Oracle® Communications Service Catalog and Design

Design Studio Modeling OSM Processes

Release 8.1
F96246-01
July 2024

ORACLE

Oracle Communications Service Catalog and Design Design Studio Modeling OSM Processes, Release 8.1

F96246-01

Contents

7 Working with Tasks

10 Working with Orders

11 Working with Behaviors

12 Working with Jeopardy and Event Notifications

13 Packaging and Deploying OSM Cartridges

A Automation and Compensation Examples

Preface

This document contains information about the procedures and tasks that are necessary to configure and deploy Oracle Communications Order and Service Management (OSM) process entities and cartridges using Oracle Communications Service Catalog and Design - Design Studio.

Audience

This guide is intended for business analysts, architects, development managers, developers, and designers who are responsible for system integration or solution development involving the Oracle Communications operational support systems applications.

Ideally, you should be knowledgeable about your company's business processes, the resources you need to model, and any products or services your company offers.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Diversity and Inclusion

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

1

Getting Started with Design Studio for OSM Processes

When modeling orders, you can use Oracle Communications Service Catalog and Design - Design Studio to define new order fulfillment processes, tasks, order types, and order policies that control the order type life cycles. Design Studio enables you to test your Oracle Communications Order and Service Management (OSM) configurations and resolve configuration problems.

The OSM provisioning configurations that you create in Design Studio enable you to provision orders across multiple service and network domains. Design Studio enables you to define, run, and control service orders for any domain, including Broadband access (for example xDSL, fixed wireless, WiMAX), IPTV, VoIP, Voice, IP-VPN, Mobile, and emerging services. You create these configurations in Design Studio by defining and relating objects at different levels of abstraction.

> **Note:**
>
> If you upgrade to Order and Service Management 7.x from a prior version of OSM and your cartridges were developed with OSM Administrator tool, Oracle recommends that you migrate your cartridges into Design Studio. Use Design Studio as the tool to design and deploy OSM 7.x cartridges. The recommended migration procedure, common migration issues, and issue resolutions are documented in release 7.3.2 *Design Studio Order and Service Management Cartridge Migration Guide*, which is available in the Oracle Help Center:
>
> `http://docs.oracle.com/en/industries/communications/design-studio/index.html`

See the following topics when getting started with Design Studio for OSM processes:

- About Order Modeling Users and Tasks
- Reviewing Design Studio Sample Cartridges
- Creating a Cartridge for Orders That Use Processes

About Order Modeling Users and Tasks

The following is a list of roles and the tasks each role typically performs in Design Studio for OSM processes:

Business Analyst Tasks

Business analysts are responsible for describing the features and services associated with marketing products and communicating this to the users responsible for building these products. The business analyst may simply name and describe products and pass this product

description to a configuration modeler, or may perform basic configuration modeling tasks in Design Studio. This involves the following tasks:

Task	For More Information
Configure OSM cartridges	"Working with OSM Cartridge Projects"
Model processes	"Working with Processes" "Working with Order Lifecycle Policies"
Model roles	"Working with Roles"

Configuration Modeler Tasks

OSM modelers are responsible for creating, modeling, refining, building and deploying an OSM cartridge. This involves the following tasks:

Task	For More Information
Create OSM cartridges	"Working with OSM Cartridge Projects"
Model processes	"Working with Processes" "Designing Subprocesses" "Designing Workstream Processes"
Model roles	"Working with Roles"
Model Tasks	"Working with Manual Tasks" "Working with Automated Tasks"
Model Orders	"Working with Orders" "Working with Order Lifecycle Policies"
Refine processes	"About Task States and Statuses"
Build, deploy, and undeploy to and from development environments	"Packaging and Deploying OSM Cartridges"

Deployment Manager Tasks

Deployment managers are responsible for build management and are usually advanced users of the version control system. Deployment managers should be familiar with the tools that Eclipse provides to interface with the version control system and understand how to package items from source control and build and deploy configurations to OSM environments. Activities performed by deployment managers in Design Studio include:

Task	For More Information
Build, deploy, and undeploy to and from development environments	"Packaging and Deploying OSM Cartridges"
Configure the development environment	"Packaging and Deploying OSM Cartridges"

See the Eclipse documentation for more information about version control.

Cartridge Developer Tasks

Cartridge developers assist OSM configuration modelers in modeling and refining an OSM cartridge. This involves the following tasks:

Task	For More Information
Model the Data Dictionary	"Modeling Data"
Model Tasks	"Working with Manual Tasks" "Working with Automated Tasks"
Model Orders	"Working with Orders" "Working with Order Lifecycle Policies"
Refine the process	"About Task States and Statuses"

Reviewing Design Studio Sample Cartridges

Design Studio includes two OSM sample cartridges intended to provide examples of the type of configuration that you can model in a Design Studio OSM cartridge project. You can use these sample cartridges to familiarize yourself with OSM functionality, as a reference aid when first learning Design Studio, and as a starting point when modeling OSM cartridges. Each cartridge contains corresponding documentation to describe the cartridge content.

To open and review a sample cartridge:

1. Install the **Design Studio Samples** feature from your internal update site.

   For information about installing features, see the *Design Studio Installation Guide*.
2. From the **File** menu, select **New**, and then select **Example**.

   The New Example dialog box is displayed.
3. Expand the **Design Studio Order and Service Management Provisioning Examples** directory.
4. Select a sample cartridge.

   There are two OSM sample cartridges. Select one of the following:

   - Provisioning Broadband and Order Change Demo
   - Provisioning View Framework Demo
5. Click **Next**.
6. Select the example project to contain the sample cartridge.

   Do one of the following:

   - For the Provisioning Broadband and Order Change Demo, select **bb_ocm_demo**.
   - Provisioning View Framework Demo, select **view_framework_demo**.
7. Click **Finish**.

   The project is added to the Studio Projects view. You can expand the project directory to explore the entities included in the sample cartridge.
8. Access the sample cartridge documentation.

   Each cartridge is delivered with a corresponding file that contains documentation to describe the cartridge functionality. To access the documentation:

   a. From the **Window** menu, select **Show View**, and then select **Package Explorer**.

   b. In the Package Explore view, expand the sample cartridge project directory.

For example, expand either the **bb_ocm_demo** or the **view_framework_demo** directory.

c. Expand the **doc** directory.

d. Double-click the file to open the documentation.

> **Note:**
>
> To run **view_framework_demo** in an OSM environment, you must define your Design Studio language preference as **English (United States) [en-us]**. See "Defining Language Preferences" for more information.

Creating a Cartridge for Orders That Use Processes

The procedure below outlines the basic steps to create an OSM cartridge configuration in Design Studio.

After you create a cartridge project, you can model the process flow first or define your data first (see "Modeling Data" for information about defining data). The steps required to create the cartridge configuration are the same; only the starting points differ. For example, if you elect to model your process first, you start by creating a new cartridge project, then model the process in the Process editor. If you want to define data first, you can open the Data Schema editor to define as much data as necessary.

The following procedure demonstrates how to create an OSM cartridge, modeling the process first.

To create a cartridge for orders that use processes:

1. Create an OSM cartridge project.

   The OSM cartridge project is your working area for the OSM configuration. When you create a new cartridge project, Design Studio displays the new cartridge project in the Studio Projects view of the Design perspective. In addition to the newly-generated Project entity, the project contains entities for an order and for the Data Dictionary. See "Creating New Cartridge Projects" for more information.

2. Model the process.

   A process is a sequence of tasks that runs either consecutively or concurrently to fulfill an order or part of an order. Using the Process editor as a white board, you can create, sequence, and link the tasks that are required to implement the process flow. See "Working with Processes" for more information.

3. Model the roles.

   You can permit specific roles access to a standard set of functions in the Task web client. See "Working with Roles" for more information.

4. Model tasks and task data.

   A task is one step in a process. The data that you model for the task includes all of the data that the task requires to complete.

   You can define different types of tasks (manual or automation), create the task data, assign task states and statuses, add behaviors, and define other parameters as needed. You can associate the task with an existing OSM order or model a new order (and order template) for the task. See "Working with Tasks" for more information.

5. Model the order.

   You model the data for the order and add behaviors that affect the manner in which the data appears in the Task web client or Order Management web client. You can associate a default process, a creation task, and a life-cycle policy with the order, associate rules with the order, and permit specific roles access to the order. See "Working with Orders" for more information.

6. Refine the process.

   After reviewing your configuration, you may need to make changes, such as removing tasks from the process flow, renaming tasks, changing the data required at a task, updating the Data Dictionary, or adding additional states and statuses.

7. Package the cartridge.

   Packaging enables you to control which entities, libraries, and resources will be included in the cartridge when you deploy the cartridge to the OSM run-time environment. See "Packaging and Deploying OSM Cartridges" for more information.

8. Deploy and undeploy to and from development environments.

   You create an Environment project to contain the information required to connect to your run-time environment. You can deploy all of the data in your cartridge, or when possible, deploy only the changes to your cartridge project (using the Optimize Deploy feature). See "Packaging and Deploying OSM Cartridges" and "Deploying Cartridge Projects with Optimize Deploy" for more information.

2

Defining OSM Preferences

When modeling Oracle Communications Order and Service Management (OSM) cartridges in Design Studio, see the following topics:

Defining Language Preferences

Design Studio for OSM supports multiple languages for fields in the Task web client or the Order Management web client (through the use of the Information behavior). Use the language preference settings to define the languages that you intend to use in your OSM cartridges, and the language with which you prefer to work. You define language preferences in the Oracle Design Studio Preferences dialog box.

To define OSM language preferences:

1. From the **Window** menu, select **Preferences**.

   The Preferences dialog box is displayed.
2. Select **Oracle Design Studio**.
3. Click **New**.

   The Add Language dialog box is displayed.
4. Select a language from the available options.
5. Click **OK**.

   Design Studio adds the language to the Languages group.
6. (Optional) Define the language display priority.

   When multiple languages appear in the Languages group, use the **Up** and **Down** buttons to reposition the language display priority. The language display priority controls the order in which the languages appear in Design Studio language drop-down lists.
7. (Optional) Click **Remove** to delete a language from the Languages group.
8. In **Preferred Language**, select the language from your language list in which you prefer to work.
9. Click **OK**.

   Design Studio saves your language preferences and closes the Preferences dialog box.

> **Note:**
>
> If you deploy or import into OSM cartridges that support languages that you have not included in your language group, the system displays a warning message indicating that the language is not currently supported. To include that language in your language list, you can return to the Language Preferences page and add it. Also note that you must specify the English [en] language setting to use the OSM sample cartridges.

Related Topics

Defining Information Behavior Properties

Defining Information Behaviors in Multiple Languages

Defining OSM Preferences

Defining Diagrammer Preferences

To define OSM diagrammer preferences, from the **Window** menu, select **Preferences**, then expand **Oracle Design Studio** in the Preferences navigation tree, and then select **Order and Service Management Diagrammer Preferences**.

Use the diagram preferences to define aspects of a diagram layout. Offset preferences control the spacing between elements in the diagram. Segment lengths control aspects of the link shape. Layout preferences provide control over incremental layout logic. The default settings will work well in most instances.

Select **Incremental Layout** if you want the system to consider the current coordinates of the diagram nodes and links when you click the Layout All Nodes or Layout Selected Nodes buttons in the Process editor. When enabled, **Incremental Layout** retains the relative order of nodes and links and attempts to create a new layout with similar positions.

The system considers the following settings only if you enable the **Incremental Layout** option:

Field	Use
Number of link crossing sweeps	Define the number of sweeps you want the system to use to remove instances of link crossing. The system uses a complex algorithm to resolve link crossings; while the number of sweeps influences the number of resolved crossings, there is not a direct relationship between the number of sweeps and the number of resolved link crossings. In some instances, it may be useful to decrease the number of sweeps to resolve more link crossings.
Link crossing reduction	Select to reduce the number of crossing links during incremental layout. When this field is enabled, the system preserves the level structure and relative order of the nodes but reorders them within the level structure to avoid link crossings.
Long link crossing reduction	Select to reduce the number of long crossing links during incremental layout. When this field is enabled, the system preserves the level structure and the relative order of the nodes but reroutes link bends to avoid link crossings.

Field	Use
Allow node level repositioning	Select this option if you want the system to consider the original node positions during incremental layout. A primary objective of incremental layout is to achieve a balanced diagram. To achieve a balanced diagram, the system may place nodes far from their original positions. You can enable this field to ensure that the system places nodes as close to the original position as possible. This decision, however, can result in an imbalanced layout. **Note:** The system does not consider this option if you have enabled either the **Link crossing reduction** or **Long Link crossing reduction** fields. The **Allow node level repositioning** and crossing reduction fields are mutually exclusive.
Level positioning range	Define the acceptable range (in pixels) from the original node position that the system can consider when repositioning nodes during incremental layout. **Note:** The system considers this option only when you enable the **Allow node level repositioning** field.
Level positioning tendency	Define the percentage of the level positioning range to which the system should adhere when moving nodes from their original position. Define higher percentages to ensure that the nodes remain closer to their original positions, and lower percentages to enable the system more leverage for achieving a balanced diagram. A position tendency of **0** effectively disables this option. **Note:** The system considers this option only when you enable the **Level positioning range** field.

Related Topics

Defining Language Preferences

Defining Order and Service Management General Preferences

Defining OSM Preferences

Defining Order and Service Management General Preferences

To define OSM general preferences, from the **Window** menu, select **Preferences**, then expand **Oracle Design Studio** in the Preferences navigation tree, and then select **Order and Service Management Preferences**.

Defining Build-and-Deploy Modes for Automation Plug-ins

This field is only relevant for cartridges targeted to OSM server versions between 7.0.3 and 7.2.4.x. Prior to 7.0.3, only Legacy mode is used. From 7.3 onward, only Optimized mode is used.

In the **Automation plug-in Build and Deploy Mode** field, specify whether you want OSM to process all automation plug-ins in a common (**oms.ear**) EAR file (by selecting **Optimized (Default)**), process each automation plug-in in its own EAR file (by selecting **Legacy**), or build the automation components required for OSM to process automation plug-ins in either Optimized or Legacy mode according to the automation plug-in dispatch mode defined on the OSM server (by selecting **Both (Allow server preference setting to decide)**). For more information on automation plug-in build-and-deploy modes, see "Defining Build-and-Deploy Modes for Automation Plug-ins".

Setting the Problem Marker Severity for Automation Task Name to Plug-in EJB Discrepancies

In the **Automation Plugin EJB Name Compliance Level** list, specify whether you want Design Studio to display an Error or Warning problem marker when there is a discrepancy between the automation task entity name and an associated external plug-in EJB name. If such a discrepancy occurs, you can use the Quick Fix option from either the warning or error problem marker to revert the plug-in EJB name to match the automated task name.

Defining Deploy Properties

When installing Design Studio, you can use the **install.bat** script to install the Design Studio features and configure the environment automatically. If you use this script, it will automatically install the necessary libraries and configure the fields below. See *Design Studio Installation Guide* for more information about installing Design Studio and using the script.

If you install Design Studio without using the script, or if you want to change any of the values after the install, see the information in the table below.

Field	Use
WebLogic Home	Specify the installation directory for WebLogic (for example, **C:\Oracle\Middleware\wlserver**). If you are using Design Studio on a system that does not already have the correct version of Oracle Middleware on it, consult the software requirements in *OSM installation Guide* for information about the correct version of the application server to install, and then consult *Fusion Middleware Installation Guide for Oracle WebLogic Server* for information on installing the core WebLogic software. Do not install a WebLogic domain for Design Studio, which only needs the core software.
Java SDK Home	Specify the installation directory for the Java JDK (for example, **C:\Program Files\Java\jdk1.7.0_85**). Select the JDK in the WebLogic Server installation.
OSM SDK Home	Specify the installation directory for the OSM SDK (for example, **C:\OSM\SDK**). If you are using Design Studio on a system that does not already have the correct version of the OSM SDK on it, first ensure that you have unpacked the correct version of JBoss on your machine. Then run the OSM installer on your machine, selecting a **Custom Installation** and then selecting only the **SDK** and (if desired) **SDK Samples** options when asked to select components to install. See *OSM Installation Guide* for information about the correct version of JBoss to use and for more information about running the installer. You should specify the location of a version of the SDK that is compatible with the value for **Target Version** that you are selecting in your projects. See "Project Editor Properties Tab" for more information about the **Target Version** field.

Defining Orphaned Task Reference Preferences

In the **Delete Orphaned Task References with Activity** field, specify whether you want to delete related task entities when removing tasks from the Process editor. When you enable this option and delete tasks in the Process editor, the system checks whether deleted tasks are referenced elsewhere in the workspace. If no task references exist in the workspace, the system displays a list from which you can select related entities for removal.

