can manage revision orders in the Task Web client
- Which data needs to be compensated, and which does not

About Update Order Transactions

In addition to revision orders, you can update order data in the following ways:

- Manually, by sending an XML API update order from a CRM. See OSM XML API Developer’s Guide for more information.
- Automatically, by configuring an automated task automator to send a Web Service or XML API update order. See OSM Developer’s Guide for more information about update orders and using automations.

Update orders can:

- Update a complete order. The existing order is updated (elements added, changed, or deleted) to match the supplied order. Order level order updates are typically sent in the context of order level notification, jeopardy, or event automation automators.
- Update nodes in an order. Elements can be added or changed. Deletes are not performed using the update node functionality. The nodes are supplied in the format of the existing order and are typically sent as part of task level automation automators.
- Add, delete, or change element data values that are typically sent as part of task level automation automators.

See "About Order-Level and Task-Level Compensation Analysis" for more information about how update orders impact compensation at the order and task level.

About Amendment Processing

To revise or cancel in-flight orders, OSM performs amendment processing. Amendment processing analyzes the requested changes, determines how to make the changes, and processes them. Amendment processing functions as follows:

1. A base order is submitted and is currently processing; it is not in the Not Started, Completed, or Aborted state. The upstream system submits a revision or a cancel
order. The new version of the order includes all of the data relevant to the order, not just changed data. The upstream system does not need to identify the changes to OSM or explicitly provide the discrepancies; OSM determines the discrepancies during amendment processing by comparing the new version with the version of the order currently being processed.

To submit the revision order, the upstream system can use either the CreateOrder Web services operation or the CreateOrderBySpecification Web services operation. The new version of the order can:

- Change existing data
- Remove existing data
- Add new data

**Note:** You can create revision orders by using the Task Web client. This is typically used only for testing or for low-volume order processing.

2. OSM receives the revision order. OSM checks to see if the base order is amendable. You enable this on the order specification. If the base order is not amendable, the order is not a revision order.

**Note:** When you model orders, ensure that orders that are expected to be amended are configured to be amendable. If not, an order that is sent as a revision order is instead processed as a new order. This can cause errors during fulfillment because there are two orders fulfilling the same services for the same customer.

3. OSM checks in-flight orders for a matching value to an order key. For example, you can specify to use the sales order number as the order key. In that case, when OSM processes an order, it looks for an in-flight order that has the same sales order number. If OSM finds an in-flight order with a matching sales order number, OSM treats the new incoming customer order as a revision on the existing order. See "About Order Keys" for more information.

OSM now has two orders to work with: the revision order and the base order.

4. OSM performs further checks on the base order to determine if the order is allowed to be amended. OSM does the following:

- OSM checks to see if the base order is in a state that can be amended. Orders in the Not Started, Completed, or Aborted state cannot be amended. You can customize the allowed transitions to the amending order state by configuring the order life-cycle policy. See "About OSM Order Life-Cycle Management" for more information.

- OSM checks to see if the base order has not passed the point of no return. The **point of no return** is the point in the processing of an order item after which order amendments are either impossible or incur some penalty. In this case, a revision order might not be possible. See "About Point of No Return" for information.

- OSM checks to see if the incoming customer order has a version identifier. If OSM has a version identifier, OSM compares the value of the version to the
version of the in-flight order. If the version of the in-flight order is greater than the version of the incoming customer order, the incoming revision is ignored.

If a revision cannot be processed, or if the order life-cycle has not allowed the revision, the revision order is set to the Failed state, and the base order continues to be processed.

5. OSM determines whether amendment processing is needed by analyzing order data at the following levels:
   - OSM compares the revision order data and the base order data (or the revision order data and the last submitted revision order data) to see if a compensation is needed. (Compensation defines the actions that need to be taken to perform amendment processing; for example, undo and redo.) See “About Order-Level and Task-Level Compensation Analysis” for more information.
   - During compensation, OSM compares task data for each task in the order process to further validate the compensation requirements. See “About Order-Level and Task-Level Compensation Analysis” for more information.
   - OSM uses the significance of the data to determine if compensation is needed at both stages. Data significance allows you to optimize amendment processing in a way that compensation is considered only for changes to data that is marked as significant. Data that is not marked significant is updated but does not get included in the compensation plan if its value is changed. See “About Data Significance” for more information.

6. After determining that amendment processing is needed, OSM transitions the order to the Amending state.

---

**Note:** OSM queues orders that need amending. This means it is possible that multiple revisions are submitted and queued for the same order while the order is in the queue. If amending the order is allowed, OSM chooses the latest version of the amendments in queue by looking at the optional version identifier (if configured) or by the date and time in which they were received if there is no configured version identifier.

In the Process Amendment state, OSM determines the compensation required. For example, OSM might redo a task with different values for one or more data elements on the task data that were used for input into the task.

For process-based orders, the tasks are analyzed to find the impact of the changes. That impact determines the compensation plan. For example, OSM might need to redo a task with different data values or undo a task if it is no longer required. The data comparison is based on the data in the creation task of the base order and the revision order. See “About Compensation” for information.

For orchestration orders, the order components of the orchestration plan are analyzed to determine which order components need to be redone, undone, or done for the first time (amend do). The tasks of the sub-processes run for each of those order components to be compensated are also analyzed.

7. OSM handles the base order and the revision order as follows:
   - For the base order, OSM creates a new orchestration plan that includes the order components and their dependencies. Any order components with data that has changed as a result of the revision are redone. Any order components that have been processed but are no longer required in the revision are undone.
in reverse dependency sequence. Any order components that are inserted as new requirements are fulfilled. The order state is set to Amending.

- For the revision order, OSM transitions it to the Completed state because its only purpose was to revise the base order.

8. OSM processes the changes according to the compensation plan it calculated and re-calculates the compensation plan needed after every change. OSM performs the necessary undo, redo, and amend do operations on order components (for orchestration orders) and on tasks (for both orchestration orders and process-based orders).

Figure 7–1 shows a simplified amendment processing flow.

A simple example of a revision order is as follows:

1. A customer orders a DSL service at 3 MBps. An order is created and sent to OSM.

   Figure 7–2 shows the start of the process. In this example, the process begins with the Verify_ADSL_Service task and then transitions to the Assign_Port task.
2. OSM verifies that the 3 MBps service is available and transitions to the next task, **Assign Port**.

3. While the order is waiting for port assignment, the customer calls back and asks a customer service representative (CSR) to change the order to 5 MBps. The CSR creates a revision order in the CRM system with the revised bandwidth value of 5MBps and submits the order to OSM.

4. OSM receives the incoming customer order, and detects that it is a revision to an in-flight order.

5. OSM accepts the revision order, calculates the compensation plan, and begins to run it. OSM knows that compensation is necessary because the data (bandwidth) that was on the order as input data when this task ran previously has now changed. The revision order requests that the **Verify ADSL Service Availability** task must be redone to ensure that the 5 MBps service is available.

6. The value set by the **Verify ADSL Service Availability** task is changed. 

   Figure 7–3 shows the order displayed in the Task Web client. In this figure, the **Verify ADSL Service Availability** task has an execution mode of Redo. Because the port has not been assigned yet, the **Assign Port** task has an execution mode of Do, but it cannot be worked on until the order completes compensation for the revision.

   The task execution mode can be Undo, Redo, Do, or Amend Do. (See "About Task Execution Modes" for more information.)

7. The revision order transitions directly to the Completed state. This is because the revision order is used only for updating the base order. For revision tracking, OSM maintains a record of the revision order as part of the order history.
8. After verifying that the revised bandwidth is available, the base order continues processing.

You can monitor revisions in the Web clients. Figure 7–4 shows a revision order (Order 316) and the base order that it revised (Order 315).

**Figure 7–4  Revision Order and the Base Order it Amended**

<table>
<thead>
<tr>
<th>Order ID 316: Order Details</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Summary</strong></td>
</tr>
<tr>
<td>General</td>
</tr>
<tr>
<td>Order ID</td>
</tr>
<tr>
<td>Reference</td>
</tr>
<tr>
<td>Version</td>
</tr>
<tr>
<td>Priority</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Order ID 315: Order Details</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Summary</strong></td>
</tr>
<tr>
<td>General</td>
</tr>
<tr>
<td>Order ID</td>
</tr>
<tr>
<td>Reference</td>
</tr>
<tr>
<td>Version</td>
</tr>
<tr>
<td>Priority</td>
</tr>
</tbody>
</table>

**About Revising or Canceling Orders by Using the Task Web Client**

In most cases, revision orders are submitted from an order-source system. You can also revise and cancel orders by using the OSM Task Web client; for example, by using the Amend Order menu command. This is useful for testing revisions and cancellations within OSM, however, this method is not appropriate for production systems.

You should use the Task Web client to submit amendments only when the order was submitted from the Task Web client originally or when the upstream system cannot submit an amendment. If the upstream system submits an amendment after you manually submit an amendment, data synchronization errors can occur.

When you use the Task Web client to amend an order:

1. OSM creates another order, with a new order ID number, for the revision. The new order includes all of the creation task data from the in-flight order.

2. The Task Web client displays the revision order.
3. You can then change the data required for the revision and submit the revision order.

**Important:** If you use revision versioning, increment the revision version.

---

**About Order Keys**

When receiving an order flagged as amendable, OSM checks in-flight orders for a matching value in an order key. (You configure the order key when you model the order specification.) For example, you can specify to use the sales order number as the order key. In that case, when OSM processes an order, it looks for an existing order that has the same sales order number and amends that order.

**Tip:** Because OSM must check the order key for all in-flight amendable orders, you should make orders amendable only if they might need to be amended. That way, OSM does not need to check for an order key for orders that would not be amended.

You define the order key in the order specification as one or more XPath expressions that reference one or more data elements in the incoming customer order. If you use multiple data elements, the values are concatenated in the order key.

OSM generates an order key when the order is created. To assign an order key:

- The order key data elements must be part of the creation task data.
- The order key must be identical between the base order and the revision orders and must not change.
- The order must be flagged as amendable.

Order key values should not be modified after an order is submitted.

---

**About Submitting Multiple Revisions of an Order**

In some cases, multiple revisions to a single order are submitted. Each revision is expected to be a new revision of the in-flight order, not a cumulative comparison of previous revisions. The latest amendment is assumed to be the most complete revision containing all of the changes from earlier revisions. Intermediate revisions are not processed by OSM.

You can use versioning in the revision orders to recognize the order of the revisions as OSM receives them. For example:

- If revisions are received out of sequence, OSM ensures that the latest revision is used. If a later revision is received while the current revision is being compensated, OSM completes the compensation for the current revision before processing the latest version. If a version is received that is earlier than the current revision being processed, the earlier version is ignored.

- If several revisions are received, OSM discards interim revisions and applies the latest revision because it represents the latest customer instructions for the order and is a complete copy of the base order.

To configure revision versioning, you specify a data element on the incoming customer order that OSM checks when processing revisions for the order. You specify the data
element as an XPath expression in the order specification **Amendable** tab. For example, if the data element is `<version>`, the XPath expression is:

```
_root/version
```

**Figure 7–5** shows the version specified in an order.

**Figure 7–5  Revision Version Data Element Specified in an Order**

![Order: OsmCentralOMExampleOrder](image)

<table>
<thead>
<tr>
<th>Display Name</th>
<th>OsmCentralOMExampleOrder</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Amendable</strong></td>
<td></td>
</tr>
<tr>
<td>Not Amendable</td>
<td></td>
</tr>
<tr>
<td>Amendable</td>
<td></td>
</tr>
<tr>
<td><strong>Key</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Expression</strong></td>
<td></td>
</tr>
<tr>
<td><code>&lt;</code></td>
<td>_root/OrderHeader/numSalesOrder</td>
</tr>
<tr>
<td><strong>Remove</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Add</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Events</strong></td>
<td></td>
</tr>
<tr>
<td>Amendment Abandoned</td>
<td></td>
</tr>
<tr>
<td>Amendment Started</td>
<td></td>
</tr>
<tr>
<td>Order Created</td>
<td></td>
</tr>
<tr>
<td>Amendment Completed</td>
<td></td>
</tr>
<tr>
<td>State Changed</td>
<td></td>
</tr>
<tr>
<td>Order Removed</td>
<td></td>
</tr>
<tr>
<td><strong>Version</strong></td>
<td></td>
</tr>
<tr>
<td>_root/version</td>
<td></td>
</tr>
</tbody>
</table>

**About Controlling When Amendment Processing Starts**

You can delay amendment processing for an order. For example, the order might be in the middle of running an automated task that is executing system interactions with fulfillment systems, so you want to postpone the processing of the revision until after the tasks complete. After the system interaction is complete, OSM can begin processing the revision.

During amendment processing, the order is in the Amending state, which prevents normal processing such as task updates. This allows compensation to deal with one set of data changes without also needing to carry out normal processing activities at the same time. To manage the transition to the Amending state, OSM does the following:

1. Checks permissions to allow or postpone the processing of the revision.
2. Checks if a grace period is set to allow all order activity to settle. If so, it waits for the grace period to end.
3. Transitions the order to the Amending state.

To control when amendment processing starts, you use the order life-cycle policy to control OSM transactions. A **transaction** is an action taken by the OSM system. For example, for the In Progress state, you can prevent the Process Amendment transaction from occurring until a condition is true.
See "About Managing Order States and Transitions" for more information about transactions.

To manage amendment processing, OSM uses two order state transactions, in the following order:

1. **Submit amendment.** This transaction occurs when the revision order is submitted. You can specify conditions that determine if the order can be amended or not. Because the evaluation of the condition is triggered when the revision order is submitted, the condition does not need to be based on data, but it can include data as part of the condition.

2. **Process amendment.** If the revision order is accepted, OSM evaluates this transaction to determine if the amendment can be processed now, or if it needs to wait for a specified amount of time, or if it needs to wait until all accepted tasks are completed. This condition is evaluated based on data in the order. If the condition returns false, the amendment is queued. The condition is re-evaluated whenever the data changes. When the condition evaluates to true, the transition to the Amending state can occur.

A **grace period** specifies a period of time to wait for all accepted tasks to complete before an order can transition to a different state. For example, if an automated task has sent a request to an external system, but the external system has not responded, OSM does not know if the task has been completed and therefore does not know if the task needs to be compensated. A grace period set on the Process Amendment order state transaction can allow the order the opportunity to reach a known state for all current tasks before transitioning to the Amending state.

Grace periods are defaulted to be indefinite, so OSM waits until all currently accepted tasks are completed before transitioning to the target state. You can limit the grace period:

- You can set the grace period to zero, which specifies that OSM not wait for any accepted tasks to complete before transitioning to the target state
- You can provide a time limit; for example, one hour (to give all accepted tasks a limited time to complete before transitioning to the target state).

If an automation response is received for a task after the order has transitioned to the Amending state, an automation exception is thrown, because the automation plug-in cannot process the response when the order is in the Amending state. The automation exception is sent to the JMS response queue and is retried. When the retry limit is reached, the message is forwarded to an error destination, if one is configured. To manage exceptions that occur during amendment processing, you can review the errors to determine if the messages can be resubmitted or handled by fallout.

If there are multiple queued revisions waiting for the grace period to end, OSM selects the latest version among the queued amendments to process. The other versions are assumed to be out of date and are ignored. See "About Submitting Multiple Revisions of an Order" and "About Modeling Life-Cycle Policies" for more information. See "OSM Order States and Transactions" for information about order states.

**About Compensation**

To manage changes to an order, OSM uses order compensation. OSM analyzes the required order changes and their impact on everything that has already been completed by the in-flight order including manual updates from Task Web client users and order updates from automated tasks. OSM then creates a compensation plan to define the actions that need to be carried out to amend the in-flight order. After
compensation has ended, the in-flight order will have incorporated the required order changes and continues executing normally. You can recognize when compensation is happening to an order when the order is in the Amending state or the Cancelling state.

---

**Note:** If you submit a revision order that uses a different cartridge version from the one that the original base order was created with, OSM uses the original base order cartridge version to execute any required compensation tasks and not the cartridge version used to create the revision order.

---

When you model OSM entities such as order specifications and tasks, aspects of your model affect compensation, including the following:

- **Order and task level compensation analysis.** OSM analyzes data at the order level and at the task level to determine whether compensation is required. When you model compensation scenarios, be aware that data can come from revision orders, update order transactions, and compensations from previous revision orders. Discrepancies between revision order data, update order transaction data, and compensations from previous revision order data triggers compensation. See "About Order-Level and Task-Level Compensation Analysis" for more information about how OSM detects data discrepancies.

- **Order data position and order data keys.** OSM analyzes data from revision orders with base orders (or between this revision order data and the last submitted revision order data) based on XML order data position or based on order data keys. See "About Order Data Position and Order Data Keys" for more information.

- **Data significance.** This specifies which data needs to be considered for compensation. See "About Data Significance" for more information.

- **Task execution modes.** This specifies what can be done; for example, redo or undo. See "About Task Execution Modes" for more information.

- **Order life-cycle events.** There are several events specific to amendments that are automatically generated when enabled.

You can model compensation in the following entities:

- **Tasks.** See "Modeling Compensation for Tasks" for more information.


- **Automations.** See "Modeling Compensation for Task Automation Plug-Ins" for more information.

You cannot configure compensation for timer delays or event delays in processes.

---

**About Order-Level and Task-Level Compensation Analysis**

When the revision order is received, OSM analyzes the differences between the revision order data and the base order data (or between this revision order data and the last submitted revision order data) to see if a compensation is indicated. Changes and updates to order data can occur in the context of tasks data views or order data views.

OSM then begins analyzing impacted tasks. OSM provides the following data perspectives for each individual task which are snapshots of data that OSM uses to calculate whether a task needs to be compensated. These data perspectives are:
- **Historical order perspective (HOP):** Specifies the data used when the task last ran in Do mode and changed to the Completed state (or Redo mode if the task last ran as part of compensation for a previously submitted revision order).

- **Contemporary order perspective (COP):** Specifies the unchanged task data from the last time the task completed in do or redo mode (for example from the tasks run for the base order or for a previous revision order). COP also shows any new or changed data from the current revision order and from the tasks triggered from that revision order that compensated prior in the process flow to the compensation task currently being analyzed.

- **Real-time order perspective (ROP):** Specifies the last change to a parameter value by any task or at the order level (for example through order level updates). This perspective may be different from the COP because the COP only provides a view of task data for previously run compensation tasks and revision order data and may not represent the last change to a parameter value. For example, the COP may include unchanged data from when the parameter that was originally processed by the Task, but that same data parameter could have been updated in a later task and so the current data would have a different value than the one displayed in the COP.

Figure 7–6 describes a process-based order, where a subscriber requests ADSL service with 3MBps speed. The order is submitted to OSM and service fulfillment begins. The subscriber calls back while the base order is in-flight and has just completed the **Activate DSLAM** task and requests the order be changed from 3MBps to 5MBps speed. In this scenario, the existing port does not support 5MBps. The compensation process proceeds as follows:

1. When OSM receives the revision order, OSM compares the creation task data of the revision order with the creation task data of the base order to determine if any data changes have occurred to significant data.

2. Because the bandwidth changed from 3 to 5 MBps and the bandwidth parameter is designated as significant, OSM begins task level analysis for the first task in the process. OSM compares the **Verify ADSL Service Availability** HOP and COP and determines that the task must be redone because of the bandwidth change and because the compensation strategy for that task is redo. OSM updates the results of the task and any data changes on account of redoing the task to the ROP. The **Verify ADSL Service Availability** ROP becomes the COP for the **Ship Modem** task and the **Assign Port** task.

3. The compensation mode for **Ship Modem** is Do Nothing, so no compensation analysis occurs for that task. The compensation mode for **Assign Port** is Redo, so compensation analysis begins for that task. OSM compares the HOP and COP for the Assign Port task and determines that the task must be redone because of the bandwidth change. OSM adds the result of redoing the task to the ROP which includes the bandwidth change and a new port ID because the original port ID could not handle the increased bandwidth requirement. The ROP becomes the COP for the **Activate DSLAM** task.

4. The compensation mode for **Activate DSLAM** is Undo then do, so compensation analysis begins for that task. OSM compares the HOP and the COP for the **Activate DSLAM** task and determines that the task must be undone then redone because of the new port ID and the bandwidth changes. OSM adds the results to the ROP. Processing continues normally after this task.
Figure 7–6  Changing a Service Request

- **Verify ADSL Service Availability**
  - Mode = Do
  - Primary Phone = 555-5555
  - Bandwidth = 3 MBps
  - Result = Succeed

- **Ship Modem**
  - Mode = Do
  - Address Node
  - Result = Succeed

- **Activate DSLAM**
  - Mode = Do
  - Primary Phone = 555-5555
  - Bandwidth = 3 MBps
  - Port ID = 35
  - Result = Succeed

- **Assign Port**
  - Mode = Do
  - Primary Phone = 555-5555
  - Bandwidth = 3 MBps
  - Port ID = 35
  - Result = Succeed

- **Add ADSL Siebel Creation Task**
  - Address Node
  - Primary Phone = 555-5555
  - Bandwidth = 3 MBps

- **Revision Order**
  - **Add ADSL Siebel Creation Task**
    - Address Node
    - Primary Phone = 555-5555
    - Bandwidth = 5 MBps
  - **Verify ADSL Service Availability**
    - Mode = Redo
    - HOP
      - Address Node
      - Primary Phone = 555-5555
      - Bandwidth = 3 MBps
    - COP
      - Address Node
      - Primary Phone = 555-5555
      - Bandwidth = 5 MBps
    - ROP
      - Address Node
      - Primary Phone = 555-5555
      - Bandwidth = 5 MBps
    - Result = Succeed
  - **Task Level Compensation Analysis**
    - **Activate DSLAM**
      - Mode = Undo then do
        - HOP
          - Primary Phone = 555-5555
          - Bandwidth = 3 MBps
          - Port ID = 35
        - COP
          - Primary Phone = 555-5555
          - Bandwidth = 5 MBps
          - Port ID = 45
        - Result = Succeed
    - **Assign Port**
      - Mode = Redo
      - HOP
        - Primary Phone = 555-5555
        - Bandwidth = 3 MBps
        - Port ID = 35
      - COP
        - Primary Phone = 555-5555
        - Bandwidth = 5 MBps
        - Port ID = 45
      - Result = Succeed
As described in "About Update Order Transactions", you can use update order transactions to make changes to order data using automation plug-ins from the task context (this includes automated task, task event, and task jeopardy notification automations) and also from the order context (this includes order level notification, event, and jeopardy automations). OSM captures any data update made from a task context in the HOP and COP and are therefore guaranteed to be reflected in any compensation analysis for that task initiated by new revision orders. Order updates can also be applied to the order level data by referencing the view for that order data defined in the query task that you can associate to an order in the Order Specification editor Permissions tab, Query Task sub tab (see "About Query Tasks for Order Automation Plug-ins" for more information about query tasks for order level data).

Updates at the order data level should be done with care because these updates are not included as part of compensation analysis and do not generate a HOP or COP. OSM attempts to apply any order level change to the closest task that has been created or completed, but these updates are not guaranteed deterministically like the task level update are. For more information about how update orders can be used in automation plug-ins, see OSM Developer’s Guide.

OSM does the following when discrepancies occur between the contemporary order perspective and the historical order perspective:

- Adds revision order nodes if they do not match nodes of the last submitted order data or the nodes in the historical task perspective.
- Changes revision order node values if the nodes do match the values found in the last submitted order data or the nodes in the historical task perspective.
- Deletes nodes if the nodes are in the last submitted order data or in the historical task perspective but not in the revision order.

In Design Studio, you can model compensation strategies for manual and automated tasks statically from a predefined list or dynamically from revision order data. If you model the compensation task dynamically, you can create an XQuery that has access to order data provided in the contemporary and historical perspectives as well as a comparison between the two. You can use the results of this comparison to dynamically select an appropriate task-level compensation strategy. For more information about compensation strategies, see "Modeling Compensation for Tasks".

**About Order Data Position and Order Data Keys**

OSM compares order data in the following ways:

- By comparing the position of the XML nodes of the base order (or last submitted revision order), with the position of the XML nodes in the current revision order.
- By comparing order data keys in the Order Template editor for specific data elements. When OSM receives a revision order, it compares the order data keys from the revision order with the order data keys in the base order (or last submitted revision order). When OSM finds a matching order data key, it compares the values in each element.

**Note:** OSM uses order data keys to determine order data changes during compensation and to identify pivot nodes that generate multiple task instances based on multi-instance data nodes (see "Generating Multiple Task Instances from a Multi-Instance Field") and should be distinguished from order keys used to match base orders with revision orders (see "About Order Keys").
To set an order key for a data element value, you must specify the data element as an XPath expression in the Key subtab on the Order Template Node editor.

Oracle recommends using order data keys for multi-instance data nodes to differentiate between instances of the same data node. For example, the data structure in Figure 7–7 can be used multiple times to identify different fulfillment patterns. You can associate an order data key to the children nodes of specification to uniquely identify each instance of a fulfillment pattern contained in a customer order.

Figure 7–7 Order Data Key Defined in Design Studio

You could also make every instance of specification unique by setting a key on specification that points to the name child node. The expression for this key would be:

\[ ./name \]

About Data Significance

During amendment processing, OSM identifies all tasks in the order that are affected by the changed order data. It then determines whether the data being changed is flagged as significant. (When you define orders or tasks, you can mark data as
**Significant or Not Significant.** By default, all data is flagged significant.) OSM compensates only those tasks that process significant data.

If any of the data changes are significant, OSM transitions the order to the Amending state and builds a compensation plan based on all affected tasks, creating redo or undo compensation tasks as necessary.

Changes to non-significant data are updated on the in-flight order. For example, if the customer’s preferred contact method (email or text message) is marked as non-significant, a revision order that changes only that data does not trigger amendment processing. Instead, the base order is changed, and the revision order is completed without starting amendment processing. The next task that uses the changed data uses the updated values.

You can configure data significance at the following levels:

- Data Dictionary
- Order template data
- Task data

Each level can inherit or override the significance flag of its parent level. The Data Dictionary is at the top parent level.

The order template can inherit or override the data significance specified in the Data Dictionary. This allows one order type to consider the data significant while another order type does not.

The task data can inherit the data significance set in the order template only to override it as non-significant data. This allows data to be significant in one task and not significant in another. In that case, a revision with that one data element changed would cause the task that considers the data element significant to be compensated: the task that does not consider it significant will not be compensated.

It is not possible to specify a data element as not significant at the order level and significant at the task level.

*Figure 7–8* shows how data significance can be inherited and overridden.
Figure 7–8  Data Significance Inheritance

Figure 7–9 shows data significance specified in a data schema. Because this is the top level, there is nothing to inherit the significance from, so there is no inheritance option.

Figure 7–9  Data Significance Specified in a Data Schema

Figure 7–10 shows data significance specified in the order template. In this example, the significance is inherited from the Data Dictionary.
Figure 7–10  Data Significance Specified in the Order Template

![Order Template Node - /CustomerDetails/areaCode](image)

Figure 7–11 shows significance specified in the task data. Note that the significance is either inherited, or it is not significant. There is no option for significant: instead, that value is inherited from the order template.

Figure 7–11  Data Significance Specified in the Task Data

![Task Data Node/CustomerDetails/areaCode](image)

About Task Execution Modes

Tasks run in the following modes:

- **Do.** This is the normal execution mode of a task when the order is in the In Progress state.

- **Undo.** This execution mode is used when the task must undo work that has already been done; for example, to un-assign a port when an order is canceled.

Undoing tasks is performed in reverse order to how they were run. For example, if task B was completed after task A, then task B is undone before task A is undone.

Undo is used when the order component in the base order has completed, and the revision order has no corresponding order component. A cancellation order, therefore, can include no order components. This causes all of the order components in the base order to be undone. The **Orchestration Plan** tab in the
Order Management Web Client displays nothing when this is the case, indicating that the order may have been canceled.

- **Redo.** This execution mode is used when the task must redo work that has already been done; for example, a port assignment task that needs to be performed again because the input value of bandwidth is different as a result of the revision order. Redoing tasks is performed as an optimization of the Undo and Do operations for a task in a single operation.

  The Redo execution mode is used when an order component has completed in the base order, and the revision order has the same order component, but specifies different order items or data values.

- **Amend Do.** This execution mode is used when a new task must be performed while the order is in the Amending state. For example, the revision order might specify to add a service that was not in the base order. Because normal processing is not allowed during amendment processing, the Do mode cannot be used; Amend Do is used instead.

  The Amend Do execution mode functions like the Do execution mode. When a task runs in the Amend Do mode, all of the permissions and automation plug-in logic for the Do mode of that task apply.

  See "Example 4: Amend Do Compensation" for an example of how the Amend Do execution mode is used.

---

**Note:** You can specify which tasks can be amended by the Redo and Undo compensation modes, but Amend Do is not configurable. This is because OSM determines when Amend Do is required, and the logic followed is that of the Do mode.

---

Table 7–1 summarizes the possible combinations and the required compensation for a revision order.

### Table 7–1 Compensation Types

<table>
<thead>
<tr>
<th>Base Order Component</th>
<th>Revision Order Component</th>
<th>Compensation Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exists</td>
<td>Does not exist</td>
<td>Undo</td>
</tr>
<tr>
<td>Does not exist</td>
<td>Exists</td>
<td>Do (run after compensation is complete) or Amend Do (while the order is in the Amending state)</td>
</tr>
<tr>
<td>Exists</td>
<td>Exists, no changes found</td>
<td>No compensation required</td>
</tr>
<tr>
<td>Exists</td>
<td>Exists, changes found</td>
<td>Redo</td>
</tr>
</tbody>
</table>

---

**Modeling Compensation for Tasks**

To perform compensation, OSM must identify the tasks that need to be compensated and then do or undo them in the appropriate sequence.

A task needs to be compensated if it was completed and a change to at least one significant data element in the task’s data has been made. Tasks in the Received, Accepted, Assigned, or a user-defined state are not compensated. This is because OSM cannot determine what work has been accomplished. For example, an automated task might have sent a request to an external system, but a response has not been received yet. If those tasks have not completed before the grace period expires, they would be
started again after the compensation has completed and the order resumes to the In Progress state.

---

**Note:** When a task is compensated, all its successors must be compensated, whether or not they have significant changes.

---

In the Design Studio **Task Editor Compensation** tab (see Figure 7–12), you can model:

- Static amendment processing compensation strategies for manual and automated tasks using a predefined list. Static compensation strategies are appropriate when the compensation requirements for a task are invariable.

- Dynamic amendment processing compensation strategies for manual and automated tasks based on revision order data using an XQuery expression. Dynamic compensation strategies are appropriate when more than one compensation strategy is required for a task. For example, you could model the XQuery expression to select an **Undo then do** compensation strategy if the revision order bandwidth parameter is greater than 50 MB, and only a **redo** compensation strategy if the bandwidth parameter is less than 50 MB. For more information about dynamically modelling amendment processing compensation strategies, see the **OSM Developer’s Guide**.

---

**Figure 7–12 Task Compensation Options**

As shown in Figure 7–12, there are two scenarios that need to be compensated:
The task needs to be re-evaluated. This means that the task includes significant data and needs to be compensated.

The static amendment processing compensation options are:

- Redo in one operation. This option is recommended because it performs the fewest number of Undo and Do operations necessary for compensation.
  
  In the case of a manual task, the task will appear in the worklist in Redo mode, and the user can display the historical perspective and the contemporary perspective of the task data (from the last time the task was run) in two separate tabs. The user updates the data on the Contemporary Perspective tab and completes the task.

- Undo and Redo in two operations. Use this option when you need to roll back all order changes and perform the task again from the beginning. This option is useful when interacting with an external system that has no redo action but can process equivalent do and undo actions (for example, in the external system, implement and cancel).

- Do nothing. Use this option if redoing the task is not necessary. For example, a task that sends a customer survey email would not need to be redone, even if it includes significant data.

In addition, you can select the Compensation Expression check box and enter an XQuery that dynamically selects an amendment processing compensation option at run time based on order data. The dynamic compensation takes precedence over the static compensation unless there is an error in the XQuery itself. If there is an XQuery error, then OSM reverts to the compensation selected with the static options.

The task is no longer required. This occurs when an order is canceled or when a branch of completed tasks in a process becomes obsolete due to a revision.

Figure 7–13 shows a process that has two potential paths. In this example, the base order followed the path from Task_1 to Task_2_1. The revision caused the path to change to follow Task_1 to Task_2_2. This means that Task_2_1, Task_3_1, and Task_4_1 do not need any compensation, because they are no longer on the path required to fulfill the order.
The static amendment processing compensation options are:

- **Undo.** This option rolls back the task by executing the task in Undo mode to perform the roll-back operation. In the case of a manual task, this requires that the rollback be acknowledged manually in the Task Web client. You cannot update the task data for an undo of a manual task in the Task Web client, because the system will automatically put the data back to what it was prior to the task executing.

- **Do nothing.** This option rolls back the task automatically, without creating an undo task.

In addition, you can select the Compensation Expression check box and enter an XQuery that dynamically selects an amendment processing compensation option at run time based on order data. The dynamic compensation takes precedence over the static compensation unless there is an error in the XQuery itself. If there is an XQuery error, then OSM reverts to the compensation selected with the static options.

### About Managing Compensation in the Task Web Client

When an automated task is redone, it is redone automatically. When a manual task is redone, the Task Web client displays the task with an execution mode of Redo. The manual task must be processed in the Task Web client.
To manage compensation in the Task Web client, you can do the following:

- Perform manual undo and redo operations.
- Display the execution mode under which the tasks is running (Do, Redo, or Undo).
- Display the order state; for example, Amending.
- Display the historical data (the data as it was when the task last run) in the historical order perspective when editing a task.

**Note:** You can assign roles in Design Studio to specify who can redo and undo tasks in the Task Web client. OSM also supports the ability to assign the different execution modes of a task to different roles. This is useful because OSM can compensate using both manual and automated tasks. For example, the regular processing of a task in Do mode could be automated, and the Undo and Redo modes for the same task could be set to a special role to be done manually.

See *OSM Task Web Client User’s Guide* for more information.

**Modeling Compensation for Rules in Processes**

You can specify to redo a rule in a process, or to do nothing. Because rules only evaluate data, and therefore do not modify data or interact with other systems, there is no undo necessary for a rule.

*Figure 7–14* shows rule properties in the Design Studio Process editor when a rule is selected.

*Figure 7–14  Rule Properties*

<table>
<thead>
<tr>
<th>Region1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Property</td>
</tr>
<tr>
<td>Compensation</td>
</tr>
<tr>
<td>Condition</td>
</tr>
<tr>
<td>Description</td>
</tr>
<tr>
<td>Display Name</td>
</tr>
<tr>
<td>Join Type</td>
</tr>
<tr>
<td>Process History</td>
</tr>
</tbody>
</table>

**Modeling Compensation for Task Automation Plug-Ins**

An automated task can include multiple automation plug-ins; for example, senders and receivers. Each automation plug-in can be associated with one or more execution modes. For example, if you create an automated task to activate a service, you can use the same logic to handle the initial activation request and the redo compensation for the activation request.

*Figure 7–15* shows compensation for the sender automation plug-in for an automated task.
Each automated task can have separate plug-ins for each of the three modes; Do, Redo, and Undo. When an automated task runs in Redo or Undo mode, OSM provides information about the task data that was present when the task was last ran. For redo tasks, the Automation framework provides the historical data, the contemporary data, and the delta to the automation plug-in for use in the plug-in logic you write. See *OSM Developer’s Guide* for more information.

### About Compensation and Orchestration

OSM performs compensation on both process-based orders and orchestration orders. When compensating an order that has an orchestration plan, the compensation can change the orchestration plan.

Each orchestration order has its own unique orchestration plan, generated specifically for that order. Therefore, to manage a revision order for the base order, OSM must generate a new orchestration plan for the revision order. The orchestration plan for the revision order can be different from the orchestration plan for the base order; for example, it might include different order components, with different dependencies and different order items.

By contrast, a process-based order has a predefined process; the process is not generated when the order is created. The tasks that make up that process and the flow of those tasks in the process do not change. The data values for those tasks change as a result of a revision, and the path through the predefined process may change as a result of compensation, but the overall process remains the same.

To manage compensation for an orchestration plan, OSM needs to recognize and account for dependencies between the order components in the order that is being amended. The compensation required depends on whether components exist in one or
both orders’ (revision order and base) orchestration plans and on whether changes to the contents of those order components (such as different order items) exist.

Redoing an order component in an orchestration plan is performed by redoing the tasks run by the order component. In redoing order components, OSM follows the sequence of dependencies in the orchestration plan. OSM takes into account the dependencies from the revised orchestration plan, unless a successor component has previously started in the original base order, in which case the dependency is considered resolved.

OSM analyzes the order component compensation type and component dependencies to determine the sequence of component compensation. OSM performs order component compensations in the following stages:

- **Reverse compensation:** In this stage, OSM performs only undo compensation tasks for order components. OSM performs undo tasks for order components in the reverse order of dependencies that existed between the components in the original.
  
  For example, OSM performs undo tasks for order component B before performing any undo task for order component A if B was dependent upon A in the base order.

- **Forward compensation:** In this stage, OSM fulfills order components that have changed (redo) or been introduced (amend do) based on the order of dependencies, which is derived after taking into account dependencies from the revised orchestration plan.

  The revised orchestration plan may include new components to be completed using Amend Do and Redo compensation types.

---

**Note:** When switching from reverse to forward compensation, OSM identifies the new order components that need to be completed using the Amend Do compensation type. These components participate in the compensation plan as compensation items. This facilitates appropriate compensation sequencing for compensation tasks of existing downstream order components or other components that require amend do compensation.

---

All processing not related to compensation is suspended for an orchestration plan until compensation is complete. After compensation is complete, the order is restored from the Amending state to an In Progress state and normal processing continues.

### Compensation Considerations for Tasks and Processes

You can plan for compensation when designing tasks. When an order amendment is received, tasks should be undone and redone only if the order change requires it. For example, a provisioning system task might have a data element that contains the full customer name. You might include this information in the task, even if it is not something that needs to be sent to another system. However, a change to this data element would not require any action in OSM for the provisioning system. The same data element might, however, be significant in a task for a billing system. So, when designing tasks it is important to ensure that the correct data for the task has been marked as significant. This optimizes the calculation of the deltas for each task, and ensures that only the necessary tasks are compensated.
When modeling processes, you can use subprocesses to control the scope of compensation. For example, Figure 7–16 shows a process that includes only tasks, with no subprocesses.

**Figure 7–16 Process Example with No Subprocesses**

In this figure, there is a choice between two possible flows: one through Task_2_1 and the other through Task_2_2, depending on the exit status of Task_1.

Assume this flow has completed Task_1, Task_2_2, Task_3, and Task_4 and has reached Task_Final when a revision is received. The revision indicates that the process now should go through Task_2_1 instead of Task_2_2.

In this scenario, the compensation plan undoes Task_4, then undoes Task_3 and then undoes Task_2_2, before re-processing with the correct data.

Now consider the following flow, which is modified so that Task_1, Task_2_1, Task_2_2, and Task_3 are now contained in a subprocess.
Assume that the same processing has taken place as before. Task_1, Task_2_2, and Task_3 are run in the subprocess, and Task_4 is run in the main process, which reaches Task_Final when a revision is received. With this configuration, the compensation for SubProc1 takes place before the compensation for Task_4. Task_4 does not need compensation, because the compensation is isolated in the subprocess.

**Figure 7–17 Process Example with Subprocess**

Compensation Examples

The following examples show different compensation scenarios.

**Example 2: Compensation During Provisioning**

*Figure 7–18* shows a compensation scenario for an orchestration order. In this example, OSM is running in the central order management role, fulfilling multiple functions.

1. The base order requires provisioning, billing, and customer account order components (sync customer).

2. A revision order is submitted while the order is carrying out provisioning tasks. The revision order replaces a medium-capacity service (3 MBps) with a high-capacity service (8 MBps). In this case, the content of the order components has changed in the revision order’s orchestration plan, but the order components it contains and their dependencies remain the same.
3. Because the revision order was received during the base order provisioning, the compensation specifies that the provisioning order component must be redone, after which the order returns to the In Progress state, and the billing and sync customer components are then run with no compensation required.

**Figure 7–18  Example of Compensation that Occurs During Provisioning**

**Example 3: Compensation During Billing**

*Figure 7–19* shows a compensation scenario for billing:

1. The base order includes provisioning order components for a fixed-line service and a cable broadband service.

2. A revision order is received after the provisioning order components are complete but while the billing order components are being processed. The revision order cancels the fixed-line service.

3. The compensation plan specifies to undo the fixed-line service and to redo the billing order components. The cable broadband service requires no compensation. Following the redo of the billing order components, the order resumes normal processing.
Example 4: Amend Do Compensation

Figure 7–20 shows a scenario for Amend Do compensation:

1. The base order includes provisioning order components for a broadband service.
2. A revision order is submitted while billing order components are being processed. The revision order adds a fixed-line service.
3. The compensation plan creates a new dependency between the fixed-line service and the broadband service. Therefore, OSM must use Amend Do to first perform the new task and then process the broadband service order components. The billing order components are redone, and processing continues normally.
Examples of Changes to Orchestration Plans

You can use the OSM Order Management Web client to see how compensation affects an order’s orchestration plan.

Figure 7–21 shows how an orchestration plan changes when a single service attribute changes. In this example, the connection speed changes from 8 MBps to 16 MBps. The order components remain the same, but the value of the connection speed changes in the provisioning component and in the billing component.
Figure 7–21  Orchestration Change Due to Revision: Change Service Attribute

Figure 7–22 shows how an orchestration plan changes when a revision order removes a service from the base order. In the example, the Fixed service was ordered in the base order, but it was removed in the revision order. The provisioning and billing components are removed, and the DSL provisioning component no longer has a dependency on the Fixed order component.
Figure 7–23 shows how an orchestration plan changes when a service is added by a revision order. In this example, the Fixed service is added. This creates a new dependency for the DSL provisioning component.
About Point of No Return

In some cases, there may be a point in the order process after which it becomes impossible or undesirable to make changes to an order. This is called a point of no return.

There are two types of point of no return in OSM.

- **A hard point of no return** indicates that amendments to the relevant part of the order are either impossible or undesirable. In the case of a hard point of no return, a revision order is not possible. Instead, you can create a follow-on order. See "About Inter-Order Dependencies" for more information about follow-on orders.

  **Note:** A follow-on order is not a change to an in-flight order but is an alternative when revising the in-flight order is not possible. Follow-on orders are used to make changes to items on an order that have not yet been completed but are past the point of no return. OSM manages follow-on orders to ensure they do not run until the order items upon which they depend are completed.

- **A soft point of no return** indicates that order amendment processing is still possible, but there are consequences for the customer. For example, you can specify to bill a customer for an extra charge if the order is revised after the soft point of no return has been reached.

You can define multiple point of no return milestones in an order’s fulfillment flow. For example:

- For a fixed-line service, a point of no return after provisioning.
- For a broadband service, a point of no return after billing.
A point of no return is typically set at the order item level. This allows order components with varying processing durations to run, instead of stopping the entire order at the first order item with a point of no return.

Figure 7–24 shows two different point of no return scenarios.

**Figure 7–24  Point of No Return for Different Services**

![Diagram of Fixed-line Service and Broadband Service](image)

**Fulfillment Pattern Point of No Return**

There are two ways to set a point of no return. The first, only available for orchestration orders, is to set it on the fulfillment pattern using fulfillment states. The second is to write an expression in the order life-cycle policy.

When you use the fulfillment pattern to set a point of no return, the point of no return rules set a point of no return value for that order component. Order life-cycle policy conditions can then leverage this point of no return value for restricting order amendments.

When you create a point of no return, model the following in Design Studio:

- Define fulfillment states. These are required before configuring a point of no return on the fulfillment pattern. See "About Fulfillment States" for more information.

- Define a point of no return value list in the fulfillment pattern. Create a name for your point of no return and indicate whether it is a hard point of no return or not. Alternatively, you can create a point of no return value on the fulfillment pattern extended by your fulfillment pattern and allow the point of no return values to be inherited.

- Define point of no return rules for the point of no return values you created. Point of no return rules are specified at the order component level. Point of no return rules involve selecting one or more fulfillment states to map to the specified point of no return value. Additionally, because order component definitions are hierarchical, a sub-component of the order component associated with the orchestration plan inherits the point of no return rules defined on the orchestration plan.
About Modeling Order Change Management

Managing Changes to Orders

plan order component. This sub-component may also specify its own additional point of no return rules.

Life-Cycle Policy Point of No Return

When you use life-cycle policies to set a point of no return, you define the point of no return as an expression in the order life-cycle policy, by setting conditions on the Submit Amendment transaction.

The following example shows a simple point of no return expression:

```
declare namespace oms="urn:com:metasolv:oms:xmlapi:1";
declare namespace osm="http://xmlns.oracle.com/communications/ordermanagement/model";

let $taskData := fn:root(.)/GetOrder.Response
let $rootData := $taskData/_root

return
if (($rootData/PoNR/text() = "HARD"))
then (true())
else (false())
```

For an expanded example of a point-of-no-return expression, see the Design Studio Modeling OSM Processes Help.

When a revision order is received, OSM checks the life-cycle policy to see if there are any point of no return conditions preventing the transition to the Amending order status. If OSM finds any point of no return conditions that are met, the order is not allowed to be amended. In the example above, if the broadband service is billed before the fixed-line service is provisioned, the order has passed the point of no return, even though the fixed-line service has not passed its point of no return.

If the life-cycle policy determines that the revision is not allowed, an OrderTransactionNotAllowedFault message is returned to the order-source that submitted the revision order.

About Modeling Order Change Management

When you model order change management, you configure the following OSM entities:

- Data dictionary. When you create data elements, you can assign them significance. If data is significant, it is considered for amending. See "About Data Significance" for more information.
- Order specification. When you create an order specification, you configure the following:
  - Data significance at the order level. You can inherit significance from the Data Dictionary, or you can define order-specific significance. See "About Data Significance" for more information.
  - If the order is amendable or not.
  - The order key. See "About Order Keys" for more information.
  - The data element that defines the order version. See "About Submitting Multiple Revisions of an Order" for more information.
About Modeling Order Change Management

- Whether or not to publish order events about amendment processing. You can choose to publish events when an amendment is started, completed, or abandoned. An amendment can be abandoned when it is queued for processing and a subsequent amendment supersedes it. See "Order Life-Cycle Events" for more information.

Tasks. You can configure the following:
- Data significance at the task level. See "About Data Significance" for more information.
- How the tasks should be compensated. See "Modeling Compensation for Tasks" for more information.
- The roles that can redo and undo tasks.
- If automated, the automation plug-ins for redo and undo modes of tasks.

Rules in processes. You can configure if the rule should be redone or not. See "Modeling Compensation for Rules in Processes" for more information.

Order life-cycle policy. You configure the conditions that allow an order to be amended. See "About Controlling When Amendment Processing Starts" and "About Point of No Return" for more information. See "OSM Order States and Transactions" for information about order states.

Troubleshooting Order Change Management Modeling

You can use the following methods to troubleshoot your order change management modeling:

- You can use the following OrderManagementDiag.wsdl Web services operations:
  - GetOrderCompensations: Returns a list of compensations against a given order.
  - GetOrderProcessHistory: Returns multiple process history perspectives.
  - GetCompensationPlan: Returns a set of compensation tasks and the dependencies between them.

- See the PONR_{OrderID}.xml file. This file is generated when a Submit Amendment transaction is called.
This chapter describes how to model roles in Oracle Communications Order and Service Management (OSM).

**Note:** Roles are also known as workgroups in the OSM Web clients and in the OSM Administrator application.

See *OSM Administrator Application User’s Guide* for information about assigning users to workgroups.

### About Roles

You create roles to define who can perform OSM order management functions. To use roles, you do the following:

1. Create roles in Oracle Communications Design Studio. When you create roles, you define permissions; for example, you can specify who can display reports in the Task Web client or who can change the priority of a task during order processing.

2. Assign users to workgroups by using the Administrator application.

3. Assign tasks to workgroups by using the Task Web client.

For example, you might create roles based on what a user can do in the Task Web client. You can assign the same tasks to multiple workgroups, but users in each workgroup can work with it differently; for example, you might have a workgroup specifically for fallout management.

Roles are also used by the automation framework. For example, automations use OSM roles to restrict who can receive a notification. If a notification is sent to an external system by using an automation plug-in, ensure that you include the role whose credentials are used when running the automation plug-in.

*Figure 8–1* shows a role defined in Design Studio. In this example, members of workgroups assigned to this role can generate online reports, search for orders, and access the Task Web client Worklist display.
Roles are defined globally in OSM. Using the example shown above, any user assigned to the OrderDisplay workgroup can generate reports. You can also assign permissions to specific order types and tasks. For example, you can specify which roles can create each type of order. This gives you more control over how orders are processed; for example, a fallout role can be allowed to create only fallout-related orders.

Table 8–1 shows the functions that can be assigned to roles in Design Studio.

**Table 8–1 Functions Assigned to Roles**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Versioned Orders</td>
<td>Enables users to create orders for different versions of cartridges. If not granted this permission, users can create orders only for the default version of the cartridge.</td>
</tr>
<tr>
<td>Exception Processing</td>
<td>Enables users to alter the flow of a process by applying exception statuses at any time throughout the process.</td>
</tr>
<tr>
<td>Online Reports</td>
<td>Enables users to view summarized reports on all orders and tasks on the system.</td>
</tr>
<tr>
<td>Order Priority Modification</td>
<td>Enables users to modify the priority of a task in an order.</td>
</tr>
<tr>
<td>Reference Number Modification</td>
<td>Enables users to modify the reference number of an order.</td>
</tr>
<tr>
<td>Search View</td>
<td>Enables users to access the order Query function. See &quot;Specifying Which Data to Display in the OSM Web Clients&quot; for more information.</td>
</tr>
<tr>
<td>Task Assignment</td>
<td>Enables users to assign tasks to others.</td>
</tr>
<tr>
<td>Worklist Viewer</td>
<td>Enables users to display the worklist in the Task Web client.</td>
</tr>
</tbody>
</table>

**Figure 8–1 Role Defined in Design Studio**

Roles are used in the following OSM entities:

- Orders
About Modeling Permissions in Design Studio

- Tasks
- Order components
- Order item specification

When you model orders, you can specify the orders that roles can display in the Web clients and the data in the order that can be displayed. See "About Setting Permissions for Orders" for more information.

Figure 8–2 shows roles defined in an order specification. In this example, members of BillingUpdateRole are allowed to display orders for customers in the 408 and 510 area codes.

**Figure 8–2  Roles Defined in an Order Specification**

![Roles Defined in an Order Specification](image)

When you model tasks, you can specify which roles can perform which task execution modes (Do, Redo, and Undo). See "About Task Execution Modes" for more information.

Figure 8–3 shows roles used in a task specification.

**Figure 8–3  Task Permissions**

![Task Permissions](image)

When modeling order components and order item specifications, you can define permissions to allow specific roles access to order component search queries and order item search queries in the Order Management Web client. You can also define the data set that their queries return.

Figure 8–4 shows roles used in an order component. In this example, members of ProvisionRole can perform queries based on ProvisioningFunctionTask and display the data in both the summary and detail views in the Order Management Web client. The method for applying roles in an order item specification is identical.
Figure 8–4  Roles Used in an Order Component Specification
This chapter describes the Oracle Communications Order and Service Management (OSM) order transformation manager.

Understanding the Order Transformation Manager

The order transformation manager provides users with the ability to transform order items. For example, you can use the order transformation manager to transform customer-focused order items (what the customer bought) to service-focused order items (the services that equate to what the customer bought). It enables you to set up guidelines for order transformation that do not need to be changed due to product changes. Instead of writing a lengthy XQuery, users can model the order transformation in Oracle Communications Design Studio. The order transformation manager also provides visibility in the Order Management Web client into service processing, making it easier to see how customer services are being transformed into the services being processed by OSM. In addition, the order transformation manager enables you to propagate data upstream and assists in status consolidation.

Order Transformation Manager in Runtime

In runtime, when the order transformation manager is triggered, OSM initiates the following process for each domain that has order items associated with it:

1. The appropriate transformation sequence is accessed to determine the appropriate transformation stages.

2. The transformation stages are executed in order. For each transformation stage:
   a. The stage condition is evaluated to determine whether the stage should be run. If not, OSM moves to the next stage.
   b. The list of source order items is gathered: both context order items (the order items to be transformed) and related order items (order items that might contribute data to the transformed order items).
   c. The list of mapping rules that apply to the named relationships for the transformation stage is gathered.
   d. The mapping rules are processed, creating transformed order items and mapping parameters to them.

3. The transformed order items are processed in the same way as original order items, for example being processed by order components.
The Order Transformation Manager and the Conceptual Model

The order transformation manager requires some of the configuration of the conceptual model. Conceptual model items are not built into OSM cartridges or deployed to the OSM server directly. They are included into OSM by something called realization. Realization refers to converting the abstract entities in the conceptual model into actual instances in the OSM configuration. Conceptual model entities represent abstractions of services, so you cannot deploy conceptual model entities to OSM. Rather, you convert conceptual model entities into OSM model entities. This conversion process is called realization. For more information about conceptual model entities, see Design Studio Concepts.

Entities are realized into the OSM cartridges by different means. Following is a description of how the different entities are realized into OSM or referenced by OSM.

- Provider Functions: Provider functions in the conceptual model are realized into OSM as transformation managers.
- Named Relationships: These entities are realized into OSM when they are referenced by OSM entities, such as mapping rules.
- Domains: Domains are referenced in OSM by transformation managers and mapping rules.
- Products and Customer-Facing Services: These entities are realized into OSM when they are included in relationships that are used by the order transformation manager and when their parameters are mapped to OSM order items using order item parameter bindings.

Figure 9–1 depicts general relationships between conceptual model entities and OSM entities that are used by the order transformation manager.
OSM Entities Used in the Order Transformation Manager

The order transformation manager uses several entities in Design Studio for OSM.

- Transformation manager: The transformation manager entity enables you to select the transformation sequences for the service domains within a provider function. This entity is the entry into the order transformation functionality.

- Transformation sequence: The transformation sequence enables you to define the transformation stages and the logic to be used at each transformation stage. Transformation stages define the source and target order items and the relationship between them for each step of the transformation.

- Order item specifications: You must define an original (source) order item specification that defines the structure of the incoming order items and a transformed (target) order item specification that defines the structure of the output of the order transformation for the order transformation manager. If the same structure is used for both, the same order item specification can be defined for both original and transformed order items. See "About Order Items" for more information about configuring and using order items.
Mapping rules: Mapping rules define the way that original order items are transformed into transformed order items. You use mapping rules to define how transformed order items are generated and how their parameters and properties are populated. The data elements you can use as a source for the mappings are the parameters on the original order item in addition to the parameters on the actions defined for the order item. There are many different ways to generate the parameters and properties for the transformed order items. These methods include:

- You can map parameters from the source order item to the target order item. You can copy the value from the source to the target, transform the value of the source parameter or property to a value on the target based on pre-defined value mappings or on the units of measure for each, and you can write XQuery expressions to do the value transformation.

- You can map order item instances from the source order item to parameters or properties on the target order item. You can either set up a specific value to use on the transformed order item based on the presence of the source order item, or you can use XQuery to determine the value for the parameter or property on the transformed order item.

For more information about mapping rule types, see the Modeling OSM Orchestration Help.

Mapping rules also enable you to map actions for the transformed order item either using the actions defined in the named relationship or defining the actions specifically for the mapping rule, based on the input, output, and current actions of the order items.

Order Item Parameter Bindings: The order item parameter bindings enable you to bind the parameters from a conceptual model entity to parameters on an order item. They also enable you to determine the mapping between the parameters on the conceptual model entity and the properties on an order item. In addition, they enable you to transform the parameters from the customer order line before they are added to the conceptual model entity. One use for this would be to transform name/value-pair-type parameters from the incoming order into more strongly typed parameters on the conceptual model entity.

Transformation Tasks: If you want to call the order transformation manager from a process instead of before the orchestration plan is generated, you do this using a transformation task. See "Calling the Order Transformation Manager" for more information. The transformation task is very much like an automated task, except that by default it has an appropriate automation plug-in defined for it and provides the ability to define the transformation manager to call.

**Calling the Order Transformation Manager**

There are two methods for calling the order transformation manager:

- If you want the order transformation manager to run before the orchestration plan is generated, select **Invoke Order Transformation Manager** in the Orchestration Process and select a provider function. This is the recommended practice, as it causes the order transformation manager to be run in the context of the whole order and with one call.

- If you want to call the order transformation manager at a different place in the order process, you can include a transformation task in an OSM process. The transformation task calls a specific transformation manager that you define in the task. This option provides flexibility in the following ways:
- It enables you to call the order transformation manager multiple times in the process flow for different provider functions. You should not call the order transformation manager more than once for the same provider function.

- It provides the option not to persist the results of the transformation to the order template. This is useful if the order transformation manager results are transient or going to be passed through directly to a southbound system. Additionally, this gives the user the flexibility to format any results that are going to be persisted in whichever structure they want.

- It provides the ability to filter the order items passed into the order transformation manager. This enables a user to ensure that the order transformation manager only processes relevant order items.

The order transformation manager works the same regardless of the way it is called.

**Using the Distributed Order Template with the Order Transformation Manager**

When you are using the order transformation manager, you must use the distributed order template for the order item specification that contains transformed order items. For the order item specification that contains original order items, using the distributed order template is optional. See "About Using the Distributed Order Template" for general information about the distributed order template.

The distributed order template uses namespaces to determine the data structure that should be used. For transformed order items, the namespace depends on the source of the data for the transformed order item. Data that is defined in the order item specification itself will use the namespace for the order item specification, the same way that data would be referenced for an input order item. Following is an example of an XQuery reference to the `lineItemID` property on the `OutputOrderItem` order item with the namespace `http://ex_output.com`:

```
/ControlData/OrderItem[@type='{http://ex_output.com}OutputOrderItem']/lineItemID
```

Data that has been derived from a common model entity, for example an action, will use a different format. In the following situation:

- Order item namespace: `http://ex_output.com`
- Order item name: `OutputOrderItem`
- Name of the parameter assigned as the Dynamic Parameter Property in the order item specification: `dynamicParams`
- Conceptual model cartridge name: `Model_Broadband`
- Conceptual model cartridge version: `1.0.0.0`
- Conceptual model entity (in this case an Action) name: `SA_Add_Internet`
- Parameter name on `SA_Add_Internet`: `serviceLevel`

The reference would look like this:

```
/ControlData/OrderItem[@type='{http://ex_output.com}OutputOrderItem']/dynamicParams[@type='{Model_Broadband/1.0.0.0}SA_Add_InternetType']/serviceLevel
```

The parameters from the conceptual model entity are contained in the `dynamicParams` element on the transformed order item. The type for the parameters contained in the conceptual model entity has the string “Type” appended to the name of the entity. Thus, the type contains `SA_Add_InternetType` rather than just `SA_Add_Internet`. 
About the Calculate Service Order Provider Function

Calculate Service Order is a specific provider function that is delivered via design patterns in Design Studio. The Calculate Service Order provider function is the functional module that transforms customer orders into service orders.

Using Calculate Service Order has two parts. First, you must run the relevant design patterns to set up the framework, and then you must configure the other required entities that are specific to your implementation.

Calculate Service Order Design Patterns

Calculate Service Order includes two design patterns:

- The Design Studio core software contains a design pattern (Common Model Base Data) that sets up the base data for the conceptual model. The following entities that are created in the conceptual model support Calculate Service Order:
  - A Design Studio project to contain the conceptual model entities (optional, an existing project can be used)
  - The Calculate Service Order provider function (see "About the Calculate Service Order Provider Function")
  - The Primary and Auxiliary relationship types (see "About Calculate Service Order Relationship Types")

For more general information about these entities, see the information about designing solutions in Design Studio Concepts.

- Design Studio for OSM contains a design pattern (Calculate Service Order) that contains OSM entities to support Calculate Service Order:
  - A Design Studio project to contain the OSM entities (optional, an existing project can be used)
  - The Calculate Service Order transformation sequence (see "About the Calculate Service Order Transformation Sequence")

About the Calculate Service Order Provider Function

The Calculate Service Order provider function is a logical entity that groups all the metadata required to perform the transformation. It also provides the ability to determine what types of entities and relationships can be used in the transformation and the method used to realize the provider function into OSM.

The Calculate Service Order provider function defines the following associations:

- The input (Product) and output (Customer Facing Service and Resource) conceptual model entities
- The relationship types (Primary and Auxiliary)

About Calculate Service Order Relationship Types

Calculate Service Order also contains the definitions of the following relationship types:

- Primary: In this relationship type, transformed order items are created from original order items. Action codes are normally transferred to the target without being changed, or you can define rules to change the action types.
Auxiliary: In this relationship type, transformed order items are enriched, but no new transformed order items are created. Action codes are translated based on the action type of the source item combined with the current action type of the target item. If the target action type is None, the source action type will be transferred to the target without being changed. If the source and target action types are both defined to something other than None, the action code of the target is changed to Modify. Otherwise, the target action code is unchanged.

These action types are the default for the relationship type. In a mapping rule, you can either use the default from the relationship type or you can define specific rules for a named relationship to be used for the mapping rule.

About the Calculate Service Order Transformation Sequence

The transformation sequence (CalculateServiceOrder) that is created by the OSM design pattern for Calculate Service Order contains the following transformation stages. These stages process order items based on an order item hierarchy. See “About Order Item Hierarchy” for more information about the way order items can be arranged in hierarchies. You can edit these stages using Design Studio, if you need the transformation to work differently.

1. ProcessPrimaryRelationships: This stage creates transformed order items from original order items. Parameters from the original order item are also mapped to parameters on the transformed order item.

2. ProcessDescendantItems: This stage looks at child order items of the original order items and uses them to provide auxiliary data on the transformed order items. This can happen in two ways: the child order item itself may map to a data element on the transformed order item, or parameters from the child order item may map to parameters on the transformed order item. The child order items considered in this stage are not only the immediate children of the original order item, but also their children, to the bottom of the order item hierarchy.

3. ProcessSiblingItems: This stage is similar to the ProcessDescendantItems stage, except that the order items that are contributing data to the transformed order item are the siblings, rather than the descendants, of the original order item. As in the ProcessDescendantItems stage, the order items can provide auxiliary data by the sibling order item mapping to a data element on the transformed order item, or by parameters from the sibling order item mapping to parameters on the transformed order item.

4. ProcessAncestorItems: This stage is also similar to the ProcessDescendantItems stage. In this stage, the order items considered are the parent order items instead of the children. As in the ProcessDescendantItems stage, the order items can provide auxiliary data by the parent order item mapping to a data element on the transformed order item, or by parameters from the parent order item mapping to parameters on the transformed order item. The parent order items considered in this stage are not only the immediate parents of the original order item, but also their parents, to the top of the order item hierarchy.

User-Created Entities for Calculate Service Order

In addition to the entities created by the design patterns, you must also create entities with information specific to your implementation. Some of these entities are in the conceptual model, and some are in OSM.

In the conceptual model, you will need to model at least some of the following:

- Products
About Configuring the Order Transformation Without Calculate Service Order

If the supplied Calculate Service Order order transformation does not transform the order items the way you need, to such an extent that you do not think that editing the supplied entities would work for your situation, you have the option of configuring the order transformation manager from scratch instead.

To configure the order transformation manager if you are not using Calculate Service Order:

1. Model conceptual model entities. For more information about modeling these entities, see the Design Studio Platform Help.
   a. Create a provider function.
   b. Create relationship types.
   c. Create one or more functional areas.
   d. Create a domain in the conceptual model.
   e. Model customer-facing services in the conceptual model.
   f. Model products in the conceptual model.
   g. Model named relationships in the conceptual model.
   h. Add the products to the domain in the conceptual model.
   i. Model a provider function in the conceptual model.
   j. Model data in the conceptual model, including keys for conceptual model entities.

2. Model the order item specifications for the original and transformed order items:
   a. Model the order item recognition. This is usually a parameter on the customer order line item, such as Fulfillment Item Code.
   b. Model order item properties, including a property for order item recognition, a property to contain dynamic parameters created by the order item parameter binding, and properties for the order item ID and action.
3. Model order item parameter bindings to create typed and named parameters from parameters that may have been in name/value pairs in the incoming customer order line item.

4. Model mapping rules. These rules create order items and order item parameters on transformed order items based on original order items (that is, the order items and parameters from the customer order). The following types of mappings are available:
   - Entity-to-entity mapping: This creates a new transformed order item from an original order item. For example, you can use this to create a transformed order item representing a line from an original order item representing a major service.
   - Attribute-to-attribute mapping: This type of mapping creates new parameters on the transformed order item based on parameters on the original order item.
   - Entity-to-attribute mapping: This type of mapping creates new parameters on the transformed order item based on the presence of particular original order items. For example, an original order item representing a feature might be mapped to a parameter for that feature on an order item representing a new line.