Defining Order Template Inheritance Preferences

When you enable order template inheritance preferences, when an order is extended, the significance and keys defined on the order are inherited; that is, significance and keys are included in the information from the base order. In addition to keys and significance defined on the base order being inherited on the extended order, the significance of an inherited data element within the order template is also inherited from the OSM entity that contributes to it. If these preferences are not enabled, the significance of an inherited data element within the order template is inherited from the data schema rather than from the OSM entity that contributes to it.

Offering inheritance within the order template for inheriting significance and keys enables a level of inheritance that is more complete and increases development convenience. As such, enabling these preferences is recommended for new cartridge development. If you are upgrading existing cartridges, refer to the discussion on order template inheritance and upgrade impacts in *OSM Installation Guide* before enabling these preferences.

To define order template inheritance preferences:

1. From the **Window** menu, select **Preferences**.

   The Preferences dialog box is displayed.
2. In the Preferences navigation tree, expand **Oracle Design Studio** and then select **Order and Service Management Preferences**.
3. In the **Order Template Inheritance** area, define your order template inheritance preferences.

   When an inherited data element in the order template has its significance set as **Inherited**, enabling and disabling the preference for significance inheritance from order contributors works as follows:

   - Disabled: Significance is inherited from the data schema.
   - Enabled: Significance is inherited from the order template of the data contributor(s). Note that if the significance of the data contributor is set as **Inherited**, it inherits its significance from the data schema.
4. Click **Apply** and then click **OK**.

   Design Studio saves your preferences.

Related Topics

Defining Language Preferences

Defining Diagrammer Preferences

Defining OSM Preferences

Defining Orchestration Preferences

To define OSM orchestration preferences, from the **Window** menu, select **Preferences**, then expand **Oracle Design Studio** in the Preferences navigation tree, then select **Order and Service Management Preferences**, and then select **Orchestration Preferences**.

Use orchestration preferences to set options related to orchestration.

Field	Use
Product Specification Mapping Folder	Specify the folder in which Design Studio is to store product-specification-to-fulfillment-pattern mappings. When you import common model products or OSM orchestration product specifications, Design Studio creates (in the resources folder of the cartridge project) a **productSpecMapping** folder that contains all product-specification-to-fulfillment-pattern mappings. You can reference this folder location in XQuery expressions when you configure the order item properties for the orchestration fulfillment pattern. If you want to maintain the **productSpecMapping** folder at an external location (and reference the folder using a Data Instance), specify the location of the folder here. See "About XQuery Expressions for Mapping Product Specifications and Fulfillment Patterns" for more information.
Prompt To Create Orchestration Data Dictionary	Specify whether you want to be prompted to create the **OracleComms_OSM_CommonDataDictionary** model project in your workspace; this model project contains a predefined data schema which is recommended for modeling data structures used by orchestration. If you select the check box, Design Studio prompts you to create the model project in your workspace when you open or create an orchestration entity. See "About Autogeneration of Order Component Control Data" for more information about this model project.

Defining Web Browser Preferences

When you submit test orders from Design Studio to a run-time environment, Design Studio opens a browser window in the Design Studio editor area and displays the Task web client log-in window. For the Task web client to work correctly from inside Design Studio, you must configure Design Studio to use a supported external browser.

To specify an external browser:

1. From the **Window** menu, select **Preferences**.

   The Preferences dialog box is displayed.
2. In the Preferences navigation tree, select **Web Browser**.
3. In the **Web Browser** area, select **Use external web browser**.
4. In **External web browser:**, select a supported external browser, such as Internet Explorer.
5. Click **OK**.

   Design Studio saves your preferences.

Related Topics

Submitting Test Orders to Run-time Environments

Additional Configuration To Support OSM Versions 7.5.0 Onwards

In order to enable Design Studio to perform feature configuration for OSM 7.5.0 or newer, an SDK library needs to be copied from the OSM SDK to the Eclipse installation.

To copy the SDK library to Eclipse:

1. In the OSM SDK, navigate to **SDK/OpenAPISDK/***version***/** and locate the **osm-openapi-sdk-full.jar** file.

   > **Note:**
   >
   > If this file does not exist, ensure that you are using the appropriate OSM SDK for your project.

2. Close any open instances of Eclipse or Design Studio.
3. Identify the Eclipse installation directory.
4. Copy the **osm-openapi-sdk-full.jar** file from OSM SDK to the *Eclipse_installation_directory***/plugins** directory.

   > **Note:**
   >
   > You can delete or move any pre-existing file with the same name into a separate backup folder.

5. Start Eclipse or Design Studio.

> **Note:**
>
> For tracing, auditing and debugging purposes, the version of OpenAPI SDK is in the **META-INF/MANIFEST.MF** of the **osm-openapi-sdk-full.jar** file.

3

Working with OSM Cartridge Projects

An Oracle Communications Order and Service Management (OSM) cartridge defines all of the entities and system interfaces required to fulfill an order in the OSM run-time environment.

The OSM cartridge is your working area for the OSM configuration. You store all simple data elements and structured data elements in data schemas. The combined data schemas are referred to as the Data Dictionary. You use subsets of data from the Data Dictionary to create order data templates. You use data from the order template to model specific tasks. You associate tasks with a process required to fulfill incoming customer orders. You can model different types of tasks, such as automated tasks and manual tasks, and define rules for tasks at the order level. You package all configured elements and components required for deployment.

When working with OSM cartridge projects, see the following topics:

- Creating New Cartridge Projects
- Closing Projects
- Opening Projects
- Managing Project Dependencies
- Working with Existing OSM Models
- Working with XML Catalogs
- Order and Service Management Project Editor

Working with Existing OSM Models

If you have existing OSM models, you can import the model into Design Studio as a single file or as multiple files. When importing models, Design Studio generates a cartridge project and maps all of the order's data into the equivalent Design Studio directory structure.

When working with existing OSM models, see the following topics:

- About Importing Design Studio Cartridges
- About Importing Cartridges Created in OSM Administrator
- About Import Summary Reports
- Importing Existing OSM Models

About Importing Design Studio Cartridges

You can import into Design Studio existing OSM models. When importing OSM models, Design Studio generates a new cartridge project and maps the metadata for each order into an equivalent Design Studio directory structure.

> **Note:**
>
> Cartridge dependencies must exist in the workspace before importing a cartridge project. For example, when importing an OSM project that has a dependency on a Model project, you must import the Model project first to avoid problem markers during the project build. Also, when importing cartridges that have specific languages defined in the OSM model, you must add those languages to your language preferences. See "Defining Language Preferences" for more information.

Creating Design Studio Projects

When you import XML models into a workspace, Design Studio creates the following projects to contain the data:

- A common data model project.

  The default name is **OracleComms_OSM_CommonDataDictionary**. This project contains the data schemas associated with the imported cartridges, as well as behaviors and data providers defined at the Data Dictionary level.

- An OSM project for the cartridge.

  These projects contain orders, processes, tasks, order life cycle policies, custom automation plug-ins, and data providers.

For example, consider that you are importing an XML model that contains a cartridge called **DSLService**. Upon import, Design Studio creates the following projects:

- A Data Dictionary project.
- A cartridge project for **DSLService**.

> **Note:**
>
> Design Studio does not import entities that are defined in the import model but not associated with any orders. For example, if the import model contains a task that is not used in any processes or referenced by any of the orders, Design Studio will not import the task. Additionally, Design Studio does not import entities that are not supported (such as test categories or rules), even if they are associated with orders.
>
> Design Studio logs all entities it does not import in the Import Summary report.

Importing Data Types

In Design Studio, you can define numeric data elements as type *int*, *double*, *float*, or *decimal*. OSM does not directly support these data types. When you deploy a cartridge containing these data types, OSM converts them to *numeric*. Conversely, when you import numeric types into Design Studio, they are converted to *decimal*.

> **Note:**
>
> Importing cartridges originally created in Design Studio, but exported from OSM using the XML Import/Export application is not supported.

Improving Design Studio Performance

To improve Design Studio performance during import and during modeling, you can separate large cartridges into multiple projects that exist in different workspaces. During the first import, you create the common Data Dictionary that all of the smaller projects will reference and select a subset of orders to add to the first project. Create a new workspace for subsequent imports and continue to add subsets of orders. When you are finished, Design Studio will include multiple projects in multiple workspaces, each sharing the same Data Dictionary, each with different subsets of orders from the original cartridge. For example, you might include all VoIP orders in a single workspace, all Internet orders in another, and so forth.

About Importing Cartridges Created in OSM Administrator

Design Studio replaces much of the functionality previously included in the old OSM Administrator application. OSM administrator functionality, for example managing workgroups, schedules, and calendars, is provided in the Administration area of the OSM Order Management web client. For information about migrating cartridges that were created in the OSM Administrator into Design Studio, see *Design Studio Order and Service Management Cartridge Migration Guide*.

The following table describes how entities defined in the Administrator and imported into Design Studio are mapped into the Design Studio environment:

Entity	Considerations
Cartridge Name and Version	At import, Design Studio preserves the namespace and version, using namespace as the project name, and version as the version number of the cartridge. You cannot import into Design Studio a cartridge with a name identical to an existing Design Studio cartridge (even if it is a different version), per workspace.
Data Dictionary and Master Order Template	Design Studio does not require that you build a master order template or explicitly model order data. As you model task data, the system automatically builds the order template. Design Studio maps all data elements to a Data Dictionary that you specify in the Import wizard, and maps order template elements to the Order editor **Order Template** tab.
Order Source	Design Studio imports the order source. It is visible in the Order editor **Details** tab.
Workgroups	Design Studio maps workgroups to roles. Permissions are visible in the Role editor and also in the Order editor and Task editor **Permissions** tabs.
Tasks	Manual and automated tasks appear as separate task entities in the project directory. If you are importing a cartridge that contains a task used by multiple order types and sources (using different views in OSM Administrator), Design Studio creates duplicate tasks, one for each order, and renames each task with the original task name concatenated with the order type and source. Design Studio updates all references to the task. If the task name is referenced in an automation map, Design Studio creates duplicate entries.

Entity	Considerations
Views	In Design Studio, views are not independent entities, but are implicit in the configuration of data nodes in tasks and order templates. Upon import, the data associated with an order or task appear in the Order and Task editors, respectively.
Processes	If you associate a process with multiple order type and sources using rules, Design Studio duplicates the process for each associated order type and source. On import, the type/source-to-process mappings are replaced by an equivalent top-level process and subprocess. If you are importing a cartridge that contains a process used by multiple order types and sources (using different views in OSM Administrator), Design Studio creates duplicate processes, one for each order, and renames each process with the original process name concatenated with the order type and source. Design Studio updates all references to the process, including any references in the automation map. Rules, event delays, timer delays, subprocesses, and process exceptions appear in the Process editor, not as separate entities.
Behaviors	Upon import, Design Studio saves behaviors to the following editors: • Behaviors defined at the task level appear in the **Behaviors** tab of the Task editor. • Behaviors defined at the order level appear in the **Behaviors** tab of the Order editor. • Behaviors defined at the data element level appear in the Data Dictionary's **OSM** tab for the element definition.

Rules	In Design Studio, rules are defined within a specific order. Upon import, Design Studio determines which rules are used by an order and adds those rules to the Order editor **Rules** tab.
	If you are importing text-based rules (SQL rules), Design Studio imports the text-based rules as separate text files, using *name_of_the_rule*.**sql** as the rule name. Design Studio saves the rule to the **resources** folder and displays the rule with the other rules in the Order editor **Rules** tab. To modify text-based rules, click the name of the file to access the rule in a SQL editor.
	Rule expressions defined with order type and source operands are not supported in Design Studio. On import, these operands are mapped to an expression with the same data element on each side. If the order-based operand matches the imported order, the generated expression evaluates to true. For example:
	`/VPN_Name = /VPN_Name`
	Otherwise, it evaluates to false:
	`/VPN_Name_Count ! /VPN_Name_Count`

Notifications	Design Studio imports polled-type notifications as jeopardy notifications, and all others as event notifications. Design Studio does not support or import system notifications (notifications that are not associated with any order, task or activity) or mixed transition notification types (for example, notifications are both transitional and polled).

Entity	Considerations
Data Providers	Design Studio imports inline, external, and XQuery instances that are defined as part of an Information rule as a Data Provider entity type. In Design Studio, you can use a Data Instance behavior in conjunction with a Data Provider to retrieve information from an external system.
Process Exceptions	Design Studio maps process exceptions to the Process editor under the following conditions: • A restriction defined in the OSM Administrator must be valid in the imported order, and the type and source of the imported order must match the restriction; otherwise, it will not be imported. • The process exception status used in the OSM Administrator must be available in the Design Studio process. • Activities defined for the process exception must be available in the Design Studio process.
Automation Maps	Design Studio generates automation maps automatically, based on the configurations of automated tasks. If you are importing a cartridge with custom automation plug-ins, you can specify which automation maps to include from within the Import wizard.
Responsibility and Category	Design Studio does not support or import Responsibility and Category entities created in the OSM Administrator.
Entity Names	If Design Studio detects entity name conflicts, it automatically renames the entity. Review the Import Summary report to acquire a list of all entity name changes made during import. You may be required to edit references to the affected entity names in Java code or XSLT files.

Note:

Cartridges created in the OSM Administrator are not necessarily valid upon import to Design Studio. Design Studio performs logical validations to ensure that errors are detected before deploying a cartridge to an OSM run-time environment. For example, if you import a cartridge that contains a rule to check values of a specific data node, Design Studio ensures that the data node exists in the corresponding order data. You must resolve all cartridge errors before deploying to a run-time environment.

Related Topics

Working with Existing OSM Models

About Importing Design Studio Cartridges

Working with OSM Cartridge Projects

About Import Summary Reports

When you import an existing project into Design Studio, the system generates a summary report that describes the errors generated and the actions Design Studio took to resolve the error. Design Studio creates an **importReport** directory in the project. Then, Design Studio

uses the name of the cartridge to generate a default file name for the Import Summary report and saves the report to the **importReport** directory.

The Import Summary report lists the entities in the model that were not imported, (such as entities that are not associated with orders) and entities that are not supported in Design Studio (for example, test category or rule entities, even if they are associated with orders).

Also, if you import a model that includes a task that is used in multiple orders, Design Studio renames the task and notes in the Import Summary report the original name; the new name; and the associated order type, source, and view name.

> **Note:**
>
> The Import Summary report lists all entity name changes made during import. You may be required to edit references to the affected entity names in Java code or XSLT files.

Related Topics

Working with Existing OSM Models

About Importing Design Studio Cartridges

Working with OSM Cartridge Projects

Importing Existing OSM Models

You can import a single order type into a cartridge project, or multiple order types into a cartridge project. After importing multiple order types into a cartridge project, you can deploy all or a specified number of the order types to an OSM run-time environment within the context of a single project.

> **Note:**
>
> Cartridge dependencies must exist in the workspace before importing a cartridge project. For example, when importing an OSM project that has a dependency on a Data Dictionary project, you must import the Data Dictionary project first to avoid problem markers during the project build.

For example, if you have defined a project with metadata to support the DSL services Add, Delete, and Modify for orders that come from two different sources (Siebel and Oracle Communications Billing and Revenue Management, for example), you can deploy the entire configuration to a run-time environment with a single deployment.

To import an OSM model into Design Studio:

1. From the **Studio** menu, select **Show Design Perspective**.
2. Right-click in the Solution view or Studio Projects view and select **Import**, then select **Import Order and Service Management Model**.

   The Import Order and Service Management wizard is displayed.
3. Click **Browse**.

An import dialog box opens.

4. Select the file to import.
5. Click **OK**.

   Design Studio returns you to the Import Order and Service Management dialog box.
6. Click **Next**.
7. Select the cartridge to import from the **Source Cartridge** list.

   If multiple cartridges exist in the XML model, you must select which cartridge to import.
8. In the **Target Name** field, edit the default project name.

   The project name must be unique among project entity types. Two projects cannot share the same name, even if they are different versions.
9. In the **Target Data Dictionary** field, select the dictionary to which the data elements will be added.

   You can add the new data elements to a Data Dictionary that is common to all cartridges, or to a data dictionary defined for a specific cartridge. To create a new dictionary, enter the name of a new data dictionary in the **Target Data Dictionary**.
10. Click **Next**.
11. Select the orders to import.

    Select orders in the Available column and use the arrow buttons to move them to the Selected column.
12. Click **Next**.
13. Add one or multiple automation maps to the cartridge.

    Click **Add** to navigate to and select one or multiple automation maps. Click **Remove** to delete an automation map from the table.

    When you configure automation plug-ins in Design Studio, the system automatically generates an automation map. However, if you are importing cartridges that contain custom automation plug-ins, you can include in the import the automation maps that define the configuration for the custom plug-ins. For each plug-in, the automation map defines whether the plug-in is associated with a task, notification, or data change event, the class name of the plug-in, and whether the plug-in receives information from OSM or from an external system.