5. Model a transformation sequence. This involves modeling a series of transformation stages. Each transformation stage includes the following steps:
   a. Identify context order items for the transformation stage. These nodes are the original order items that will be available for transformation. You can select these nodes either by selecting an order item property that the original order items will have in common or by defining an XQuery expression to select them.
   b. Identify related order items for the transformation stage. These order items will be able to contribute data to the transformed order items. You can select these nodes either by their relation to the context order items (parent, sibling, child) or using an XQuery expression. The relationships between the order items will be based on the physical order item hierarchy defined in the order item specification.
   c. Select the relationship and relationship type that will be available to the transformation stage. For example, the transformation stage may be set up to include a Primary relationship between the Broadband product and the BroadbandInternetAccess customer-facing service.
   d. Determine whether the stage should be conditional, and if so, write a condition for it.

6. Create a transformation manager that links the service domains and transformation sequences that you have created.
This chapter describes how to use behaviors to manipulate data used by orders and tasks in Oracle Communications Order and Service Management (OSM).

About Behaviors

You can use behaviors to specify how OSM manages data. For example:

- You can specify the maximum allowed number of characters for text string data.
- You can add the values of multiple fields and display the sum in another field.
- You can specify the minimum and maximum times that a data element can be used in an order. For example, an order might require that exactly two IP addresses are added.

In Design Studio, you can create behaviors for data elements at three levels:

- Data element level
- Order level
- Task level

To set behaviors at the data element level, open a data schema and click the data element to which you would like to add a behavior. Click the OSM tab to see the area where behaviors can be added.

Figure 10–1 shows a behavior modeled in an order in Oracle Communications Design Studio. This behavior is used by an order to display a tool tip for the payment information field.
Figure 10–1 Information Behavior Modeled in Design Studio

Figure 10–2 shows how behaviors are modeled in a task that enters payment information. In this figure, the field that shows the payment total uses two behaviors:

- A Calculation behavior that adds values in multiple other fields to create the total payment value.
- A Read Only behavior that makes the field read-only in the Task Web client.

**Note:** The examples are for illustrative purposes only; OSM is not typically used for payment handling.
Table 10–1 lists the OSM behaviors.

Table 10–1  OSM Behaviors

<table>
<thead>
<tr>
<th>Behavior Name</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calculation</td>
<td>Computes the value of a field value based on a formula that references order data.</td>
</tr>
<tr>
<td></td>
<td>See &quot;Using the Calculation Behavior&quot; for more information.</td>
</tr>
<tr>
<td>Constraint</td>
<td>Specifies a condition that must be met for the data to be considered valid.</td>
</tr>
<tr>
<td></td>
<td>See &quot;Using the Constraint Behavior to Validate Data&quot; for more information.</td>
</tr>
<tr>
<td>Data Instance</td>
<td>Declares an instance that can be used by other behaviors.</td>
</tr>
<tr>
<td></td>
<td>See &quot;Using the Data Instance Behavior to Retrieve and Store Data&quot; for more information.</td>
</tr>
<tr>
<td>Event</td>
<td>Specifies an action that is performed when data is modified.</td>
</tr>
<tr>
<td></td>
<td>See &quot;Using the Event Behavior to Re-evaluate Data&quot; for more information.</td>
</tr>
<tr>
<td>Information</td>
<td>Specifies the label, hint, and help information for the data element instance.</td>
</tr>
<tr>
<td></td>
<td>See &quot;Using the Information Behavior to Display Data and Online Help&quot; for more information.</td>
</tr>
<tr>
<td>Lookup</td>
<td>Specifies a set of dynamically generated choices from which you can select.</td>
</tr>
<tr>
<td></td>
<td>See &quot;Using the Lookup Behavior to Display Data Selection Lists&quot; for more information.</td>
</tr>
</tbody>
</table>
About Behavior Evaluation

It is possible that multiple behaviors can be applied to the same data. At run-time, OSM determines which behavior should be applied by evaluating the conditions defined for behaviors using a combination of server rules and behavior attributes that you model by using Design Studio configuration options. The following configuration options affect the manner in which OSM evaluates behaviors at run-time:

- The level at which you define the behavior. See "Evaluating Behavior Levels" for more information.

- The manner in which you define the Design Studio Override and Final configuration options. See "Evaluating Design Studio Final and Override Options" for more information.

- The type of behavior defined for the element. See "Evaluating Behavior Type Precedence and Sequence" for more information.

- Whether multiple behaviors of the same type are defined for an element at the same level.

Table 10–1 (Cont.) OSM Behaviors

<table>
<thead>
<tr>
<th>Behavior Name</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>Specifies whether a value can be modified or not. See &quot;Using the Read-Only Behavior&quot; for more information.</td>
</tr>
<tr>
<td>Relevant</td>
<td>Specifies whether data is visible or hidden. See &quot;Using the Relevant Behavior to Specify if Data Should Be Displayed in the Web Client&quot; for more information.</td>
</tr>
<tr>
<td>Style</td>
<td>Specifies the visual appearance of fields. See &quot;Using the Style Behavior to Specify How to Display Data in the Task Web Client&quot; for more information.</td>
</tr>
</tbody>
</table>

Note: The style behavior is the only behavior applied to Redo, Undo, and Do Nothing compensation strategies and the historical order perspective displayed in the Task Web client. See "Managing Changes to Orders" for more information about compensation strategies and the historical order perspective.

About Behavior Evaluation

Evaluating Behavior Levels

In Design Studio, you can create behaviors for data nodes at three levels:

- Data element level (most general)
- Order level (more specific)
- Task level (most specific)

OSM evaluates behaviors from the general level to the specific level. For example, OSM evaluates behavior conditions defined at the data element level first, and evaluates behaviors defined for data nodes at the task level last. At run-time, OSM determines which level to use for a behavior type and data node combination and evaluates rules from that level only.

For example, consider that you create a Calculation behavior at the data element level, and for the same data node you create a Calculation behavior at the order level. In this
scenario, OSM would never evaluate the conditions defined for the Calculate behavior at the order level (unless you force evaluation using the Override or Final options), even if all of the conditions defined for the behavior at the data element level evaluate to false.

OSM does, however, evaluate different types of behaviors defined for a data node at different levels. For example, if for the same data node you define a Calculation behavior at the data element level and a Constraint behavior at the order level, OSM evaluates the conditions for both behaviors at run-time.

---

**Note:** The Constraint behavior is an exception to the way in which behaviors are evaluated. When the run-time environment evaluates Constraint behaviors, it evaluates all of them, regardless of the level at which they are declared.

---

**Evaluating Design Studio Final and Override Options**

You can force local, specific exceptions to the way behaviors are evaluated for a given node by selecting the **Override** and **Final** check boxes on the appropriate Behaviors Properties view Details tab in Design Studio. You can select the **Override** attribute to allow the behavior to take precedence over any other behavior:

- Of the same type
- For the same node
- Declared at the same or more general level

For example, consider that you have a data element called `customer` that you declare twice: at the data element level and at the task level. For each occurrence of `customer`, you create a behavior called `styleBehaviorType`. At the specific task level, you select the behavior’s **Override** check box. At run-time, OSM evaluates the behavior conditions defined at the task level, as the task-level version of `styleBehaviorType` overrides the data element-level version of the same behavior type.

---

**Note:** Override does not function if the behavior that you are trying to override has the **Final** check box selected.

---

When selected, the **Final** check box prevents another behavior of the same type, for the same node, declared at the same or more specific level, from overriding that behavior.

For example, you define the element `customer` at the data dictionary level (highest), and add it at the task level (lowest). For each occurrence of `customer`, you define a Style behavior. On the data dictionary level (most general) of the behavior definition, you select the **Final** check box. On the task level (lowest) of the behavior definition, you select the **Override** check box. When OSM evaluates the behaviors, the selection of the **Final** check box at the data dictionary level prevents the task level (lowest) definition of the Style behavior from overriding the data dictionary level (highest) definition of the behavior.

**Evaluating Behavior Type Precedence and Sequence**

OSM automatically evaluates behaviors whenever you retrieve, save, or transition an order. OSM evaluates the behaviors in a specific nested sequence, as outlined below:

1. The system evaluates all behaviors for a given node before moving to the next node in the order.
The next node in the order is based on a depth first, left-to-right traversal. Figure 10–3 shows the element selection order.

Figure 10–3  Element Selection

2. Behaviors within a given node are evaluated based on the following precedence of type:
   - 1st: Calculate
   - 2nd: Style
   - 3rd: Information
   - 4th: Relevant
   - 5th: Lookup
   - 6th: Constraint
   - 7th: Read-only
   - 8th: Event
3. Within an order, within an element, within a specific behavior type, all behaviors defined at a specific data level are evaluated before moving to the next data level. The evaluation process prioritizes data levels, which are evaluated in the following order:

- Data dictionary level
- Order level
- Task level

Behaviors defined on a task can override behaviors defined on an order if you have enabled the behavior’s Override option at the task level and if you have disabled the behavior’s Final option at the order level.

**Note:** The Constraint behavior is an exception to the way behaviors are evaluated: When OSM evaluates Constraint behaviors, it evaluates all of them, regardless of the level at which they are defined.

---

**Evaluation Process**

Within an order, within an element, within a behavior type, within a data level, the evaluation proceeds as follows:

1. **Is the behavior enabled?**
   - If the behavior is enabled, the final and override options are evaluated simultaneously.
   - If the behavior is not enabled, the behavior is not applied.

2. **Is the behavior finalized or overridden?**
   - If the behavior is not finalized and not overridden at a lower level, the condition defined for the behavior is evaluated.
   - If the behavior is finalized and not overridden at a lower level, the behavior is final and the condition defined for the behavior is evaluated.
   - If the behavior is finalized and overridden at a lower level, the override has no affect; the behavior is final and the condition defined for the behavior is evaluated.
   - If the behavior is not finalized and is overridden at a lower level, the condition defined for the overridden behavior is evaluated (not the condition defined for the behavior that is currently being evaluated). If the condition is met, the overridden behavior is applied.
   - If the behavior is not finalized and is overridden by more than one lower level, the condition defined for the lowest level overridden behavior is evaluated (not the condition defined for the behavior that is currently being evaluated). If the condition is met, the overridden behavior is applied.

---

**Note:** Relevant rules can prevent other rules from being evaluated. For example, if the Relevant rule of a data node evaluates to false, then rule types with a precedence lower than the Relevant rule are not evaluated (the Lookup, Constraint, Read-only, and Event rules). Additionally, if a data node’s Relevant rule evaluates to false, no rule evaluation is done for any descendents of that node.
3. Is the condition defined for the behavior met?
   ■ If the condition is met, the behavior is applied.
   ■ If the condition is not met, the behavior is not applied.

   **Note:** If you define two or more behaviors for an element at the same level, to avoid unpredictable behavior you should define mutually exclusive conditions. OSM does not guarantee the order of evaluation for the same behavior types defined at the same level.

4. The evaluation process continues.
   ■ If a condition is met, and a behavior is applied, the evaluation process no longer checks lower levels; it moves to the next occurrence of the behavior.
   ■ If a condition is not met, the evaluation process continues with the next occurrence of this behavior type defined at this data level. If there are no more at this level, the evaluation process moves to the next lower level. If there are no lower levels, the evaluation process continues with the next occurrence of this behavior type defined at the highest level, and so on. When there are no more occurrences of the behavior type, the evaluation process moves to the next behavior type, and so on. When there are no more behavior types, the evaluation process moves to the next element.

When the evaluation process determines that a behavior is to be applied at a particular level, some behavior types stop evaluating behaviors of the same type, while others continue evaluating behaviors of the same type at that level for the same element.

For example, you define three behaviors of the same type on the same data element at the same level, and all go through the evaluation process ending with the condition being met (the behavior is applied). For behaviors that stop evaluating, only the first behavior is applied. For behaviors that continue evaluating, multiple behaviors of the same type may be applied, and their effect on the UI is cumulative.

The following behaviors stop evaluating behaviors of the same type after a condition is met and a behavior of the type is applied:
   ■ Calculation
   ■ Lookup

The following behaviors continue evaluating behaviors of the same type after a condition is met and a behavior of the type is applied:
   ■ Constraint
   ■ Event
   ■ Information
   ■ Read Only
   ■ Relevant
   ■ Style

   **Note:** The behaviors in both lists above are presented in alphabetical order, not in behavior type evaluation order.
For example, if three Constraint behaviors are defined, and all go through the evaluation process ending with the behavior being applied, all three Constraint violation messages display in OSM. In another example, if three Read Only behaviors are defined, if any of them get applied, the field is set to read-only (even if prior and/or subsequent Read Only behaviors evaluate to false). Style and Information behaviors are a bit more complicated in that they have multiple facets. The end result is the cumulative effect of these facets. For example, you can define hints and labels with an Information behavior. If one behavior has a hint and another behavior has a label, the end result is that both are applied. If two behaviors define hints, then the second behavior’s hint is applied.

**Evaluating Multiple Behaviors of Similar Type and Level**

When modeling behaviors of the same type, at the same level, for the same data node, ensure that the conditions you define for each behavior are mutually exclusive. When evaluating behaviors of the same type and defined on the same data node and level, the OSM run-time server has no ability to guarantee a predictable order of evaluation. When modeling behaviors for a data node, when it’s necessary to define behaviors of the same type at the same level, ensure that you configure conditions that do not rely on a specific order of evaluation.

Additionally, the OSM server evaluates the conditions of each behavior until the conditions of one behavior evaluate to true. Subsequently, OSM does not continue to evaluate any conditions defined for behaviors of the same type and for the same data node.

**About Setting Conditions in Behaviors**

Conditions enable you to specify when a behavior should function. You set a condition by defining an XPath expression. If the XPath expression evaluates to false at run time, the condition is not met and the behavior is not applied. If the XPath expression evaluates to true at run time, the condition is met and the behavior may or may not be applied, depending on the outcome of evaluation of the behavior at run time.

If no conditions are defined, OSM considers the condition to be met. If multiple conditions are defined, all conditions must evaluate to true for the condition to be met.

---

**Note:** The Constraint behavior is the only exception to the way conditions are handled.

Constraint behaviors specify a condition that must be met for the data to be considered valid.

---

**XPath Examples**

This section provides XPath examples that are applicable to setting a condition on any behavior type.

- This example shows a condition that evaluates to true when the value of myNumericField is less than 100, and evaluates to false when the value of myNumericField is 100 or greater:

  ```xml
  ../myNumericField<100
  ```

- This example shows a condition that evaluates to true when the value of myTextField is populated, and evaluates to false when the value of myTextField is an empty String:
This example shows a condition that evaluates to true when the value of all three fields are zero, and evaluates to false if any one of three fields are greater than or less than 0:

```
../myNumericField1=0 and myNumericField2=0 and myNumericField3=0
```

Using the Calculation Behavior

You use the Calculation behavior to calculate a field’s value based on a formula that references other field values. You can perform numeric operations and string concatenations.

OSM supports the Calculation behavior in the Task Web client and in the Order Management Web client Data tab.

For example, you can use the following expression in a Calculation behavior to calculate the current balance for a customer:

```
../amount_owing - sum(../../payment_information/payment_amount)
```

In this example, the current balance displays the value from the `amount_owing` field after subtracting the value from the `payment_amount` field; the balance = (amount owed) - (amount paid).

Figure 10–4 shows an XPath expression that combines the `first_name` field and the `last_name` field. The Calculation behavior is applied to a field that contains the card-holder name field, where the first and last names are combined into a single field value.

**Figure 10–4  Calculate Behavior Formula for Combining String Values**

```
concat(../..../account_information/first_name, ' ', ../..../account_information/last_name)
```

Figure 10–5 Shows how the fields appear in the Task Web client.
Calculation Behavior XPath Examples

The following examples show how to use XPath statements in the Calculation behavior.

- This example shows how to set a constant value of 100 for a numeric field (whatever number you specify is the number that displays for the field):
  
  100

- This example shows how to prefix a constant value to a text field (whatever text you define is the text that displays along with the text value of the field):
  
  append('any text here',../fieldName)

- This example shows how to display a numeric field as a result of adding three other numeric fields:
  
  ../fieldName1 + ../fieldName2 + ../fieldName3

- This example shows how to see the user name of the user who accepted a task:
  
  /GetOrder.Response/AcceptedUserName

Calculation Behavior Overview

Table 10–2 shows Calculation behavior attributes.

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order of evaluation</td>
<td>1st</td>
</tr>
<tr>
<td>Default value</td>
<td>None</td>
</tr>
<tr>
<td>Applies to</td>
<td>All elements</td>
</tr>
</tbody>
</table>
Using the Constraint Behavior to Validate Data

You can use the Constraint behavior to validate data that is entered in an order. For example:

- Validate the format of a field. For example, 10 digits for a telephone number, 5 digits for a ZIP code, or an IP address format.
- Validate the range of a field. For example, ensure that a numeric value is between 0 and 100.
- Validate the field value is within a valid list.

In addition to specifying how data is validated, you can:

- Configure messages that indicate the results of the validation; for example, a warning or error message.
- Specify how the order should be processed if the validation fails; for example, stop processing or continue processing.

For example, you might want to ensure that value in a Payment Amount field is less than the amount owed and greater than 0. The Constraint behavior would include this condition:

```
. <= ../../account_information/amount_owing and . >= 0
```

The same Constraint behavior would include the following message to display if the behavior was not met:

```
concat('Invalid payment amount['',.,']'. Payment must be greater than 0 and less than amount owing of ['',../../account_information/amount_owing,']')
```

OSM supports constraint rules in the Task Web client.

Displaying Constraint Behavior Error Messages

OSM only displays a Constraint behavior error message if there is a constraint violation caused by the failure of a Constraint behavior condition or by an exception that occurred during the behavior evaluation while you are attempting to either:

- Save an order with invalid field content
- Transition an order with invalid or null values

Otherwise, OSM cues you that a field requires some value by placing a red dot to the left of the field label.
Evaluating Constraint Behaviors

OSM always evaluates Constraint behaviors except when the element or parent element is not relevant, as defined through the Relevant behavior. OSM does not evaluate the Constraint behavior when the task to which the Constraint behavior is associated is at the rollback status. In cases when data is rolled back, it is understood that the Constraint behavior was already evaluated.

Constraint behavior evaluation is different from that of other behaviors. Constraint behaviors are evaluated only when one or more specified conditions evaluate to false. All other behaviors are either:

- Always evaluated
- Evaluated only when one or more specified conditions evaluate to true.

In addition, when OSM does evaluate Constraint behaviors, it always evaluates all of the Constraint behaviors, regardless of where they are defined. This is different from other types of behaviors, where only the first instance of each behavior is selected and applied. However, the Override and Final check boxes give you control over inheritance. See “Evaluating Design Studio Final and Override Options” for more information.

Constraint Behavior Overview

Table 10–3 shows Constraint behavior attributes.

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order of evaluation</td>
<td>6th</td>
</tr>
<tr>
<td>Default value</td>
<td>True</td>
</tr>
<tr>
<td>Applies to</td>
<td>All elements and structures</td>
</tr>
<tr>
<td>Parent/child inheritance</td>
<td>Does not inherit</td>
</tr>
<tr>
<td></td>
<td>(This applies to element relationships within a structure, which is different than the inheritance of behaviors between the data dictionary, order, and task levels.)</td>
</tr>
</tbody>
</table>

Using the Data Instance Behavior to Retrieve and Store Data

You can use the Data Instance behavior to get data from external sources. For example, an order processor using the Task Web client can retrieve a set of available ports in real time from an ADSL inventory system.

This behavior differs from all other behaviors in that it has no affect on the UI display of the element for which the behavior is defined. You can think of the Data Instance
behavior as a “supporting” behavior because it provides functionality that can be used with other behaviors.

You can use the Data Instance behavior to:

- Store data from an external system and make it accessible to other behaviors.
- Store data that is defined in-line in an XML or XQuery and make it accessible to other behaviors.
- Store data from OSM that is housed in multiple fields but commonly referenced collectively as a single field and make it accessible to other behaviors. For example, the fields `first_name` and `last_name` can be combined in a new data instance `customer_name`.

When you use the Data Instance behavior, you need to specify the data provider that you get data from. OSM supports several data providers; for example, Oracle Communications Unified Inventory Management (UIM), XML files, and data in the incoming customer order. You can also configure your own data provider.

See "About Mapping Order Items to Fulfillment Patterns" for an example of how to use a Data Instance behavior.

Evaluating Data Instance Behaviors

When a Data Instance behavior is defined for an element, regardless of the data level, the container is available to the element on all data levels. Because of this:

- The Override and Final check boxes have no effect on the Data Instance behavior.
- The Data Instance behavior is not part of the evaluation process in terms of prioritization of behavior type, or in terms of prioritization of data level.

Data Instance Behavior XML, XPath, and XQuery Examples

This section provides XML, XPath, and XQuery examples that are applicable to defining a Data Instance behavior.

- This example shows an in-line XML that defines constant values (this could be used to define the values that appear in a dropdown field):

  ```xml
  <bookStore>
    <books>
      <titles>
        <AlgebraForDummies> <price>30</price> </AlgebraForDummies>
        <GeometryForDummies> <price>35</price> </GeometryForDummies>
        <TrigonometryForDummies> <price>40</price> </TrigonometryForDummies>
      </titles>
    </books>
  </bookStore>
  ```

- This example shows an XPath expression that selects data from an XML file that defines elements (nodes) of bookstore, book, price, and title. This example returns a list of titles with a price greater than $30:

  ```xml
  xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
  xmlDoc.async=false;
  xmlDoc.load("books.xml");
  xmlDoc.selectNodes("/bookstore/book[price>35]/title");
  ```

- This example shows an XQuery expression that selects data from an XML file that defines elements (nodes) of bookstore, book, price, and title. This example returns
a list of ordered titles with a price greater than $30. The list is returned in variable $x$:

```xml
for $x$ in doc("books.xml")/bookstore/book
where $x$/price>30
order by $x$/title
return $x$/title
```

### Data Instance Behavior Overview

Table 10–4 shows Data Instance behavior attributes.

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order of evaluation</td>
<td>Not applicable. The data instance type is unique in that it doesn't perform any action. It's just a container for data provider instances.</td>
</tr>
<tr>
<td>Default value</td>
<td>None</td>
</tr>
<tr>
<td>Applies to</td>
<td>All elements and structures</td>
</tr>
</tbody>
</table>
| Parent/child inheritance    | Children inherit instances declared on parent
                          \(\text{(This applies to element relationships within a structure, which is different than the inheritance of behaviors between the data dictionary, order, and task levels.)}\) |

### Using the Event Behavior to Re-evaluate Data

You can use the Event behavior to save or refresh data when the data changes. This is useful when a change in a field can cause a behavior to automatically occur in the same field or in another field. For example, you might include an Event behavior in the `account_information/country` field, that causes the data to refresh. That refreshed data might in turn be used by a Relevant behavior assigned to the address details fields that display address information based on the country.

**Refreshing** causes OSM to re-evaluate all the behaviors associated with the task but does not save the order. **Saving** re-evaluates the behaviors and automatically saves the contents of the order.

**Figure 10–6** shows an Event behavior defined in Design Studio. In this figure, the Event behavior refreshes the data in the `account_information/country` field.
Figure 10–6  Event Behavior Defined in Design Studio

See "Using the Relevant Behavior to Specify if Data Should Be Displayed in the Web Client" for more information on the Relevant behavior, and this scenario in particular.

OSM supports the Event behavior in the Task Web client.

Event Behavior Overview

Table 10–5 shows Event behavior attributes.

Table 10–5  Event Behavior Attributes

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order of evaluation</td>
<td>8th</td>
</tr>
<tr>
<td>Default value</td>
<td>None</td>
</tr>
<tr>
<td>Applies to</td>
<td>All elements</td>
</tr>
<tr>
<td>Parent/child inheritance</td>
<td>Does not inherit</td>
</tr>
<tr>
<td></td>
<td>(This applies to element relationships within a structure, which is different than the inheritance of behaviors between the data dictionary, order, and task levels.)</td>
</tr>
</tbody>
</table>

Using the Information Behavior to Display Data and Online Help

You can use the Information behavior to specify how data is displayed in the OSM Task Web client. You can do the following:
Set an alternative label for the field. For example, instead of the standard label State, the field can be changed to State or Province when processing the type of order that uses this behavior setting.

- Localize the field label to one or more different languages.
- Set a tooltip on a field.
- Provide an online help topic for the field.

In the Order Management Web client, any information rule on the first instance of a group node that uses a table layout style is used to determine the text of the table panel header. The first instance of each of this group instance’s child field nodes are used to determine the column header text for that field node. Hint text for the group instance row and child field instance cells are displayed as tooltip text. Help defined for the group can be executed with either a menu item in the table’s Actions menu or a row-level context menu and displays help in a modal window in the page containing the table. The implementation of this help behavior differs from the task client implementation, which uses a linked icon in each table cell to load the help in a separate browser window.

OSM triggers information rules when the data element or structure contains data, (for example, from the incoming order or derived from other data sources). If the data element or structure is empty, OSM does not display any label, hint, or help topic information behavior associated with the empty element or structure. For example, if you defined a label for an element, the label does not appear when the element does not contain a value. Instead, the OSM uses the Display Name of the element as defined in the data dictionary.

Information Behavior XPath Examples

This section provides XPath examples that are applicable to defining an Information behavior.

- This example shows an Information behavior label that could be used in conjunction with a Calculation behavior that calculates the current balance based on other fields such as endingBalance + currentCharges + fees - payments:
  "Current Balance"

- This example shows an Information behavior label that displays in place of the existing label assigned to the element. For example, the existing label “State” can be changed to display as:
  "State or Province"

- This example shows an Information behavior hint that displays when you hover over the Current Balance data field:
  "The current balance reflects the customer’s ending balance, plus any current charges and fees, minus any applied payments."

- This example shows an Information behavior hint that displays when you hover over the Billing Address data field:
  "The billing address is the address of the party responsible for payment of account. The billing address may differ from the service address. For example, the service address may be a college student’s address, and the billing address may be the student’s parents address."
Information Behavior Overview

Table 10–6 shows Information behavior attributes.

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order of evaluation</td>
<td>3rd</td>
</tr>
<tr>
<td>Default value</td>
<td>None</td>
</tr>
<tr>
<td>Applies to</td>
<td>All elements and structures</td>
</tr>
<tr>
<td>Parent/child inheritance</td>
<td>Does not inherit&lt;br&gt;(This applies to element relationships within a structure, which is different than the inheritance of behaviors between the data dictionary, order, and task levels.)</td>
</tr>
</tbody>
</table>

Using the Lookup Behavior to Display Data Selection Lists

You can use the Lookup behavior to display data in a GUI field that users can select from. You can specify the order of the labels in the list, such as alphabetically.

You can look up data from the following sources:

- Data that is in the incoming customer order.
- Data from an internal source, such as an XML file.
- Data from an external data provider.

Data can be retrieved dynamically based on input. For example, you can look up and populate a list of phones that cost less than $100, where $100 is a value obtained from another field in the order.

Figure 10–7 shows a Lookup behavior that creates a choice of cities. The upper part of the figure shows the behavior in Design Studio, and the lower part of the figure shows how the data displays in the Task Web client.
Using the Lookup Behavior to Display Data Selection Lists

**Figure 10–7  Example of Defining a Lookup Behavior**

![Data Schema: bb_ocm_demo](image)

**Lookup Behavior XPath Example**

This section provides an XPath examples that is applicable to defining a Lookup behavior.

This example shows an XPath expression that selects data from an XML file that defines elements (nodes) of bookstore, book, price, and title. This example returns a list of titles with a price greater than $35:

```javascript
xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
xmlDoc.async=false;
xmlDoc.load("books.xml");
xmlDoc.selectNodes(/bookstore/book[price>35]/title);
```

**Lookup Behavior Overview**

Table 10–7 shows Constraint behavior attributes.
You can use the Read Only behavior to specify that data displayed in the Task Web client is read only. You can specify that data can be read only based on conditions; for example, data can be read only depending on other data in the order.

You typically create read-only fields for fields where the value is derived from other fields. For example, in your order display, you might have two windows: an account window and a payment window. Both windows might have an Amount Owed field, which displays the same data. However, you could make the Amount Owed field in the payment window the field where the data is collected, and the Amount Owed field in the account window read only. In that case, the field in the account window uses two behaviors:

- A Calculate behavior, to get the data from the payment window.
- A Read Only behavior.

### Using the Read-Only Behavior

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order of evaluation</td>
<td>7th</td>
</tr>
<tr>
<td>Default value</td>
<td>The default specified by the static read-only value.</td>
</tr>
<tr>
<td>Applies to</td>
<td>All elements and structures</td>
</tr>
<tr>
<td>Parent/child inheritance</td>
<td>If any ancestor evaluates to true, this value is treated as true. Otherwise, the local value is used. (This applies to element relationships within a structure, which is different than the inheritance of behaviors between the data dictionary, order, and task levels.)</td>
</tr>
</tbody>
</table>
Using the Relevant Behavior to Specify if Data Should Be Displayed in the Web Client

You can use the Relevant behavior to specify if data should be displayed in the Task Web client or in the Order Management Web Client Data tab, based on specified conditions.

For example, you can use the Relevant behavior to display address-input fields appropriate to the country that the order applies to. In this example, the Relevant behavior can be used as follows:

- The data for the customer’s country is included in the order’s account_information/country field. This data is displayed in the Task Web client in the Country/Region field.
- Based on the data in the account_information/country field, the customer address fields (address_information) can include different values, depending on the country, as shown in Figure 10–8.

**Note:** The account_information/country field includes an Event behavior, which refreshes the data in the field, making it available to the Relevant behavior.

![Figure 10–8 Address Fields Based on Relevant Behavior](image)

**Figure 10–9** shows the address fields for the United States (address_details_us) and Canada (address_details_ca). The Relevant behavior applies to the selected data, address_details_ca.
Using the Relevant Behavior to Specify if Data Should Be Displayed in the Web Client

Figure 10–9  Address Fields in Design Studio

![Address Fields in Design Studio](image)

Figure 10–9 shows the XPath expression that specifies the condition (country = Canada) under which the Relevant behavior is enabled.

Figure 10–10  Relevant Behavior Properties

![Relevant Behavior Properties](image)

In the Order Management Web client, if a group instance displayed with a table style behavior is not relevant, then the entire associated table row is omitted. If a particular field is not relevant, the associated table cell is rendered empty.

Relevant Behavior Overview

Table 10–9 shows Relevant behavior attributes.

Table 10–9  Relevant Behavior Attributes

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order of evaluation</td>
<td>4th</td>
</tr>
<tr>
<td>Default value</td>
<td>True</td>
</tr>
<tr>
<td>Applies to</td>
<td>All elements and structures</td>
</tr>
</tbody>
</table>
Using the Style Behavior to Specify How to Display Data in the Task Web Client

You can use the Style behavior to specify where and how to display data in the Task Web client. You can do the following:

- Control the placement of an element on a specific page.
- Specify to display data on tabbed pages. You can display data in columns and tables.
- Hide or mask sensitive data; for example, passwords or credit-card information. You can specify who can read passwords, and you can display a history of password changes. Masked data appears similar to *******.
- Control the layout of a multi-valued field, such as a list of buttons to choose from.
- Apply cascading style sheets (CSS style sheets) to specify how to display data. For example, you could make the current account balance display in red when the data value is greater than zero.

### Important:
If you define a behavior that contains an apostrophe (') character, OSM will throw an exception error when loading the data. To prevent this from happening, you must include the escape character before and after the apostrophe.

**Example:**

```
'L'Information De Carte de credit'
```

**should be**

```
"L""Information De Carte de credit"
```

Figure 10–11 shows how the Style behavior changes the appearance of the Current Account Balance field in the Task Web client.

**Table 10–9 (Cont.) Relevant Behavior Attributes**

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent/child inheritance</td>
<td>If any ancestor evaluates to false, this value is treated as false. Otherwise, the local value is used. (This applies to element relationships within a structure, which is different than the inheritance of behaviors between the data dictionary, order, and task levels.)</td>
</tr>
</tbody>
</table>
Figure 10–12 shows the condition that determines if the Style behavior should be applied. In this case, the Style behavior is applied if the account balance is the same as the amount owed.