    Design Studio imports external receivers as a list of XML files into the **customAutomation** folder, each containing one external receiver XML fragment.

    See "Working with Automation Plug-Ins" for more information.
14. Click **Finish**.

    Design Studio imports the cartridge and adds the new project to the Studio Projects view.

> **Note:**
>
> If Design Studio detects entity name conflicts on import, it automatically renames the entity in the imported cartridge. The Import Summary report lists all entity name changes made during import. You may be required to edit references to the affected entity names in Java code or XSLT files. See "About Import Summary Reports" for more information.

Related Topics

About Importing Design Studio Cartridges

About Importing Cartridges Created in OSM Administrator

Creating New Cartridge Projects

Order and Service Management Project Editor

Working with the Orchestration Model Project

The **OracleComms_OSM_CommonDataDictionary** model project enables auto-generation of orchestration control data. Use the data elements of this model project to model control data for order items. See "About Modeling Order Component Control Data" for more information.

Creating the OracleComms_OSM_CommonDataDictionary Model Project

When you open or create an orchestration entity, Design Studio prompts you to create the data schema of the **OracleComms_OSM_CommonDataDictionary** model project in your workspace.

If you choose not to create the data schema in your workspace, dismiss the prompt and select the **Do not show this prompt in the future** check box. When you are ready to create the data schema in your workspace at a later time, see "Defining OSM Preferences" for information on re-enabling the prompt.

Working with XML Catalogs

In Design Studio, you model behaviors such as business rules that satisfy the business requirements of order processing. The business rules are often contained in resource files such as XQuery files, XSLT files, custom JAR files, third-party JAR files, and XML files. There can be a large quantity of resources, and some of those resources need to reference each other. Resources in OSM can be referenced through URI locators in your data model.

Because a URI must be a physical location on a server, and because the location of the resource may change depending on the run-time system to which you deploy your cartridges, using XML Catalogs in OSM is very useful. XML Catalogs provide a redirection from a URI to another URI. At run time, when OSM processes a URI you specify as part of the OSM data model, OSM first attempts to resolve the URI against the XML Catalogs you specified. Based on the mapping defined in the XML Catalogs, OSM updates the URI to adapt to the environment by resolving the location of the URI in your data model with the new URI you mapped for it in the XML Catalogs.

When you specify URIs for resources that will be redirected to other URIs at run time by way of XML Catalogs, one strategy is to treat the URIs defined in your cartridge design as logical URIs that are replaceable tokens. Using this strategy, it is useful to have a well-defined naming convention for these URIs; for example, the URI schema would include your organization name, project name, type of cartridge, and type of data entity. For more information on packaging resources when using XML Catalogs, see *OSM Developer's Guide*.

You can use XML Catalogs for any of the URIs you specify in the Design Studio editors. You can specify XML Catalogs in your OSM cartridge projects as well as on the OSM server for different purposes. For detailed information on how to use XML Catalogs in OSM, see the discussion on XML Catalogs in *OSM Developer's Guide*.

Related Topics

Enabling and Disabling XML Catalogs for a Cartridge Project

Specifying XML Catalogs for a Cartridge Project

Enabling and Disabling XML Catalogs for a Cartridge Project

If your target run-time software version is OSM 7.0.3 or later (**Target Version** field is set to **7.0.3** or higher), XML Catalog support is enabled by default for all cartridge projects and it is required to be enabled. Do not disable XML Catalog support.

If your target run-time software version is OSM 7.0.2 or earlier (**Target Version** field is set to **7.0.1** or earlier), enable or disable XML Catalog support for a cartridge in your workspace as follows:

1. From the **Studio** menu, select **Show Design Perspective**.
2. In the Studio Projects view, double-click the cartridge project entity for which you want to enable or disable XML Catalog support.

   The cartridge project opens in the Project editor.
3. Click the **Cartridge Management Variables** tab.
4. In the Name column, click the **XML_CATALOG_SUPPORT** variable.
5. In the Default Value column, do one of the following:
   - To enable the XML Catalog for this cartridge, enter **enable**.
   - To disable the XML Catalog for this cartridge, enter **disable**.
6. Click **Save**.

> **Note:**
>
> If your **Target Version** field is set to **7.0.1** or earlier, you can also enable XML Catalog support for a cartridge by adding an empty file entitled **enableXMLCatalogSupport** in the root directory that contains the cartridge project *cartridgeProject***xmlCatalogs\enableXMLCatalogSupport**. If this file is present, and you have defined the XML_CATALOG_SUPPORT cartridge management variable, OSM uses the value you configured for the cartridge management variable to disable XML Catalog support.

For instructions on specifying XML Catalogs for a cartridge, see "Specifying XML Catalogs for a Cartridge Project".

For more information on how to use the XML Catalog in OSM, including how to specify XML Catalogs on the OSM server and how to define catalog entries, see *OSM Developer's Guide*.

Related Topics

Working with XML Catalogs

Specifying XML Catalogs for a Cartridge Project

Specifying XML Catalogs for a Cartridge Project

To specify XML Catalogs for a cartridge project:

> **Caution:**
>
> XML Catalogs are system-wide entities. An XML Catalog specified in one cartridge project is used when processing requests for orders on other cartridges. Ensure URI/URL naming conventions are established across cartridges so that OSM resolves URIs as you require for each cartridge.

1. In the Package Explorer view in Design Studio, navigate to the *cartridgeProject***xmlCatalogs\core** directory.
2. Copy the XML Catalog template file *cartridgeProject***xmlCatalogs\core\xmlCatalogCoreTemplate.xml** into the same directory.

   > **Note:**
   >
   > You can have multiple XML Catalog files within the **xmlCatalogs\core** directory if you wish to organize different sets of catalog entries by file.

3. When prompted, rename the file to any filename you want and use the suffix **.xml** (for example, **catalog.xml**).

   > **Note:**
   >
   > You must use the file extension **.xml**. OSM automatically searches for files ending in **.xml** within the **xmlCatalogs\core** directory and loads any such files as XML Catalogs.

4. Open the file and enter the XML Catalog entries you require.

   You can use any standard XML Catalog entry, but the rewriteURI entry is the most commonly used for OSM. For information on how OSM uses the rewriteURI entry, see the discussion on rewriteURI entries in *OSM Developer's Guide*.

   > **Caution:**
   >
   > It is important to ensure that resources are always uniquely identifiable to a single XML Catalog entry to guarantee that the correct resource is located. For information on how to avoid defining mappings that can be satisfied by more than one entry, see the discussion on defining rewriteURI entries in *OSM Developer's Guide*.

5. Save the file.

Related Topics

Working with XML Catalogs

Enabling and Disabling XML Catalogs for a Cartridge Project

Order and Service Management Project Editor

Use the Order and Service Management Project editor to configure cartridge projects. As your required data becomes available, you can configure specifications for packaging and deploying.

See *OSM Developer's Guide* for information about the Java perspective, including information about the Package Explorer view, the **resources** folder, and the **src** folder.

To access the Project editor, click any OSM Project entity in the Studio Projects view to display the Project editor in your workspace.

When you create a cartridge, you collect the OSM XML model, automation plug-ins, task assignment behaviors, and resource files into a single archive file and deploy the file to the OSM run-time environment.

When configuring cartridge projects, see the following topics:

- Project Editor Properties Tab
- Project Editor Copyright Tab
- Project Editor Dependency Tab
- Project Editor Tag Tab
- Project Editor Packaging Tab
- Project Editor Locations Tab
- Project Editor Model Variables Tab
- Project Editor Cartridge Management Variables Tab
- Project Editor Manifest Tab

Project Editor Locations Tab

Use the Project editor **Locations** tab to view the location and folder names of your Java libraries and resources folders.

Project Editor Manifest Tab

Use the Project editor **Manifest** tab to manage entity-level dependencies in a cartridge.

> **Note:**
>
> Design Studio populates the entity list based on project-level dependencies defined in the **Dependency** tab. If you have not defined any project dependencies, the entity list will be empty. See "Project Editor Dependency Tab" for more information.

The cartridge manifest declares two types of entities:

- Exported entities are available, or visible, to other cartridges. By default, all entities defined in the cartridge are made public for other cartridges to use. You can remove entities that you do not want to make public.
- Referenced entities are entities that are provided by other cartridges. By default, all referenced entities in other cartridges are read-only.

The manifest is used by the deploy and undeploy processes to resolve all required references. If these dependencies are not resolved, the cartridge cannot be deployed or undeployed.

Exported Entities

Field	Use
Entity Type	Displays a list of all possible entity types for the cartridge. Select an entity type to display the entities in the cartridge. **Note:** Exported entity types are predefined and cannot be removed or opened.
Entity List	Displays the entities for the selected entity type. If the cartridge is sealed, the entity list is read-only. To exclude entities from the list, deselect the **Include all from project** check box, then click **Select**, and then from the selection list select only those entities that you want to include in the solution. For example, you may not want to make a particular manual task available to other cartridges and would therefore exclude it. After the list is populated, you can click **Remove** to remove entities or **Open** to open the corresponding editor. **Note:** Excluding an entity in the **Manifest** tab removes it from the validation process only; packaging is not impacted. If the entity is referenced in the run-time environment, it is available.
Include all from project	By default, the check box is selected. The entity list is automatically populated with every entity defined in the cartridge. To customize the entity list, deselect the check box, and then click **Select** to select one or more entities from the selection list.

Referenced Entities

Field	Use
Entity Type	Displays a list of all entity types that are referenced by entities in this cartridge. Select an entity type to display the referenced entities. **Note:** Referenced entity types are predefined and cannot be removed or opened.
Entity List	Displays the referenced entities for the selected entity type. The list displays in read-only mode. Right-click an entity type and select **Open** to open the corresponding editor.

Related Topics

Order and Service Management Project Editor

4

Modeling Data in OSM

Oracle Communications Order and Service Management (OSM) is a model-driven software system. The data you model drives the behavior of your overall OSM solution.

You model data for OSM entities, such as tasks and orders, using simple and structured data elements from the Design Studio Data Dictionary.

When modeling data for OSM entities, see the following topics:

- About Modeling Data in OSM Cartridge Projects
- About Modeling Control Data
- About Contributing Task Data to a Cartridge Project
- About OSM Data in Model Projects
- About Modeling Data in the Order Template
- About the Order Template Context Menu
- About the Task Editor Task Data Context Menu
- Data Schema Editor OSM Tab
- Using Masks
- Defining Behaviors at the Data Schema Level

See "Modeling Data" for general information about modeling data in Design Studio.

About Modeling Data in OSM Cartridge Projects

Modeling data in OSM cartridge projects is the process of defining the order data of your solution. Order data is data on the incoming sales order, control data used for orchestration, and any other data used in the order.

You model data in two primary areas:

- Within the data schemas in your workspace

  You define data element information in the data schemas of model projects, OSM cartridge projects, and other Design Studio application feature cartridge projects; for example, in the data schema of a Design Studio Activation cartridge project.

- Within OSM entities

  You model the data elements represented by orders, tasks, products, order components, and order item specifications in the OSM editors associated with these OSM entities (by adding data elements from the data schemas).

When you model data in an OSM cartridge project, you first define data element information within the data schemas of the projects in your workspace and then use that data element information to model data within OSM entities.

OSM entities can use any data element defined in any data schema in the workspace, including data schemas for projects defined outside of OSM. For example, OSM entities can use an atomic action defined in the data schema of a Design Studio Activation cartridge

project. The data elements OSM can use are visible in the Data Element view, which displays data schemas and entity types. You can drag data elements from the Data Element view to OSM entities.

Data is modeled within OSM entities as follows:

- Orders

  You add data elements from the data schema of projects onto the Order editor **Order Template** tab. OSM uses the data you model here to drive the fulfillment, provisioning, and system interactions of the order. The **Order Template** tab is the hub of modeling data in context of the order and is the focal point for modeling OSM solution data.

  Other OSM entities contribute to data modeled on the **Order Template** tab of the order. For example, the order item specification and order components contribute data to the **ControlData** structured data element defined on the order.

- Tasks

  You add data elements from the data schema of projects or from the Order editor **Order Template** tab of orders onto the Task Data area of the Task editor. OSM uses the data you model here to run tasks.

- Order items

  You add data elements from the data schema of projects onto the Order Item Specification editor **Order Template** tab for every order item property on the order item specification that is required for OSM orchestration. The structure you model here is referred to as order item control data. OSM uses order item control data to add order items into the OSM order from the customer orders that come from the customer relationship management system; order item control data serves as the storage area on the order for each order item property. See "About Modeling Control Data" for more information on order item control data.

  The data schema recommended for modeling order item control data structures is the predefined data schema of the **OracleComms_OSM_CommonDataDictionary** model project. The data schema of this model project includes the base structure for order item control data (**ControlData/OrderItem**). See "About the OracleComms_OSM_CommonDataDictionary Model Project" in the Modeling OSM Orchestration Help for information on this model project.

- Order components

  Design Studio automatically adds data elements onto the Order Component Specification editor **Order Template** tab. The structure Design Studio models here is referred to as the order component control data. Order component control data is automatically generated for an order component that is associated with an orchestration process and associated to an orchestration fulfillment pattern that is part of the orchestration plan. Each order component that is used in orchestration requires order component control data. OSM uses the order component control data for OSM orchestration; it serves as the storage area for the order component on the order.

  Design Studio automatically adds order component control data to the order component and the order template of the order. You do not manually add data elements to the Order Component Specification editor **Order Template** tab unless you do not use the **OracleComms_OSM_CommonDataDictionary** model project. See "About Modeling Order Component Control Data" for more information.

- Composite cartridge views

  You add data elements from the data schema of projects or from the Order editor **Order Template** tab onto the Task Data area of the Composite Cartridge view editor. A composite cartridge view is used when you use composite cartridges to add task data and behaviors

to a solution without having to directly modify the existing component cartridges of that solution. The task data in the composite cartridge view is additive task data to the overall solution. OSM uses the task data to run tasks for function order components you add to a solution. See "Working with Composite Cartridge Projects" for more information on composite cartridge views.

Data structures organized in OSM editors, such as structures represented by orders, tasks, order components, and so on, and the behaviors you apply to data elements organized in OSM editors are not available for reuse in the workspace by other (non-OSM) Design Studio application feature cartridge projects.

The data element information defined in the data schemas of the workspace cannot be overridden in OSM editors. You can augment data elements after you drag them from data schemas into OSM editors, but you cannot change them. However, you can configure OSM-specific extensions to schema data elements by using the **OSM** tab of the Data Schema editor. See "Data Schema Editor OSM Tab" for more information.

A data element is typically defined at the root level in its associated data schema. If a data element is defined within another element in the schema, the path of the data element in the data schema is upheld as the relative path in the editor of the OSM entity in which the data element can be organized into any data structure. By defining a data element at the root level in its data schema and upholding its relative path within OSM entities, you can reuse the data element in multiple entity types without having to duplicate it in other paths or in other data schemas.

When you add a data element from a data schema of another Design Studio application feature cartridge project onto an OSM editor, double-clicking the data element opens the editor in which the data element is defined. For example, if you are in the **Order Template** tab of the Order editor, double-clicking a data element that is part of a service action opens the ASAP Service Action editor, in which the data element is defined.

Related Topics

About the Order Template Context Menu

About the Task Editor Task Data Context Menu

Design Studio Common Editor Tabs

About Modeling Control Data

Control data is the data OSM requires to perform orchestration. There are two kinds of control data:

- Order item control data is order items from the customer order that are required in orchestration
- Order component control data is order components that participate in orchestration

You model order item control data when you configure your order item specification. Design Studio models order component control data automatically for order components that are included in the orchestration fulfillment pattern that is part of the orchestration plan.

Control data is data located in the **ControlData** structure on the order. Control data consists of data required to perform OSM orchestration. For example, when you model order item control data for an IP services order, you include the name of the service in the control data but exclude the port number for the connection to be provisioned. Though the port number must be on the order for the service to be activated, it is not included in order item control data because it is not used by the orchestration process.

Order item properties and order components are OSM entities that contribute to the **ControlData** structure on the order by using the following structures in the data model:

- **ControlData/OrderItem/**

  Order item property control data: Order items from the customer order are stored here and included in the order.

- **ControlData/Functions/**

  Order component control data: An order component that participates in an orchestration plan must have control data defined in the order template of the order.

  Order component control data requires order item control data. Rather than copying the order item data to each order component, OSM creates in the order component control data a reference node back to the **ControlData/OrderItem/***Order_Item_Property_Name* structure. A reference node back to the original order item keeps the order components updated with any new order item properties you might add to your order item specification.