**Figure 10–12  Condition Defined in a Style Behavior**

<table>
<thead>
<tr>
<th>Behavior</th>
<th>Style Behavior - Style</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Appearance</td>
</tr>
<tr>
<td>This behavior will be triggered if all of the conditions below are met.</td>
<td></td>
</tr>
<tr>
<td>Conditions</td>
<td>XPath Expression</td>
</tr>
<tr>
<td>Balance = Owing</td>
<td>. &gt; 0 and not(, &gt; ../amount_owing)</td>
</tr>
<tr>
<td>Remove</td>
<td>Add</td>
</tr>
</tbody>
</table>

Figure 10–13 shows the style definitions to apply to a field. The definitions change the text and background color to shades of orange for the value and the label.

**Figure 10–13  Field Display Colors Defined in a Style Behavior**

<table>
<thead>
<tr>
<th>Behavior</th>
<th>Style Behavior - Style</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Appearance</td>
</tr>
<tr>
<td>Value</td>
<td>CSS Style Attribute</td>
</tr>
<tr>
<td>CSS Style Attribute</td>
<td>color: #FFA500; BACKGROUND-COLOR: #FFFFDE</td>
</tr>
<tr>
<td>CSS Class Name</td>
<td></td>
</tr>
<tr>
<td>Label</td>
<td>CSS Style Attribute</td>
</tr>
<tr>
<td>CSS Style Attribute</td>
<td>color: #FFA500; BACKGROUND-COLOR: #FFFFDE</td>
</tr>
<tr>
<td>CSS Class Name</td>
<td></td>
</tr>
</tbody>
</table>

Figure 10–14 shows how three different conditions can change how the field is displayed. If the balance is zero, the field is green. If the balance is the same as the amount that the customer owes, the field is orange. If the balance is less than zero, or greater than the customer owes, the field is red.
About Style Behavior Layouts

This section provides additional information on table layouts, which you can choose to set as **None**, **Page Layout**, or **Table Layout**.

The **Page Layout** option gives you the ability to organize structures elements onto separate pages that you can access directly, through the use of tabs. This is particularly useful for improving access where there are numerous large structures by eliminating the need to scroll through a single page to find the required structure.

The **Table Layout** option displays multi-instance structures in a grid format. By default, **Table Layout** displays all of the child elements in the structures. However, you can prevent a given child element from being used as a column by setting its hidden attribute to true.

Child elements within the structure are represented by columns, and instances of the structure are represented by rows. **Table Layout** displays the columns from left to right in the same order that they appear from top to bottom when displayed without a table layout. If you need to change the order in which the columns appear, you do so by changing their order in the Design Studio order template. The table uses the same child element label to form the column header that it does when displayed without a table layout.

**Note:** If you use an Information behavior to dynamically change the child element labels, **Table Layout** uses the label associated with the first data instance it encounters.

If you need to hide the value of an individual cell in the resulting table, you can do so by declaring a Relevant behavior for the corresponding child element. See "Using the Relevant Behavior to Specify if Data Should Be Displayed in the Web Client".
The following figures illustrate the different types of available layouts for the Task Web client. Each figure shows the same structure with a different layout type:

- **Figure 10–15, “Task with No Layout in the Task Web Client”** shows the structure with no layout applied. With this option, the elements in the structure display within a group box on the original page.

- **Figure 10–16, "Page Layout in the Task Web Client"** shows the structure with the Page Layout option applied. With this option, the elements in the structure display within a group box on a new page that is accessed through a tab on the original page.

- **Figure 10–17, "Table Layout in the Task Web Client"** shows the structure with the Table Layout option applied. With this option, the elements in the structure display within a grid on a new page that is accessed through a tab on the original page.

The following figures illustrate the different types of available layouts for the Task Web client. Each figure shows the same structure with a different layout type:
**Figure 10–16  Page Layout in the Task Web Client**

![Figure 10–16](image1)

**Figure 10–17  Table Layout in the Task Web Client**

![Figure 10–17](image2)

The following figures illustrate the different types of available layouts for the Order Management Web client. Each figure shows the same structure with a different layout type:

- **Figure 10–18, “No Layout in the Order Management Web Client”** shows the structure with no layout applied.
- **Figure 10–19, “Table Layout in the Order Management Web Client”** shows the structure with the **Table Layout** option applied.
Figure 10–18  No Layout in the Order Management Web Client

Figure 10–19  Table Layout in the Order Management Web Client
About Style Behavior Password Fields

Behaviors that define password fields can ensure unauthorized users cannot view the contents of elements containing sensitive information. For example, by using this feature you can define a password field in such a way that users in an activation work group can not see the information, but users in the system administrator’s work group can.

How Password Fields Display

If you define a password behavior on a writable field, OSM displays the contents of the field as specified by the browser, such as a line of asterisks (*) within a text box. If you define this feature on a read only field, OSM displays the data as specified by the browser, such as line of eight asterisks next to the field label, but not within a text box.

If you open the data history, OSM displays when and by whom the data was modified. When this feature is applied to a field, OSM displays the password field values as specified by the browser, such as a line of eight asterisks.

While you can define a Style behavior on all types of elements, this feature of the Style behavior has no effect on structures.

Do Not Use Password Field Feature with Boolean and Lookup Fields

Because this feature is designed for use with free form entry fields, as opposed to fields that force you to select from a limited number of choices, Oracle recommends that you do not use this feature with Boolean and lookup fields. If you do, you risk exposing confidential information to unauthorized users. This is because OSM displays the value that was previously set in a Boolean or lookup field, even if the field defines this feature through a Style behavior.

Displaying the Data History of Password Fields

OSM only evaluates behaviors at the Web UI level, so any password field that you save (that is, create, update, or delete) through the XML API/Automator is not treated as a password field, even if it is defined as such. This can introduce some complexity into how OSM displays the data history for password fields. Use the following general guidelines and examples to understand how OSM displays password field data history.

General Guidelines

1. OSM displays a line of eight asterisks in the data history for any field that it evaluates as a password field (providing the field actually contains data; if the field is empty, OSM displays nothing).

2. If OSM does not evaluate a field as password field, the data history values are shown in plain text.

3. If OSM evaluates a data field as a password field at the time of saving, and the field is later deleted, OSM displays a line of eight asterisks in the data history (providing the field actually contains data; if the field is empty, OSM displays nothing).

4. If OSM evaluates a data field as a non-password field at the time of saving, and the field is later deleted and evaluated as a non-password field at the time of deletion, the data history is displayed as plain text.

Examples

1. If you save the value of a password field through OSM, and OSM is still evaluating the field as a password field when you display the data history, OSM displays the value of the password field as eight asterisks.
2. If you save the value of a password field through the XML API/Automator, and it is still present in the order editor (that is, it has not been deleted by the XML API/Automator) when you display the data history, OSM displays the value as eight asterisks.

3. If you create and delete the password field values through the XML API/Automator, OSM displays the data history values as plain text.

4. If you enter data in a non-password field through OSM and a user subsequently deletes the value through OSM or the XML API/Automator (and OSM evaluates the field as a non-password field at the time of deletion), the history values of this field are displayed as plain text.

**Style Behavior Overview**

Table 10–10 shows Style behavior attributes.

**Table 10–10 Style Behavior Attributes**

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order of evaluation</td>
<td>2nd</td>
</tr>
<tr>
<td>Default value</td>
<td>Data type specific:</td>
</tr>
<tr>
<td></td>
<td>■ For Boolean type fields: Compact</td>
</tr>
<tr>
<td></td>
<td>■ For Lookup type fields: Minimal</td>
</tr>
<tr>
<td>Applies to</td>
<td>Elements of data type:</td>
</tr>
<tr>
<td></td>
<td>■ Boolean</td>
</tr>
<tr>
<td></td>
<td>■ Lookup</td>
</tr>
<tr>
<td></td>
<td>Elements with Lookup behaviors that display only one column.</td>
</tr>
<tr>
<td>Parent/child inheritance</td>
<td>Does not inherit</td>
</tr>
<tr>
<td></td>
<td>(This applies to element relationships within a structure, which is different than the inheritance of behaviors between the data dictionary, order, and task levels.)</td>
</tr>
</tbody>
</table>
About Notifications

This chapter describes Oracle Communications Order and Service Management (OSM) event notifications and jeopardy notifications.

About Notifications

You can use notifications to alert users and external systems to events that occur in the order process or to tell users that an action must be carried out.

There are two types of notifications:

- **Use jeopardy notifications** when you want to alert users that an order might have a problem. To trigger jeopardy notifications, OSM checks order or task conditions at specified intervals. If an action has not occurred as expected, OSM sends a notification. Jeopardy notifications are displayed in the Task Web client Notifications page. See "About Jeopardy Notifications" for more information.

- **Use event notifications** to alert users of changes to the order based on its progress. Event notifications are based on changes that occur to an order. Event notifications based on a change to order data are displayed, but event notifications generated by order milestones and changes to task status are not displayed. See "About Event Notifications" for more information.

Figure 11–1 shows notifications displayed in the Task Web client. You can specify which workgroups can see the notifications.

![Notifications Displayed in the Task Web Client](image)

About Notification Priority

You can specify a priority for most types of notifications. For example:

- Notifications can be prioritized to control how they are sorted in the Task Web client. You should prioritize Jeopardy notifications higher than information messages.
About Notifications

- Prioritizing notifications sent to external systems helps those systems process the more important notifications first.

OSM evaluates notifications with the highest priority first (1 is the highest priority). For notifications that are sent to external systems, the notification priority represents the JMS queue priority.

About Sending Notifications in Email

You can deliver notifications in email. The email message consists of the same information that is displayed in the Notifications window in the Task Web client. You cannot customize the message or add information to it. The message template is:

You have a notification for Order ID {ID number} and notification ID {notification ID}. Use the following URL to connect to the notification details: {url}

For most types of notifications, you specify to send email by selecting a check box in the notification configuration. For event notifications that are used only for running an automation plug-in, you configure the automation plug-in to send the email. See OSM Developer’s Guide for information about automation.

To specify who to send the email to, you do the following:

- When configuring the notification in Oracle Communications Design Studio, or in your automation plug-in, specify the roles that receive the notification.
- Configure the email recipients for the roles by using the OSM Administrator application. (Roles are called workgroups in OSM Administrator.)

About Using Order Rules in Notifications

All jeopardy notifications and most event notifications use order rules to determine if the notification should be triggered. (Event notifications that are used only for running an automation plug-in do not use order rules.)

Figure 11–2 shows an example of a rule defined in Design Studio. This rule finds the city that the customer lives in and the type of account, (Business or Residential). When the jeopardy notification uses this rule, the notification is sent only if the order came from a residential customer in Sao Paulo.
You can use rules such as the one shown in Figure 11–2 to route notifications to specific roles. For example, you can combine rules and roles as follows:

**Table 11–1 Example Rule and Role Combinations**

<table>
<thead>
<tr>
<th>Notification Type</th>
<th>Triggered By</th>
<th>Rule Specifies</th>
<th>Sent to Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Notification_Residential</td>
<td>Expected duration exceeded</td>
<td>Residential account</td>
<td>Residential</td>
</tr>
<tr>
<td>Notification_Business</td>
<td>Expected duration exceeded</td>
<td>Business account</td>
<td>Business</td>
</tr>
</tbody>
</table>

In this example, two identical notifications are created, both triggered by the order processing time exceeding the expected duration. If the order is for a residential account, the notification is triggered and sent to the role that handles residential accounts.

OSM uses a system-based `null_rule`. This rule always evaluates to true. Therefore, if you do not specify a rule for a notification, the null_rule is used; because it is set to true, the notification is triggered. If you do not specify any conditions to trigger the notification, and the notification uses the null_rule, the notification is triggered every time it is polled.

**Note:** The polling interval cannot be changed at run time.

See "About Order Rules" for more information about rules.

**About Using Notifications to Run Automation Plug-Ins**

All types of notifications can run automation plug-ins to send the notification to an external system. For example, you can trigger a notification that runs an automation plug-in that sends a message that a task has completed to the order-source system. An automation plug-in can also perform custom logic, update an order, send email, and send a message to display on the Task Web client Notifications page.
An automated notification triggers a specific automation plug-in when a notification is created in OSM, which can occur several different ways depending on the notification definition, such as a task reaching a specified state or an order reaching a specified milestone. An automated notification receives a message internally from OSM, and the information contained in the message is made available to the automation plug-in. The automation plug-in can then use that information to perform custom logic, update an order, or send a message to the OSM Notifications page, to a user email address, or to an external system. Automated notifications are not capable of receiving external messages back from an external system.

Automations use OSM roles to restrict who can receive the notification. If the notification is sent to an external system by using an automation plug-in, ensure that you include the role whose credentials are used when running the automation plug-in.

Automation plug-ins run by notifications are always defined as internal event receivers because notifications are used to notify OSM users or other areas of the OSM system of something happening within OSM. To notify users on external systems, you need to configure an automation plug-in.

When you use automation plug-ins, you configure the automation plug-in properties; for example, the automation type and the event type. See OSM Developer’s Guide for more information.

About Configuring Notifications

You define notifications when you model orders, tasks, and processes. There is no OSM notification entity, so you cannot model notifications and reuse them.

Before you configure notifications, you need to configure the following entities:

- You must create the roles to assign notifications to.
- To trigger notifications based on a change to order data, you must first model the data. See "About Using Order Data Changes to Trigger Notifications" for information.

You can model automation plug-ins as you define notifications, but modeling automation plug-ins before you configure notifications is more efficient.

Strategies for Using Notifications

Some common uses for notifications are:

- In general, jeopardy notifications are used for alerting order management personnel about something that should have happened but did not happen. By contrast, event notifications are based on events that have happened, and they are used more for communicating status information and for directing the order fulfillment process to the next step.

- Communication with external systems is usually handled by automation plug-ins run by event notifications. For example, the progress of an order is typically monitored in external systems by tracking which parts of the order have been completed. To communicate that, you typically configure event notifications based on a change to order data or a change to task status.

- Notifications intended for an internal audience (OSM users) are typically created using a notification type that, by default, sends a notification to the Task Web client. The only notification types that do not are event notifications based on order data change and task state change notifications that run an automation plug-in. See "About Using Task States and Rules to Trigger Event Notifications" for
About Jeopardy Notifications

A jeopardy notification is a message that is sent to OSM users or users on other systems (for example, to return status to a CRM system). Jeopardy notifications are not event-driven; they use polling at specified intervals to identify processes or tasks in jeopardy.

OSM uses three methods to deliver jeopardy notifications:

- By displaying a notification in the Task Web client.
- By sending email to users.
- By using an automation plug-in to notify an external system. Each order jeopardy notification can map to one automation plug-in.

Jeopardy notifications can be defined for an order or for a task. Many of the jeopardy properties are the same for orders and tasks; for example, you can specify the roles to notify and the rule to trigger the notification. However, defining a jeopardy for an order or a task allows you use the order or task properties. For example:

- You can trigger a notification based on the state of the order.
- You can trigger a notification if a task has exceeded its expected duration.

You can use two methods to trigger a jeopardy notification:

- Conditions; for example, if the order processing time has exceeded the expected duration.
- Order rules; for example, you can define an order jeopardy notification based on a rule that evaluates the data condition `orderMilestone <> completion and dueDate > SpecifiedDate`. This checks to see if there are any orders that are not completed but that are supposed to be completed by today.

To trigger a notification, OSM follows this process:

1. OSM polls in-flight orders and tasks to determine if a condition has been met. For example, the condition might be that a task has been in progress for longer than one hour. If the condition is met, OSM begins to process the notification.
2. OSM checks to see if there are any rules that might restrict the notification from being triggered. For example, you might configure two types of jeopardy notifications, one that is triggered for orders from only business accounts and one that is triggered for orders from only residential accounts. Each notification might have a different email recipient, so the notification is only triggered for the correct recipient.
3. If the rule evaluates to true, the notification is triggered.

You can specify how often OSM should poll to re-evaluate the jeopardy condition. You can specify a polling interval in hours, days, weeks, or months. You can specify the day of the week (for example, Monday), or the day of the month (for example, the first day of the month). You can specify a date and time for OSM to begin polling. The default is the current date.
About Jeopardy Notifications

Tip: When configuring notifications, consider the performance impact from polling for jeopardy notifications. For example, a configuration that polls every minute on one million orders has a much greater performance impact than polling every hour on one thousand orders.

About Jeopardy Notification Conditions

You can trigger jeopardy notifications based on an order or task condition. For example, you can specify to send a jeopardy notification if a task has exceeded its expected duration.

The conditions you can use are different, depending on if you define the jeopardy notification in an order or in a task.

Specifying Jeopardy Notification Conditions for an Order

When you define an order, you can specify to trigger a jeopardy notification based on the following:

- The number of days that an order has been in the In Progress state. For example, you can trigger a notification if the order has been in the In Progress state for longer than 30 days.
- The number of days that an order has been in the Completed state. For example, you can trigger a notification if the order has been in the Completed state for longer than 30 days.
- If the process duration has exceeded the expected duration. The value is based on elapsed time, regardless of the order states that the order might transition in and out of.
- If the process duration has exceeded a duration that you define; for example, five days. This duration value starts at the creation task.

To determine the duration that the order has been in any of these conditions, OSM polls the system at an interval that you define.

Specifying Jeopardy Notification Conditions for a Task

When you define a task, you can specify to trigger a jeopardy notification based on the following:

- If the process that the task is associated with has exceeded the expected duration.
- If the process that the task is associated with has exceeded a duration that you define. This duration is measured starting with the creation task.
- If the task has exceeded the expected duration.
- If the task has exceeded a duration that you define.
- If the order has exceeded a specified amount of time past when it was received (when the order is created in OSM).

To determine the duration that the order has been in any of these conditions, OSM polls the system at an interval that you define.

When you define a jeopardy notification in a task, and the task can have multiple instances, you can specify if the notification should be triggered for every task instance.
Example of a Jeopardy Notification

In this example, the order processing for a service requires that a customer service manager enter payment information. This has been configured as a manual task. It is expected that the task should be completed quickly. Therefore, a jeopardy notification is configured that triggers if the task is in the Received state for longer than one hour.

Figure 11–3 shows the configuration for a jeopardy notification in Design Studio. This jeopardy notification sends an email notification to members of a workgroup.

Figure 11–4 shows the conditions under which the jeopardy notification is triggered. In this case, the given notification specifies to trigger notification if the task has been in the Received state for longer than one hour.
About Event Notifications

Event notifications are triggered by events. You do not specify polling intervals for event notifications. You can configure them to occur in the following cases:

- When a task transitions through a task status. For example, you might trigger an event notification when a task transitions to the Failed status.

Event notifications triggered by transitions can be sent to a workgroup. See "About Using Task Transitions to Trigger Event Notifications" for more information.

- When a task reaches a specified state. You can use two methods:
You can use the task state to trigger an automated event notification. In this case, only the task state is evaluated (no rules are applied to evaluate a condition), and the notification runs an automation plug-in that handles the notification actions. For example, when a task reaches the Assigned state, you can automate an external lookup before allowing the workflow to continue. You do not specify roles or email delivery for the notification. See "About Using Task States to Trigger Automated Event Notifications" for more information.

You can use the task state in combination with rules to trigger the event notification. In this case, you can specify a rule to evaluate conditions, the priority, if the notification can be delivered by using email, and the workgroups that receive the notification. See "About Using Task States and Rules to Trigger Event Notifications" for more information.

When you use the task state to trigger an automated event notification, the notification is run from all processes that include the task. When you configure a notification based on a task state change in a process, the notification is applicable only to the task within the process in which it is defined.

■ You can trigger an event notification when an order passes an order milestone. You use this type of notification to trigger an automation plug-in that handles the notification actions. You do not specify roles or email delivery for the notification. See "About Using Order Milestones to Trigger Event Notifications" for more information.

■ You can trigger an event notification when a change is made to order data. You typically use these notifications to update external systems (such as a CRM) with information about the progress of the order when a specific data element in the order data is changed. See "About Using Order Data Changes to Trigger Notifications" for more information.

About Using Task Transitions to Trigger Event Notifications

An event notification based on a task transition does not apply to all instances of the task. It applies to a task only as it is used in a specific process. Therefore, to configure an event notification based on a task transition, you edit the process that includes the transition and apply the event notification to the transition. Figure 11–6 shows the configuration for a success transition in Design Studio. In this figure, the success transition is selected, and the event notification properties are defined below the process window.
About Event Notifications

Figure 11–6  Event Notification Based on Task Transition

The event notification for a status change works as follows:

1. When the task status changes to the status that you define for the notification, the notification runs a rule to evaluate if the conditions are true.

2. If the conditions are true, the event notification is triggered.

When you use a task transition to trigger an event notification, you can specify an automation plug-in that the notification runs; however, an automation plug-in is not required.

About Using Task States and Rules to Trigger Event Notifications

An event notification triggered by a task state change and rules works as follows:

1. When the task state changes to the state that you define for the notification, the notification runs a rule to evaluate if the conditions are true.

2. If the conditions are true, the event notification is triggered.

For example, you can specify that when the Completed task state is reached, a rule evaluates if the billing address is in California.

This type of notification does not apply to all instances of the task. It applies to a task only as it is used in a specific process. Therefore, you create this type of notification when you create processes in Design Studio. Figure 11–7 shows how to assign an event notification to a task in a process. In this figure, the EnterAccountInformation task is selected, and the rule and state are defined in the window below.
You can specify an automation plug-in that the notification runs; however, an automation plug-in is not required.

**About Using Task States to Trigger Automated Event Notifications**

You can use a task state to trigger an automated event notification. In this case, only the task state is evaluated (no rules are applied to evaluate a condition), and the notification triggers an automation plug-in which handles the notification actions. This type of notification runs for every instance of the task, independent of the process that it is in. Event notifications triggered by task states are not displayed in the Task Web client.

For example, you can define an automated notification that sends a notification when the task reaches the Assigned state. The event notification works as follows:

1. When the task reaches the Assigned state, a notification is created.
2. When the notification is created, the automation plug-ins run.

*Figure 11–8* shows an event notification configured in Design Studio. Any time this task runs, the event notification is triggered when the task reaches the Completed state.
For more information about automation plug-ins, see "About Using Notifications to Run Automation Plug-Ins".

**About Using Order Milestones to Trigger Event Notifications**

You can use an order milestone to trigger an event notifications. Figure 11–9 shows an event notification based on an order milestone.
About Event Notifications

Only the order milestone is evaluated (no rules are applied to evaluate a condition), and the notification triggers an automation plug-in that handles the notification actions. Each event notification maps to one or more automation plug-ins. For more information about automation plug-ins, see "About Using Notifications to Run Automation Plug-Ins".

Event notifications triggered by order milestones are not displayed in the Task Web client.

For example, you can define an event notification that specifies the Completion milestone. The event notification works as follows:

1. After all tasks within a process successfully complete for an order, the order Completion milestone is reached and a notification is created.
2. When the notification is created, the automation plug-ins run.

Note: You cannot define custom order milestones. Order milestones are based on order states; for example, the Completion milestone occurs when the order transitions to the Completed state.

When you create event notification that is triggered by an order milestone, you specify the order milestone that triggers the notification. You can use the following order milestones:
About Event Notifications

- **Creation:** The order was created in the OSM system.
- **Completion:** The final task in the order has completed, and the order transitioned to the Completed state.
- **Deletion:** The order was removed from the OSM system by transitioning to the Deleted state.
- **Exception:** A process exception or fallout was initiated.
- **State change:** The order transitioned to a different state.

**About Using Order Data Changes to Trigger Notifications**

You define event notifications based on order data changes when you create orders in Design Studio. For example, you can define an event notification that sends a notification when a telephone number is entered. Event notifications triggered by data changes are shown in the Task Web client.

When you create an event notification based on order data changes, you can specify the data field that triggers the notification when the data is changed. Any change to the field causes the notification to trigger. However, this value is not evaluated for content. To trigger the notification based on the value of the data, you must configure a rule to evaluate it.

For example, to trigger a rule when the billing address is changed to California, you specify the billing address field as the field that triggers the notification and run a rule that evaluates if the address was changed to California.

You can specify an automation plug-in that the notification runs; however, an automation plug-in is not required.

Figure 11–10 shows an event notification based on data change in an order. In this example, when a credit card number changes, the notification is triggered.

*Figure 11–10  Event Notification Based on Data Change in an Order*
### Summary of Notification Functionality

Table 11–2 shows a summary of notification functionality.

<table>
<thead>
<tr>
<th>Notification Type</th>
<th>Sends Email</th>
<th>Displays in Task Web Client</th>
<th>Can Be Evaluated By a Rule</th>
<th>Can Be Sent to Different Roles</th>
<th>Runs Automation Plug-in</th>
<th>Has a Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jeopardy - Task</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Optional</td>
<td>Yes</td>
</tr>
<tr>
<td>Jeopardy - Order</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Optional</td>
<td>Yes</td>
</tr>
<tr>
<td>Event - Task status</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Optional</td>
<td>No</td>
</tr>
<tr>
<td>Event - Task state, automation</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Defined by automation plug-in only</td>
<td>Mandatory</td>
<td>Yes</td>
</tr>
<tr>
<td>Event - Task state; in a process</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Optional</td>
<td>Yes</td>
</tr>
<tr>
<td>Event - Order milestone</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Defined by automation plug-in only</td>
<td>Mandatory</td>
<td>No</td>
</tr>
<tr>
<td>Event - Order data change</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Optional</td>
<td>Yes</td>
</tr>
</tbody>
</table>
This chapter describes Oracle Communications Order and Service Management (OSM) order fallout handling.

Understanding Order Fallout

**Order fallout** occurs when an order fails during processing. Order fallout is often called **order failure**. **Fallout management** is the ability to resolve fallout and allow an order to continue processing. You can model automated fallout management, which corrects errors by compensation, or you can model manual fallout management, which supports manual intervention to correct errors.

OSM can manage fallout that occurs both internally during OSM processing, such as errors in internal data, and as the result of an error returned by an external fulfillment system. The most common fallout scenarios are:

- Failure in a downstream system; for example, a failure in an activation system. Scenarios include:
  - The data was received, but was missing or incorrect and could not be processed. When a downstream system detects missing or incorrect data received from OSM, it returns an error, which in turn fails the order.
  - An internal error unrelated to the data occurred.

  See "Managing Fallout Generated from a Failure in a Downstream System" for more information.

- Failure in a network or system resource; for example, a network connectivity failure or a database failure. See "Managing Fallout Generated from a Failure in a Network or System Resource" for more information.

- Failure when an order is created in OSM; for example, recognition or validation errors. If OSM receives a corrupted order from an external system, it accepts the order but immediately places it in Failed state. See "Managing Fallout Generated from Failure When an Order Is Created in OSM" for more information.

- Failure in run-time OSM execution; for example, an unresolved dependency that prevents an order from being processed. See "Managing Fallout Generated from Failure in Run-Time OSM Execution" for more information.

Managing fallout typically follows three stages: detection, notification, and recovery.

- **Detection.** You must be able to detect when an order has failed. For example:
  - A process can transition an order to Failed state. When it does, you can specify to send a message to a fallout specialist.
– Order management personnel can monitor the progress of an order in the Task Web client or in the Order Management Web client. For example, the progress of an order might indicate that something failed. If an order uses orchestration, a problem in an order component can prevent dependant order components from processing.

When an order is identified as fallout, the order state is typically changed to the Failed state.

- **Notification.** Depending on the error, you might need to notify other systems that a problem occurred and why. For example, the CRM system must know if an order has failed because of incorrect data. You can configure OSM to send email notifications to fallout managers or to notify an external trouble-ticketing application.

- **Recovery.** Corrective measures can be taken automatically or manually to fix the problem and then resume or restart the order. Using the Order Management Web client, you can search for failed orders and examine their data to determine the cause of the error.

Order recovery can be carried out in various ways. For example:

- A revision order can be submitted with corrected data. For example, if an order fails because the specified type of telephone is not available, a revised order can be submitted that changes only the type of telephone for the order.
- Order management personnel can edit the order data in the Task Web client and resume processing of the order.
- The order can be terminated and a new order submitted.

OSM in the central order management role can function as the central fallout management system, coordinating fallout activity for all of the order processing in your system. For example, in an order-to-activate solution, OSM can be integrated with the Siebel Service module and other systems. In such a solution, OSM might do the following:

- Receive error information from fulfillment systems. This can include receiving errors from OSM running in the service order management role.
- Generate trouble tickets in a trouble-ticketing system when there is order fallout.
- Issue updates to trouble tickets for multiple issues on the same order item.
- Update trouble ticket status when milestones are reached, such as when an issue is resolved.

If your OSM implementation includes both central order management and service order management, order fallout can occur and be managed in both instances. An order failure can affect each role independently, or the same failure can affect OSM running in both roles. For example:

- An error returned from a billing system can fail an order in the central order management role without affecting service order management.
- An error returned from a provisioning system can fail an order in service order management, and, if the provisioning is part of an order component, the failure might create an unresolved dependency in central order management.

**Managing Fallout Generated from a Failure in a Downstream System**

Failures from a downstream system are typically caused by the following:
■ An invalid message request caused by insufficient or incorrect data.

■ The request cannot be completed because of the state of the user’s account; for example, an inactive account.

■ The network inventory does not represent the resources in the actual network; for example, a port has been assigned that is already in use.

■ An internal application error has occurred in the downstream system.

These scenarios typically must be resolved directly; retrying or waiting does not resolve the problem.

A failure in a downstream system is usually detected by using an automation plug-in. The failure message is received in the JMS receiver queue and correlated to an order and task context. Details about the failure are stored in the order data.

---

**Note:** OSM can maintain a history of the communications with external systems. However, because of the work that OSM must do to manage and store the messages, this can impact performance.

---

In addition to automated detection, order managers can use the Web clients to find failed orders.

Notification of fallout from failures in downstream systems is typically handled by the following:

■ Generating a trouble ticket in a trouble-ticketing application.

■ Sending the failure information to the order-source system.

■ Sending email notifications to fallout administrators.

Notifications can be based on changes to order milestones, order data, task states, or task status. See "About Notifications" for more information.

To recover from a failure in a downstream system, actions include:

■ Set the order state to Failed. This is often called a **hard failure.** This stops all processing on the order while recovery is undertaken. You can then resolve the problem in the downstream system. When it is fixed, reset the status of the order to In Progress by, for example, using the ResolveFailure Web services operation.

■ Set the failed task to a user-defined failed status. This is often called a **soft failure,** because the order is not failed. The order remains in the In Progress state, and other tasks can still be carried out while the recovery is managed. To correct the problem, you can manually complete the task or reset the state of the task to Received, which retries the task.

■ Transition the task to a manual fallout recovery task. This provides a specific set of data that applies to redoing the task. You can then transition the failed task back to the Received state to retry it. This option is not always applicable, because it can require the order manager to maintain data integrity instead of allowing OSM to handle the compensation.

■ Change the order to the Aborted state and resubmit it from the order-source system.

With all of these options, you can specify query tasks and roles to restrict the recovery to fallout managers.
Managing Fallout Generated from a Failure in a Network or System Resource

A failure in network or system resources is typically one of the following:

- Network connection failure
- Middleware failure
- Database failure
- Fulfillment system down or running slow

These failures are usually temporary, and can be resolved by waiting or addressing the network problem. Order processing might be delayed, but is usually not otherwise affected.

These failures can be detected by using an automated response notifying OSM of the failure. These failures are usually handled by a jeopardy notification, triggered by a task or order not completing in the expected amount of time.

Network and system failures are usually managed by the network administrator or system administrator. They usually do not affect the status of the order; the order retries processing and resumes when the network is available. A typical recovery scenario is as follows:

1. The automated task receives an error.
2. The automation framework fails the transaction, rolls the transaction back, and sends a JMS message to the queue.
3. The transaction is retried. You can configure how many times to retry and how long to wait before retrying.
4. When the configured limit or retries is reached, a JMS error message is sent to the queue. (The default configuration is unlimited retries and no error sent.)

Managing Fallout Generated from Failure When an Order Is Created in OSM

Failures in order creation can occur because of the following:

- Failure when the CreateOrder Web services operation is used, usually while processing a validation rule or transformation rule. The order fails, and the original incoming customer order is attached. You can publish an event based on order failure.

  In the case of validation errors, revise the order request and resubmit it. In the case of transformation errors, troubleshoot and fix the transformation logic, and resubmit the order.

- Failure when the CreateOrderBySpecification Web services operation is used, usually because the input data is not valid or permissions are not correctly set. The error response can be:
  - InvalidOrderSpecificationFault
  - InvalidOrderDataFault

  The error response includes error details.

  If either of these two faults is returned, revise the order and resubmit it.

- The order is a revision order, and the point of no return based on order state transition has been reached on the base order. In this case, TransactionNotAllowedFault is returned. If you have configured a follow-on order
for this scenario, you can submit the follow-on order. Otherwise, you can submit a new order.

You can specify to display a message in the Task Web client if an order fails during validation and transformation. To do so, specify the fail-order reason when you model the recognition rule in Oracle Communications Design Studio.

**Tip:** An order that fails to be recognized by any recognition rule is rejected by OSM and lost. No record of it is sent to the order-source system. To ensure that all orders are captured in OSM, create a recognition rule that accepts all incoming customer orders. Prioritize it at the lowest level (0) and prioritize all other recognition rules higher so they are processed first. Using this lowest-level recognition rule, an invalid order is recognized, and then it fails during validation. It then transitions to Failed state and is kept by OSM.

Recognition rules are global entities. An incoming customer order could be recognized by a recognition rule deployed in the system that you did not intend to be matched if you are not careful with the relevancy settings and the recognition rule.

### Managing Fallout Generated from Failure in Run-Time OSM Execution

Failures in run-time OSM execution are typically caused by OSM modeling errors, usually in the design of automations, rules, and orchestration. These errors are usually resolved in test systems and should not be a common occurrence in a production system.

A failure in an automation results in an exception. You can configure how the error is handled; for example, fail the order, transition the task to a user-defined failed state, or transition to a fallout recovery task.

A failed rule results in the associated tasks being marked as INVALID. Notifications are provided as system events, displayed in the Task Web client.

Orchestration failures typically result in incorrect or empty orders.

### About the Order States Used for Fallout Management

When an order fails, the Fail Order transaction transitions the order to the Failed state. An order can transition to the Failed state from the following states:

- Not Started
- In Progress
- Waiting for Revision
- Suspended

When the problem is fixed, the order can be moved out of the Failed state as follows:

- If the order was failed from the Not Started, In Progress, or Waiting for Revision states, the Manage Order Fallout transaction moves the order back to the state it was in before being failed.
- If the order was failed from the Suspended state, the order is transitioned back to the Suspended state.

If the order needs a revision to be fixed, the Submit Amendment transaction places the order in the amendment queue, after which the Process Amendment transaction transitions the order to the Amending state. A revision can come from two sources:
- The originating order-source system can enter a revision order.
- A process exception, which includes redo and undo operations, can run.

If the order must be restarted, the Cancel Order transaction transitions the order to the Cancelling state, and then to the Cancelled state. This operation undoes all changes and returns the order to the creation task.

**Note:** If the order has an orchestration plan, it cannot be restarted after being canceled. The Cancelled state is a final state for orders that have an orchestration plan.

If the order cannot be revised or restarted, the Abort Order transaction transitions the order to the Aborted state.

See "About Managing Order States and Transitions" for more information.

**Example of Discovering Fallout**

The following example shows how fallout management combined with amendment processing can resolve an order failure:

1. A customer places an order for 8 MBps DSL service. The customer service representative takes the order but neglects to find out if the service is available before entering the order in the system.

2. OSM receives the order and begins processing it. It sends fulfillment requests to the various fulfillment systems (for example, billing, mobile provisioning and DSL provisioning).

3. The DSL provisioning system receives the request for HSD 8 MBps, but it cannot support that speed. It responds with an error.

4. At this point, the order is still in the In Progress state. This allows other components to continue processing. However, the order has an expected completion date that is not met, so a jeopardy notification is sent to the order manager.

5. The fallout manager receives the notification and opens the order in the Order Management Web client. Upon reviewing the order, the order manager sees that the fixed provisioning, mobile provisioning, and their respective billing functions are complete, but the DSL Provisioning function remains in the Started state.

6. Upon investigating the DSL provisioning order component, the order manager sees that the DSL provisioning system returned an error code of “1” and a message of “Not enough speed for HSD 8M”.

7. A status update is sent to the CRM system alerting the order entry personnel that a problem has occurred, and the 8 MBps speed is not available. The order remains in the In Progress state, but no further processing can happen until the DSL provisioning is resolved. There is no reason for the fallout manager to suspend or fail the order at this point.

8. The CRM system sends a revision order, which is based on the original order. Instead of an 8 MBps speed, the revision order requests a 1 MBps speed.

9. OSM receives the revision order, processes it, and the order resumes.
About Configuring Fallout

A typical fallout management configuration allows you to do the following:

- Follow a predefined recovery process for orders. To do so, you can configure processes and tasks that fallout managers complete in the Task Web client. The fallout recovery tasks allow you to edit the order and then manually transition the order back into processing.

- Configure amendment processing to handle changes that are reinstated from the originating upstream system, such as revision orders and order cancellation. See "Managing Changes to Orders" for information about amendment processing.

- Model failure flows into your processes. For example, a failure flow might consist of these steps:
  1. Wait for amendment.
  2. Retry.
  3. Resume processing.

- Model orders that are specifically designed to handle order fallout. For example, you might create an order type that collects data about an incoming customer order that failed, with processes and tasks that allow a fallout manager to interface with a trouble-ticketing application.

- Recognize incoming error messages from external systems and use them to trigger notifications in OSM.

- Alert internal fallout managers and external systems about order failure and recovery status. To do so, you can configure event notifications.

- Define roles to allow specific users to handle fallout-management tasks.

- Create automated fallout messages based on order milestones. Figure 12–1 shows an automated fallout message triggered by the exception order milestone. For more information about order milestone notifications, see "About Using Order Milestones to Trigger Event Notifications".
You can use the Order Management Web client to review failed orders to determine why they failed. You can configure fallout entities in Design Studio to specify the data that you want to display in the Order Management Web client. To do so, when modeling an order, create a fallout entity and include it in the order model. A fallout entity includes one or more data elements that you want the Order Management Web client to display.

Figure 12–2 shows a fallout configured in OSM. In this example, the fallout is named PortAlreadyAssigned. It is used when a task for activating a service fails because a port was assigned that is not available. The data element is `asdl_service_details/port_id`. 
After you configure fallouts in the order specification, you can assign those fallouts to manual tasks. This association enables OSM to identify the task that generated the error, transition the order to the Amending state, and initiate amendment processing.

To resolve fallout, OSM follows the same process as when it performs amendment processing: It builds a compensation plan, and then applies the required changes.

Fallout can be triggered based on a single incorrect field in a single task. Because fallout can be mapped to one or more data elements, it is possible to have multiple errors in a single task view.

You can also create fallout groups to simplify assigning fallout data to orders. A fallout group is a group of fallout specifications, each of which includes a set of data to display in the Order Management Web client. This enables you to review multiple fallouts together in the Order Management Web client when the corresponding types of fallout occur.

To trigger fallout in an automated task, use the XML API FalloutTask.Request through com.mslv.oms.automation.OrderContext.processXMLRequest.

**About Configuring Fallout Processes and Tasks**

You can configure fallout-management processes and tasks to enable your fallout specialists to manage fallout by using the Task Web client. Fallout management tasks include such tasks as submitting and responding to trouble tickets, resubmitting orders, failing orders, and undoing tasks following a failure.

Figure 12–3 shows a fallout process (OrderCreationFalloutSubprocess) included in the AIA Order-to-Activate fallout cartridge. This process manages the fallout in the event of an order creation error. For more information, see *OSM Cartridge Guide for Oracle Application Integration Architecture*. 

---

**Figure 12–2 Fallout Defined in an Order**

![Image of fallout defined in an order](image-url)
In addition to creating processes and tasks specifically to manage fallout, you can add failure flows to any type of process; for example, provisioning processes.

Managing Fallout in the OSM Web Clients

You can use both the Order Management Web client and the Task Web client to manage order fallout.

- You typically use the Order Management Web client to examine order dependencies and history to determine the cause of the failure. You can also manually suspend or fail an order. See "Managing Fallout in the OSM Order Management Web Client".

- You typically use the Task Web client to run fallout management tasks; for example, fallout recovery tasks that use undo and redo tasks. See "Managing Fallout in the Task Web Client".

Both clients can be used for fallout management, but the primary differences are:

- You use the Order Management Web client to search for orders that have failed and find problems based on orchestration; for example, unresolved dependencies. You can view the problem that is causing the order to fail, but you cannot resolve the order failure by changing order data.
Managing Fallout in the OSM Web Clients

You use the Task Web client to manage problems with tasks and processes; for example, you can manage failed orders by redoing and undoing tasks. You can change order data that may resolve the order failure.

For orchestration orders, you sometimes must work in both clients to manage fallout because you require both an orchestration view and a task-level view of the order. For example, you must find and examine an order component of an order in the Order Management Web client and then drill down to the worklist to see the task-level view of the order component’s tasks in the Task Web client.

Each client can launch the other client when required. To learn more about navigating between the clients so you can quickly access the orchestration view and task-level view of an orchestration order, see the getting-started discussions in each Web client’s user guide.

Managing Fallout in the OSM Order Management Web Client

In the Order Management Web client, you can do the following:

- Find orders that are in the In Progress state but have components that have not started because a dependency has not been resolved. An unresolved dependency can indicate a problem in the fulfillment process.
- Find orders that are in the Failed state.
- View details about the order to determine why it failed. For example, you can look for unresolved dependencies or display the order history.

To correct the error that caused the failure, you often must use the OSM Task Web client to run fallout-related tasks, such as fallout recovery tasks. You might also need to work with external systems. There is no functionality in the Order Management Web client to manually run tasks.

While resolving the order failure, you can suspend the order to prevent processing on it. You can resume the order when you are ready to resolve the failure.

After the problem has been resolved, you use the Order Management Web client to resolve the order. When you do, you enter a description of how the order was resolved in the Resolve Order Failure dialog box. The order state is reset to In Progress. Using this feature does not perform the steps required to resolve the failure; it tells OSM that the problem has been resolved and that the order can be resumed.

---

**Note:** If the order failed because of a recognition rule failure or after reaching its point of no return, it cannot be resolved. Also, the ability to suspend, cancel, or terminate an order depends on its life-cycle policy.

---

If you cannot resolve the order, you can use the Order Management Web client to cancel or terminate the order:

- Canceling an order immediately stops its processing and sets the order state to Canceled. Any tasks that have already completed for the order are rolled back. If the order has an orchestration plan, the order cannot be resumed. If the order does not have an orchestration plan, it can be resumed.
- Terminating an order immediately stops its processing and sets the order state to Aborted. The order cannot be resumed. Unlike canceling an order, terminating an order does not roll back any tasks that have already completed. As a result, clean-up may be required.
Managing Fallout in the OSM Web Clients

Note: Consider the impact on other systems of canceling or terminating orders. Depending on how your solution is configured, upstream systems may not be aware that an order has been canceled or terminated.

In most cases, orders are failed automatically. You can also use the Order Management Web client to fail an order manually. Failing an order stops its processing and sets its state to Failed. It is not possible to change the state of a failed order or to make other changes until you resolve the order failure. Orders you fail manually are treated the same way as orders that are failed automatically by the system. They are considered fallout.

Note: In most environments, fallout-handling rules detect processing problems and automatically fail orders. Manually failing orders is not normally required. There may be some situations and environments when it is necessary to manually fail orders, however.

Ensure you understand how other systems in your order processing solution handle failed orders. Depending on how your solution is implemented, upstream systems may not be aware that an order has been manually failed.

Managing Fallout in the Task Web Client

You can initiate fallout in the Task Web client by raising an exception. An exception is a mechanism used to interrupt or stop an order or to redirect it to any task in the same process or any other process. You can use two types of exceptions: process exceptions and fallout exceptions.

You can use a process exception to stop or redirect an order. Process exceptions are typically part of the configured order flow and can be used to manage the order manually. Figure 12–4 shows the Process Exception page in the Task Web client. In this example, an error has been made in the order processing, and the process exception redirects the order to correct it.
You can use a fallout exception to initiate fallout to correct an error. A fallout exception allows you to initiate fallout from a particular task to correct an error caused by a previous task. When you raise a fallout exception, the system identifies the task that generated the error, transitions the order to the Amending state, and initiates amendment processing.

To recover from order fallout, the order might require a revision order to redo some of the order processing. Figure 12–5 shows how the system manages compensation tasks due to fallout.

In this scenario, Task B is responsible for the error and Tasks C and D include the error data. The fallout exception is raised at Task G.
In this figure:

1. An order is processed using the above workflow following the path A, B, C, D, G.
2. A fallout exception is raised at Task G.
3. OSM determines that Task B first output the error and initiates amendment processing as follows:
   - **Same branch**: If, during the redo processing of Task B, the task completes with the same completion status as it did in normal processing, subsequent Tasks C and D are also redone and the flow is complete.
   - **Different branch**: If, during the redo processing of Task B, the task completes with a different completion status causing Task E to be the next task, the obsolete branch of Tasks C and D must be undone and the new branch of Tasks E and F must be done while still in the Amending state.

**Note:** If the error data was generated by the creation task, the order transitions to the Failed state. No compensation tasks are created and the order must be corrected through an external amendment.
You use XQuery expressions in various locations to implement key aspects of the Oracle Communications Order and Service Management (OSM) orchestration functionality. For information about these XQuery expressions, refer to the following topics:

- General XQuery Information
- Order Recognition Rule XQuery Expressions
- Decomposition XQuery Expressions
- Dependency XQuery Expressions
- Order Transformation Manager XQuery Expressions

**General XQuery Information**

This topic contains general or reference information about XQuery that applies the same in different situations.

When working with XQuery expressions, see the following topics:

- About Creating XQuery Expressions with Design Studio
- OSM XQuery Functions
- Referencing Items from a Distributed Order Template in XQuery Expressions

**About Creating XQuery Expressions with Design Studio**

In general, the way you enter XQuery information into editors in Oracle Communications Design Studio is the same, regardless of the editor. The XQuery control in Oracle Communications Design Studio generally has three tabs: XQuery, Instances, and Information. Following are general instructions for entering XQuery information into each of these tabs in Design Studio.

**Using the XQuery Tab**

The **XQuery** tab allows you to configure XQuery-based rules or elements, or identify the source of the XQuery-based rules or elements. Select one of the following options:

- Select **None** if the XQuery configuration is optional and not configured. When you select this option, Design Studio disables the remaining options in the subtabs.

- Select **Expression** and enter the XQuery expression in the corresponding text box. Click **Edit** to open the Edit XQuery dialog box, which displays the configured XQuery expression in a larger and resizeable text box. You can edit the expression
in the Edit XQuery dialog box and click **OK** to save your changes, or click **Cancel** to dismiss the dialog box without saving the changes.

---

**Note:** Design Studio provides XQuery validation on basic syntax and semantics, and denotes errors with Problem markers.

---

- Select **File** to denote that the XQuery configuration is located in a file saved to the project **resources** directory. This option enables you to write your XQuery expressions using any XQuery editing application you have installed in your Eclipse environment. See the Eclipse online Help topic *Associating editors with file types* for more information.

  Click **Select** to open the Select XQuery File dialog box, which displays all XQuery files contained in the project **resources** directory. Select the appropriate XQuery file and click **OK**.

- Select **URI** to denote that the XQuery configuration is located in a remote URI location. For example, you might enter:

  **http://osm_server/AIARecognitionRule.xqy**

  Click **Properties** to open the Properties view, where you can define the following information for the XQuery:

  - **Annotation:** The optional XML annotation element allows you to provide information about the XQuery. Enter information (for example, HTML-formatted information) for external systems into the **Annotation <appinfo>** field. Enter information for human users into the **Annotation <documentation>** field.

  - **Language:** When you work with multiple languages, you can select a different language for displaying the description and annotation. For more information, see "Defining Language Preferences" in *Modeling OSM Processes* Help.

**Using the Instances Tab**

You can define a Data Instance behavior to obtain data that is not included in the order data and make that data available to the rule. Click **Add** to add a Data Instance behavior. Select the Data Instance behavior and click **Properties** to configure the Data Instance behavior.

For more information, see "Defining Data Instance Behavior Properties" in *Modeling OSM Processes* Help.

**Using the Information Tab**

Use this tab if you want to describe the intended use of the rule. For example, you might describe the functionality of a complex rule or provide instructions on its use.

**OSM XQuery Functions**

OSM-specific XQuery functions are available to you when writing XQuery expressions. These XQuery functions are contained in classes that you can declare in the prolog of your XQuery expression.

To see specifics about the functions available, install the OSM SDK and extract the **OSM_home/SDK/osm7.x.y.z-javadocs.zip** file (where **OSM_home** is the directory in which the OSM software is installed and x, y, and z are the software release, patch, and build numbers respectively). This file contains the OSM Java Docs. See *OSM Installation Guide* for more information about installing the OSM SDK.
The specific classes that contain XQuery functions you might use are:

- **OrchestrationXQueryFunctions**: This class contains XQuery functions that are used in OSM Orchestration. To declare this class, put the following declaration in the prolog of your XQuery expression:

  declare namespace osmfn = 
  "java:oracle.communications.ordermanagement.orchestration.generation.OrchestrationXQueryFunctions";

- **XQueryFunctions**: This class contains XQuery functions that are used in the order transformation manager. To declare this class, put the following declaration in the prolog of your XQuery expression:

  declare namespace otmfn = 
  "java:oracle.communications.ordermanagement.orchestration.transformation.XQueryFunctions.";

### Referencing Items from a Distributed Order Template in XQuery Expressions

The distributed order template is an option you can set on an order item specification to modify the method used to store order item data. For more general information about the distributed order template, see OSM Concepts.

When using a distributed order template, any XQuery expressions that reference order item data must be in a particular format.

For any order item that is not a transformed order item, you must include the namespace of the order item specification. Following is an example of an XQuery reference to the `lineItemID` property on the `InputOrderItem` order item with the namespace [http://ex_input.com](http://ex_input.com):

```
/ControlData/OrderItem[@type='{http://ex_input.com}InputOrderItem']/lineItemID
```

For transformed order items, the format depends on the source of the data for the transformed order item. Data that is defined in the order item specification itself will use the namespace for the order item specification, the same way that data would be referenced for an input order item. Following is an example of an XQuery reference to the `lineItemID` property on the `OutputOrderItem` order item with the namespace [http://ex_output.com](http://ex_output.com):

```
/ControlData/OrderItem[@type='{http://ex_output.com}OutputOrderItem']/lineItemID
```

Data that has been derived from a common model entity, for example an action, will use a different format. In the following situation:

- Order item name: `OutputOrderItem`
- Order item namespace: [http://ex_output.com](http://ex_output.com)
- Conceptual model entity (in this case an Action) name: `SA_Add_Internet`
- Conceptual model cartridge name: `Model_Broadband`
- Conceptual model cartridge version: `1.0.0.0`
- Parameter name on `SA_Add_Internet`: `serviceLevel`

The reference would look like this:

```
/ControlData/OrderItem[@type='{http://ex_output.com}OutputOrderItem']/dynamicParams[@type='{Model_Broadband/1.0.0.0}SA_Add_InternetType']/serviceLevel
```
Note that the type for the parameters contained in the conceptual model entity has the string "Type" appended to the name of the entity. Thus, the type contains **SA_Add_InternetType** rather than just **SA_Add_Internet**.

**Order Recognition Rule XQuery Expressions**

The following topics provide reference information about order recognition rule XQuery expressions:

- About Recognition Rule XQuery Expressions
- About Validation Rule XQuery Expressions
- About Order Priority XQuery Expressions
- About Order Reference XQuery Expressions
- About Order Data Rule XQuery Expressions

**About Recognition Rule XQuery Expressions**

This topic describes how to use the Order Recognition Rule editor Recognition Rule area **XQuery** tab to write an expression that specifies a customer order and associates it with an OSM target order type. The XQuery has the following characteristics:

- **Context**: The input document for the Recognition Rule **XQuery** is the customer order. For more information about typical customer order structures, see **OSM Concepts**.

- **Prolog**: You can declare the namespace for the customer order if you want to use the contents of the order as part of the recognition rule or you can omit the declaration if you only want to check the incoming customer order namespace. For example:

  ```xml
declare namespace im="http://xmlns.oracle.com/InputMessage";
  ```

- **Body**: You must match the namespace you want to select for order processing with the namespace of the incoming customer order. For example, the following expression retrieves the namespace URI from the incoming customer order (**fn:namespace-uri(.)**) and compares it with this URI: 'http://xmlns.oracle.com/InputMessage':

  ```xml
  fn:namespace-uri(.) = 'http://xmlns.oracle.com/InputMessage'
  ```

If you have declared a namespace in the prolog, you can also check to see if specific values exist in the order. For example, you can use the **fn:exists** function to check that an element exists. Or you can use a comparison expression such as **=`** (equal to) or **!=** (not equal to) to compare a value in the customer order with a value in the XQuery.

**Tip**: Recognition rules are global entities within OSM, meaning that they can apply to any CreateOrder operation. Configure the relevancy settings and the recognition rule carefully to avoid having an incoming customer order recognized by a recognition rule that you do not intend. For more information about relevancy, see **OSM Concepts**.

For example, in a simple scenario, the XQuery is based on a namespace:

```xml
fn:namespace-uri(.) = 'http://xmlns.oracle.com/InputMessage'
```
The input message XML file includes the following line, which matches the namespace specified in the recognition rule:

```xml
<im:order xmlns:im="http://xmlns.oracle.com/InputMessage"
```

The XQuery expression returns a Boolean expression, for example, `fn:true()` or `fn:false()`

The following example searches in a specific type of order:

```
fn:namespace-uri(.) = 'http://xmlns.oracle.com/communications/sce/dictionary/CentralOMManagedServices-Orchestration/CustomerSalesOrder'
```

In a more complicated scenario, you might create an XQuery expression that looks for a specific namespace and also interrogates the data within the incoming customer order. The following example shows a recognition rule that recognizes an order based on the following criteria:

- Namespace
- Value of the `typeCode` data element in the incoming customer order. In this case, the value must be OSM-BDB. This indicates an OSM business-to-business order.
- The value of the `FulfillmentModeCode` data element in the incoming customer order. In this case, the value can be DELIVER, CANCEL, or TSQ.

```xquery
declare namespace provord=";http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/ProvisioningOrder/V1";
declare namespace corecom=";http://xmlns.oracle.com/EnterpriseObjects/Core/Common/V2";
fn:namespace-uri(.) = 'http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/ProvisioningOrder/V1'
and
and
../provord:ProcessProvisioningOrderEBM/provord:DataArea/provord:ProcessProvisioningOrder/provord:TypeCode/text() = 'OSM-BDB'
and
{
  or
  or
}
```

For more information about order recognition rules see *OSM Concepts*.

### About Validation Rule XQuery Expressions

This topic describes how to use the Order Recognition Rule editor Validation Rule area XQuery tab to write an expression that specifies nodes in the incoming customer order that must evaluate to true to accept the customer order into the system. The XQuery has the following characteristics:

- Context: The input document for the Validation Rule XQuery is the customer order. For more information about typical customer order structures, see *OSM Concepts*.
Prolog: The input document for the Validation Rule XQuery is the customer order. You can declare the customer order namespace in the XQuery prolog. For example:

```xquery
declare namespace im="http://xmlns.oracle.com/InputMessage";
```

Body: The validation rule must specify customer order parameters or parameter values to evaluate to true for the validation to be successful. If the validation fails, the expression should return an error message.

In addition, if the Validation Rule fails, OSM automatically creates the order and sets the order state to Failed. The inbound message and validation failure output are attached to the order for reference. You can display and manage the order failure in the Order Management Web client.

The following sample XQuery checks for the existence of a sender ID:

```xquery
if (fn:exists(./header/c:Sender/c:ID) and ./header/c:Sender/c:ID != '')
then (true())
else concat("SEVERE", "Message Header should contain Sender ID", header/Sender/ID")
```

The following sample XQuery checks for correct values in the typeCode data element in the incoming customer order:

```xquery
if (fn:exists($orderLine/im:ItemReference/im:TypeCode)
and
$orderLine/im:ItemReference/im:TypeCode != '')
then
{
    if ($orderLine/im:ItemReference/im:TypeCode = "PRODUCT" or
    $orderLine/im:ItemReference/im:TypeCode = "OFFER" or
    $orderLine/im:ItemReference/im:TypeCode = "BUNDLE") then ()
    else
        local:reportIssue("ERROR", "Product Type should be one of: PRODUCT, OFFER, BUNDLE", $lineNum, "ProcessProvisioningOrderEBM/DataArea/ProcessProvisioningOrder/ProvisioningOrderLine/ItemReference/TypeCode")
}
```

Given this XQuery sample, the following part of a customer order would evaluate to true because the typeCode element value is BUNDLE.

```xml
<!-- FIXED BUNDLE - BUNDLE -->
<im:salesOrderLine>
  <im:lineId>2</im:lineId>
  <im:promotionalSalesOrderLineReference>1</im:promotionalSalesOrderLineReference>
  <im:serviceId></im:serviceId>
  <im:requestedDeliveryDate>2001-12-31T12:00:00</im:requestedDeliveryDate>
  <im:itemReference>
    <im:name>Fixed Bundle</im:name>
    <im:typeCode>BUNDLE</im:typeCode>
    <im:specificationGroup />
  </im:itemReference>
</im:salesOrderLine>
```

For more information about validation rules see OSM Concepts.

**About Order Priority XQuery Expressions**

This topic describes how to use the Order Recognition Rule editor Order Priority area XQuery tab to write an expression that specifies an element value in the incoming
About Order Reference XQuery Expressions

This topic describes how to use the Order Recognition Rule editor Order Reference area XQuery tab to write an expression that specifies an element value in the incoming customer order that identifies the order reference. The XQuery has the following characteristics:

- Context: The input document for the Order Reference XQuery is the customer order. For more information about typical customer order structures, see OSM Concepts.
- Prolog: You can declare the customer order namespace in the XQuery prolog. For example:

```xquery
declare namespace im="http://xmlns.oracle.com/InputMessage";
```

- Body: The Order Reference body must specify the node that contains the order reference value.

The following example shows a transformation rule XQuery expression that retrieves the order reference number (as a string) from the `numSalesOrder` field in the incoming customer order:

```xquery
declare namespace im="http://xmlns.oracle.com/InputMessage";
let $order := ../im:order
return $order/im:numSalesOrder/text()
```

For more information about order reference, see OSM Concepts.

About Order Data Rule XQuery Expressions

This topic describes how to use the Order Recognition Rule editor Order Data Rule area XQuery tab to write an expression that specifies nodes in the incoming customer order that must be used in the creation task. The XQuery has the following characteristics:

- Context: The input document for the Order Data Rule XQuery is the customer order. For more information about typical customer order structures, see OSM Concepts.
- Prolog: You can declare the customer order namespace in the XQuery prolog. For example:
Body: The Order Data Rule body must map the customer order element values to the corresponding creation task Task Data values.

The following example shows the fields in an incoming customer order:

```xml
<im:customerAddress>
  <im:locationType>Street</im:locationType>
  <im:nameLocation>Jangadeiros</im:nameLocation>
  <im:number>48</im:number>
  <im:typeCompl>floor</im:typeCompl>
  <im:numCompl>6</im:numCompl>
  <im:district>Ipanema</im:district>
  <im:codeLocation>5000</im:codeLocation>
  <im:city>Rio de Janeiro</im:city>
  <im:state>RJ</im:state>
  <im:referencePoint>Gen. Osorio Square</im:referencePoint>
  <im:areaCode>22420-010</im:areaCode>
  <im:typeAddress>Building</im:typeAddress>
</im:customerAddress>
```

Following is a sample order data in a creation task. In the example, the following data is contained in a CustomerDetails element:

- locationType
- nameLocation
- number
- typeCompl
- numCompl
- district
- codeLocation
- city
- state
- referencePoint
- areaCode
- typeAddress

The following XQuery expression specifies a variable for the location of the customerAddress node in the customer order that can then be used to map customerAddress child element values to CustomerDetails task data elements:

```xml
let $details := $customer/mes:customerAddress
```

The following XQuery expression performs this mapping:

```xml
return
<root>
  <CustomerDetails>
    <locationType>{$details/im:locationType/text()}</locationType>
    <nameLocation>{$details/im:nameLocation/text()}</nameLocation>
    <number>{$details/im:number/text()}</number>
    <typeCompl>{$details/im:typeCompl/text()}</typeCompl>
    <numCompl>{$details/im:numCompl/text()}</numCompl>
    <district>{$details/im:district/text()}</district>
  </CustomerDetails>
</root>
```
In the following example, the XQuery expression returns the \_root\ portion of the order. The ControlData portion of the order is populated by the system during the generation of the orchestration plan.

```
declare namespace cso="http://xmlns.oracle.com/communications/sce/dictionary/CentralOMManagedServices-Orchestration/CustomerSalesOrder";
let $customer := //cso:CustomerAccount
return
\_root
<OrderHeader>
<AccountIdentifier>{$customer/cso:AccountID/text()}</AccountIdentifier>
</OrderHeader>
</\_root>
```

For more information about order data rules, see OSM Concepts.

**Decomposition XQuery Expressions**

This topic includes information about order recognition rule XQuery expressions related to order decomposition:

- About Orchestration Sequence XQuery Expressions
- About Order Item Specification XQuery Expressions
- About Fulfillment Pattern Order Component XQuery Expressions
- About Decomposition Rule Condition XQuery Expressions
- About Component Specification Custom Component ID XQuery Expressions
- About Component Specification Duration XQuery Expressions
- About Fulfillment Pattern Duration XQuery Expressions (deprecated)
- About Fulfillment Pattern Component Duration XQuery Expressions

**About Orchestration Sequence XQuery Expressions**

The Orchestration Sequence editor provides the following areas to define XQuery expressions related to order decomposition:

- About Order Sequence Order Item Selector XQuery Expressions
- About Order Sequence Fulfillment Mode XQuery Expressions

**About Order Sequence Order Item Selector XQuery Expressions**

This topic describes how to use the Orchestration Sequence editor Order Item Selector area XQuery tab to write an expression that specifies which node-set to use from the customer order as order items and has the following characteristics:
Decomposition XQuery Expressions

- **Context:** The input document for the Order Item Selector XQuery is the customer order. For more information about typical customer order structures, see OSM Concepts.
- **Prolog:** You can declare the customer order namespace in the XQuery prolog.
- **Body:** The XQuery body must specify the customer order node-sets that OSM then uses as order items.

The following example shows an order item selector XQuery where the `<salesOrderLine>` node-set is specified. OSM can now use the data in the `<salesOrderLine>` node-set in the incoming customer order in the order items. There can only be one node-set selected per sequence.

```
declare namespace im="http://xmlns.oracle.com/InputMessage";
//im:salesOrderLine
```

### About Order Sequence Fulfillment Mode XQuery Expressions

This topic describes how to use the Orchestration Sequence editor Fulfillment Mode area XQuery tab to write an expression that specifies the fulfillment mode for the orchestration sequence from a customer order element and has the following characteristics:

- **Context:** The input document for the Fulfillment Mode Expression area XQuery is the customer order. For more information about typical customer order structures, see OSM Concepts.
- **Prolog:** The input document for the Fulfillment Mode Expression area XQuery is the incoming customer order. You must declare the customer order namespace in the XQuery prolog.
- **Body:** The XQuery body must specify the fulfillment mode.

Typically, the fulfillment mode is specified in the order header. For example:

```
<im:FulfillmentModeCode>Deliver</im:FulfillmentModeCode>
```

In the following example, the XQuery looks in the incoming customer order (SalesOrder) for the `<FulfillmentModeCode>` element. It returns the text contained in that element.

```
declare namespace im='http://xmlns.oracle.com/InputMessage';
<osm:fulfillmentMode name="{fn:normalize-space(//im:SalesOrder/im:DataArea/im:FulfillmentModeCode/text())}"/>
```

This is the XML in the incoming customer order:

```
<im:FulfillmentModeCode>Deliver</im:FulfillmentModeCode>
```

In this case, the XQuery returns Deliver.

### About Order Item Specification XQuery Expressions

The Order Item Specification editor provides the following areas to define XQuery expressions related to order decomposition:

- About Order Item Specification Order Item Property XQuery Expressions
- About XQuery Expressions for Mapping Product Specifications and Fulfillment Patterns
- About Order Item Specification Order Item Hierarchy XQuery Expressions
About Order Item Specification Condition XQuery Expressions

About Order Item Specification Order Item Property XQuery Expressions
This topic describes how to use the Order Item Specification editor, Order Item Properties tab, Property Expression area, XQuery tab to write an expression that specifies order item properties based on the input context. These expressions have the following characteristics:

- Context: The Property Expression area XQuery input document is a node from the node-set returned by the order item selector (see “About Order Sequence Order Item Selector XQuery Expressions”). OSM runs every order item Property Expression area XQuery against each node (starting with the first and ending with the last node) in the node-set returned by the order item selector.