You model the control data structure for order item properties (**ControlData/OrderItem/***Order_Item_Property_Name*) manually in the **Order Template** tab of the Order Item Specification editor. See "Modeling Order Item Control Data" for information on modeling order item control data.

Design Studio automatically models the control data structure for order components (**ControlData/Functions/***Order_Component_Name*) associated with orchestration orders on the **Order Template** tab of the Order Component Specification editor. See "Modeling Order Component Control Data Automatically" for more information on how Design Studio models the order component control data.

Data modeled in the **Order Template** tabs of the Order Component Specification editor and the Order Item Specification editor contribute to the order template of the order.

When working with composite cartridge projects, control data is deployed to your runtime environment as part of the OSM composite cartridge. As a result, the complete control data becomes available to the new tasks that are contributed through the component cartridges of the composite cartridge. See "Working with Composite Cartridge Projects" for information on creating composite cartridges.

Related Topics

About Modeling Order Component Control Data

About Modeling Order Item Control Data

About Contributing Task Data to a Cartridge Project

You may want to contribute task data to a cartridge project without directly modifying the modeled data within it. For example, you cannot directly modify the modeled data in a sealed cartridge project, but you may want to add task data to the existing tasks within it. You can contribute task data to a cartridge project without directly modifying it by including it as a component cartridge within a composite cartridge project. See "Working with Composite Cartridge Projects" for more information.

About OSM Data in Model Projects

Model projects are collections of data elements that can be referenced by other projects in a workspace. The data elements you define in a model project represent the foundational data elements of the entire data model and are product-agnostic (not specific to any one Design

Studio application feature cartridge project). Although data elements in a model project are intended to be product-agnostic, you can configure OSM-specific extensions to these data elements. Doing so defines OSM-specific configuration at the root data element, which allows the configuration to be inherited into a product-related context. OSM behaviors are an example of valid extensions to schema data elements in a model project. OSM-specific extensions to schema data elements are configured in the **OSM** tab of the Data Schema editor. See "Data Schema Editor OSM Tab" for more information.

The main benefit of model projects is that they provide a common and centralized repository of data models across Design Studio application feature cartridge projects, which enables consistent data typing in message interactions across your Operational Support Systems (OSS) environment.

When you right-click a model project and select **Select Data Structure Definition**, you either select an existing data structure definition or create a new data structure definition. After you create a data structure definition entity, you open it and define its attributes. Data structure definitions allow you to model complex data types in OSM. Complex data types, which can contain child elements, allow for the generic and reusable definition of both abstract (extendable) and concrete (final) data structures. See "Defining Order Data" for information about adding data structure definitions to an order.

About Modeling Data in the Order Template

When you model data in the **Order Template** tab of the Order editor, you add from the Data Dictionary data elements, including data types such as atomic actions, and organize them in a way that makes sense in the context of the order.

When you drag a data element from the Data Element view onto a data element in the order template, all selected nodes of that data element appear in the order template underneath the data element that was dropped. If a child node of a data element is selected in the Data Element view, the child node and all its parent nodes up to the root of the data schema are automatically included.

When you right-click in the order template and select **Select from Dictionary**, the **Select Data Elements** dialog box is displayed. The **Select Data Elements** dialog box shows *all* data elements available in the workspace, unlike the Data Element view, which shows only data elements and entity types based on the filters you set for the view. For example, based on the dependencies you defined in the Project editor **Dependencies** tab. If you add a data element from the **Select Data Elements** dialog box onto the order template from a project that is not defined as a dependency, Design Studio creates a problem marker. See "Managing Project Dependencies" for information on defining dependencies for a project.

The order template context menu contains actions specific to simple and structured data elements defined on the order. To access these actions, right-click in the Order Template area of the Order editor **Order Template** tab. See "About the Order Template Context Menu" for information about these actions.

About the Order Template Context Menu

The Order Template context menu contains actions specific to simple and structured data elements defined on the order. To access these actions, right-click in the Order Template area of the Order editor **Order Template** tab. The actions specific to the Order Template are listed below. For information about the standard data modeling context menu options, see "Modeling Data Using Context Menus".

Field	Use
Select Data Structure Definition	Select to add an existing data structure definition or create a new data structure definition. Data structure definitions allow you to model complex data types in OSM. If no data structure definitions are displayed in the **Matching items** area, you must define the dependency of the data structure definition to the model project before you add it to the order template.
Disable Merge Mode/ Enable Merge Mode	Select **Disable Merge Mode** when adding a data element from the Data Dictionary onto an order so that Design Studio does not merge the data element with a data element that is identical, and instead, adds the data element as a child of the identical data element (creating a recursive structure). This setting stays in effect for all OSM editors in which data elements are added until you change it.
Add To Tasks/Views...	Select to add one or more data elements from the order template to one or more tasks or one or more views in a single operation. Use this option for task inheritance and view inheritance of data elements defined in an order. **Important:** Ensure source control is set up. You cannot undo this action in a single operation. You must individually remove data elements from each task or view after they are added.
Set Significance	Select to set significance on multiple data elements in a single operation.
Copy Mnemonic Path	Select to copy the path of the selected entity to the clipboard. This path is then available for you to paste. If the selected OSM entity has derived complex types, you can copy the path to the derived type.
Open In...	Select to open the OSM entity that contributes an inherited data element. This option is available when you select data elements that are inherited from different OSM entities (these data elements are grayed out). This option provides quick access from the order template tree of the order to the order template tree of the base entity where the inherited data element is defined. For example, in orchestration orders, the order template can have data elements (such as **ControlData**) that are contributed from multiple OSM entities (parent orders, order components (fulfillment functions), order item specifications, and so on. Using this option, you find out which entities the data element is inherited from and can quickly open the order template of those entities.

About the Task Editor Task Data Context Menu

The Task Data context menu contains actions specific to simple and structured data elements defined on the task. To access these actions, right-click in the Task Data area of the Task editor **Task Data** tab. The actions specific to the Task editor are listed below. For information about the standard data modeling context menu options, see "Modeling Data Using Context Menus".

Field	Use
Select Data Structure Definition	Select to add a data structure definition to the task. You can select an existing data structure definition, or create a new data structure definition. Data structure definitions allow you to model complex data types in OSM. If no data structure definitions are displayed in the **Matching items** area, you must define the dependency of the data structure definition to the model project before you add it to the task.
Disable Merge Mode/ Enable Merge Mode	Select **Disable Merge Mode** when adding a data element from the Data Dictionary or order template onto a task so that Design Studio does not merge the data element with a data element that is identical, and instead, adds the data element as a child of the identical data element (creating a recursive structure). This setting stays in effect for all OSM editors in which data elements are added until you change it.
Set Read Only	Select to set multiple data elements on the task as read only in a single operation. You often must make task data elements read only. For example, in a manual task, you do not want users to update certain data values such as the account ID.
Set Significance	Select to set significance on one or more data elements in a single operation.
Open In...	Select to open the editor from which the data element is inherited. This option applies when you select an inherited data element that is contributed through a solution or base order.

Data Schema Editor OSM Tab

Use the Data Schema editor **OSM** tab to define values for data elements that are associated with your OSM runtime system. See "Data Schema Editor" for information on the other tabs in the Data Schema editor.

Field	Use
Significant Element	Specify whether the OSM server should consider the corresponding element during amendment processing. During amendment processing, the OSM system compensates only for task instances that use significant data elements as inputs. If an element is not specified as significant, the system updates the order only with the changed data (no compensation is required). Data significance is supported at the Data Dictionary (data schema), order template, and task-view levels.
Read Only	Specify to indicate that the data element is read only. This option applies only to tasks. **Note**: If you are defining an attribute of a data structure definition element, do not use the Read Only check box to define the element as read only. Instead, deselect this check box and use the Behavior field to define a ReadOnly behavior for the element.
XML Type	Specify to indicate that the data structure definition is an XML data type. Structures defined as XML data types in the data structure definition can contain XML documents.

Field	Use
Maximum Numeric Digits	Enter the number of digits you want to allow in the OSM user interface for the corresponding element. This field is not available for editing if you define an OSM element mask for the corresponding field. For data elements that inherit data from a base type, this field is read-only.
OSM Element Mask	Specify the string of characters used to define the format of the data in the field of the Task web client. When you create an OSM element mask, you can restrict user input for a field to a specific format. Use a mandatory mask character to create a field where a user must enter information in the appropriate format to complete the task. For data elements that inherit data from a base type, this field is read-only. See "Using Masks" for more information.
Behaviors	Right-click to define behaviors for the corresponding data element. When you define behaviors at the Data Dictionary (data schema) level, the behavior can apply to all orders and tasks in which the data element appears. Behaviors inherited from a base type are read-only. See "Defining Behaviors at the Data Schema Level" for more information.

Related Topics

Using Masks

Defining Behaviors at the Data Schema Level

Design Studio Common Editor Tabs

Using Masks

You use masks in Design Studio to restrict Task web client user input for a field to a specific format. When using masks, refer to the following topics:

- About Masks
- Defining Masks for Task Web Client Fields

About Masks

Masks enable you to restrict Task web client user input for a field to a specific format. Using the Data Dictionary editor **OSM** tab, you can specify the strings of characters used to define the format of the data in the field and use a mandatory mask character to create a field where a user must enter information in the appropriate format to complete the task. See the "Data Schema Editor OSM Tab" for more information.

Use the following mask characters for text fields:

Mask Character	Description
#	Mandatory digit

Mask Character	Description
9	Optional digit
A	Mandatory alphanumeric
a	Optional alphanumeric
?	Mandatory alpha
z	Optional alpha

Mask characters serve as placeholders in Task web client fields. All non-mask characters appear in text type fields; you cannot edit them. For example, if you enter *TEXT* in the **OSM Element Mask** field, it appears in the Task web client field as **TEXT**. To use mask characters as literals, you must enter "\" in front of the character.

Use the following mask characters for numeric fields:

Mask Character	Description
9	Optional digit
0	Use to the right of the decimal. Displays **0** if no value is entered.
.	Decimal placeholder
,	Numeric placeholder

> **Note:**
>
> You must include at least one numeric mask character for the field to be valid. You cannot use quotation marks (" ") for numeric masks.

The following table illustrates how numeric mask characters defined in the Data Schema editor would affect data entered into the Task web client:

Mask Defined in Data Dictionary	Entered into the Web Client	Displayed in Web Client
9,999	1234	1,234
99.99	34.12345	34.12
99.00	12	12.00
99.00	34.1256	34.13

Related Topics

Defining Masks for Task Web Client Fields

Data Schema Editor

Defining Masks for Task Web Client Fields

Use masks in Design Studio to restrict Task web client user input for a field to a specific format.

To define a mask for a Task web client field:

1. From the **Studio** menu, select **Show Design Perspective**.
2. Click the **Data Element** tab.
3. Select the data schema entity that contains the field for which you want to define the mask.

   The entity opens in the Data Schema editor.
4. In the Dictionary area, select the data node for which you want to define the mask.
5. Click the Data Schema editor **OSM** tab.

   See "Data Schema Editor OSM Tab" for more information about the fields on this tab.
6. In the **OSM Element Mask** field, specify the string of characters used to define the format of the data in the Task web client field.

   Use a mandatory mask character to create a field where a user must enter information in the appropriate format to complete the task. See "About Masks" for more information.For elements that inherit data from a base type, this field is read-only. See "Leveraging Existing Data Information" for more information.
7. Click **Save**.

Related Topics

Data Schema Editor

Modeling Data

Defining Behaviors at the Data Schema Level

On the "Data Schema Editor OSM Tab", you can define behaviors for data, which enable you to extend the functionality and appearance of order data. Each behavior type performs an action; for example, calculating or validating data or displaying fields in read-only mode.

To define behaviors for data:

1. Double-click an element from the Data Element view.

   The details for the selected element are displayed in the Data Schema editor.
2. Click the **OSM** tab.
3. Select the data element for which you will define the behavior.

   The Add Behavior dialog box is displayed.
4. In the Behaviors area, right-click and select **Add Behavior**, and then select the behavior type.

   The newly created behavior appears in the **Behaviors** list.
5. Select the behavior from the list and click **Properties**.

   The **Properties** tab opens with the set of properties that you can define for this behavior type.

   See "Working with Behaviors" for information on how to set behavior properties.

Related Topics

Data Schema Editor OSM Tab

Data Schema Editor

Modeling Data

5
Working with Roles

When assigning permissions, you can permit specific roles access to functions in the Oracle Communications Order and Service Management (OSM) Task web client. When modeling roles, see the following topics:

- Creating New Roles
- Adding Roles to Multiple Tasks
- Role Editor Role Tab

Creating New Roles

You create roles to permit specific user groups access to functions in the Task web client.

To create a role:

1. From the **Studio** menu, select **New**, select **Order and Service Management**, select **Order Management**, then select **Role**.

   The Role wizard is displayed.
2. In the **Project** field, select the OSM project in which to save this entity.
3. In the **Name** field, enter a name for the role.

   The name must be unique among the role entities in the same namespace.
4. (Optional) Select a location for the role.

   By default, Design Studio saves the role to your default workspace location. You can enter a folder name in the **Folder** field or select a location different from the default. To select a different location:

   a. Click the **Folder** field **Browse** button.

   b. Navigate to the directory in which to save the entity.

   c. Click **OK**.
5. Click **Finish**.

   Design Studio adds the role to the project in the Studio Projects view.

Related Topics

Role Editor Role Tab

Adding Roles to Multiple Tasks

When you create a task, you assign a role to it. If you create new roles that require access privileges to existing tasks, you can add the roles to multiple tasks in a single operation.

When you add roles to tasks in a single operation, roles are added with all task permissions granted. If you add a role to a task where the same role is already added, any task permissions that are not granted to the existing role remain not granted.

To add roles to multiple tasks in a single operation:

> **Note:**
>
> Ensure source control is set up. This action cannot be undone in a single operation.

1. In the Studio Projects view, select the roles to add to existing tasks.
2. Right-click on the selected roles and select **Add Role(s) to Tasks**.

   The Select Tasks dialog box is displayed.
3. Select the tasks to which you are adding the roles.
4. Click **OK**.

   The roles are added to all tasks that were selected, and the roles are added with all task permissions granted (**Do**, **Redo**, and **Undo**).
5. For roles that should not have all task permissions granted, open each task to which that role was added and set task permissions as needed.

To remove a role from a task, use the **Permission** tab of the Task editor.

Related Topics

Creating New Roles

Role Editor Role Tab

Role Editor Role Tab

You use the Role editor to assign permissions to role entities. To access the Role editor, double-click any role entity in the Studio Projects view. The Role editor enables you to modify the name that displays for the role in the Task web client and to assign to the corresponding role any combination of the following permissions.

Field	Use
Create Versioned Orders	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.
Exception Processing	Enables users to alter the flow of a process by applying exception statuses at any time throughout the process.
Online Reports	Enables users to view summarized reports on all orders and tasks on the system.
Order Priority Modification	Enables users to modify the priority of a task in an order.
Reference Number Modification	Enables users to modify the reference number of an order.
Search View	Enables users to access the order Query function.
Task Assignment	Enables users to assign tasks to others.
Worklist Viewer	Enables users to access the Worklist function.

Related Topics

Creating New Roles

6

Working with Processes

An Oracle Communications Order and Service Management (OSM) process is a representation of the activities, or tasks, required to offer a specific service to a customer. The process representation includes all of the work that must be performed to complete the order. Rules that you predefine in Design Studio determine which process an incoming order uses. OSM may include an unlimited number of processes.

When working with processes, see the following topics:

About the Process Editor

The Process editor is a canvas where you can configure new services quickly and with minimal data; designing with the Process editor is analogous to capturing your workflow requirements on a white board. Designers using the Process editor to model new services do not need to understand the OSM technology behind the processes and integrations. As a designer begins sketching out the initial processes and tasks in the Process editor, Design Studio simultaneously builds in the background the corresponding artifacts necessary for deployment to the OSM system.

You can use the diagrammed representations that you design in the Process editor to illuminate patterns and identify inefficiencies in processes. Process editor shapes, colors, and presentation can communicate information about the flows and processes.

Related Topics

Working with Processes

Working with Process Editor Menu Controls

The Process editor context menu provides access to specific actions that enable creation of OSM processes. Right-click in the Process editor to access the Process editor context menu.

When working with Process editor menu controls, see the following topics:

- About Task Controls
- About Zoom Controls
- About Layout Controls
- About Print Controls
- About Selection Controls

About Task Controls

Use the following context menu options to control the Process editor task-related features:

Field	Use
Rename	Select to modify the display name that represents the entity in the Process editor. **Note:** You can also press the F2 key to rename an entity in the Process editor.
Assign Order	Select to associate the task with an order.
Clear Activity Reference	Select to remove from the task any existing entity associations. The task activity remains in the Process editor if you clear the reference or if you delete the referenced entity.
Convert to	Select to convert the task to a different task type. **Important:** When converting from one type of task to another, the system displays a prompt if the potential for data loss exists (for example, when converting from an automated task to a manual task). Consider your task conversions carefully before implementing.

Related Topics

Working with Processes

Working with Process Editor Menu Controls

About Zoom Controls

The following zoom tools are available in the Process editor context menu and Process editor toolbar.

> **Note:**
>
> In addition to the menu controls, you can press the Control key and simultaneously rotate your mouse wheel to zoom in and out of a graphic.

Field	Use
Zoom In, Zoom Out	Select to magnify and reduce the size of a graphic, respectively.
Reset Zoom	Select to reset the graphic to the original size.

Field	Use
Fit to Contents	Select to increase or decrease the size of a graphic so that it fills as much of the editor as possible.
Zoom Box	Select to magnify a section of the Process editor. Drag a selection rectangle with the **Zoom Box** tool, and that part of the image will be magnified to fill the editor. If the **Selection** tool is active, you can activate the **Zoom Box** tool by pressing the Control and shift keys simultaneously and dragging a selection rectangle in the Process editor to magnify that area.

Related Topics

Working with Processes

Working with Process Editor Menu Controls

About Layout Controls

Use the following context menu options to control the Process editor layout features:

Field	Use
Layout All Nodes	Select to automatically arrange all nodes in a standard flow chart format.
Layout Selected Nodes	Select to automatically arrange a selection of the process in a standard flow chart format. Use a drag selection to activate multiple tasks for selection.

Related Topics

Working with Processes

Working with Process Editor Menu Controls

About Print Controls

Use the following context menu options to control the Process editor printing features:

Field	Use
Print	Select to print a diagram or a selection of the diagram. **Note:** The first time that you invoke a print function, the system queries the current status of the installed printer, which may cause an initial delay.
Print Preview	Select to display each page to be printed and their corresponding page margins. On the **Print Preview** page, click the **Setup** button to access the **Page Setup** page, where you can scale the selection to fit a specific number of pages, and modify the layouts, margins, header and footers, and page order. You can access the page setup properties from Process editor context menu by selecting **Page Setup**.
Print Diagram to Image File	Select to save the image in JPG or PNG format.
Page Setup	Select to scale the selection to fit a specific number of pages, and modify the layouts, margins, header and footers, and page order.

Field	Use
Set Print Area	Select to print a selection of the diagram. Using the Selection tool, click and drag over the area you want included in the print.
Clear Print Area	Select to clear the selected print area and dismiss the selection.

Related Topics

Working with Processes

Working with Process Editor Menu Controls

About Selection Controls

Use the following context menu options to control the Process editor selection features:

Field	Use
Make Select Active	Select to enable the **Selection** tool.The **Selection** tool enables you to make an object active. You can also press the Esc key to activate the **Selection** tool when other tools in the **Tool** drawer are active.
Pan	Select to reposition the contents in the process editor. Click the Pan icon to make the tool active, then drag the graphic into the desired position. If the **Selection** tool is active, you can activate the **Pan** tool by pressing the Control key and clicking the left mouse button. **Note:** The **Pan** tool remains active for a single use only. When you release the left mouse button, the **Pan** tool is deactivated and the **Selection** tool becomes active. If you want the **Pan** tool to remain active for multiple instances, activate the **Sticky Mode** tool.

Related Topics

Working with Processes

Working with Process Editor Menu Controls

Working with the Process Editor Palette

In addition to the actions available on the Eclipse Workbench toolbar (for example, New, Save, Search, External Tools), the Design Studio for OSM Process editor palette provides access to specific actions that enable modeling of OSM processes. The palette appears as a collapsible sidebar in the Process editor. If you prefer to access the palette tools from a view, you can include the Palette view in your perspective.

The Process editor palette contains four drawers. Click a drawer to expand or collapse the drawer. Click the **Pin Open** button to pin the drawer into the open position. The tools in the Process editor palette drawers are grouped by type:

- Tools
- Activities
- Flows
- Exception Paths

Related Topics

Working with Processes

About the Process Editor Tool Drawer

About the Process Editor Activities Drawer

About the Process Editor Flow Drawer

About the Process Editor Exception Paths Drawer

About the Process Editor Tool Drawer

The **Tool** drawer contains tools for selecting and positioning activities within the Process editor.

Field	Use
Selection Tool	Select existing components. Click the **Selection** tool, then click any activity in the Process editor to make that activity active. Press the Esc key to activate the **Selection** tool when other tools in the **Tool** drawer are active. This tool is the default cursor tool.
Pan Tool	Select to reposition the contents in the Process editor. To make the **Pan** tool active, click the Pan Tool icon in the Process editor tools palette, then drag the graphic into the desired position. If the **Selection** tool is active, you can activate the **Pan** tool by pressing the control key and clicking the left mouse button.
Zoom Tool	Select to magnify a section of the Process editor. Drag a selection rectangle with the **Zoom** tool, and that part of the image will be magnified to fill the editor. If the **Selection** tool is active, you can activate the **Zoom** tool by pressing the control key and the shift key simultaneously. To reset the graphic to the original size, right-click in the Process editor and select **Reset Zoom.** Additionally, you can press the Control key and use a mouse wheel to zoom in and out of a graphic. **Note:** You can magnify and reduce the size of a graphic by using the **Zoom In** and **Zoom Out** buttons in the Process editor toolbar. Click the **Zoom In** or **Zoom Out** button to increase or decrease the size of the graphic.
Magnify Tool	Use to magnify the area of the diagram positioned under the pointer. Press the Alt key and simultaneously click and hold the left mouse button to activate the **Magnify** tool. Release the mouse button and Alt key to deactivate the tool. While this tool does not appear in the tools palette, it is available for use in the Process editor and accessible by the keystroke shortcut noted above.

Related Topics

Working with Processes

Working with the Process Editor Palette

About the Process Editor Activities Drawer

An activity is a unit of work that the system performs. Activity types include processes, subprocesses, and tasks.

Field	Use
Task	A task is an activity that cannot be broken into a finer level of detail. To place a task activity within the Process editor, click **Task**, then click inside of the Process editor. The Create Task wizard appears, where you can define attributes for a new task or select an existing task.
Subprocess	A subprocess is an activity that is included within a process. Subprocesses can be broken into a finer level of detail (a process) through a set of activities. To place a subprocess activity in the Process editor, click the **Subprocess** button, then click inside of the Process editor. Right-clicking the subprocess activity lets you edit the display name and assign an order.
Rule	Rules are tasks that evaluate data to determine if a specified condition exists. Rule tasks are evaluated by the system and have completion statuses of true or false. **Note:** Before you create a rule task, you must first define the data elements in the Order editor **Order Templates** tab. To place a rule activity in the Process editor, click the **Rule** button, then click inside of the Process editor. Right-clicking the rule activity lets you edit the display name, assign an order, and create an activity reference.
Timer Delay	A timer delay pauses an operation until a specified order rule evaluates as true. Timer delays and event delays work identically, but differ in how the rule evaluation is triggered: • A timer delay is evaluated at specified time intervals. • An event delay is evaluated only when the data referenced in the rule changes. To place a timer delay activity in the Process editor, click the **Timer Delay** button, then click inside of the Process editor. Right-clicking the timer delay activity lets you edit the display name, assign an order, or create an activity reference. By default, timer delays use the **null_rule**. Oracle recommends using a custom order rule instead. See "Designing Timer Delays" for more information. **Note:** The frequency at which the OSM server evaluates a delay rule is determined by your OSM server configuration. See "Installing OSM in GUI Mode" in *OSM Installation Guide* for more information.

Field	Use
Event Delay	An event delay pauses an operation until a specified order rule evaluates as true. Timer delays and event delays work identically, but differ in how the rule evaluation is triggered: • A timer delay is evaluated at specified time intervals. • An event delay is evaluated only when the data referenced in the rule changes. To place an event delay activity in the Process editor, click the **Event Delay** button, then click inside of the Process editor. Right-clicking the event delay activity lets you edit the display name, assign an order, or create an activity reference. See "Designing Event Delays" for more information about event delays.
Join	Enables you to combine a set of flows into a single flow. The unified flow can join based on all transitions completing or any one transition completing. To place a join activity in the Process editor, click the **Join** button, then click inside of the Process editor. Right-clicking the join activity lets you edit the display name, assign an order, or create an activity reference.
End	An event is an occurrence during the course of a business process. Events affect the flow of the process and usually have a cause (trigger) or an impact (result). The end event indicates where a process will end. To place an end event in the Process editor, click the **End** button, then click inside of the Process editor. Right-clicking the end event lets you edit the display name, assign an order, or create an activity reference.
Redirect	A redirect activity describes a mechanism used to redirect an operation to a different process. To place a redirect activity in the Process editor, click the **Redirect** button, then click inside of the Process editor. Right-clicking the redirect activity lets you edit the display name, assign an order, or create an activity reference.

Related Topics

Working with Processes

Working with the Process Editor Palette

About the Process Editor Flow Drawer

Flows describe how tasks are completed and determine the order of tasks in the process. To describe the flow between any two activities in the Process editor, click the appropriate flow button in the Process editor palette, then click the source activity in the Process editor. A dynamic flow line appears, enabling you to connect the source activity to any other activity in the Process editor. Click a destination activity to create the directional flow.

You can use the following options to describe the flow within a process. Each of these flow options describes a different transition status:

Field	Use
Flow	Refers to the flow that originates from a start event and continues through activities via alternative and parallel paths until it ends at an end event.
True Flow	Denotes that the activity processed and completed with a result that allows the process to continue to the next activity or end.
False Flow	Denotes that the activity processed and completed with a result that prevents the process from continuing to the next activity.
Next Flow	Denotes advancement to the next activity.
Back Flow	Denotes a return to the previous activity.
Finish Flow	Denotes the completion of an operation.
Cancel Flow	Denotes the cancellation of an operation.
Success Flow	Denotes that an operation completed successfully.
Failure Flow	Denotes that an operation did not complete successfully.

> **Note:**
>
> Flows are represented in the Process editor by transition arrowheads. When the Mandatory Check option for a corresponding flow is not enabled, the flow is represented as a hollow arrowhead. When Mandatory Check is enabled, the flow is represented as a solid black arrowhead. To ensure that the system verifies that mandatory fields are present when a task completes, enable the Mandatory Check option for the corresponding flow in the Properties view. To access the Properties view, right-click the corresponding flow and select Show Properties, or see "Process Editor Flow Properties General Tab" for more information.

Related Topics

Working with Processes

Working with the Process Editor Palette

About the Process Editor Exception Paths Drawer

Exception paths are used in conjunction with redirect and end activities to define process exceptions. Process exceptions let you alter the normal process flow from anywhere within a process (or subprocess) at any time while running the process. You can also model an exception with role restrictions, thus allowing only selected roles to throw the exception.

See "Designing Exception Paths" for more information about exception paths.

Related Topics

Working with Processes

Working with the Process Editor Palette

Creating New Processes

You create processes to represent the activities required to offer a specific service to a customer. The process representation includes all of the work that must be performed to complete the order.

To create a new process entity:

1. From the **Studio** menu, select **New**, select **Order and Service Management**, select **Order Management**, then select **Process**.
2. In the **Project** field, select the OSM project to which to add the process.
3. In the **Order** field, associate the process to an order.

   If no order exists (the field is blank), you can create the process entity and then later create an order to associate with the process.
4. In the **Name** field, enter a name for the process.

   Ensure that the process name is unique among the process entity types. Two processes cannot share the same name.
5. (Optional) Select a location for the process.

   By default, Design Studio saves the process to your default workspace location. You can enter a folder name in the **Folder** field, or select a location different from the system-provided default. To select a different location:

   a. Click the Folder field **Browse** button.
   b. Navigate to the directory in which to save the entity.
   c. Click **OK**.
6. Click **Finish**.

   Design Studio displays the process entity under the selected project in the Studio Projects view.

Related Topics

Working with Processes

Modifying Process Editor Start Properties

You can define properties at the process level, including the expected duration of the process, the reference name, and whether it is a workstream process.

To modify Process editor start properties:

1. In the Process editor, right-click the Start node and select **Show Properties**.

   The Properties view opens for the process.
2. For each property, click inside the **Value** field.

   Design Studio displays a list of values for the corresponding property. See "Process Editor Start Properties General Tab" for more information about the fields and values.
3. Select the desired value.
4. Click **Save**.

> **Note:**
>
> You can double-click the Start entity in the Process editor to open the associated order in the Order editor.

Related Topics

Working with Processes

Designing Workstream Processes

Working with Orders

Process Editor Start Properties General Tab

Use the Process Editor Start Properties General tab to define properties at the process level, including the expected duration of the process, the reference name, and whether it is a workstream process.

Field	Use
Duration	Define the expected duration of the process in weeks, days, hours, minutes, and seconds.
Process History	Select **True** if you want this process to appear in the Process History - Summary Table window in the Task web client. If you do not want this process to appear in the Process History - Summary Table window in the Task web client, select **False**.
Reference	Displays the order with which this process is associated. To change the association, click the **Select** button to access a list of orders.
Workstream	Select **True** to define the process as a workstream process. See "Designing Workstream Processes" for more information.
X, **Y** coordinates	Indicates the present X and Y pixel coordinates for the Start entity.

Related Topics

Working with Processes

Modifying Process Editor Start Properties

Designing Tasks and Activities

A task is one step in a process. You design process flows with tasks, subprocesses, rules, or other process information. There are multiple methods for including new tasks and activities into a Process editor design model.

To add tasks to a process:

1. From the Process editor palette, select **Task** from the Activities drawer.
2. Click inside the Process editor.

   The Create Task wizard is displayed.
3. Select the task to add to the process.

   You can create a new task or select an existing task.

To create and add a new task to the process, select **Create a New Task**. See "Creating New Tasks" for more information.

To select an existing task to add to the process:

a. Select **Select an existing Task**.

b. Click **Select**.

The Task Selection dialog box is displayed. To view all available tasks for all projects, enter an asterisk (*) into the **Select an item to open** field.

To filter for specific tasks, type any character or string of characters contained in the task name to display only the tasks containing those characters.

c. Select the desired task and click **OK**.

The name of the selected task is displayed in the **Task** field.

4. Click **Finish**.

The new task appears in the Process editor and under the selected project in the Studio Projects view.

5. (Optional) Drag existing tasks from the Studio Projects view into your process design.

When you drag existing tasks from the Studio Projects view onto the Process editor, the system copies to the new process the associated data defined for the existing task.

6. (Optional) Copy existing tasks from a different process design model.

To use an existing task or set of tasks from a different process design model, use the **Select** tool in the Process editor palette to select the tasks you want to use, copy those tasks (select **Edit, Copy**), then paste the tasks into the new process design model. The system copies to the new process the associated data defined for the existing task.

7. Click **Save**.

To include other activity types in a process:

1. From the Process editor palette, select an activity from the Activities drawer.
2. Click inside the Process editor to place the object.
3. Right-click on the activity.

The context menu is displayed. You can edit the display name, assign an order to the activity, and create or clear an activity reference.

4. (Optional) Double-click subprocess entities to open them in a separate Process editor tab.

In the new tab, edit the display name and model the tasks, activities, and flows associated with the subprocess.

5. Repeat these steps as appropriate.
6. Click **Save**.

Related Topics

Designing Subprocesses

Working with Process Editor Menu Controls

Working with the Process Editor Palette

Working with Processes

Working with Tasks

Process Editor Activities Properties General Tab

Use the Process editor Properties view **General** tab to define values for the Activities drawer entities, including tasks, rules, delays, and join entities.

See "Designing Subprocesses" for information about subprocess properties. See "Designing Exception Paths" for more information about the redirect entity.

Field	Use
Compensation	Appears for rule entity properties only. Define how the OSM run-time environment evaluates the corresponding entity during compensation. Select **Redo** if the OSM server should undo the original operation and re-evaluate it. Select **Do nothing** if you do not want to OSM server to re-evaluate the entity operation.
Condition	Appears for delay and rule entity properties only. Select the predefined rule which must evaluate to true for the rule or delay entity to transition. For timer delay entities, Oracle recommends selecting a custom order rule rather than using the default **null_rule**. See "Defining Order Rules" for more information about order rules.
Description	(Optional) Enter a description or the intended use for the corresponding task.
Display Name	Enter the name of the task that represents how an entity appears in the Task web client and throughout the Design Studio editors.
Join Type	Select **All** to have the task begin when all transitions flowing to the task have completed or select **Any** to have the task begin when any one transition flowing to the task has completed. Selecting **Any** will create one instance of the task for each incoming transition.
Process History	Select **True** if you want this task to appear in the Process History - Summary Table window in the Task web client. Otherwise, select **False**.
Reference	Displays the task associated with the selected task entity. To change the association, click the corresponding ellipsis button to access a list of Tasks.
X, **Y** coordinates	Indicates the present X and Y pixel coordinates for the task entity.