- Prolog: You can declare the following variables within the prolog to access additional context information:
  - The $inputDoc variable can be declared in the prolog of an OSM XQuery to provide access to the original input customer order. This external function can be useful if you need to generate order item properties based on elements outside of the order item node-set defined in the order item selector. The format for declaring this variable in the XQuery prolog is:
    
    ```
    declare variable $inputDoc as document-node() external;
    ```

    You can then access this variable within the XQuery body. For example, the following XQuery body uses $inputDoc to define the ItemReferenceName value:

    ```
    let $inputOrderData:= $inputDoc/GetOrder.Response/_root
    fn:normalize-space(cso:ItemReferenceName/text())
    ```

    For more information about typical customer order structures, see OSM Concepts.

  - The $salesOrderLines variable can be used in an OSM XQuery to provide access to original order item node-set before it is selected by the orchestration sequence’s order item selector. This can be useful if the order item selector XQuery changes the selected order item node-set (for example, by rearranging the order of the elements). The format for declaring this variable in the XQuery prolog is:

    ```
    declare variable $salesOrderLines as document-node() external;
    ```

    You can access this variable within the XQuery body. For more information about typical customer order structures, see OSM Concepts.

- Body: The XQuery body must specify the order item element that provides the values for each order item property you define.

After these XQuery expressions have run against an order item, the order item and the order item properties become internally accessible as an XQuery context for other OSM entities. For example,

```xml
<osm:orderItem
xmlns:osm="http://xmlns.oracle.com/communications/ordermanagement/model" id="1288881040699">
  <osm:name>Commercial Fixed Service [Add]</osm:name>
  <osm:orderItemSpec xmlns="http://xmlns.oracle.com/communications/ordermanagement/model">
    <osm:name>CustomerOrderItemSpecification</osm:name>
    <osm:namespace>
      http://oracle.communications.ordermanagement.unsupported.centralom
    </osm:namespace>
  </osm:orderItemSpec>
</osm:orderItem>
```

XQuery Examples   A-11
The following examples show some ways to map data in an incoming customer order to an order item property. The current context is a single node from salesOrderLines, which is one of the nodes returned by executing the orchestration sequence order item selector against the input message (see "About Order Sequence Order Item Selector XQuery Expressions").

Order management personnel need to know what the requested delivery date is for order items. Adding the date to the order item allows the order management personnel to see it in the OSM Web clients. In addition, OSM needs the requested delivery date to calculate the order start date.

To retrieve the requested delivery data for an order item, OSM looks in the incoming customer order for the `<requestedDeliveryDate>` element:

```xml
<im:requestedDeliveryDate>2001-12-31T12:00:00</im:requestedDeliveryDate>
```

The definition of the `<requestedDeliveryDate>` order item property includes the following XQuery, which returns the text of the `<requestedDeliveryDate>` element:

```xml
declare namespace im="http://xmlns.oracle.com/InputMessage";
fn:normalize-space(im:requestedDeliveryDate/text())
```

Order management personnel need to identify order items in the OSM Web clients. The `<lineItemName>` order item property includes the following XQuery:

```xml
declare namespace im="http://xmlns.oracle.com/InputMessage";
fn:normalize-space(fn:concat(im:itemReference/im:name/text(),' [',im:serviceActionCode/text(),' ])')
```

This XQuery looks for two elements, `<name>` and `<serviceActionCode>`:

```xml
<im:name>Fixed Caller ID</im:name>
```

It then concatenates the text retrieved from the two elements to form the order item name, in this case Fixed Caller ID [Add].
Order management personnel need to identify the products or product specification from the customer order so that order items can be mapped to fulfillment patterns (see "About XQuery Expressions for Mapping Product Specifications and Fulfillment Patterns"). The following example shows the product specification data in the message, contained in the `<primaryClassificationCode>` element:

```
<im:primaryClassificationCode>Mobile Service Feature Class</im:primaryClassificationCode>
```

The `productClass` order item property uses the following XQuery expression to get the data:

```
declare namespace im='http://xmlns.oracle.com/InputMessage';
fn:normalize-space(im:itemReference/im:primaryClassificationCode/text())
```

About XQuery Expressions for Mapping Product Specifications and Fulfillment Patterns

The order item property specified in the Fulfillment Pattern Mapping Property field for the order item must map to an existing OSM fulfillment pattern entity. The value could be contained in a customer order, but more often, it is derived from other customer order parameter. This property is mandatory.

The construction of the fulfillment pattern mapping order item property follows the same rules as other order item property XQuery expressions. See "About Order Item Specification Order Item Property XQuery Expressions" for more information about the XQuery context, prolog, and body.

The following describes a common scenario for deriving fulfillment patterns from product or product specification data contained in an order. In other scenarios, the mapping from product or product specification to fulfillment pattern might be simpler; or, there might be cases where some order line items have no product specification, in which case the product specification can be derived from the context of the order item.

You typically create conceptual model products in your OSM system by importing them. (See OSM Concepts for more information.) When you import products, Design Studio creates the `productClassMapping.xml` and `productSpecMapping.xml` files. These files contain mappings between conceptual model products and OSM product specifications and fulfillment patterns. The `productClassMapping.xml` file is provided for backward compatibility, so in this topic it is assumed that you are using the `productSpecMapping.xml` file. These files are created in one of the following directories:

- If you have specified a value for the Product Specification Mapping Folder field of the Orchestration Preferences in Eclipse, it will create the two files in the directory specified.
- If no value is specified for that field, OSM will create the `productClassMapping.xml` file in the `resources/productClassMapping` directory and the `productSpecMapping.xml` file in the `resources/productSpecMapping` directory.

You can retrieve this mapping data from one of these files by creating a data instance provider that can be referenced from an XQuery expression body using a data instance behavior.
Decomposition XQuery Expressions

Note: The element names are not the same between the productClassMapping.xml and productSpecMapping.xml files. Ensure that you are using the correct element names for the file you are referencing. The names in this topic are correct for the productSpecMapping.xml file.

For example, the following XQuery creates the $productSpecMap variable that references the data instance that points to the productSpecMapping.xml file:

```xquery
let $productSpecMap := vf:instance('ProductSpecMap')
```

The following code creates a variable that references the product specification value from the customer order. For example:

```xquery
let $productSpecName := fn:normalize-space(im:itemReference/im:primaryClassificationCode/text())
```

You can now create an expression that matches the product specification from the order with the product specification contained in the productSpecMapping.xml file and returns the fulfillment pattern associated with it or else defaults to the Non.Service.Offer fulfillment pattern. For example:

```xquery
return if ($productSpecName != '')
  then fn:normalize-space($productSpecMap/productSpec
    [fn:lower-case(@name)=fn:lower-case($productSpecName)]/productClass/text())
  else 'Non.Service.Offer'
```

In the following example, OSM retrieves the product specification Mobile Service Feature Class from the incoming customer order. OSM uses the order item property specified in the Fulfillment Pattern Mapping Property field for the order item to map the product specification to a fulfillment pattern.

The order item property specified in the Fulfillment Pattern Mapping Property field for the order item includes the following XQuery expression:

```xquery
declare namespace im="http://xmlns.oracle.com/InputMessage";
(: Use the ProductSpecMap data instance behavior to retrieve the data in the productSpecMapping.xml file: :)  
let $productSpecMap := vf:instance('ProductSpecMap')
let $productSpecName := fn:normalize-space(im:itemReference/im:primaryClassificationCode/text())
return if ($productSpecName != '')
  then fn:normalize-space($productSpecMap/productSpec
    [fn:lower-case(@name)=fn:lower-case($productSpecName)]/productClass/text())
  else 'Non.Service.Offer'
```

The productSpecMapping.xml file includes the <productSpec> element, that maps the Mobile Service Feature Class product specification to the Service.Mobile fulfillment pattern:

```xml
<productSpec name="Mobile Service Feature Class"
  cartridge="OsmCentralOMExample-ProductSpecs">
  <fulfillmentPattern<Service.Mobile</fulfillmentPattern>
```

Note: The element names are not the same between the productClassMapping.xml and productSpecMapping.xml files. Ensure that you are using the correct element names for the file you are referencing. The names in this topic are correct for the productSpecMapping.xml file.
To summarize, to map an order line item in an incoming customer order to a fulfillment pattern, you configure the following:

- In the order item specification:
  - A property that retrieves the conceptual model product or the OSM product specification from the incoming customer order.
  - The order item property specified in the **Fulfillment Pattern Mapping Property** field, that maps the product or product specification to the fulfillment pattern. To do so, OSM uses the ProductClassMap data instance behavior.
- The ProductSpecMap data instance behavior (and the data provider that supports it), that retrieves data from the `productSpecMapping.xml` file.
- The `productSpecMapping.xml` file used by the ProductClassMap data instance behavior, that maps products and product specifications to fulfillment patterns.

When you update your product catalog, you might need to add new fulfillment patterns. In that case, you need to:

- Create new fulfillment patterns and conceptual model products, if necessary.
- Add mappings to the `productSpecMapping.xml` file.

You do not need to change the order item specification or the data instance behavior.

**About Order Item Specification Order Item Hierarchy XQuery Expressions**

This topic describes how to use the Order Item Specification editor **Order Item Hierarchies** tab, Key Expression and Parent Key Expression areas, **XQuery** subtabs to write expressions that specify the relative hierarchy of order items, in the same order or between different orders, based on an order item value, such as `lineId` and `parentLineId` and has the following characteristics:

- **Context**: The Key Expression and Parent Key Expression area XQuery input document is the order item. Specifically order item properties that indicate the relative hierarchy, such as order item `lineId` and `parentLineId` properties. For example:

```xml
<osm:orderItem
  xmlns:osm="http://xmlns.oracle.com/communications/ordermanagement/model"
  id="1288881040699">
  ....
  <osm:properties
    xmlns:im="http://oracle.communications.ordermanagement.unsupported.centralom">
    <im:typeCode>PRODUCT</im:typeCode>
    <im:parentLineId>3</im:parentLineId>
    <im:requestedDeliveryDate>2013-06-31T12:00:00</im:requestedDeliveryDate>
    <im:lineItemName>Commercial Fixed Service [Add]</im:lineItemName>
    <im:lineId>4</im:lineId>
    <im:ServiceActionCode>UPDATE</im:ServiceActionCode>
    <im:productClass>Fixed Service Plan Class</im:productClass>
    <im:serviceId>5521313131313</im:serviceId>
    <im:lineItemPayload> [34 lines]
    <im:region>Sao Paulo</im:region>
  </osm:properties>
</osm:orderItem>
```
Prolog: You can declare the order item specification namespace and the OSM namespace in the XQuery prolog. For example:

```
declare namespace osm="http://xmlns.oracle.com/communications/ordermanagement/model";
declare namespace im="http://oracle.communications.ordermanagement.unsupported.centralom";
```

You can declare the OrchestrationXQueryFunctions class in the prolog to use the ancestors method that returns the current node and all ancestors of the current node based on the specified hierarchy definition. This method can be useful when creating dependencies between order items based on hierarchy. For example:

```
declare namespace osmfn = "java:oracle.communications.ordermanagement.orchestration.generation.OrchestrationXQueryFunctions";
```

See "OSM XQuery Functions" for more information about the OrchestrationXQueryFunctions class. See OSM Concepts for an example of how the ancestors method is used.

Body: The XQuery body must specify an order item property defined in the order item specification.

For example, for the Key Expression, you can identify a unique key for each order item, typically the order item line ID:

```
fn:normalize-space(osm:properties/im:LineId/text())
```

For example, for the Parent Key Expression, you can identify a parent order line item, typically the line ID for the parent order line item:

```
fn:normalize-space(osm:properties/im:parentLineId/text())
```

In the following example, the key expression uses the parent order line item’s <lineId> element from the order item property customer order:

```
declare namespace im="http://oracle.communications.ordermanagement.unsupported.centralom";
declare namespace osm="http://xmlns.oracle.com/communications/ordermanagement/model";
fn:normalize-space(osm:properties/im:lineId/text())
```

The parent key expression uses the child order line item’s <parentLineId> element from the incoming customer order:

```
declare namespace im="http://oracle.communications.ordermanagement.unsupported.centralom";
declare namespace osm="http://xmlns.oracle.com/communications/ordermanagement/model";
fn:normalize-space(osm:properties/im:parentLineId/text())
```

About Order Item Specification Condition XQuery Expressions

This topic describes how to use the Order Item Specification editor Orchestration Conditions tab, Condition Expression area, XQuery subtab to write expressions that specifies an order item property value as a condition that you can then use in an order decomposition rule or in a fulfillment pattern to determine whether an order item gets included in an order component. The XQuery for the condition has the following characteristics:

- Context: The Condition Expression area XQuery input document is the order item properties you want to use as conditions. For example, the following order item
contains the region and the ServiceActionCode order item properties, that could be associated to conditions:

```xml
<osm:orderItem
  xmlns:osm="http://xmlns.oracle.com/communications/ordermanagement/model"
  id="1288881040699">
  ....
  <osm:properties
    xmlns:im="http://oracle.communications.ordermanagement.unsupported.centralom">
    <im:typeCode>PRODUCT</im:typeCode>
    <im:parentLineId>3</im:parentLineId>
    <im:requestedDeliveryDate>2013-06-31T12:00:00</im:requestedDeliveryDate>
    <im:lineItemName>Commercial Fixed Service [Add]</im:lineItemName>
    <im:lineId>4</im:lineId>
    <im:ServiceActionCode>UPDATE</im:ServiceActionCode>
    <im:productClass>Fixed Service Plan Class</im:productClass>
    <im:serviceId>552131313131</im:serviceId>
    <im:lineItemPayload> [34 lines]
    <im:region>Sao Paulo</im:region>
  </osm:properties>
</osm:orderItem>
```


- **Prolog:** You can declare the order item specification namespace and the OSM namespace in the XQuery prolog. For example:

  ```xml
  declare namespace osm="http://xmlns.oracle.com/communications/ordermanagement/model";
  declare namespace im="http://oracle.communications.ordermanagement.unsupported.centralom";
  ```

- **Body:** The XQuery body must evaluate an order item property defined in the order item specification. These order item properties are available from the OSM namespace using the properties element. For example, the following expression evaluates to true if the value of region is anything other than Sao Paulo and the order item gets included in the order component. If the region were set to Sao Paulo, then the order item would not be included in the order component.

  ```xml
  fn:not(fn:normalize-space(osm:properties/im:region/text())='Sao Paulo')
  ```

  Another condition could be created that would only evaluate to true if the value of region was set to Sao Paulo. In this case, the order item would only be included in the order component if the region were set to Sao Paulo.

### About Fulfillment Pattern Order Component XQuery Expressions

The Fulfillment Pattern editor provides the following areas to define XQuery expressions related to order decomposition:

- About Fulfillment Pattern Order Component Condition XQuery Expressions
- About Associating Order Items Using Property Correlations XQuery Expressions
Note: The XQuery expressions discussed in this chapter also apply to the Orchestration Dependency editor.

About Fulfillment Pattern Order Component Condition XQuery Expressions
This topic describes how to use the Fulfillment Pattern editor, Orchestration Plan tab, Order Components subtab, Conditions subtab XQuery subtab to write an expression that specifies whether to include or exclude an order item from an order component. You can create a new fulfillment pattern from the Fulfillment Pattern editor or select from conditions created in the Order Item Specification. See “About Order Item Specification Condition XQuery Expressions” for more information about the context, prolog, and body of condition XQuery expressions.

The following example XQuery expression only evaluates to true if the value of ServiceActionCode is not NONE or UPDATE. For example, if the value of ServiceActionCode were ADD, then the order item would be included in the order component.

```
fn:boolean
(
(osm:properties/im:ServiceActionCode/text()!='NONE' and
osm:properties/im:ServiceActionCode/text()!='UPDATE') or
(
```

About Associating Order Items Using Property Correlations XQuery Expressions
This topic describes how to use the Fulfillment Pattern editor, Orchestration Plan tab, Order Components subtab, Order Item Association subtab, Property Correlation selection, XQuery subtab to write an expression that associates order items to order components that are not assigned by their fulfillment pattern. These order item associations are typically required when external systems need a specific context for an order item and includes the following characteristics:

- Context: The Order Item Association subtab XQuery input documents are multiple order items in the order after decomposition contained in the fromOrderComponent element and the entire set of order items included in the order contained in the toOrderComponent element. You can make an XQuery association based on the contents of these order items that create an association between the unique order item IDs. For example:

```
<fromOrderComponent xmlns="">
<osm:orderItem
xmlns:osm="http://xmlns.oracle.com/communications/ordermanagement/model"
id="1234">
<osm:name>Speed By Demand [Add]</osm:name>
......
<osm:properties
xmlns:im="http://oracle.communications.ordermanagement.unsupported.centralom">
<im:typeCode>PRODUCT</im:typeCode>
<im:parentLineId>3</im:parentLineId>
<im:requestedDeliveryDate>2013-06-31T12:00:00</im:requestedDeliveryDate>
<im:lineItemName>Commercial Fixed Service [Add]</im:lineItemName>
<im:lineId>4</im:lineId>
<im:SiteID>10</im:SiteID>
<im:ServiceActionCode>UPDATE</im:ServiceActionCode>
<im:productClass>Speed by Demand class</im:productClass>
<im:serviceId>552131313131</im:serviceId>
```
Decomposition XQuery Expressions

XQuery Examples

Prolog: You can declare the order item namespace and the OSM namespace in the XQuery prolog. For example:

```xquery
declare namespace osm="http://xmlns.oracle.com/communications/ordermanagement/model";
declare namespace im="http://oracle.communications.ordermanagement.unsupported.centralom";
```

Body: The XQuery body must specify a dependency between the order item and the associated order item using something similar to the following syntax:

```xquery
let $fromItem := osm:fromOrderComponent/osm:orderItem[osm:name/text()='Speed By Demand [Add']]
let $toItem := osm:toOrderComponent/osm:orderItem[osm:name/text()='Broadband Bundle [Add]' and osm:properties/im:SiteID/text() = $fromItem/osm:properties/im:SiteID/text()]
```
Decomposition XQuery Expressions

return
<osm:dependency fromOrderItemId='{$fromItem/@id}'
  toOrderItemId='{$toItem/@id}'/>

where

- **osm:fromOrderComponent**: Returns the set of order items included in the
  order component after the decomposition phase.
- **osm:toOrderComponent**: Returns the entire set of order items included in the
  order.
- **osm:orderItem**: These are the order items in the fromOrderComponent or
  toOrderComponent categories.
- **osm:dependency fromOrderItemId='{$fromItem/@id}' toOrderItemId='{$toItem/@id}'**: The output of the
  XQuery specifying the source order item ID for the association.
- **toItem='{$childOrderItem/@id}'**: The output of the XQuery specifying the
  target order item ID for the association.

Given the sample provided in the context bullet, this XQuery would return the following association:

<osm:dependency fromOrderItemId='1234' toOrderItemId='5678'/>

The following example shows an XQuery that associates all child order items with
their parent items. (See **OSM Concepts** for more information.) The output of the
XQuery expression returns a node-set of

<osm:dependency fromOrderItemId='{$fromOrderItem/@id}'
  toOrderItemId='{$toOrderItem/@id}'/>

where item IDs are the @id attribute of the order item.

declare namespace osm="http://xmlns.oracle.com/communications/ordermanagement/model";
declare namespace prop="http://oracle.communications.ordermanagement.unsupported.centralom";

(:   $fromOrderItemList contains all order items in the selected order component:   :) for $fromOrderItem in $fromOrderItemList
let $fromOrderItemList := osm:fromOrderComponent/osm:orderItem
(:    $childOrderItems contains all children for the current $fromOrderItem:    :) let $childOrderItems := osm:toOrderComponent/osm:orderItem/osm:properties
[prop:ParentLineID/text() = $fromOrderItem/osm:properties/prop:LineID/text()]
 (:    Returns the association between all parents and their children:    :) for $childOrderItem in $childOrderItems
return
<osm:dependency fromOrderItemId='{$fromOrderItem/@id}'
  toOrderItemId='{$childOrderItem/@id}'/>

About Decomposition Rule Condition XQuery Expressions

This topic describes how to use the Decomposition Rule editor, **Conditions** tab,
**Conditions Details** subtab, **XQuery** subtab to write an expression that associates a
condition with a decomposition rule. You can create the condition in the order item
specification and select them in the decomposition rule, or you can create them
directly in the decomposition rule. See "**About Order Item Specification Condition XQuery Expressions**" for more information about the context, prolog, and body of
condition XQuery expressions.

The following is an example of two decomposition rules, each having a condition set
that determines whether an order item is included in the target order component or
not. In this example:

- The decomposition rule that targets the target system order component for region
  1 has the following decomposition condition:
The decomposition rule that targets the a target system order component for region 2 has the following decomposition condition:

**isOtherRegion**

The XQuery for the **isRegion1** decomposition rule condition is:

```xquery
declare namespace im="http://oracle.communications.ordermanagement.unsupported.centralom";
declare namespace osm="http://xmlns.oracle.com/communications/ordermanagement/model";
fn:normalize-space(osm:properties/im:region/text())='Toronto')
```

This condition specifies the value of the **region** order item property. If the value is **Toronto**, the decomposition rule condition is true, and the order item is included in the region 1 target system order component.

The XQuery for the **isOtherRegion** decomposition rule condition is:

```xquery
declare namespace im="http://oracle.communications.ordermanagement.unsupported.centralom";
declare namespace osm="http://xmlns.oracle.com/communications/ordermanagement/model";
fn:not(fn:normalize-space(osm:properties/im:region/text())='Toronto')
```

This condition also specifies the value of the **region** order item property, but evaluates to true only if the value is not **Toronto**. All order items that have any other value are included in the region 2 target system order component.

The following example includes a variation on the **isRegion1** decomposition rule that specifies that all the order items from the source order component to the target order component that have at least one order item with a **region** property of **Toronto** are included in the order component. Otherwise, if the condition evaluates to false then none of the order items in **fromOrderComponent** are included in the resulting order component.

```xquery
declare namespace im="http://oracle.communications.centralom";
declare namespace osm="http://xmlns.oracle.com/communications/ordermanagement/model";
fn:exists(osm:fromOrderComponent/osm:orderItem[fn:normalize-space(osm:properties/im:Region/text())='Toronto'])
```

For some functions, there is only one target system in the topology. For example, if you have only one collections system in the topology, you will have one dependency rule that uses a simple mapping from the source collections function order component to the collections target system order component, and no decomposition condition is necessary.

### About Component Specification Custom Component ID XQuery Expressions

This topic describes how to use the Order Component Specification editor, **Component ID** tab, Component ID area, **XQuery** subtab to write an expression that specifies a custom component ID for an order component. These custom component IDs are typically required when the default component IDs are not sufficiently specific (see **OSM Concepts** for more information about the default component ID). The Component ID XQuery includes the following characteristics:

- **Context**: The **Component ID** tab XQuery input document is the order item and the order item properties you want to use to create a custom component ID with. For example, the following order item contains the **SiteID** and
**requestedDeliveryDate** order item properties. In a simple scenario, you can use this element to group all order items that share the same **SiteID** value and further delineate groups based on **requestedDeliveryDate** date range.

```xml
<osm:orderItem
xmlns:osm="http://xmlns.oracle.com/communications/ordermanagement/model"
    id="1288881040699">
    .....  
    <osm:properties
xmlns:im="http://oracle.communications.ordermanagement.unsupported.centralom">
        <im:typeCode>Bundle</im:typeCode>
        <im:parentLineId>3</im:parentLineId>
        <im:requestedDeliveryDate>2013-06-31T12:00:00</im:requestedDeliveryDate>
        <im:lineItemName>Commercial Fixed Service [Add]</im:lineItemName>
        <im:lineId>4</im:lineId>
        <im:SiteID>10</im:SiteID>
        <im:ServiceActionCode>UPDATE</im:ServiceActionCode>
        <im:productClass>Fixed Service Plan Class</im:productClass>
        <im:serviceId>552131313131</im:serviceId>
        <im:lineItemPayload> [34 lines]
        <im:region>Sao Paulo</im:region>
    <osm:properties>
</osm:orderItem>
```

- **Prolog:** You can declare the order item namespace and the OSM namespace in the XQuery prolog. In more complicated XQuery expressions, you can also use the **OrchestrationXQueryFunctions** OSM Java package to specify component IDs based on order item hierarchies, order item requested delivery date, order component duration, order component minimum duration separation, or a combination of some or all of them. For example:

```xml
declare namespace osm="http://xmlns.oracle.com/communications/ordermanagement/model";
declare namespace im="http://oracle.communications.ordermanagement.unsupported.centralom";
declare namespace osmfn = "java:oracle.communications.ordermanagement.orchestration.generation.OrchestrationXQueryFunctions";
```

See “**OSM XQuery Functions**” for more information about the **OrchestrationXQueryFunctions** class.

- **Body:** The body must return a string. Every order item that ends with the same string gets included in the order component. For example, if you wanted to group all order items based on the **SiteID** value, you could specify the following XQuery:

```xml
return osm:properties/im:SiteID/text()
```

The following topics describe OSM Java package methods.

For more information about how the OrchestrationXQueryFunctions are used in custom Component ID XQuery expressions and for more complicated custom group ID generation scenarios that use Orchestration XQueryFunction, see the following topics:

- **Custom Order Component IDs Based on Hierarchy**
- **Custom Component IDs Based on Requested Delivery Date and Duration**
Custom Component IDs by Duration and Minimum Separation Duration

Combining Order Item Hierarchy with Duration-Based Groupings

Custom Order Component IDs Based on Hierarchy

A more common scenario where custom order component IDs can be used is when you need additional groupings of order components at the granularity level. For example, three levels of decomposition from Function, System, to Bundle, results in the following component IDs:

- BillingFunction
- BillingFunction.BillingSystem

If you had order items in the Bundle order components that were part of different bundles that go to different the billing system, you would need to separate each order item bundle into different bundle order component. A component ID for such a scenario could look like this:

- For billing system 1: BillingFunction.BillingSystem.Bundle.2/BundleGranularity
- For billing system 2: BillingFunction.BillingSystem.Bundle.6/BundleGranularity

To create custom component IDs for this scenario, you could use the following order item properties:

- **typeCode**: This property specifies if the order line item is an offer, bundle, or product. This element also defines the product hierarchy of the order line items. For example:

  OFFER
  BUNDLE
  PRODUCT

- **lineId** and **parentLineId**: These properties specify the hierarchical relationship between the bundle and product order line items. You can create separate component IDs for every order item bundle and associate all product order items with their corresponding bundle component ID. To identify all ancestor order items that may be a bundle, you can use the XQuery `ancestors` function, as explained later.

For example, the following four order items include two bundles and two associated products. These order items have the following characteristics:

- **Order Item 1** includes:
  - **typeCode**: BUNDLE
  - **lineId**: 2
  - **parentLineId**: 1 (for example, an order item with an OFFER typeCode. This order item is not specified in this example).

```xml
<osm:orderItem xmlns:osm="http://xmlns.oracle.com/communications/ordermanagement/model"
               id="1234">
  <osm:name>FIXED BUNDLE - BUNDLE</osm:name>
  ......
  <osm:properties xmlns:im="http://oracle.communications.ordermanagement.unsupported.centralom">
    <im:typeCode>BUNDLE</im:typeCode>
  </osm:properties>
</osm:orderItem>
```
Order Item 2 includes:
- **typeCode**: PRODUCT
- **lineId**: 3
- **parentLineId**: 2 (This matches the lineId of order item 1 indicating that order item 1 is the parent of order item 2).

Order Item 3 includes:
- **typeCode**: BUNDLE
- **lineId**: 6
- **parentLineId**: 1 (This indicates that both order item 1 and order item 3 share the same parent).
Order Item 4 includes:
  - **typeCode**: PRODUCT
  - **lineId**: 7
  - **parentLineId**: 6 (This matches the lineID of order item 3 indicating that order item 3 is the parent of order item 4).

The customer order includes two bundles with two products. The hierarchy is:

- Fixed Bundle - order item 2
  - Fixed Caller ID - order item 5
- Broadband Bundle - order item 6
  - BroadBand Service - order item 7

To create the separate customized component IDs for the bundle order items 1 and 3, and include all their corresponding children order items you need to:

- Return a separate component ID for each BUNDLE **typeCode**. This causes BUNDLE order components to be generated.
- Ensure that the PRODUCT **typeCode** for that bundle are included in its parent order item.
To do so, the XQuery uses the `ancestors` function to find whether the order item has a BUNDLE `typeCode` or has a BUNDLE `typeCode` in one of its parent order items. If the order item is a bundle, then OSM creates a component ID for the bundle. If the order item has a bundle in one of its parent order items, then OSM includes the order item in its parent order item component ID. The following example shows an XQuery that does this.

```xquery
declare namespace osm="http://xmlns.oracle.com/communications/ordermanagement/model";
declare namespace prop="http://oracle.communications.ordermanagement.unsupported.centralom";
declare namespace osmfn = "java:oracle.communications.ordermanagement.orchestration.generation.OrchestrationXQueryFunctions";

(: The following part of the XQuery identifies the order line hierarchy definition and retrieve all of the predecessor order line items in the bundle: :) let $ancestors :=
  osmfn:ancestors('CustomerOrderItemSpecification','default','http://oracle.communications.ordermanagement.unsupported.centralom')

(: The following part of the XQuery finds the BUNDLE order item and generates an ID based on the bundle order item lineID: :) return
  if (fn:exists($ancestors[osm:properties/prop:typeCode='BUNDLE'])))
    then (concat($ancestors[osm:properties/prop:typeCode=('BUNDLE')][1]/osm:properties/prop:lineId/text(),'/BundleGranularity'))
  else ('ALL_OFFERS_AND_NON_SERVICE_BILLING/BundleGranularity')
```

This XQuery finds the child order line items, finds their parent order line items, and creates a bundle order component for each of the bundle lines. The component keys are:

- BillingFunction.BillingSystem.Bundle.6/BundleGranularity

In another example, there is one offer with two bundles and two products in each bundle. The following table shows the hierarchy of bundles and products. The component keys use the line IDs of the two bundle items.

<table>
<thead>
<tr>
<th>Line Number</th>
<th>Line Name</th>
<th>Line typeCode</th>
<th>Parent Line ID</th>
<th>Value to Use in Component Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Triple Play</td>
<td>OFFER</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>Fixed Bundle</td>
<td>BUNDLE</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2.1</td>
<td>Fixed Service</td>
<td>PRODUCT</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2.2</td>
<td>Call Forwarding</td>
<td>PRODUCT</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>Broadband Bundle</td>
<td>BUNDLE</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>5.1</td>
<td>Broadband Service</td>
<td>PRODUCT</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5.2</td>
<td>High-Speed Internet</td>
<td>PRODUCT</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>
Custom Component IDs Based on Requested Delivery Date and Duration

In some scenarios, you may want to create custom Order Component IDs based on order item requested delivery date and duration. For example, the following custom component ID XQuery creates order component grouping based on the order item requested delivery dates:

```
declare namespace osm="http://xmlns.oracle.com/communications/ordermanagement/model";
declare namespace prop="http://oracle.communications.ordermanagement.unsupported.centralom";
declare namespace osmfn = "java:oracle.communications.ordermanagement.orchestration.generation.OrchestrationXQueryFunctions";
let $groupDuration := "P2D"
return osmfn: getGroupIdByDateTime ($groupDuration)
```

The XQuery creates a new order component for an order item based on the order item’s requested delivery date and includes all order items within this group that fall within two days of the first order item’s requested delivery date in the group. The XQuery does the same thing for all other order items within the order.

The following table shows how five order items would be grouped given a custom Order Component ID XQuery that creates a new component IDs.

<table>
<thead>
<tr>
<th>Order Item</th>
<th>Requested Delivery Date</th>
<th>Group ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>June 9, 2014</td>
<td>Group1</td>
</tr>
<tr>
<td>B</td>
<td>June 10, 2014</td>
<td>Group1</td>
</tr>
<tr>
<td>C</td>
<td>June 11, 2014</td>
<td>Group2</td>
</tr>
<tr>
<td>D</td>
<td>June 12, 2014</td>
<td>Group2</td>
</tr>
<tr>
<td>E</td>
<td>June 12, 2014</td>
<td>Group3</td>
</tr>
</tbody>
</table>

Note: The group ID names are static with the first order component always called Group1 and the next Group2, and so on.

See “About Component Specification Custom Component ID XQuery Expressions” for more information about the context, prolog, and body of this XQuery. See "OSM XQuery Functions" for more information about the OrchestrationXQueryFunctions class.