Related Topics

Working with Processes

Modifying Process Editor Start Properties

Process Editor Task Properties Events Tab

Use the Process Editor Task Properties view **Events** tab to create event notifications for a single task instance in a specific process that only triggers when the task reaches a specific state or status and (optionally) if a specific rule evaluates to true. Use the **Details** sub-tab to choose the task and transitional events, specify the rule that triggers the event, set the priority level, enable or disable the event, and specify whether to send the notification by email. Use the **Automation** sub-tab to create automation plug-ins to perform the work of the notification.

See "Properties Events Detail Tab" for more information about the fields on the **Detail** sub-tab. See "Creating Process-specific Task Event Notifications" for information about creating event notifications at the Process editor level.

Related Topics

Working with Event Notifications

Configuring Automation Plug-In Properties

Designing Timer Delays and Event Delays

Timer delays and event delays pause an operation until an order rule evaluates as true. Timer delays and event delays work identically, but the timing for the rule evaluation differs as follows:

- The order rule on a timer delay is evaluated at specified time intervals. The frequency at which the OSM server evaluates the timer delay rule is determined by your OSM server configuration. See *OSM Installation Guide* for more information.
- The order rule on an event delay is evaluated only when the data referenced in the rule changes.

> **Note:**
>
> Compensation ignores timer delays and event delays when undoing and redoing the tasks in a process.

See "About the Process Editor Activities Drawer" for information about adding timer delays and event delays to a process.

Designing Timer Delays

By default, timer delays use the **null_rule**. To use OSM resources efficiently and minimize database table entries, Oracle recommends that you define a custom order rule and apply it to the timer delay. By applying an order rule to the timer delay, you avoid creating a separate automated task to evaluate the rule.

See "Defining Order Rules" and "Applying Order Rules to Timer Delays" for more information.

Applying Order Rules to Timer Delays

To apply an order rule to a timer delay:

1. In the Process editor, right-click the timer delay.
2. Select **Show Properties**.

   The Properties view for the timer delay opens.
3. Select the **Condition** property and click inside the **Value** field.
4. Select a custom order rule for the timer delay.
5. Click **Save**.

Designing Event Delays

Because event delays can only detect data changes that happen after the event delay starts, you must model your cartridges so that the event delay starts before the task or process that triggers the data change.

For example, imagine you have a cartridge project with two parallel processes. Process A includes an event delay that pauses until Process B updates some order data. If you model the project so that Process A reaches the event delay before Process B begins, when Process B updates the data, the event delay detects the data change, evaluates the order rule, and resumes Process A if the rule evaluates as true.

However, if you model the project so that Process B updates the data before Process A reaches the event delay, the event delay does not detect the data change, does not evaluate the order rule, and continues to delay Process A indefinitely.

Designing Subprocesses

A subprocess is a type of task that represents a previously defined process. You can reuse existing processes by dragging them onto subprocess entities in the Process editor. The process associated with the subprocess task is triggered by the evaluation of a rule. You can use a subprocess task within a process or it can be run outside the process, as another process on its own.

To design a subprocess:

1. From the Studio Projects view, drag one or multiple existing processes onto the subprocess entity in the Process editor.
2. Right-click the subprocess and select **Assign Order**.

   The Order Selection dialog box is displayed.
3. Select the order type to associate with the subprocess.
4. Click **OK**.
5. Right-click the subprocess entity and select **Show Properties**.
6. In the Properties view **General** tab, define the basic information about the subprocess.

   For example, you can define subprocess description and display name. See "Subprocess Properties General Tab " for more information.
7. In the Properties view **Process** tab, associate the subprocesses to the rules that trigger their rules processing.

   You define process and rule combinations that determine which process the system initiates when a rule evaluates to *True.* The system evaluates each rule in the order that you specify. When a rule evaluates to *True*, the system runs the corresponding process and ignores the remaining processes. See "Subprocess Properties Process Tab " for more information.
8. In the Properties view **Exception Map** tab, define how the subprocess task handles process exceptions.

   For example, you can map subprocess exceptions to the completion statuses of the subprocess task, or to exceptions on the parent process. See "Subprocess Properties Exception Map Tab " for more information.
9. Click **Save**.

> **Note:**
>
> Do not deploy updated process flows (for example, adding additional parallel tasks to a subprocess) to an OSM run-time environment until all orders submitted to the process have completed.

Related Topics

Subprocess Properties General Tab

Subprocess Properties Process Tab

Subprocess Properties Exception Map Tab

Designing Exception Paths

Working with Processes

Working with Orders

Subprocess Properties General Tab

You use the Subprocess Properties General tab to define general information about the subprocess, such as the description and display name.

Field	Use
Description	Enter a description of a task to help differentiate between tasks used more than once in the same process flow.
Display Name	Enter the name that you want to appear in the subprocess task entity in the Process editor.
Process History	Select **True** if you want this task to appear in the Process History - Summary Table window in the Task web client. Otherwise, select **False**.
X, **Y** coordinates	Coordinates that indicate where in the Process editor the subprocess task exists.

Related Topics

Designing Subprocesses

Working with Processes

Subprocess Properties Process Tab

When creating a subprocess task, you can associate the subprocess with a list of process and rule combinations, and sequentially order them. The first rule an order satisfies defines the process that is used as the subprocess.

Select the Properties view Process tab to define the values for the following:

Field	Use
Pivot Node	Click **Select** to access the Order Template Selection dialog box, where you can select the data element on which OSM will spawn the individual instances. For example, if you have subprocess that will create an email address for every person in a list, you might select the node Person as the pivot node, so that the subprocess repeats, spawning an instance for each person.
Sequential	If you anticipate that a large number of task instances will appear in the Task web client Worklist, and you prefer that the system display the instances in the worklist one at a time, you can specify a sorting option. Specifying a sorting option here causes OSM to run the task instances sequentially, instead of in parallel, and to display the individual instances in the worklist one at a time, in the specified order. As you complete each task instance, OSM spawns the next instance. You can select one of the following sorting methods: **Non Sequential:** The subprocess initiates every instance simultaneously. **No Sorting**: The Task web client Worklist displays the instances of the task in the order that they are spawned. **Ascending**: The Task web client Worklist displays the instances of the task in ascending order, based on the attribute (date, alpha, or numeric) of the data element that you select in the **Sort Element** field. **Descending**: The Task web client Worklist displays the instances of the task in descending order, based on the attribute (date, alpha, or numeric) of the data element that you select in the **Sort Element** field.
Sort Element	If you selected a reusable structure as the pivot node, click **Select** to identify which of the data elements the system should use to determine the order in which to display the individual task instances in the Task web client Worklist. This option is available only when you sort the task instances in descending or ascending order. For example, consider that you have three levels of DSL service: Regular, Gold, and Platinum. You want to ensure that customers ordering the Platinum level of service have priority over the lower two levels. In the **Pivot Node** field, you might select the reusable structure called *service_type*, which contains the attributes *regular*, *gold*, and *platinum*. In the **Sequential** field, you can select *ascending*, then select the value *platinum* in the **Sort Element** field to ensure that the orders with the Platinum level of service are initiated first.
Continue if rule failed	Select if want the process to continue if none of the rules associated with the subprocesses equate to *True*.

Field	Use
Association	The Association table lists the process and rule combinations that determine which process the system will initiate when a rule in the table evaluates to *True.* The system evaluates each rule in the order that they appear in the Association table. When a rule evaluates to *True*, the system runs the corresponding process and ignores the remaining processes. You can prioritize the processes listed in the table by using the **Move Up** and **Move Down** buttons. Click **Add** to display the Add Rule and Process Association dialog box. In the dialog box, indicate which rule and process combination triggers the subprocess. To change the order of the process rule combinations, highlight a combination and click the **Move Up** or **Move Down** button. Select a process rule combination and click the **Modify** button to change the combination pair, or click the **Remove** button to delete from the list. If you drag a process from the Studio Projects view on to a subprocess entity, the system automatically adds that process to the Association table, and uses *null_rule* as the default rule. Select the table row and click **Modify** to change the rule and process combination. **Note:** You can manipulate ordered tables in Design Studio using keyboard controls. For example, you can launch the Add Rule and Process Association dialog box by pressing the Insert key on your keyboard. You can highlight a table row and press the Delete key to delete a process and rule combination from the table. Use the Control key in conjunction with the arrow keys to select one or multiple table rows and move those rows up or down in the index order.

Related Topics

Designing Subprocesses

Working with Processes

Subprocess Properties Exception Map Tab

When you use a subprocess task in a process, you can define how the subprocess task handles process exceptions. You can map subprocess exceptions to the completion statuses of the subprocess task, or to exceptions on the parent process. Using the subprocess mappings, a parent process can raise a process exception or complete the subprocess task. Additionally, a parent process can set the exception status, terminate the subprocesses, and set the process status. The list of subprocess mappings is ordered by priority. If a lower priority subprocess exception occurs after a higher priority exception, the lower priority exception is ignored. The subprocess task does not complete until all subprocess tasks are completed or terminated.

For example, consider that you 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. When *validate_address* throws the *invalid_address* exception, you can instruct it to complete or to raise an exception. If you are creating a VPN for a business that contains multiple physical locations, you might have multiple instances invoked for the

validate_address subprocess. 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.

Use the Properties view Exception Map tab to define values for the following:

Field	Use
Process	Select the subprocess for which you want to map exceptions. You can map exceptions to any of the subprocesses included in the Association table on the Properties view Process tab.
Exception	Select an exception to associate with the subprocess. Only those exceptions defined for the subprocess you selected in the **Process** field are available for mapping.
Action	Select an option to determine what action the parent process should take when the subprocess defined in the **Process** field throws the exception defined in the Exception field. Select: • *Complete Task* to complete the task and indicate normal flow. • *Raise Exception* to raise an exception status defined in the parent process. If you select this option, you must also select an exception throw value. The values available in the **Throw** field are those exception options that are defined at the parent process level.
Terminate	Define how invoked threads should be affected by the exception. Select from the following options: • *All instances (All)*: Select to instruct the system to stop all subprocess threads. • *Excepting instance only (One)*: Select to instruct the system to stop the subprocess thread that raised the exception. • *None (Ignore)*: Select to instruct the system to continue normal processing.

Related Topics

Designing Subprocesses

Working with Processes

Designing Workstream Processes

A workstream process enables Task web client users to run a series of sequenced order tasks through a wizard-like interface. When a process, or subprocess, is defined as a workstream, Task web client users flow from task to task after each transition without being returned to the worklist to initiate the next transition. Instead, the next task is invoked automatically, eliminating the need to manually navigate through the worklist to retrieve the next task.

Workstreams are designed by business analysts or process modelers in much the same way standard processes are designed, using the Process editor. To make a process a workstream process, right-click the **Start** node in the Process editor and select **Show Properties** to display the Properties view. Then, click the **Workstream** property and change the value to True.

Related Topics

Working with Processes

Designing Process Sequence and Flow

When you create a process, you can use the Process editor to describe the sequence in which tasks should complete and the manner in which the process flows from one task to the next. You create the structure of the process by manually dragging tasks into place, or by using the Process editor layout tools. To describe the link between tasks, you use the elements from the Process editor Flows and Exception Path drawers. Flows describe how tasks are completed and determine the order of tasks in the process.

To sequence tasks:

1. Drag the tasks and activities into the desired order.

   When multiple tasks are active, you can move them simultaneously. To make multiple tasks active, drag a selection rectangle around the tasks that you want to move, then move the group of task into the new position.

2. (Optional) Click the **Layout All Nodes** icon, located in the Process editor toolbar.

   Alternatively, you can right-click in the Process editor to access **Layout All Nodes** from the context menu. The **Layout All Nodes** feature automatically arranges the process nodes in a standard flow chart format.

3. (Optional) Select multiple tasks in the Process editor and click the **Layout Selected Nodes** icon, located in the Process editor toolbar.

   Alternatively, you can right-click in the Process editor to access **Layout Selected Nodes** from the context menu. The **Layout Selected Nodes** feature automatically arranges only a selected section of the process.

4. Click **Save**.

To link tasks:

1. In the Process editor palette Flows drawer, click a flow activity.
2. In the Process editor, click the **Start** node and then click the first task to link the two objects together.
3. From the palette, use the actions in the Flow and Exception Paths drawers to link all of the tasks.

   Click the **Toggle Sticky Tool Mode** button to repeatedly link tasks in the Process editor.

4. Right-click a flow and select **Status**.

   Select the task exit status for the flow. The available options include the statuses you previously defined for the task in the Task editor States/Statuses tab.

5. (Optional) Click the **Layout All Nodes** button in the Process editor toolbar.

   The **Layout All Nodes** feature automatically displays the process flow in an organized and neat arrangement.

   To arrange only a section of the process, use a drag selection to activate multiple tasks and click the **Layout Selected Nodes** button in the Process editor toolbar.

6. Click **Save**.

Related Topics

Working with Process Editor Menu Controls

Designing Exception Paths

Working with Processes

Working with Tasks

Process Editor Flow Properties General Tab

Use the **Process** editor Flow Properties view **General** tab to define attributes for process flows.

Field	Use
Condition	This property applies only to transitions that are part of a workstream process. You can apply rules to the transition that must evaluate to True for the transition to occur. Select the Condition row to access all rules defined in the order. **Note**: Selecting a condition in a transition not part of a workstream process will produce a build warning. **Note**: If all of a task's transitions include conditional rules that evaluate to false during run-time, the OSM server considers the task to be the terminal task, as there are no additional valid transitions to consider.
Mandatory Check	Select **True** to ensure that the system verifies that mandatory fields are present when a task completes.
Reporting Status	Enter the reporting status that you want to display in the Task web client. This status is tracked in the client's History. The **Reporting Status Value** field is an open text field. In the **Value** field enter a status name to indicate how the transition should be reported.
Status	Select the task exit status that represents this flow. The available options include the statuses you previously defined for the task in the Task editor States/Statuses tab.

Related Topics

Working with Processes

Modifying Process Editor Start Properties

Process Editor Flow Properties Events Tab

Use the Properties Events Details tab to create event notifications for a single task transition in a specific process that only triggers if a specific rule evaluates to true. When defining event notifications for task transitions in the Process editor Properties Events tab, you can name the task and transitional events, specify the rule that triggers the event, set the priority level, enable or disable the event, specify whether to send the notification by email, and create automation plug-ins to perform the work of the notification. See "Properties Events Detail Tab" for more information about the fields on the Events Detail tab. See "Creating Task Status-Based Event Notifications" for information about creating event notifications at the Process editor level.

Related Topics

Working with Event Notifications

Configuring Automation Plug-In Properties

Designing Exception Paths

An exception is a mechanism used to interrupt or stop an order, or to redirect it to a task in the process to a different process. The choices are defined by the system administrator and identified by the exception statuses. Exceptions may be used to cancel an in-flight order, to add supplemental information to an order and redirect the order to an earlier task in the process, or to take other actions defined in the original process.

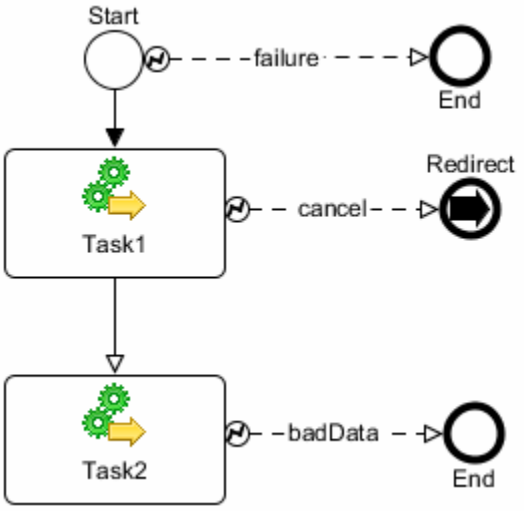
Exception statuses are user-defined statuses used to alter a process flow from anywhere in the process. The exceptions can be defined with restrictions that allow only specified workgroups, activities, or order type/sources (or combinations of these) to raise the exception.

> **Note:**
>
> If you have previously defined a process exception in the OSM Administrator, and intend to import it into Design Studio, you must ensure that:
>
> - You define (in the OSM Administrator Process Exception Definition tab) the status used in the process exception in any task that can initiate the exception.
> - You create a process exception restriction (in the OSM Administrator Process Exception Restriction tab) that specifies the tasks that can raise the exception.

Exception paths are used in conjunction with the **Redirect** and **End** activities to define process exceptions:

- The *End* activity stops the work order from continuing.
- The *Redirect* activity redirects the work order to another task in the same process or to a different process.