Custom Component IDs by Duration and Minimum Separation Duration

You can specify a minimum duration separation value for order items that fall very close to a custom Order ID grouping based on order item requested delivery date and duration. For example, the following XQuery adds a minimum separation value of one day:

```
declare namespace osm="http://xmlns.oracle.com/communications/ordermanagement/model";
declare namespace prop="http://oracle.communications.ordermanagement.unsupported.centralom";
declare namespace osmfn = "java:oracle.communications.ordermanagement.orchestration.generation.OrchestrationXQueryFunctions";
```

See "About Component Specification Custom Component ID XQuery Expressions" for more information about the context, prolog, and body of this XQuery. See "OSM XQuery Functions" for more information about the OrchestrationXQueryFunctions class.
let $groupDuration := 'P2D'
let $minSeparationDuration := 'P1D'
return osmfn: getGroupIdByDateTime ($groupDuration, $minSeparationDuration)

All order item requested delivery dates that fall within one day of a two day grouping, would be included in the two day grouping.

The following table shows how the five order items would be grouped given a one day minimum separation duration.

<table>
<thead>
<tr>
<th>Order Item</th>
<th>Requested Delivery Data</th>
<th>Group ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>June 9, 2014</td>
<td>Group1</td>
</tr>
<tr>
<td>B</td>
<td>June 10, 2014</td>
<td>Group1</td>
</tr>
<tr>
<td>C</td>
<td>June 11, 2014</td>
<td>Group1</td>
</tr>
<tr>
<td>D</td>
<td>June 12, 2014</td>
<td>Group2</td>
</tr>
<tr>
<td>E</td>
<td>June 12, 2014</td>
<td>Group2</td>
</tr>
</tbody>
</table>

See "About Component Specification Custom Component ID XQuery Expressions" for more information about the context, prolog, and body of this XQuery. See "OSM XQuery Functions" for more information about the OrchestrationXQueryFunctions class.

**Combining Order Item Hierarchy with Duration-Based Groupings**

You can combine the function to create custom Component IDs based on order item requested delivery date, duration, and minimum duration separation, or a combination of these functions with order component ID generation based on order item hierarchy. The following example creates separate component IDs for order items that, although they have the same requested delivery date, are part of different order item hierarchical groupings:

```
declare namespace osm="http://xmlns.oracle.com/communications/ordermanagement/model";
declare namespace prop="http://oracle.communications.ordermanagement.unsupported.centralom";
declare namespace osmfn = "java:oracle.communications.ordermanagement.orchestration.generation.OrchestrationXQueryFunctions";
let $groupDuration := 'P2D'
let $minSeparationDuration := 'P1D'
return osmfn: getGroupIdByDateTime ($groupDuration, $minSeparationDuration)
return fn:concat($rootAncestorId, '/', $groupId)
```

The following table shows how five hierarchically divided order items would be grouped given a one day minimum separation duration.

<table>
<thead>
<tr>
<th>Order Item</th>
<th>Requested Delivery Data</th>
<th>Group ID</th>
<th>Component ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.1</td>
<td>June 9, 2014</td>
<td>Group1</td>
<td>A/Group1</td>
</tr>
<tr>
<td>A.1.1</td>
<td>June 11, 2014</td>
<td>Group1</td>
<td>A/Group1</td>
</tr>
</tbody>
</table>
### About Component Specification Duration XQuery Expressions

This topic applies to the Order Component editor, **Duration** tab, Duration Expression area, **XQuery** subtab.

- **Context:** There is no input document for this expression.
- **Prolog:** There is no prolog required for this expression.
- **Body:** The XQuery body returns a duration value based on the XQuery you enter:

  \[
  PTyYMmDTmMmSS
  \]

  where
  - \(P\) begins the expression.
  - \(yY\) specifies the year.
  - \(mM\) specifies the month.
  - \(dD\) specifies the day.
  - \(T\) separates the parts of the expression indicating the date from the parts of the expression indicating the time.
  - \(hH\) specifies the hour.
  - \(mM\) specifies the minutes.
  - \(sS\) specifies the seconds.

The following example is a hard-coded duration expression for seven hours:

\[
PT7D0M0S
\]

For more information about how OSM uses these fields to calculate order component durations, see *OSM Concepts*.

### About Fulfillment Pattern Duration XQuery Expressions

This topic applies to the Fulfillment Pattern editor, **Orchestration Plan** tab, **Duration** subtab, Duration Expression area, **XQuery** subtab. The functionality for this tab has been deprecated and is displayed to provide backward compatibility with older cartridges.

For the recommended functionality for configuring order component durations, see "About Fulfillment Pattern Component Duration XQuery Expressions" and "About..."
Component Specification Duration XQuery Expressions”.

About Fulfillment Pattern Component Duration XQuery Expressions

This topic applies to the Fulfillment Pattern editor, Orchestration Plan tab, Order Components subtab, Duration subtab, Duration Expression area, XQuery subtab.

- Context: There is no input document for this expression.
- Prolog: There is no prolog required for this expression.
- Body: The XQuery body returns a duration value based on the XQuery you enter:

  \[ P Y m M d D T h H m M s S \]

  where
  - \( P \) begins the expression.
  - \( Y \) specifies the year.
  - \( m \) specifies the month.
  - \( D \) specifies the day.
  - \( T \) separates the parts of the expression indicating the date from the parts of the expression indicating the time.
  - \( h \) specifies the hour.
  - \( m \) specifies the minutes.
  - \( s \) specifies the seconds.

The following example is a hard-coded duration expression for three hours:

  \[ PT3H0M0S \]

  For more information about how OSM uses these fields to calculate order component durations, see OSM Concepts.

Dependency XQuery Expressions

This topic includes information about Orchestration XQuery expressions related to orchestration dependencies:

- About Order Item Dependency Property Correlation XQuery Expressions
- About Wait Delay Duration XQuery Expressions
- About Wait Delay Date and Time XQuery Expressions
- About Order Data Change Wait Condition XQuery Expressions
- About Order Item Inter-Order Dependency XQuery Expressions

About Order Item Dependency Property Correlation XQuery Expressions

This topic describes how to use one of the following fields:

- Fulfillment Pattern editor, Orchestration Plan tab, Dependencies tab, Order Item Dependency subtab, XQuery subtab for the Property Correlation selection
- Orchestration Dependency editor, Order Item Dependencies tab, XQuery subtab for the Property Correlation selection
to write an expression that specifies dependencies between different order items using order item properties. The Property Correlation XQuery has the same context, prolog, and body structure as the Fulfillment Pattern editor, Order Components tab, Order Item Association subtab, XQuery subtab. See "About Associating Order Items Using Property Correlations XQuery Expressions" for more information.

The following example shows a dependency that requires provisioning of an Internet service before shipping a modem. This involves two order items: provision Internet service and ship modem. The correlating property is the order item ID.

```xml
declare namespace osm='http://xmlns.oracle.com/communications/ordermanagement/model';
declare namespace im='http://sample.broadband';
let $bbProvision := osm:fromOrderComponent/osm:orderItem[osm:name='Internet Service']
let $bbModem := osm:toOrderComponent/osm:orderItem[osm:name/text()='Broadband Modem'
    and osm:properties/im:SiteID/text() = $bbProvision/osm:properties/im:SiteID/text()]
return <osm:dependency fromOrderItemId='{$bbProvision/@id}' toOrderItemId='{$bbModem/@id}'/>
```

In this example:
- $bbProvision contains the broadband service order item in the blocking Provision order component.
- $bbModem is the broadband modem in the waiting Ship order component.
- The XQuery returns a dependency from the Internet Service order item to its associated Broadband Modem order item, identified by $bbProvision/@id and $bbModem/@id.

If the order item IDs are:
- $bbProvision/@id = 1301589468772
- $bbModem/@id = 1301589468785

Then the XQuery returns the following:

```
<osm:dependency fromOrderItemId='1301589468772' toOrderItemId='1301589468785'/>
```

### About Wait Delay Duration XQuery Expressions

This topic describes how to use one of the following fields:
- Fulfillment Pattern editor, Orchestration Plan tab, Dependencies subtab, Wait Condition subtab, Wait Delay area, Duration Expression area XQuery subtab for the Duration selection
- Orchestration Dependency editor, Wait for Condition tab, Wait Delay area, Duration Expression area XQuery subtab for the Duration selection

To write an expression that specifies the duration of delay, based on an order item property, before starting a waiting order component after all dependencies have been resolved.

- Context: The Duration XQuery input document is the entire set of order items included in the order contained in the toOrderComponent element. You can return the value of requestedDeliveryDate to help determine the wait delay duration. For example:

```xml
<toOrderComponent xmlns=''
<osm:orderItem [35 lines]
<osm:orderItem [37 lines]
<osm:orderItem [42 lines]
```
Prolog: You can declare the order item namespace and the OSM namespace in the XQuery prolog. For example:

```xquery
declare namespace osm="http://xmlns.oracle.com/communications/ordermanagement/model";
declare namespace im="http://oracle.communications.ordermanagement.unsupported.centralom";
```

Body: The XQuery body returns a duration value based on the requestedDeliveryDate order item property:

```xquery
let $mydate := osm:toOrderComponent[1]/osm:orderItem[1]/osm:properties[1]/*[namespace-uri()='http://oracle.communications.ordermanagement.unsupported.centralom' and local-name()='requestedDeliveryDate'][1]/text() return if (fn:current-dateTime() - xs:dateTime($mydate) < xs:dayTimeDuration('PT10H')) then 'PT10H' else 'PT10M'
```

where

- `osm:toOrderComponent`: Provides the entire set of order items included in the order.
- `osm:orderItem`: These are the order items in the `toOrderComponent` category. The remainder of this expression identifies the namespace of the order item specification and returns the value of the requestedDeliveryDate element.
The if statement checks to see if the value of the requestedDeliveryDate is less than the hard-coded dayTimeDuration value. These values conform to the XSD duration data type.

The then statement returns 10 hours if the if statement evaluates to true.

The else statement returns 10 months if the if statement evaluates to false.

The following example shows the sample XQuery to return a duration value.

```xml
declare namespace osm="http://xmlns.oracle.com/communications/ordermanagement/model";
declare namespace im="http://oracle.communications.ordermanagement.unsupported.centralom";

let $mydate :=
  osm:toOrderComponent[1]/osm:orderItem[1]/osm:properties[1]/*[namespace-uri() = 'http://oracle.communications.ordermanagement.unsupported.centralom' and local-name()='requestedDeliveryDate'][1]/text()
return
  if (fn:current-dateTime() - xs:dateTime($mydate) < xs:dayTimeDuration('PT10H'))
     then
        'PT10H'
   else
        'PT10M'
```

### About Wait Delay Date and Time XQuery Expressions

This topic describes how to use one of the following fields:

- Fulfillment Pattern editor, Orchestration Plan tab, Dependencies subtab, Wait Condition subtab, Wait Delay area, Duration Expression area XQuery subtab for the Date Time Expression selection

- Orchestration Dependency editor, Wait for Condition tab, Wait Delay area, Duration Expression area XQuery subtab for the Date Time Expression selection to write an expression that specifies the date and time, based on an order item property, for starting a waiting order component after all dependencies have been resolved.

- Context: The Date Time Expression XQuery input document is the entire set of order items included in the order contained in the toOrderComponent element. You can use the requestedDeliveryDate order item property to determine the date and time that the XQuery should start after all blocking items have resolved. For example:

```xml
<toOrderComponent xmlns="">
  <osm:orderItem [35 lines]
  <osm:orderItem [37 lines]
  <osm:orderItem [42 lines]
  <osm:orderItem
    xmlns:osm="http://xmlns.oracle.com/communications/ordermanagement/model"
    id="5678">
    <osm:name>Broadband Bundle [Add]</osm:name>
    ....
    <osm:properties
      xmlns:im="http://oracle.communications.ordermanagement.unsupported.centralom">
      <im:typeCode>PRODUCT</im:typeCode>
      <im:parentLineId>3</im:parentLineId>
      <im:requestedDeliveryDate>2013-06-31T12:00:00</im:requestedDeliveryDate>
      <im:lineItemName>Broadband Bundle [Add]</im:lineItemName>
      <im:lineId>4</im:lineId>
      <im:SiteID>10</im:SiteID>
      <im:ServiceActionCode>UPDATE</im:ServiceActionCode>
```
<im:productClass>Broadband Bundle Class</im:productClass>
<im:serviceId>1112223333</im:serviceId>
<im:lineItemPayload>[34 lines]</im:lineItemPayload>
<im:region>Sao Paulo</im:region>
<osm:properties>
<osm:orderItem>[57 lines]</osm:orderItem>
<osm:orderItem>[57 lines]</osm:orderItem>
<osm:orderItem>[42 lines]</osm:orderItem>
<osm:orderItem>[37 lines]</osm:orderItem>
<osm:orderItem>[37 lines]</osm:orderItem>
<osm:orderItem>[57 lines]</osm:orderItem>
</toOrderComponent>

■ Prolog: You can declare the order item namespace and the OSM namespace in the XQuery prolog. For example:

```xml
declare namespace osm="http://xmlns.oracle.com/communications/ordermanagement/model";
declare namespace im="http://oracle.communications.ordermanagement.unsupported.centralom";
```

■ Body: The XQuery body returns a date and time value based on the requestedDeliveryDate order item property:

```xml
osm:toOrderComponent[1]/osm:orderItem[1]/osm:properties[1]/*[namespace-uri()='http://oracle.communications.ordermanagement.unsupported.centralom' and local-name()='requestedDeliveryDate'][1]/text()
```

**osm:toOrderComponent**: returns the entire set of order items included in the order and returns the requested delivery date of all order items for the wait delay date and time.

The following example shows the sample XQuery to return a date time value.

```xml
declare namespace osm="http://xmlns.oracle.com/communications/ordermanagement/model";
declare namespace im="http://oracle.communications.ordermanagement.unsupported.centralom";

osm:toOrderComponent[1]/osm:orderItem[1]/osm:properties[1]/*[namespace-uri()='http://oracle.communications.ordermanagement.unsupported.centralom' and local-name()='requestedDeliveryDate'][1]/text()
```

### About Order Data Change Wait Condition XQuery Expressions

This topic describes how to use one of the following fields:

- Fulfillment Pattern editor, Orchestration Plan tab, Dependencies subtab, Wait Condition subtab, Wait for Condition area, XQuery subtab for the Data Change Notification selection.

- Orchestration Dependency editor, Wait for Condition tab, Wait for Condition area, XQuery subtab for the Data Change Notification selection

to write an expression that specifies a value that must exist in order item property (typically a blocking order item property) before a waiting order item starts.
Context: The **Data Change Notification** XQuery input document is the task view task data that was changed using an update order transaction.

Prolog: You can declare the $blockingIndexes variable in the XQuery prolog that contains an index of data element for all blocking order items. For example:

```
declare variable $blockingIndexes as xs:integer* external;
```

Body: The XQuery body returns a specific value and will wait until all blocking order items have the corresponding value and the XQuery returns true.

The following example shows the XQuery that evaluates the data change. The dependency is met when all blocking order items have reached a state of PROVISION STARTED.

```
(: The $blockingIndexes variable contains data element indexes for all blocking order items: :) 
declare variable $blockingIndexes as xs:integer* external;
(: Specify "PROVISION STARTED" as the data value that must be met: :) 
let $expectedMilestoneCode := "PROVISION STARTED"
(: $milestoneValues contains a set of milestones for all blocking order items: :) 
let $milestoneValues := 
/GetOrder.Response/_root/ControlData/Functions/ProvisioningFunction/orderItem/orderItemRef[ fn:index-of($blockingIndexes, xs:integer(@referencedIndex)) != 0]/milestone[text() eq $expectedMilestoneCode]
(: Return true only if all the milestones in ProvisioningFunction/orderItem/orderItemRef are PROVISION STARTED: :) 
return fn:count($milestoneValues) eq fn:count($blockingIndexes)
```

The following example returns true when at least one blocking item is completed.

```
declare namespace oms="urn:com:metasolv:oms:xmlapi:1";
declare variable $blockingIndexes as xs:integer* external;
let $component := //ControlData/Functions/NetworkProvisionFunction
let $lineItem := $component/orderItem/orderItemRef[fn:index-of($blockingIndexes, xs:integer(@referencedIndex)) != 0]
return 
if (fn:exists($lineItem))
then
    let $statusValue := $lineItem/OrderItemStatus/text() = "completed"
    return 
    if (fn:count($statusValue)>0)
    then
        fn:true()
    else
        fn:false()
    else
        fn:false()
```

About **Order Item Inter-Order Dependency** XQuery Expressions

This topic describes how to use the Order Item Specification editor, **Order Item Dependency** tab, Order Item Selector area, **XQuery** tab to write an expression that creates dependencies between order items across orders.

Context: The **Order Item Selector** XQuery input document is typically an order item on a follow-on order (the waiting order).

Prolog: You can declare the OSM namespace, the cartridge namespace for the target order, and the namespace of the task that contains the data you want to query. For example:

```
declare namespace oms="http://xmlns.oracle.com/communications/ordermanagement";
```
Body: The XQuery body returns the order ID of the target order and the order item property:

```xml
<osm:dependency fromOrderId="{$parentOrderId}"
fromOrderItemId="{$parentOrderItemId}"/>
```

where

- `<osm:dependency fromOrderId>`: Returns the target order ID.
- `fromOrderItemId`: Returns the target order item property value that controls the dependency.

The following example shows an XQuery for an inter-order dependency.

```xml
let $dependingLineId := fn:normalize-space(osm:properties/im:DependingSalesOrderBaseLineId)
return
  if(fn:not($dependingLineId = ''))
  then
    (: Use the data instance behavior "findOrder" to find the base order: :) 
    let $parentOrderId := fn:normalize-space(vf:instance("findOrder")/ord:Order[last()]/ord:Amendments/ord:AmendedOrderSummary/ord:Id/text())
    (: Use the data instance behavior "getOrder" to find the associated order item ID in the base order: :) 
    let $parentOrderItemId := fn:normalize-space(vf:instance("getOrder")/ord:Data/osmc:_root/osmc:ControlData/osmc:OrderItem[osmc:BaseLineId=$dependingLineId]/@index)
    return
      if(fn:not($parentOrderId = '') and fn:not($parentOrderItemId = ''))
      then
        (: Return the dependency: :) 
        <osm:dependency fromOrderId="{$parentOrderId}"
fromOrderItemId="{$parentOrderItemId}"/>
      else()
        else()
```

Order Transformation Manager XQuery Expressions

The following topics provide reference information about order transformation manager XQuery expressions.

- About Transformation Sequence XQuery Expressions
- About Mapping Rule XQuery Expressions
- About Order Item Parameter Binding XQuery Expressions
About Transformation Sequence XQuery Expressions

When working with Transformation Sequence editor, see the following topics for information about defining XQuery expressions related to transformation sequences:

- About Order Item Context XQuery Expressions
- About Related Order Item Selector XQuery Expressions
- About Stage Condition XQuery Expressions

About Order Item Context XQuery Expressions

This topic describes how to use the Transformation Sequence editor, Dependencies tab, Order Item Context subtab, Expression area, XQuery subtab to write an expression that defines the context order items for the order transformation. To see the Order Item Context subtab, you must select a transformation stage in the tree on the Dependencies tab.

- Context: The input document is the complete set of source order items.
- Prolog: You can declare the order item namespace in the XQuery prolog. For example:
  
  ```
  declare namespace prop='http://oracle.communications.centralom';
  ```

- Body: The XQuery body returns the source order items that should be considered the context for the transformation stage.

The following example shows an XQuery expression for selecting an order item context.

```
declare namespace prop='http://oracle.communications.centralom';
osm:orderItem[osm:properties/prop:serviceIntance = 'Y']
```

About Related Order Item Selector XQuery Expressions

This topic describes how to use the Transformation Sequence editor, Dependencies tab, Related Order Item Selector subtab, Expression area, XQuery subtab to write an expression that defines the related order items for a particular context order item. To see the Related Order Item Selector subtab, you must select a transformation stage in the tree on the Dependencies tab.

- Context: The input document is a context order item.
- Prolog: You can declare the order item namespace and the namespace for the order transformation manager functions in the XQuery prolog. For example:

  ```
  declare namespace prop='http://oracle.communications.broadband';
  declare namespace otmfn="java:oracle.communications.ordermanagement.orchestration.transformation.XQueryFunctions.";
  ```

- Body: The XQuery body returns the source order items related to the context order items.

The following example shows an XQuery expression that returns sibling order items as related order items to the order transformation.

```
declare namespace prop='http://oracle.communications.broadband';
declare namespace otmfn="java:oracle.communications.ordermanagement.orchestration.transformation.XQuenquery = 'Y'
```


eryFunctions;”;
let $siblings := otmfn:siblings (.,
'\{http://oracle.communications.broadband}\default')
return $siblings[! fn:exists(osm:properties[prop:serviceInstance = 'Y'])]

For more information about the transformation.XQueryFunctions class, install the OSM SDK and see the OSM Java docs located in the OSM_home/SDK/osm7.x.y.z-javadocs.zip file (where OSM_home is the directory in which the OSM software is installed and x.y.z are the software release, patch, and build numbers). See OSM Installation Guide for more information about installing the OSM SDK.

About Stage Condition XQuery Expressions

This topic describes how to use the Transformation Sequence editor, Dependencies tab, Stage Condition subtab, Expression area, XQuery subtab to write an expression that determines whether a particular transformation stage should be run. To see the Stage Condition subtab, you must select a transformation stage in the tree on the Dependencies tab.

- **Context:** The input document is the complete set of target order items.
- **Prolog:** You can declare the order item property and parameter namespaces in the XQuery prolog. For example:

  declare namespace prop='http://oracle.communications.broadband';
  declare namespace parm='http://oracle.communications.broadband';

- **Body:** The XQuery body returns a Boolean, with true meaning that the transformation stage should be run and false meaning that the transformation stage should not be run.

The following example shows an XQuery expression that returns true if certain parameters have not been defined, and false if the parameters are already defined.

```
declare namespace prop='http://oracle.communications.cso';
declare namespace parm='http://oracle.communications.broadband';
not(fn:exists(osm:properties/prop:Parameters[fn:exists(parm:uploadSpeed) and fn:exists(parm:downloadSpeed)])
```

About Mapping Rule XQuery Expressions

When working with Mapping Rule editor, see the following topics for information about defining XQuery expressions related to order decomposition:

- About Mapping Condition XQuery Expressions
- About Action Mapping XQuery Expressions
- About Entity-to-Entity Advanced Mapping XQuery Expressions
- About Entity-to-Data-Element Advanced Mapping XQuery Expressions
- About Data-Element-to-Data-Element Advanced Mapping XQuery Expressions
- About Reverse Mapping XQuery Expressions
- About Multi-Instance XQuery Expressions
About Mapping Condition XQuery Expressions
This topic describes how to use the Mapping Rule editor, Mapping tab, Condition subtab, Expressions area, XQuery subtab to write an expression that defines a condition that must be satisfied to apply this mapping.

- Context: The input document is a target order item.
- Prolog: You can declare the order item namespace in the XQuery prolog. For example:
  
  ```xquery
  declare namespace prop='http://oracle.communications.broadband';
  ```

- Body: The XQuery body returns a Boolean, with true meaning that the mapping rule should be run and false meaning that the mapping rule should not be run.

The following example shows an XQuery expression that will execute the rule only if the target action is None.

```xquery
declare namespace prop='http://oracle.communications.cso';
osm:properties/prop:Action/text() = 'None'
```

About Action Mapping XQuery Expressions
This topic describes how to use the Mapping Rule editor, Mapping tab, Actions subtab Action Mappings area, XQuery subtab to write an expression that defines the mapping for an action code for a particular mapping rule. To access this field, you must deselect Use Relationship Action Map and select the Advanced option.

- Context: The input document is a source order item.
- Prolog: You can declare the following variables within the prolog to determine the action code.
  - You can declare $sourceValue to access the action code of the source order item. This is the Order Item Action property value for the source order item.
  - You can declare $currentTargetValue to access the action code of the target order item. This is the Order Item Action property value for the target order item.
- Body: The XQuery body returns an action code, or returns () to leave the current value unchanged.

The following example shows an XQuery expression that returns the source action code if the target action code is not already set and otherwise leaves the target action code unchanged.

```xquery
declare $sourceValue external;
declare $currentTargetValue external;
if (! fn:empty($currentTargetValue))
  $sourceValue
else
  ()
```

About Entity-to-Entity Advanced Mapping XQuery Expressions
This topic describes how to use the Mapping Rule editor, Mapping tab, Mapping subtab, Mapping Rule Item area, XQuery subtab to write an expression that defines an advanced mapping between two entities. This field is displayed when you select the target of an entity-to-entity mapping. This is the only type of mapping available for entity-to-entity mapping.

- Context: The input document is a source order item.
Prolog: You can declare any namespaces needed to construct the target property (or properties) in the XQuery prolog. For example:

```
declare namespace prop='http://oracle.communications.cso';
```

Body: The XQuery body returns a list of order item properties to be set on the target order item. If the property already exists on the target order item, it will be overwritten by the value returned from this XQuery expression.

The following example shows an XQuery expression that returns the structured Parameters property for the target order item.

```
declare namespace prop='http://oracle.communications.cso';
<prop:Parameters xmlns:param="http://oracle.communications.broadband">
  <parm:AAAAccount>Account1</parm:AAAAccount>
  <parm:DownloadSpeed>6</parm:DownloadSpeed>
  <parm:UploadSpeed>0.6</parm:UploadSpeed>
  <parm:MAC/>
  <parm:Brand>Siemens</parm:Brand>
  <parm:Model>4200</parm:Model>
  <parm:Firewall>Y</parm:Firewall>
</prop:Parameters>
```

About Entity-to-Data-Element Advanced Mapping XQuery Expressions

This topic describes how to use the Mapping Rule editor, Mapping tab, Mapping subtab Mapping Rule Item area, XQuery subtab to write an expression that defines an advanced mapping between an entity and a data element. This field is displayed when you select the target of an entity-to-data-element mapping and select the Advanced option in the Mapping Rule Item topic.

- Context: The input document is a source order item.
- Prolog: There is no prolog for this XQuery.
- Body: The XQuery body returns a data element value or returns () to leave the current value unchanged.

The following example shows an XQuery expression that returns "Y" if a particular parameter exists, and () if it does not exist.

```
if fn:exists(vf:instance("checkMe")/somevalue)
  "Y"
else
  ()
```

About Data-Element-to-Data-Element Advanced Mapping XQuery Expressions

This topic describes how to use the Mapping Rule editor, Mapping tab, Mapping subtab Configuration subtab, XQuery subtab to write an expression that defines an advanced mapping between two data elements. This field is displayed when you select the target of a data-element-to-data-element mapping and select the Advanced option in the Mapping Rule Item topic.

- Context: The input document is a source order item during normal transformation. If invoked during forward data propagation, the input document is empty.
- Prolog: You can declare the order item namespace in the XQuery prolog. For example:

```
declare namespace prop='http://oracle.communications.centralom';
```
You can also declare the following variable within the prolog to determine the action code.

- You can declare $value to contain the values of the target data elements.

**Body:** The XQuery body returns one or more data element values or returns () to leave the current value unchanged.

The following example shows an XQuery expression that returns the target value of a data element based on the value of the source data element.

```
declare variable $value external;
if (fn:empty($value)) then ('unknown') else (fn:concat('Loc: ', $value))
```

The following example shows an XQuery expression that returns the target value of a data element based on characteristics of the source order item.

```
declare namespace prop='http://oracle.communications.centralom';
if (fn:exists(osm:properties/prop:ServicePoint/text()))
then (fn:concat('Loc: ',
fn:normalize-space(osm:properties/prop:ServicePoint/string())))
else ('unknown')
```

### About Reverse Mapping XQuery Expressions

This topic describes how to use the Mapping Rule editor, **Mapping** tab, **Mapping** subtab, **Bi-Directional Mapping** subtab, **XQuery** subtab to write an expression that defines an advanced mapping between two data elements. This field is displayed when you select the target of a data element-to-data element mapping and select the **Advanced** option in the **Mapping Rule Item** topic, if **Supports Bi-Directional Mapping** is selected in the **Details** subtab of the **Mapping** tab for the selected mapping.

- **Context:** The input document is empty.
- **Prolog:** You can declare the following variables within the prolog to determine the action code.

  - You can declare $value to access the updated target value.
  - **Body:** The XQuery body returns the updated source value.

The following example shows an XQuery expression that returns () if the return value is unknown and otherwise returns the updated value.

```
declare variable $value external;
if ('unknown' = $value) then() else (fn:substring($value, 5))
```

### About Multi-Instance XQuery Expressions

This topic describes how to use the Mapping Rule editor, **Mapping** tab, **Mapping** subtab, **Multi-Instance Expression** subtab, **XQuery** subtab to write an expression that defines key mapping for a multi-instance structure. This field is displayed when you select the target of a data element-to-data element mapping and select the **Advanced** option in the **Mapping Rule Item** topic, if the target data element is a member of a multi-instance structure.

- **Context:** The input document is a source order item.
- **Prolog:** You can declare the order item namespace in the XQuery prolog. For example:

  ```
declare namespace prop='http://oracle.communications.centralom';
  ```
Body: The XQuery body returns a key value that identifies a source order item instance.

The following example shows an XQuery expression that returns the concatenation of two source order item properties for the key value.

```xquery
fn:concat(prop:areaCode, '-', prop:localNumber)
```

### About Order Item Parameter Binding XQuery Expressions

This topic describes how to use the Order Item Parameter Binding editor, Parameter Bindings tab, Binding Expression area, XQuery subtab to write an expression that defines the bindings for one or more parameters on a conceptual model entity from an order item.

- **Context:** The input document is an input order item. Each order item element in this node set is passed into the XQuery as the context.

- **Prolog:** You can declare the namespace for the incoming order and the namespace for the conceptual model entity in the XQuery prolog. For example:

  ```xquery
  declare namespace im="http://xmlns.oracle.com/InputMessage";
  declare namespace otm="CommonModelBroadbandCart/1.0.0.0.0";
  ```

- **Body:** The body of the XQuery will return a node set of elements that correspond to the conceptual model entity data elements. Since you can have as many separate bindings between the entities as you like, this can return anything from one data element to all of them.

The following example shows an XQuery expression that returns an UploadSpeed and a DownloadSpeed parameter from two name-value pairs where the names are Upload Speed and Download Speed.

```xquery
declare namespace fulfillord="http://xmlns.oracle.com/InputMessage";
declare namespace otm="OSMCom_3Play/1.0.0.0.0";
<otm:DownloadSpeed>{fn:normalize-space(fulfillord:itemReference/fulfillord:specificationGroup/fulfillord:specification[fulfillord:name='DownloadSpeed']/fulfillord:value)}</otm:DownloadSpeed>
```

### About Transformed Order Item Fulfillment State XQuery Expressions

This topic describes how to use the Transformed Order Item Fulfillment State Composition Rule Set editor, Composition Rules tab, Source Order Item subtab, XQuery field to write an expression that defines the conceptual model entities that should be present if the condition is to be evaluated. This field is only available when you have a condition selected in the tree in the tab, and you have selected the Advanced option on the subtab.

- **Context:** The input document is the order.

- **Prolog:** You can declare `$orderItemIndex` to access the index of the order item being considered.

- **Body:** The body of the XQuery will return a Boolean value indicating whether the current rule should be used to calculate the fulfillment state.
The following example shows an XQuery expression that returns true if a particular order item property has a specific value.

```
declare variable $orderItemIndex external;

let $orderData := fn:root(.)/GetOrder.Response
let $orderItem := $orderData/_root/ControlData/OrderItem[@index=$orderItemIndex]
return
  if (fn:exists($orderItem) and fn:data($orderItem/AnyProperties) = 'ABC')
    then fn:true()
  else fn:false()
```
activation
The enabling, disabling or changing of a service or network feature.

activation task
A type of automated task designed specifically to interact with the Oracle Communications ASAP product or the Oracle Communications IP Service Activator product.

Administrator application
An OSM application used to manage user workgroups, calendars and schedules, email notifications, and system events.

amend and amendment processing
A generic term that refers to making changes to in-flight orders. Amendments are typically made when processing a revision order, or managing an order cancellation or order failure. The amendment usually performs compensation; such as redoing or undoing tasks.

Application Integration Architecture (AIA)
The Oracle Application Integration Architecture. A set of Oracle products that enable you to set up and orchestrate cross-application business processes so that multiple applications can work together. OSM integrates with Oracle AIA for Communications. Oracle AIA runs on top of Oracle Fusion Middleware.

Application Integration Architecture (AIA) Order-to-Activate Cartridges
A set of OSM cartridges that integrate with the Oracle Communications Order to Cash Integration Pack for Oracle Communications Order and Service Management (Order to Cash Integration Pack for OSM). The Order-to-Activate cartridges and the integration pack enable OSM to be part of an order fulfillment solution that cover the entire order-to-activate process from order creation to service activation.

ASAP
Automated Service Activation Program. An Oracle product used by communication service providers to activate operational support systems equipment across multiple technology domains. ASAP supports many hardware vendor’s network systems, and is integrated with OSM using activation tasks.

automated task
A task that does not require manual activity. Automated tasks handle interactions with external systems such as billing systems, shipping systems, and activation systems.
They can also perform custom calculations and other tasks. Automated tasks are implemented using **automation plug-ins**. See **manual task**.

**automation framework**

An interface that enables the integration of OSM with external applications. It is used to automate tasks and notifications to other systems. It can also be used to perform business logic (such as performing complex calculations) without interacting with an external system. The automation framework is an OSM server component that performs the work required by **automation plug-ins**.

**automation plug-in**

An OSM component that performs the operation specified by an **automated task**. For example, you can create automation plug-ins to update order data, complete order tasks with appropriate statuses, set process exceptions, react to system notifications and events, send requests to external systems, and process responses from external systems. These operations can involve communication with external systems. OSM provides several predefined plug-ins. You can also develop your own plug-ins using a custom template.

**behaviors**

OSM behaviors allow you to control the validation and presentation of data elements in the **OSM Web clients** and **data providers**. For example, you can use the **Calculate** behavior to derive the value of the data in a field by adding the values in two other fields. You could use the **Information** behavior to present a tool tip for a field in the **Task Web client**.

**cartridge**

A software package created in Design Studio to deploy functionality to an OSM run-time system. Cartridges contain order **metadata** such as **recognition rules**, **processes**, **order states**, **behaviors**, and **specification**, and other entities used for order processing.

Cartridges are created with Design Studio, but Oracle also offers customized cartridges that support integration with other common applications (for example, the Oracle Communications Order to Cash Integration Pack for Oracle Communications Order and Service Management). Oracle also offers pre-configured cartridges that demonstrate the capabilities of OSM.

**central order management**

The OSM system role that receives customer orders from one or more order-source systems such as Siebel CRM, creates an OSM order, and manages the fulfillment of the order across other enterprise systems including billing, shipping, and service fulfillment systems. OSM operating in the central order management role also receives status information from these systems and provides visibility of an order’s status back to the order-source system. The central order management role is sometimes called central fulfillment.

An OSM instance can operate in a central order management role or in a **service order management** role.

**common fulfillment state**

The **fulfillment state** list that is defined on the **States** tab of the **fulfillment state map** entity. The common fulfillment state list provides the values that can be used for both **mapped fulfillment states** and **composite fulfillment states**.
compensation
Changes that are made to accommodate revisions to in-flight orders (orders still being processed). For example, if a customer initially orders Bronze-level DSL service but upgrades to Gold-level service while the original order is in place, tasks may need to be done, redone, or undone. OSM automatically calculates the compensation required to accommodate the changes to orders.

OSM uses these types of compensation: Do, which is a change that needs to be made in addition to the original tasks. Redo, which is a change that needs to be made to redo work that was already performed by the order. Undo, which is a change that needs to be made to undo work that has already been done by the order. Amend Do, which is the same as Do, but performed during amendment processing.

composite fulfillment state
The fulfillment state that results from an order fulfillment state composition rule set or an order item fulfillment state composition rule set.

control data
Metadata in an orchestration order that is used to manage the execution of the orchestration plan. OSM extracts control data from an order. Control data provides information about order items, order components and dependencies required to create the orchestration plan. This includes status and timestamps for its order items and components. During the execution of an orchestration plan, the order data, including control data, can be updated as transactions are completed.

Design Studio automatically generates control data for order components. You manually model control data for order items.

creation task
The task that contains data required for the order. The creation task specifies what information must be provided to the order before it can start processing. This applies equally to manual order entry through the Task Web client as well as through OSM WS or XML APIs. A creation task is defined in the order specification.

In OSM Web clients, the creation task represents the step that creates and submits an order instance that starts the order workflow.

CRM
Customer relationship management. A system for managing a company’s interactions with customers, clients and sales prospects; for example, Oracle’s Siebel CRM. When used in OSM documentation, CRM refers to sales and order capture activities.

customer order
An order request received by OSM to obtain a product or products, typically generated by the CRM or some other order-source system. OSM converts the customer order to OSM format after which it is referred to as an order. Sometimes called an in-bound order.

Data Dictionary
The logical repository of data elements used in Design Studio. The Data Dictionary defines data types and structures that can be used within OSM orders. For example, you can define a simple type that represents an IP address or a phone number, or more complex types representing addresses, product attributes and so on. Data elements in a Data Dictionary are used as building blocks of an OSM order.
The data elements within a Data Dictionary project can be referenced by other projects in a workspace. Multiple data dictionaries can be used to contribute data structures to a single order definition. A single Data Dictionary can also contribute to multiple order definitions.

**data element**

An entity viewable in Design Studio. When you model simple and structured data elements in the Data Dictionary, you can create new data elements that inherit their attributes from other existing data elements. The new data element can extend the information configured in the parent data element (also referred to as the base data element). For example, if you have a structured data element called *person* with first name, last name, and social security number child elements. You can extend the *person* structured data element by using it as a base for a new structured data element called *employee*, to which you add the employee number, hire date, and department child elements.

*Simple data elements* are reusable data types that contain no child dependencies. A simple data element has no structure, and is associated—directly or indirectly—to a primitive type (int, boolean, char, and so forth). *Structured data elements* are reusable data types and are containers of simple data elements and other structured data elements.

**data provider**

An adapter that can retrieve order data from external systems in an XML format. Design Studio provides several built-in data providers to retrieve external XML instances from specific sources such as a JDBC database or a SOAP Web service. Additionally, you can create your own custom data provider. Data providers are used when defining Data Instance behaviors.

**decomposition**

The process by which a *customer order* is broken into constituent order items, which are then organized into order components. For example, OSM can use the following algorithm to achieve decomposition:

- Map order items to functions (fulfillment function order components)
- Map function order items to fulfillment systems (fulfillment system order components)
- Map fulfillment system order items to processing granularity (granular order components)

The example above is representative of what OSM is normally configured to do for decomposition, but OSM is not restricted to these three stages. Decomposition is specified through configuration and can have any number of stages through which the order is decomposed.

See also *order component*, *executable order component*, *orchestration stage*.

**decomposition rule**

Rules that determine the order items in each order component. Decomposition rules specify the conditions in which OSM decomposes one or more order components into another order component. OSM evaluates every order item in the source order component against the conditions that you define for the decomposition rule. If an order item passes all specified conditions, OSM includes the order item in the target order component.
Unlike many other OSM modeling entities, decomposition rules are not directly referenced by other parts of the model. OSM selects decomposition rules by matching the source and target order components of the decomposition rule to the order components in the orchestration stages in the orchestration sequence.

See also executable order component.

default process
The first process that runs after an order is created. A default process can either be an orchestration process (which will be backed by a dynamically generated orchestration plan) or a workflow or workstream process.

delay
A process activity that specifies that a process stops until a condition evaluates as true. In OSM there are two types of delays, timer delays and event delays. A timer delay retries the evaluation of the rule at a fixed time interval. An event delay retries the evaluation of the rule only when order data changes.

dependency
A relationship in which a condition related to one order component or order item must be satisfied before another order item can be processed. For example, it may be necessary to perform provisioning before billing can occur for the same order item. Dependencies can have the following relationships:
- Between different order components for the same order item.
- Between different order components for different order items.
- Between order items across orders.
- Time-based dependencies.

See also inter-order dependency, intra-order dependency.

Design Studio
The software application used to design, configure, and deploy OSM Order Management cartridges into OSM environments. Design Studio is based on the Eclipse Integrated Design Environment (IDE). OSM plug-ins provide the specific screens (editors), validation logic, and cartridge-build functionality that allow users to create and configure OSM cartridges.

distributed order template
A structure data type that is available only for order item specifications and (if you are using the order transformation manager) the dynamic parameter property for the order item. An order item that is being used as a transformed order item must use the distributed order template.

entity
A functional unit created and edited in Design Studio; for example, tasks, processes, behaviors, projects, and notifications. Entities are collected in a cartridge to deploy in a run-time environment to support your business processes (for example, you deploy cartridges to OSM run-time environments.

Entity names must be unique by entity type. For example, you cannot name two task entities with the same name. However, you can create identical names for different entity types. For example, you can model a task entity and a process entity with the name AddDS.
Entity names are sometimes defined just by a simple name (the filename of the entity), and sometimes by a simple name and a namespace URI. In the case where a namespace URI is provided, it is the combination of the URI plus the simple name that gives the entity its unique name.

See also data element.

event delay
See delay.

executable order component
An order component with an associated process. Typically this is a component decomposed to its final level of granularity. Executable order components are generated during the last orchestration stage in an order.
See also order component and decomposition.

expected duration
The amount of time an order, or some part of the order (order component, fulfillment pattern or task), is expected to take to complete processing.

expected start date
The date on which an order is expected to start being processed. Expected start date is determined for orchestration orders by calculating the expected order duration and factoring this in with the requested delivery date for order items on the order.

external fulfillment state
The status returned from a fulfillment system to an order component. This may be the exact status returned by the system, or automation may be used to translate the status before it is put on the order. It is a key input into a fulfillment state mapping.
See also fulfillment state.

fallout
The failure of an order during processing. Fallout occurs whenever an order encounters a situation that prevents it from being processed. Causes for fallout include missing data or the inability to access a fulfillment system. Fallout management includes detecting, investigating, and resolving failed orders.

fallout exception
A mechanism initiated from the OSM Task Client to interrupt or stop an order, or to redirect it to any task in the same process or any other process. A fallout exception halts an order at a particular task in order to correct an error caused by a previous task.

fallout management
A process that includes detecting, investigating, and resolving failed orders. Administrators perform fallout management with the OSM Web clients. The clients allow them to search for failed orders, identify the reason for the failure by viewing order details, and resolve dependencies to allow order processing to proceed. In cases where the dependency cannot be resolved, you can cancel or terminate the order.

follow-on order
An order that is submitted to modify a completed order. Follow-on orders are not processed until their order-item dependencies on the in-flight orders allow them to proceed.
fulfillment
Operations that fulfill a customer’s order. This may be providing, modifying, resuming or canceling a customer’s requested products and services.

fulfillment function
Fulfillment functions are operations that must be performed to process an order item; for example, initiating billing, shipping, or performing installation. Fulfillment functions are defined in order component specifications created in Design Studio.

fulfillment modes
An entity that represents the intent of an order. For example, the fulfillment mode could indicate whether the order is intended for qualification, delivery to fulfillment systems, testing and so on. Every customer order can specify a fulfillment mode.

Different fulfillment modes will have different orchestration sequences. If OSM receives two identical incoming customer orders with different fulfillment modes, it generates a different orchestration plan for each order. The two plans include different executable order components with different dependencies among order items.

fulfillment pattern
An entity that includes the fulfillment function order components and dependencies required to fulfill a product order. Each order item in an order is mapped to a fulfillment pattern. OSM uses the fulfillment pattern to determine the necessary fulfillment functions, order components, the dependencies that exist between them, and dependencies on other fulfillment patterns to generate an orchestration plan.

fulfillment state
The state of an order or order item aggregated and translated from status values returned by external systems. This state can be used to provide status visibility to upstream systems and to users by using the Order Management Web client.

See also common fulfillment state, composite fulfillment state, external fulfillment state, and mapped fulfillment state.

fulfillment state map
The Design Studio entity that contains both the definition of common fulfillment states and fulfillment state mappings. A common fulfillment state defined on one fulfillment state map is available to all fulfillment state mappings in the workspace.

fulfillment state mapping
The Design Studio entity that maps external fulfillment states to values from the common fulfillment state list. The resulting fulfillment state is referred to as a mapped fulfillment state.

fulfillment system
A participating system in an order management solution. Fulfillment systems can include billing, provisioning, activation, shipping and workforce management systems, among others. OSM interacts with fulfillment systems when processing orders.

fulfillment topology
The arrangement of network elements, processes, systems, and software used to fulfill a customer order. The fulfillment topology represents the types and instances of fulfillment systems involved in fulfilling an order. For example, all of the Business
Support Systems (BSS) and Operational Support Systems (OSS) that participate in order capture and order fulfillment represent the fulfillment topology.

**future-dated order**

An order that has a requested delivery date that is later than the current date and time. For example, a customer order to have a new VoIP service added at the beginning of the next month is a future-dated order. OSM uses the order orchestration plan to calculate the order start date of future-dated orders so the order can be completed by the time the customer wants it.

See also expected duration and expected start date.

**in-bound order**

See customer order.

**in-flight changes**

Changes that are made to an order that is being processed.

**in-flight order**

Any order that is not in a closed state (Closed or Aborted). An in-flight order still has the potential for further work to be performed on it.

**inter-order dependency**

A dependency between order items in different orders.

**intra-order dependency**

A dependency between order items in the same order. An intra-order dependency can refer to external information, but not to data in other orders.

**IP Service Activator**

Internet Provider Service Activator. An Oracle product used by communication service providers to define and fully automate the activation of services on large-scale multi-vendor IP networks. IP Service Activator delivers end-to-end network control and enables real time reaction to new service and customer demands.

**line item**

See order line item.

**manual task**

Tasks performed by OSM operations personnel using the Task Web client. See also automated task.

**mapped fulfillment state**

The fulfillment state that results from a fulfillment state mapping.

**mapping rule**

The Design Studio entity that enables you to map original order items to transformed order items and ensure that the correct data elements from the customer order are added to the transformed order items.

See also order transformation manager.
**metadata**
Data definitions for entities such as processes, states, and rules modeled in Design Studio. OSM uses metadata to determine how to process order data. For example, OSM uses metadata from the fulfillment pattern to determine how order items are to be grouped into order components in an orchestration plan.

**mnemonic**
A synonym for an entity name. Mnemonic is a legacy term for OSM. The proper name is entity name.

**multi-instance data element**
A data element that is permitted to have more than one instance. For example, you configure the ControlData/OrderItem structure as a multi-instance data element so that OSM can create an instance of the structure for every order line item extracted off the in-bound customer order.

**namespace**
1. An XML namespace is a method for uniquely naming elements and attributes in an XML document. Attributes and elements are identified by a fully qualified name that consists of a namespace name paired with the local attribute or element name.

2. An OSM entity namespace is a method for uniquely naming OSM entities across projects. Fully qualified OSM entity names consists of a namespace name paired with the local entity name. OSM entity namespaces allow different work groups of Design Studio users to create different entities without concern for name contention. Services can be implemented independently by a different teams, then deployed into a single OSM run-time environment.

Not every OSM entity has a separate namespace (example: tasks and processes). For these types of entities, a unique name is created by attaching the cartridge name and version number to the entity name.

3. An OSM cartridge namespace is a method for uniquely naming OSM cartridges. This allows you to identify what cartridge is deployed in an environment. For example, if you are diagnosing an order failure, it's useful to know the logic and configuration of the cartridge that processed that order. Fully qualified cartridge names consist of a namespace name paired with the cartridge name.

You can view namespaces and other details about an entity or cartridge through Design Studio.

**notification**
Messages sent by OSM to alert users of order problems (jeopardy notifications) or changes to an order's state, status or data (event notifications). By default, OSM sends most types of notifications to the Task Web client Notifications page. You can also specify that notifications be sent by e-mail.

**Oracle WebLogic Server**
Oracle's application server for building and deploying enterprise Java EE applications. The Oracle WebLogic Server hosts the OSM server, OSM integration, and related interfaces.

**orchestration**
The process OSM uses to manage the fulfillment of a complex order. Order fulfillment often requires interaction with many fulfillment systems. Various dependencies may require that these interactions be run in a specific order to ensure that order items are
sent to the proper systems, and that the required steps, in the proper sequence are run.

**orchestration order**

An order that requires an **orchestration plan** for fulfillment. Orchestration orders contain **control data** for an orchestration plan. The default process for an orchestration order is an orchestration process. See **process-based order**.

**orchestration plan**

A dynamically generated plan that is used to manage the fulfillment of an order. Order fulfillment often requires interaction with many fulfillment systems, and various dependencies may require that these interactions be run in a specific order. The orchestration plan includes the order, **order component**, and the type and the sequence of order component execution. An orchestration plan is generated for each order based on the **metadata** defined for the type of order being processed.

For example, an order is captured by Siebel CRM and is sent to OSM for processing. Using the recognition rules and other entities provided by the OSM cartridges in the Order to Cash Integration Pack for OSM, 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.

**orchestration sequence**

A set of orchestration stages for an order. Orchestration sequences specify the set of orchestration stages for an order. Orchestration stages and sequences together define how an order is decomposed.

See also **decomposition, orchestration stage**.

**orchestration stage**

A step in an orchestration sequence used to decompose an order and create an orchestration plan.

See also **decomposition, order**.

**order**

An order in the OSM format. You model orders by creating order specifications in Design Studio.

There are many order variants including:

- customer order
- follow-on order
- future-dated order
- in-bound order
- in-flight order
- orchestration order
- process-based order
- revision order
- service order

**order component**

A collection of **order items** that can be processed together because they meet some common criteria as determined by an orchestration stage. Order components are
modeled in Design Studio, based on factors such as a function that needs to be performed, the systems that need to perform that function, and what other items can be processed in the same group.

See also decomposition, executable order component, orchestration stage.

**order component ID**
An ID associated with an order component that can be used in decomposition. When implementing fulfillment systems; for example, you can configure OSM to achieve decomposition by using decomposition rules or by using the component IDs of the order items. For example, a decomposition rule can select order items from fulfillment system order components and group them into an order component to create a single bundle. OSM can then use an order component ID calculation to generate distinct bundle instances.

**order data**
The data that is used for fulfilling an order; for example, a customer name and address.

**order data key**
Uniquely identifies a data element or structure in an order by differentiating the data element or structure based on a data element value. Order data keys are important when identifying order data changes during compensation and for multi-instance data elements.

**order definition**
See order specification.

**order duration**
See expected duration.

**order entity**
See order specification.

**order fallout**
See fallout.

**order fulfillment state composition rule set**
The Design Studio entity that enables you to aggregate and evaluate the fulfillment states of root-level order items and compose them into a single composite fulfillment state for the entire order.

**order header**
Orders typically consist of two parts: an order headers containing information that is applicable to the entire order such as the customer name and address, and order line items such as the products, services, and offers requested by the customer and the action to be performed on them (Add, Suspend, Delete, Move, and so on).

**order item**
An order line item transformed so that it can be processed in OSM. Each order item includes the action required to implement it, such as Add, Suspend, and Delete. Order items are decomposed into order components based on shared characteristics defined in a fulfillment pattern.

See also order component.
order item fulfillment state composition rule set
The Design Studio entity that enables you to aggregate and evaluate the fulfillment states of order component order items and child order items and compose them into a single composite fulfillment state for the entire order item.

order item parameter binding
The Design Studio entity that enables you to map incoming order item parameters to parameters on a conceptual model entity. You can create multiple order item parameter bindings for different types of parameters on the incoming order. For example, you could create one binding for parameters that are stored as name-value pairs, and another binding for more strongly typed parameters. See Design Studio Concepts for more information about conceptual model entities.

See also order transformation manager.

order key
Unique value that enables the system to match incoming revision orders to the corresponding OSM order. If the order key matches an order that is currently in progress, the order is considered to be a revision that amends a base order. For example, you can specify to use the customer reference ID as the order key. In that case, when OSM processes an order, it looks for previous orders that have the same customer reference ID, and amends it.

The order key can be any data or combination of data associated with the order. It is configured in Design Studio as an XPath expression to a data element that will uniquely match an amended order to its corresponding OSM order. For example, you might specify a customer reference ID as the following Xpath expression: root/Cust_Ref_ID

order life cycle
The sequential states through which an order passes and the transactions it undergoes from the time it is received in OSM until the time it is resolved. States include Not Started, In-progress, Suspended, and Completed. Each order state is associated with a set of transactions that can be performed while the order is in that particular state. Transactions include Update Order, Cancel Order, Complete Task, and Raise Exception. The life cycle of an OSM order is governed by the order state model and order life cycle.

order life-cycle policy
A set of policies that controls the states in which an order can be, and the transactions allowed in those states. The order life-cycle policy also determines which roles can perform which transactions. For example, while an order is in the In Progress state, you might want your Customer Service role to be able to perform the Update Order, Cancel Order, and Suspend Order transactions, while your Fallout role is able to perform the Raise Exceptions transaction. Every order type you create must be associated with an order life-cycle policy.

order line item
Specific items such as individual products, services, and offers on an incoming customer order. OSM transforms order line items into order items.

Order Management Web client
An OSM web application that displays an order's orchestration plan, including dependency, orchestration stages, order components, order items, and processes. The Order Management Web client is used by fallout administrators responsible for
locating orders with errors, determining the causes of failures, and taking the necessary corrective actions; operations and management personnel who monitor the progress of orders; and orchestration plan designers who can use this application to test and validate orchestration-based orders during the modeling and implementation of OSM solutions.

**order priority**
A value that OSM uses to determine which orders should be given more processing resources. OSM uses order priority to determine the next thing to be done. Orders with higher priority will be processed before orders with lower priority. In situations where resources are constrained (for example, the system is using all available CPU, memory, or other resources to process orders), orders with higher priority will process faster than orders with lower priority.

**order recognition**
The process of determining the type of an incoming customer order so it can be mapped to an **order specification** in OSM. During order recognition, OSM steps through an ordered list of **recognition rules** to determine which rule applies to the customer order. Each rule is associated with an order specification.

**order reference number**
A value associated with an order specified in one of the OSM Web clients or OSM APIs. OSM uses an order reference number as an identifier to external systems. Reference numbers can be used as keys to correlate orders between systems.

**order specification**
An order **entity** defined in Design Studio. The order specification is the central entity in OSM. It defines the basic information OSM requires for it to be able to process orders. It specifies such things as what data is allowed in an order (order template), what are the range of order priorities, whether amendments are allowed and how they are processed, how to handle jeopardy, fallout, permissions and so on.

Other entities such as tasks, processes, and notifications require that you specify an order specification to which it relates. Order specifications can inherit from other order specifications, and multiple order specifications can be modeled and can exist simultaneously. Also known as an order definition and order entity.

**order state**
OSM processes each order within a set of **order states**. A state is the condition of the order. For example:

1. An order is created in the Not Started state.
2. When processing begins on the order, the state changes to the In Progress state.
3. When the order is complete, it transitions to the Completed state.

See **order life cycle**.

**order state transition**
Changes from one order state to another order state as a result of a transaction. Each order state has a set of allowable transitions. For example, when an order is completed, it transitions from the In Progress state to the Completed state.

**order template**
A part of an order specification that defines what order data OSM will use to process and fulfill the order. For example, the order template defines the data required for
order items as well as the data required in an order header. Create or modify order templates by adding data elements from one or more data dictionaries.

**order transformation**

The manipulation and enrichment of the structure and contents of a customer order through a set of rules. Transformation rules are defined as part of recognition rules, and are based on applying a series of XQueries to the in-bound order.

Three types of transformation rules are available: order priority rules, which define the priority of the order in relation to others; order reference rules, which define the order reference number; and order data rules, which add to or modify incoming customer order data. Order items on the order can also be transformed using the order transformation manager.

**order transformation manager**

An OSM feature that enables you to transform order items in a consistent, data-driven, and traceable manner. It consists of order item parameter bindings, transformation sequences, transformation managers, mapping rules, and (if fulfillment state processing is defined) transformed order item fulfillment state composition rule sets.

**order validation**

A process that occurs during order recognition that validates that an order is syntactically correct. When an inbound order is recognized, OSM validates it based on validation rules defined in the order recognition rule.

For example, a validation rule can determine that all mandatory fields are populated, that valid characters (numeric or alphanumeric) are used for fields, and that the order has a valid status code such as Open. Validation rules are implemented as XQuery expressions. Each node in the expression must evaluate to true for validation to pass.

**original order item**

In the context of the order transformation manager, an original order item is the order item that has not yet been transformed. Sometimes called a source order item.

**OSM order management Web Services API**

The primary interface for external systems to OSM. The OSM order management Web services provide for in-bound order operations such as creating, managing, retrieving, updating, or canceling an order. Web services are Web APIs that support interoperable machine-to-machine interaction over a network such as the Internet. Web services run on a remote system hosting the requested services such as OSM. Web service interfaces are described by the Web service definition language (WSDL).

**OSM security callback**

A callback interface that allows you to generate an audit trail log of users before they gain access to order data that is considered sensitive. The security callback interface is designed to intercept order access from defined functions such as getOrder, XML API WorkList.Request, and Task Web client Order Data History page.

**OSM server**

The server that manages OSM run-time functionality, including in-bound order operations and outbound communications with external systems. The OSM server is deployed on Oracle WebLogic Server.
**OSM Web clients**

The two OSM GUI applications called the **Order Management Web client** and the **Task Web client**. The Order Management Web client displays an order's **orchestration plan**, including **dependency**, orchestration stages, order components, order items, and processes. The **Task Web client** is used for monitoring and managing the **tasks** in an order.

**point of no return**

The point in the orchestration of an order item after which revisions can no longer be accepted.

**process**

A sequence of **tasks** and **subprocesses** that need to be carried out to fulfill all or part of an order. For example, an ADSL fulfillment process could include the following tasks that can take place over a number of days: assign a port, ship modem to customer, activate DSLAM, send customer survey, and verify order. The process includes definitions of the relationships between tasks and the sequence in which they are run.

**process-based order**

An order that does not include an orchestration plan. A process-based order typically handles a provisioning process. See **orchestration order**.

**processing granularity**

Decomposition groups **order items** into optimally executable **order components**. For example, if monthly fees, VoIP adapter, and VoIP phone are all billed by the same billing system, they can be grouped into a single executable order component. This is called **processing granularity**. See **decomposition**.

**product catalog**

A data repository that stores and retrieves information about your products including price lists, discount lists, and unit groups. The product catalog is used to create customer orders. Sometimes called the **master product catalog**. The product catalog exists on the **CRM** or other order-source system. An example of a product catalog is the Siebel Sales Catalog or Oracle Product Information Management Data Hub for Telco.

**product specification**

Groups of related products that share common attributes. For example, suppose you sell products for three levels of DSL service. Though there are different values to differentiate the service levels, the products are structurally identical and provisioned by the same system; they are variations of the basic DSL service. As a result, a single product-specification to fulfillment-pattern mapping can be used for all of them.

Product specifications may be defined in the Design Studio conceptual model, or imported as conceptual model product specifications from a product catalog such as the Siebel Sales Catalog or Oracle Product Information Management Data Hub for Telco. Existing product specifications that were created in OSM are still supported, but new ones should be created in the conceptual model.

**provisioning**

Providing the data necessary for enabling a service, with the actual enabling done by activation.
recognition rule
Rules that enable OSM to validate a customer order and transform it into an OSM order format.

related order
An order that contains order items that depend on another order.

reporting interface
A tool for generating reports about OSM orders, tasks, and notifications. The Reporting Interface augments the reports that are available through the OSM Web clients. See OSM Reporting Interface Guide for more information.

requested delivery date
The date on which an order is requested to be delivered.

revision order
An order that modifies a previously submitted order that is still being processed. For example, a customer may want to switch to a higher level of service before an order is completed. Revision orders may require compensation. The system can process revision orders until the original order reaches its point of no return. A revision order is sometimes called a supplemental order.

See also follow-on order.

role
A set of permissions to access to functions in the Task Web client and Order Management Web client that can be assigned to users. Functions include viewing reports, assigning tasks, and querying orders. In addition to granting OSM Web client permissions, you can also grant permissions at the order and task levels. Roles are created with the OSM Administrator application or Design Studio. The Administrator application refers to roles as workgroups, although they are both the same thing.

rule
Rules are defined as part of an order specification to work on data in the order. Rules are used in many OSM activities to evaluate conditions and determine next process steps. For example, you can specify to delay the next task in a process until a specified data element includes a certain value.

rule engine
A processing component of OSM that evaluates rule and timer delays for transition to the next task. The engine is implemented as one or more Oracle database jobs. The rule engine is configured as one or multiple jobs to improve performance.

security callback interface
See OSM security callback.

service order
An order received by an OSM instance acting in the service order management role.

service order management
The OSM system role that serves as part of a service fulfillment stack, working with inventory and activation systems to fulfill services in one or more network domains. An OSM instance can operate in a service order management role or in a central order management role. Sometimes called local fulfillment.
**source order item**
See original order item.

**specification**
See order specification.

**subprocess**
A process started by another process. A subprocess is used to organize any large process into smaller more re-usable pieces. It includes one or more tasks and realizes an executable order component.

**target order item**
See transformed order item.

**task**
An individual step that is required for the processing of an order. Tasks are defined by the order specification in Design Studio, and can be either a manual task (performed by human action) or automated task (performed by an automation plug-in).

Tasks are run as part of any process-based order. For example, a completely process-based order consists of a series of steps which are defined by tasks. By contrast, an orchestration order starts out as orchestration plan. Each order component within the orchestration plan is implemented by a subprocess which in turn contains tasks and other subprocesses.

**task state**
A state describing the milestones of a task in a process. The task state also determines how it can be worked on. OSM provides the following task states: Received, Assigned, and Accepted. You can, however, create your own task states. For example, you can define a Suspended task state to indicate the progress of automated tasks, or you can define a Completed task to indicate that user is finished with the task and the order is ready to move to the next task in the process.

**task status**
A representation of how a task can transition to the next step in a process. The task status shows how the task transitions in a process; for example, if the task transitioned the process to the next task, or if it caused the process to fail. Changing the status of the task determines the next step in the order process. The statuses that you define appear as task transition options in the OSM Web clients.

For example, if you have a task called Assign Port, and the two statuses are Port Available and Port Unavailable, the status determines whether the process can proceed to the next task. Task status also controls notifications, so when the status is Port Available, OSM can send a message saying Successful.

**Task Web client**
An OSM GUI application used for monitoring and managing the tasks in an order. This application is typically used by order processing personnel to ensure that all the tasks are completed. It is also used by order fallout managers. You can also suspend and resume orders, cancel orders, and create orders manually.

**timer delay**
See delay.
**transaction**

An action taken by the OSM system on an order. For example, the Suspend Order transaction stops all processing on the order and transitions the order to the Suspended state. Also called an order state transaction

Some other transactions are Abort Order, Complete Task, Process Amendment and Raise Exception. Most transactions perform transitions that change the state of the order to a different state. However, some transactions do not perform a transition to another state. For example, the Update Order transaction can make changes to an order without changing the order's state.

**transformation manager**

The Design Studio entity that enables you to select the transformation sequences for the service domains within a provider function. See Design Studio Concepts for more information about service domains and provider functions.

See also order transformation manager.

**transformation sequence**

The Design Studio entity that enables you to define transformation stages and the sequence in which they should be processed.

See also order transformation manager.

**transformation stage**

A component of the transformation sequence that defines the source and target order items, and the relationship between them, for one step of the transformation sequence.

**transformed order item**

In the context of the order transformation manager, a transformed order item is the order item that is the result of the transformation. Sometimes called a target order item.

**transformed order item fulfillment state composition rule set**

The Design Studio entity that enables you to perform special fulfillment state processing relating to the order transformation manager. You can perform all of the functions in the order item fulfillment state composition rule set and also to define conditions for updating the original order item associated with the transformed order item.

**trouble ticket (TT)**

A request to the trouble ticketing system indicating that an error occurred during the processing of an order. Different from a fallout task in that trouble tickets come from front-end systems such as Siebel CRM.

**unresolved dependency**

A dependency with at least one unmet condition.

**WebLogic Server**

See Oracle WebLogic Server.

**workgroup**

A group of users with assigned permissions to access to functions in the Task Web client and Order Management Web client. Functions include viewing reports, assigning tasks, and querying orders. In addition to granting OSM Web client...
permissions, you can also grant permissions at the order and task levels. Workgroups are created with the OSM Administrator application or Design Studio. Design Studio refers to workgroups as roles, although they are the same thing.

**worklist**

A list of manual tasks assigned to OSM operations personnel who use the Task Web client to manage orders. When an order arrives at a task, it is added to the worklist of all the members of all the workgroups assigned to work on that task. Users can select a task from their worklist to view the assigned task in the order process. Worklist also refers to the main page in the Task Web client used for managing orders.