> **Note:**
>
> End and Redirect activities cannot be defined as the source end of a path.



In the Process editor, visual cues enable you to distinguish flow transitions from exception paths: exception paths are represented by an exception path source marker (lightning bolt icon) and a dashed line.

> **Note:**
>
> You can only configure two or more exception paths with the same status in the same process if the following conditions are met:
>
> - All the exception paths must start from a task, a rule, or a subprocess.
> - All the exception paths terminate on the same redirect or end.
> - In the properties of the path, the Reporting Status, Role Restriction, and Order Restriction must be the same.

To model exception paths in the Process editor

1. To model a Stop exception, draw an exception path from an activity to an **End** activity.

   Linking the path from the **Start** activity applies the exception to the entire process; linking from a task within the process applies the exception to that task.
2. To model a Redirect exception, draw an exception path from an activity to a **Redirect** activity. The **Redirect** activity supports both process-level and activity-level redirection.

Related Topics

Working with Event Notifications

Configuring Automation Plug-In Properties

Exception Path Properties General Tab

Exception Path Properties Restrictions Tab

Redirect Properties General Tab

Designing Subprocesses

Creating Task Status-Based Event Notifications

Working with Processes

Exception Path Properties General Tab

Use the Exception Path Properties **General** tab to define the reporting status that will display in the Task web client and the task completion status that initiates the exception.

Field	Use
Reporting Status	Enter the reporting status that you want to display in the Process Exception section of the Task web client. This status is tracked in the client's History. The **Reporting Status Value** field is an open text field. In the **Value** field enter a status name to indicate how the exception should be reported.

Field	Use
Status	Specify the task completion status that initiates the exception. If the exception is thrown from the **Start** activity, the values available include all of the statuses defined within the process. If the exception is thrown from a specific activity, the values available include only those statuses defined for that specific activity.

Related Topics

Designing Exception Paths

Working with Processes

Exception Path Properties Restrictions Tab

Use the Exception Path Properties Restrictions tab to restrict exceptions to specific roles.

Field	Use
Role Restriction	To restrict the ability to throw an exception to specific roles, select Allow Restricted Roles, then select those roles from the Available list and move them into the Selected list using the arrow buttons.
Available, Selected	Move roles between the **Available** and **Selected** fields by highlighting one or multiple roles and clicking the appropriate arrow button. To move all roles from one column to the other, use the double arrow buttons.

Related Topics

Designing Exception Paths

Working with Processes

Redirect Properties General Tab

Use the Redirect Properties General tab to define to which process and to which task within that process you want the order redirected.

Field	Use
Reference	Select the process to which you want the order redirected. To select the process, click inside the **Value** field to access the ellipsis button. Click the ellipsis button and select a process from the entity list. **Note:** After you define the Reference process, you can double-click the **Redirect** entity to open the process in the Process editor.
Reference Task	Select the task to which you want the order redirected. When you specify a process in the **Reference** field, you can also redirect to a specific starting point within that process. The process will run to completion from the task that you select here. If you select no task here, the process will start with the first task.
X, **Y** coordinates	These coordinates indicate where on the Process editor canvas the **Redirect** activity exists.

Related Topics

Designing Exception Paths

Working with Processes

7

Working with Tasks

A task is one step in a process (a process is a sequence of tasks that run either consecutively or concurrently to fulfill an order or part of an order). In Design Studio, you model tasks and the data necessary to run them. The tasks are processed in an Oracle Communications Order and Service Management (OSM) run-time environment.

When working with tasks, see the following topics:

About Tasks

A task is a specific activity that must be carried out to complete the order. OSM has more than one type of task:

- A manual task requires user intervention, which is performed through the OSM Task web client. See "Working with Manual Tasks" for more information.
- An automated task is run without manual intervention and is used to handle interaction with external fulfillment systems. See "Working with Automated Tasks" for more information.
- An activation task is a type of automated task that interacts with Oracle Communications ASAP or Oracle Communications IP Service Activator. See "Working with Activation Tasks" for more information.
- A transformation task is a type of automated task that accesses the OSM order transformation manager. See "Working with Transformation Tasks" for more information.

All types of tasks share many of the same modeling activities, such as defining task data, details, and compensation. Some configuration steps, however, are specific to each task type.

Related Topics

Creating New Tasks

Task Editor

About Task Extensions and Inheritance

During task creation, you can base new tasks on the functionality of an existing task by using the extend feature. When you extend a task, the extended task inherits all of the data, rules, and behaviors of the parent task. 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.

You cannot edit task data inherited from a parent task. For example, if you are working with a task that includes data inherited from a parent task, you cannot remove, rename, or reposition data elements inherited from the parent task, make changes to inherited behaviors, and so forth. Nodes inherited from the extended task are represented by a black icon and cannot be removed from the task. New nodes that you add to the task are represented by a green icon and can be removed from the task.

Additionally, tasks in a cartridge project can inherit from tasks in a different cartridge project, if the order with which the task is associated extends the order from the same source cartridge. For example, a task named **Task2** in a cartridge named **Cartridge2** can inherit data from a task named **Task1** in a cartridge named **Cartridge1**, but only when the order (with which the task is associated) in **Cartridge2** extends the order (with which **Task1** is associated) in **Cartridge1**.

> **Note:**
>
> Design Studio does not permit cyclic referencing. For example, if task **T2** extends from task **T1**, and task **T3** extends from task **T2**, then you cannot extend task **T1** from task **T3**.

Related Topics

Task Editor

Creating New Tasks

About Task States and Statuses

A task *state* determines the milestone of a task in a process. The default states are:

- Received: The task has been received in the system and is waiting to be accepted.
- Accepted: The assigned user (or system) has accepted the task. The task is locked so that it cannot be modified or completed by other users and systems.
- Completed: The task is finished.
- Assigned: (manual tasks only) The task has been assigned to a user.
- Create Activation Work Order Failed: (activation tasks only) The task attempted to create a work order in the activation system but work order creation failed.

These states are mandatory and cannot be removed. You can define additional states (user-defined states) to support your business processes. If a task cannot be completed on time, you can change the task state to Suspended.

A task *status* describes how a task was completed and determines the next task in the process. Several default statuses are given for each task type. For example, the default statuses for a manual task are **Back**, **Cancel**, **Finish**, and **Next**. The default statuses for an automated or transformation task are **Failure** and **Success**. The default statuses for activation tasks are **Success**, **Activation Failed**, and **Update OSM Order Failed**. You can select from the set of additional predefined statuses (Delete, False, Rollback, Submit, Failed, and True), and you can also define your own.

> **Note:**
>
> A status represents a transition between tasks. The statuses that you define in the Task editor **States/Statuses** tab appear as task transition options in the Task web client. Statuses that you define for a task but fail to use in the process still appear in the Task web client as transition options. If a user selects one of these unmapped options, the OSM server terminates the process.

Related Topics

Assigning Task States and Statuses

About Task Rollback Status

Task Editor States/Statuses Tab

About Task Rollback Status

When you run a rollback from within a given task, the system processes an update that restores the order to the state that the order was in when the previous task finished. The system accomplishes this by deleting the new nodes, inserting the removed nodes, and restoring the updated nodes with the data they held before the current state.

If you are within a subprocess, you can roll back only to previous subprocess tasks. You cannot roll back from the first subprocess task to the previous parent task. This means that if you want to roll back an entire subprocess task, you must complete the subprocess, proceed to the next task in the main process, then roll back to the parent process task that spawned the subprocess in question.

Because the rollback runs when the Task web client user clicks the **Back** button, the order advances according to how you define the rollback status in the process.

If a task is at the rollback status, the **Mandatory Check** option is disabled for that task. See "Process Editor Flow Properties General Tab" for more information about the **Mandatory Check** option.

Related Topics

Working with Manual Tasks

Designing Workstream Processes

About Task States and Statuses

Task Editor States/Statuses Tab

About Task Compensation

Compensation refers to the changes OSM makes to an order when a customer requests a change to the order after the service fulfillment process has been initiated or when a failure occurs during order processing that triggers compensation for fallout management purposes. OSM manages these in-flight changes, called order amendments, by analyzing the tasks performed in the process and determining whether data has changed. This analysis, called Order Change Management, is necessary when, for example:

- A consumer orders high-speed internet access, then calls back midway through the service fulfillment process to change bandwidth or cancel the order.
- A business customer orders VoIP service for several office locations and then calls back multiple times to change the feature sets for the various offices as the order is being processed.
- An error occurs in an in-flight order where a user or an automation plug-in can raise an exception, trigger fallout, and make changes to compensate for errors that caused the error.

In Design Studio, you configure the manner in which OSM manages these changes to in-flight orders by indicating how the run-time environment compensates the task if it is affected by an amendment.

> **Note:**
>
> Compensation ignores timer delays and event delays between tasks while undoing or redoing the tasks in a process.

Execution Modes

Tasks can run in one of the following possible execution modes:

- *Do* is the default mode for a task that runs under normal processing.
- *Undo* reverses the effects of the associated *Do* operation.
- *Redo* combines both *Undo* and *Do* operations in a single operation.
- *Do in Fallout* is the mode that a task uses after the task has moved from the Do execution mode because of a failure condition. You can use this mode to manually investigate and resolve failure conditions or trigger automation plug-ins set to run in the fallout mode.
- *Undo in Fallout* is the mode that a task uses after the task has moved from the compensation Undo execution mode because of a failure condition. You can use this mode to manually investigate and resolve failure conditions or trigger automation plug-ins set to run in the fallout mode.
- *Redo in Fallout* is the mode that a task uses after the task has moved from the compensation Redo execution mode because of a failure condition. You can use this mode to manually investigate and resolve failure conditions or trigger automation plug-ins set to run in the fallout mode.

On the **Compensation Strategy** tab of the Task editor, you define how the run-time environment compensates tasks if they are affected by an amendment. Each section on this tab represents a scenario to consider when determining a compensation strategy for the task.

You can model a static compensation strategy from the predefined list or a dynamic compensation strategy based on an XQuery expression that compares data from the revision order with the results of a comparison between the current order perspective and the historical order perspective. Dynamic compensation strategies can be useful if you want to model different compensation types based on revision order data values. For example, you could model the XQuery 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.

You can also configure whether a task is included in compensation when it is completed or in progress, or you can use an XQuery expression that evaluates whether an in progress task can be included in compensation based on order data. This ability is important for long running tasks where a response to a request takes hours or even days but still needs to be considered in compensation. If you specify that a task can be compensated while it is in progress, you can also specify whether a grace period should be observed before performing the compensation. In addition, you must use an XQuery expression to evaluate any changes to the compensating task data to identify when the compensation has completed and the task can enter into normal do mode again.

For example, some automated plug-ins communicating with workforce management systems may involve the dispatching of personnel to perform work over several days. In such cases the automation plug-in sends the dispatch request to the workforce management system, and remains in progress until such time as the work is completed. If a revision order were to arrive that changes some aspects of the work required by the dispatched personnel, then the in progress automation plug-in responsible for sending the original request should be included in the compensation plan. You can specify an XQuery that evaluates data on the in progress task communicating to the workforce management system that determines if the task needs to be compensated. In addition, you can specify whether a wait period should be observed before starting compensation. You must also write an XQuery that determines when compensation has completed, for example, when the task receives the response from the new request indicating the workforce management system has received the new work details and has begun to processing the request.

See "Compensation XQuery Expressions" for more information about compensation XQuery expressions.

> **Note:**
>
> If you modify the default strategy settings, ensure that you analyze the task within the context of the entire workflow because modifying strategy settings in one task can adversely affect subsequent tasks and task data.

Related Topics

Task Editor Compensation Tab

About Task Fallout

Fallout refers to orders that encounter problems during fulfillment and therefore fall out of normal processing. OSM places these orders in Failed state (you can also manually fail orders in the Order Management web client).

In the Design Studio Order editor, you associate a fallout name with one or multiple data nodes whose values you will want to review (in the Order Management web client) when the

corresponding type of fallout occurs. In the Task editor, you associate tasks with the types of fallout that can occur for the task.

See "Task Editor Fallouts Tab" when modeling task fallout in Design Studio.

Ensure that the Exception Processing permission is assigned to roles that are assigned to fallout tasks.

See "Role Editor Role Tab" for information on assigning permissions to role entities.

Related Topics

Defining Order Fallout

Order Editor Fallouts Tab

Order Editor Fallout Groups Tab

About Enabling Task Web Client Users to Reassign Tasks

A Task web client user can reassign a task only after you configure the task for reassignment in Design Studio for OSM.

Before a Task web client user can reassign a task, you must ensure the following:

- You have added the **Assigned** state to the task. See ""Assigning Task States and Statuses" for more information.
- You have added a role to the task that is responsible for reassigning the task. See "Assigning Task Permissions" for more information.
- You have an Oracle WebLogic Server user account that is a member of the role entity, using the Administration area of Order Management web client, that has the **Task Assignment** permission applied. See "Role Editor Role Tab" for more information.

See *OSM Task Web Client User's Guide* for information about reassigning a task.

Creating New Tasks

You create tasks to include in processes that represent the activities required to offer a specific service to a customer. When creating tasks, you can create new tasks with minimal information or select an existing task upon which to base the new task entity.

To create new tasks:

1. From the **Studio** menu, select **New**, then select **Order and Service Management**, and then do one of the following:
   - Select **Order Management**, then select **Activation Task**
   - Select **Order Management**, then select **Automated Task**
   - Select **Order Management**, then select **Manual Task**
   - Select **Order Transformation**, then select **Transformation Task**
2. In the **Project** field, select the appropriate OSM project for this task.

   By default, the project under which the process was created is selected.
3. (Optional) In the **Extends** field, select an existing task to leverage the task data and extend the functionality of that existing task.

Click **Select** and select a task for the **Extends** field. If a suitable task does not yet exist, click **New** to create the task. See "About Task Extensions and Inheritance" for more information.

4. (For activation tasks only) In the **Activation System** field, select the activation system type that the task will be communicating with.
5. Associate the task with an order, if necessary.

   If you are creating a new task from the Design Studio main menu, you must associate the task with an order, and the **Order** field appears in the wizard. When you add tasks to the Process editor using the Task tool from the Activities drawer, the task is already associated with an order through the Process entity, and the **Order** field will not appear in the wizard.

   > **Note:**
   >
   > If you are planning to use the task for an order (OrderA) and also an order (OrderB) that is extended from that order, you must select the parent order (OrderA) here.

6. In the **Name** field, enter a name for the new task.

   The name must be unique among the task entity types. Two tasks cannot share the same name, even if they are different types of tasks. For example, an automated task and a manual task cannot share the same name.
7. (Optional) Select a location for the task.

   By default, Design Studio saves the task to your default workspace location. You can enter a folder name in the **Folder** field, or select a location different from the system-provided default. To select a different location:

   a. Click the **Folder** field **Browse** button.
   b. Navigate to the directory in which to save the entity.
   c. Click **OK**.
8. (For transformation tasks only) In the **Transformation Manager** field, select a transformation manager to be called from the task.
9. (For transformation tasks only) In the **Order Component** field, select an order component to be used in the task.
10. Click **Finish**.

    Design Studio displays the new task entity under the selected project in the Studio Projects view.

Related Topics

Task Editor

About Task Extensions and Inheritance

Defining Task Data

Task data is the data that the task requires for completion. You can add data to a task directly, or it can be provided on the order data or inherited from a different task. You can model task data in several ways using the **Task Data** tab of the Task editor. If you created the task using

the **Create Task** wizard and you specified an existing task to use, that task's data is displayed in the Task Data area.

When extending the task data, see the following topics:

- Creating Simple Data Elements, Structured Data Elements, and Data Structure Definitions
- Adding Data to a Task
- About the Task Editor Task Data Context Menu

> **Note:**
>
> Different tasks can have different default values for the same data element. Because of this, whenever you add data from the Data Dictionary to a task, the default value defined in the Data Dictionary will not be inherited by the data element on the task.

Adding Data to a Task

You can add any data defined in the Data Dictionary or the order to the task data.

To add data you have previously created:

1. Right-click inside the Task editor **Task Data** tab.

   See "Task Editor Task Data Tab" for more information about the fields on this tab. When modeling data for activation tasks, navigate to the **Task Request Data** tab and right-click in the Task Data area.

2. Select **Select from Order Template** or **Select from Data Schema**.

   A dialog box is displayed, enabling you to select data elements.

   > **Note:**
   >
   > You can alternatively select **Open Data Element view** and then drag data elements from the Data Dictionary onto the Task Data area.

3. Select which data you want to add to the task.

   > **Tip:**
   >
   > When selecting data to add to the task:
   >
   > - Press and hold the Shift key to select multiple consecutive elements. Or, press and hold the Control key to select multiple non-consecutive elements.
   > - Select a parent node to add all data elements (simple and structured data elements) in its hierarchy.
   > - Select a child node to add only the child node and its parent nodes. Design Studio automatically adds parent nodes associated to the child node up to the root of the data schema.

4. Click **OK**.

Design Studio adds the elements to the Task editor Task Data area.

5. Click **Save**.

Related Topics

Working with Orders

Task Data Node Properties View Identification Tab

Task Data Node Properties View Dictionary Tab

Task Editor

Adding a New Data Structure Definition to a Task

You can create and add a new data structure definition to a task. See "About OSM Data in Model Projects" for more information about data structure definitions.

To add a new data structure definition to a task:

1. Right-click inside the Task editor **Task Data** tab.

   See "Task Editor Task Data Tab" for more information about the fields on this tab. When modeling data for activation tasks, navigate to the **Task Request Data** tab and right-click in the Task Data area.

2. Select **Select Data Structure Definition**.

   A dialog box is displayed, enabling you to select data structure definitions.

> **Note:**
>
> You can alternatively select **Open Data Element view** and then drag data elements from the Data Dictionary onto the Task Data area.

3. Select the data that you want to add to the task.
4. Click **OK**.

   Design Studio adds the data structure definitions to the Task editor Task Data area.

5. Click **Save**.

Related Topics

Working with Orders

Task Editor

Adding an Existing Data Structure Definition to a Task

To add data structure definitions that you have previously created:

1. Right-click inside the Task editor **Task Data** tab.

   See "Task Editor Task Data Tab" for more information about the fields on this tab. When modeling data for activation tasks, navigate to the **Task Request Data** tab and right-click in the Task Data area.

2. Select **Select Data Structure Definition**.

A dialog box is displayed, enabling you to select data structure definitions.

> **Note:**
>
> If no data structure definitions are displayed in the Matching items area, you must define the dependency of the data structure definition to the model project before you add it to the task data. For more information, see "Managing Project Dependencies".

3. Select the data structure definitions that you want to add to the task.

> **Tip:**
>
> When selecting data structure definitions to add to the task:
>
> - Press and hold the Shift key to select multiple consecutive elements. Or, press and hold the Control key to select multiple non-consecutive elements.
> - Select a parent node to add all data elements (simple and structured data elements) in its hierarchy.
> - Select a child node to add only the child node and its parent nodes. Design Studio automatically adds parent nodes associated to the child node up to the root of the data schema.

4. Click **OK**.

   Design Studio adds the data structure definition, and all its child data elements and structures, to the Task editor Task Data area.

> **Note:**
>
> Derived data structure definitions are not displayed in the Task Data area.

5. Click **Save**.

Related Topics

Working with Orders

Task Editor

Assigning Task States and Statuses

When you create a task, the system assigns three mandatory processing states, which cannot be removed, to the task: Accepted, Completed, and Received. You can also select additional predefined states. Similarly, you assign completion statuses to the task by selecting from a list of predefined statuses or by adding your own.

When assigning task states and statuses, see the following topics:

- About Task States and Statuses
- About Task Rollback Status

Assigning States to Tasks

When you create a task, the system assigns three mandatory processing states, which cannot be removed, to the task: Accepted, Completed, and Received. You can also assign additional states to tasks and remove states that are assigned to tasks.

To assign predefined states to tasks:

1. In the Task editor, click the **States/Statuses** tab.

   See "Task Editor States/Statuses Tab" for more information about the fields on this tab.
2. In the States area, click the **Select** button.

   The Select a State dialog box is displayed.
3. Select a user-defined state to assign to the task.

   See "About Task States and Statuses" for more information about mandatory task states and user-defined task states.
4. Click **OK**.

To create a new state and assign it to a task:

1. In the Task editor, click the **States/Statuses** tab.

   See "Task Editor States/Statuses Tab" for more information about the fields on this tab.
2. In the States area, click the corresponding **Add** button.

   The Add State dialog box is displayed.
3. Enter a name and a display name for the new state.
4. Click **OK**.

To remove a state assignment for a task:

1. In the Task editor, click the **States/Statuses** tab.

   See "Task Editor States/Statuses Tab" for more information about the fields on this tab.
2. Select a state and click **Remove** to delete a state from the list.

Assigning Statuses to Tasks

You assign completion statuses to the task by selecting from a list of predefined statuses or by adding your own. You can also remove statuses that are assigned to the task.

To assign statuses to tasks:

1. In the Task editor, click the **States/Statuses** tab.

   See "Task Editor States/Statuses Tab" for more information about the fields on this tab.
2. In the Statuses area, click the corresponding **Select** button.

   The Select a Status dialog box is displayed.
3. Select a predefined status to assign to the task.

   See "About Task States and Statuses" for more information about statuses.

4. Click **OK**.

To create a new status and assign it to a task:

1. In the Task editor, click the **States/Statuses** tab.

   See "Task Editor States/Statuses Tab" for more information about the fields on this tab.

2. In the Status area, click the corresponding **Add** button.

   The Add Status dialog box is displayed.

3. Enter a name and a display name for the new status.

4. Select a value for the **Constraint** field.

   All statuses have a corresponding constraint severity level that determines the transition behavior of the task when a constraint violation occurs. When you add a new status and select the associated default constraint, the system displays a list of available values.

   See "Defining Constraint Behavior Properties" for more information.

5. Click **OK**.

To remove the status assignment from a task:

1. In the Task editor, click the **States/Statuses** tab.

   See "Task Editor States/Statuses Tab" for more information about the fields on this tab.

2. Select a status and click **Remove** to delete a status from the list.

Assigning Task Permissions

You assign execution modes to roles for each task to specify which roles can perform the execution mode. For example, you can restrict a particular role from performing a redo on a task.

To add a role to the Permissions table:

1. In the Task editor, click the **Permissions** tab.

2. Select an existing role or create a new role to add to the Task Permissions table.

   Do one of the following:

   - Click **Select** to add an existing role to the list.
   - Click **New** to create a new role.

     See "Creating New Roles" for more information.

> **Note:**
>
> For automated and transformation tasks, Oracle recommends that you select or create an automation role. Assign the **oms-automation** user to this role using the OSM Administration area of the Order Management web client, and assign permissions for automated tasks to the automation role only. Using an automation role ensures that only automation plug-ins process automated tasks.
>
> See *OSM Order Management Web Client User's Guide* for more information about assigning users to roles.

3. For each role listed in the Role Name column, select or deselect, as appropriate, the **Do**, **Undo**, **Redo**, **Do in Fallout**, **Undo in Fallout**, and **Redo in Fallout**, check boxes to enable or disable access to the task execution modes.

   See "Task Editor Permissions Tab" for more information about task execution modes.

4. (Optional) To view the permissions defined for a role, select a role and click **Open**. The system displays the role in the Role editor, where you can view the permissions assigned to the role. You assign permissions to a role to give the users in that role access to related functions in the Task web client.

5. (Optional) To delete a role assignment for a task, select the role and click **Remove**.

Related Topics

Working with Roles

Working with Manual Tasks

Converting Tasks

Design Studio enables process modelers to create tasks in the absence of detailed design-level information. Consequently, as you model your tasks, you may need to convert the task from one type to a different type.

You can convert between manual, automated, activation, and transformation tasks.

> **Caution:**
>
> When converting from one type of task to another, Design Studio displays a prompt if the potential for data loss exists (for example, when converting from an automated task to a manual task). Consider your task conversions carefully before implementing them.

To convert a task to a different task type:

1. In the Process editor, right-click a task and select **Convert**.
2. Select the task type to which you want to convert.

Related Topics

Deleting Unreferenced Tasks

Working with Tasks

Deleting Unreferenced Tasks

When you delete an activity from the Process editor, the system identifies all referenced task entities. If no other activities reference the task, you can define your OSM system preferences to delete these orphaned tasks.

To define OSM general preferences to delete unreferenced tasks:

1. From the **Window** menu, select **Preferences**.

   The Preferences dialog box is displayed.

2. In the Preferences navigation tree, expand **Oracle Design Studio**, and then select **Order and Service Management Preferences**.
3. In the **Delete Orphaned Task References with Activity** field, do one of the following:
   - If you want the system to automatically delete orphaned tasks, select **Always**.
   - If you want the orphaned tasks to remain in the workspace, select **Never**.
   - If you want the system to display a prompt for the user, select **Prompt**.

> **Note:**
>
> If activities other than the deleted activity reference a task, the system does not delete the task or display a prompt, because the task is still required elsewhere.

4. Click **OK**.

   Design Studio saves your preferences.

Related Topics

Converting Tasks

Working with Tasks

Working with Automation Plug-Ins

You use automation plug-ins to implement specific business logic automatically. 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.

When working with automation plug-ins, see the following topics:

- About Automation Plug-ins
- Creating New Custom Automation Plug-ins
- Configuring Automation Plug-In Properties
- Example: Modeling a Basic Automator Plug-In

Related Topics

About Tasks

Task Editor

About Automation Plug-ins

When learning about automation plug-ins, see the following topics:

- About Automation Plug-in Types
- About Automation Plug-in Association
- About Automation Message Correlation

About Automation Plug-in Types

There are two basic types of delivered automation plug-ins: Sender and Automator. Each type can be implemented using XSLT or XQuery, and each type can be defined as an internal event receiver (the JMS message that triggers the call to the plug-in is generated by OSM) or as an external event receiver (the JMS message that triggers the call to the plug-in is generated by an external system).

- Automator plug-ins receive information from OSM or from an external system, then perform some work. Depending on how you configure the plug-in, it can also update the order data. See "Predefined Automation Plug-ins" for sample XQuery and XSLT automators.
- Sender plug-ins receive information from OSM or from an external system. They perform some business logic, and they may or may not update an order, depending on your configuration. Additionally, they can produce outgoing JMS or XML messages to an external system. When generating JMS messages, you can define JMS messages to connect to a topic or queue. See "Predefined Automation Plug-ins" for sample XQuery and XSLT senders.

> **Note:**
>
> XQuery automation types cannot be implemented when using releases prior to OSM 7.0.

Related Topics

Adding Automation Plug-ins to Automated Tasks

Task Editor Automation Tab

About Automation Plug-in Association

When you add an automated task to a process, you must associate at least one automation plug-in for the task. To associate an automation plug-in for a task, you open the automated task entity in the Automated Task editor, and add the plug-in to the task in the **Automation** tab. When you deploy your cartridge to the run-time environment, the OSM server detects a task that has an automation plug-in associated with it and the server triggers the plug-in to perform its processing.

An automated task might have only a single automation plug-in associated with it. For example, you might associate a built-in Automator plug-in with the task to interrogate the task data, perform some calculation, update the order data, and transition the task. In this example, as soon as the Automator plug-in has finished processing, it updates the task with an exit status, and the OSM server moves to the next task.

An automated task can have multiple associated automation plug-ins. For example, you might want to associate multiple plug-ins with a task to represent conversations with external systems. You can associate a built-in Sender plug-in to receive the task data and send it to an external system for processing. That external system might send an acknowledgment back to a queue, where a second Automator plug-in (one that is defined as an external event receiver: it receives data from external system queues) consumes the reply and updates the order data with the response. A third Sender plug-in might send the external system a message to begin

processing, and a fourth Automator plug-in can receive the "processing complete" message from the external system, update the order, and transition the task.

Related Topics

Adding Automation Plug-ins to Automated Tasks

Task Editor Automation Tab

About Automation Message Correlation

Automation plug-ins defined as external event receivers are designed to process JMS messages from external systems. JMS messages are asynchronous, therefore external event receivers provide a method of correlating responses to requests previously delivered to enable you to map OSM orders to external system orders.

To correlate responses, the plug-in sets a property on the outbound JMS message, with the name of the value set for the correlation property in the **automationmap.xml** file and a value decided by your business logic.

For example, business logic might dictate that you correlate on a reference number. The external system copies the properties that you defined for the correlation on the request and includes that data in the response.

You can use the **Message Property Selector** field to filter messages placed on the queue and determine which automation to run. You define the **Message Property Selector** value as a Boolean expression that is a string with a syntax similar to the **where** clause of an SQL **select** statement. For example, the syntax may be:

```
"salary>64000 and dept in ('eng','qa')"
```

When the condition evaluates to true, the message is picked up and processed by the automation that defined that condition.

In a second example, consider that an external system defines five order types and OSM defines a different automation to process each order type. Each automation defines a different value in the **Message Property Selector** field, such as orderType=1, orderType=2, and so forth. When a message is sent to the queue by the external system, and the message includes the order type upon which the condition is based, the automation framework evaluates each condition until one evaluates to true. If more than one automation defines the same condition, the first one that evaluates to true is picked up and processed.

> **Note:**
>
> When you define only one automation plug-in external event receiver for each automated task, and you use the Optimized build-and-deploy mode to build and deploy automation plug-ins, you are not required to enter a selector in the **Message Property Selector** field. (For OSM 7.3 servers and later, Optimized is the only build-and-deploy mode available.) In this case, automated tasks can share the same JMS queue without a message property selector being set. You must set a message property selector when you do any of the following:
>
> - Define multiple automation plug-in external event receivers for the same automated task.
> - Set up other applications (besides OSM) to share the same queue that an external event receiver is listening on.
> - Use the Legacy build-and-deploy mode to build and deploy cartridges with automation plug-ins.
> - Use both (Allow server preference to decide) build-and-deploy mode to build and deploy cartridges with automation plug-ins and configure the OSM server dispatch mode for the Internal mode.
>
> See "Defining Build-and-Deploy Modes for Automation Plug-ins" for information on build-and-deploy modes.

Related Topics

Adding Automation Plug-ins to Automated Tasks

Task Editor Automation Tab

Creating New Custom Automation Plug-ins

Automation plug-ins enable you to extend OSM behavior by running specific business logic when events occur, sending and receiving data to and from external systems, and updating orders.

Design Studio supports two types of built-in automation plug-ins: Sender and Automator. Additionally, you can create your custom automation plug-ins using the custom automation plug-in template. You can, for example, write your own custom code to make CORBA or web services calls to external systems and register the custom plug-in against an automated task.

To create a custom automation plug-in:

1. From the **Studio** menu, select **New**, then select **Order and Service Management**, and then select **Custom Automation Plug-in**.

   The Custom Automation Plug-in wizard is displayed.
2. In the **Project** field, select the project in which to save the new custom plug-in.
3. In the **Name** field, enter a name for the plug-in.

   The name must be unique among the automation entity types within the same namespace.
4. Click **Finish**.

   The Custom Automation Plug-in editor is displayed.
5. Click **Select**.

The Select Java Class dialog box is displayed. Select from the **src** folder the Java class that implements the automation interface. See *OSM Developer's Guide* for more information about Java classes and custom plug-ins.

6. Click **OK**.
7. In the **XML template** field, enter the XML required for the plug-in's implementation.
8. In the **Documentation** field, enter information about the plug-in.

Related Topics

Example: Modeling a Basic Automator Plug-In

Working with Automated Tasks

Configuring Automation Plug-In Properties

After you add a plug-in to the Design Studio entity, you define the plug-in properties.

> **Note:**
>
> The type of automation (for example, an XSLT Sender) and the automation function (for example, a task event) determine which tabs appear in the Properties view.

To configure automation plug-ins:

1. From the **Studio** menu, select **Show Design Perspective**.
2. In an OSM entity editor Automation subtab, select an automation plug-in. You can configure automation plug-ins in the following editors:
   - The Order editor Jeopardy tab Automation subtab.
   - The Order editor Notifications tab Automation subtab.
   - The Order editor Events tab Automation area.
   - The Manual and Automated Task editor Jeopardy tabs Automation subtabs.
   - The Manual and Automated Task editor Events tabs Automation subtabs.
   - The Process editor, flow lines with status transition defined in the Properties subtab Events subtab Automation subtab.
   - The Process editor exception path flow lines in the Properties subtab Events subtab Automation subtab.
   - The Process editor Automated and Manual Tasks in the Properties subtab Events subtab Automation subtab.
3. Click **Properties**.

   The Properties view is displayed showing the automation plug-in properties tabs. If you selected properties from the process editor properties sub-tab, then the Properties view displays in a Additional Properties dialog box.
4. Click the **Details** tab.

   In the **Details** tab, you can name the plug-in and identify the user whose credentials will be used to run the automation plug-in. See "Properties View Details Tab" for more information.

5. Click the **Compensation** tab.

> **Note:**
>
> This tab is called the **Execution Mode** tab in the Manual and Automated Task editor Events tabs Automation subtabs.

6. In the **Compensation** tab, specify the execution modes in which the plug-in will run when called.

   See "Properties View Compensation Tab" for more information.

7. (For external event receivers only) Click the **External Event Receiver** tab.

   Define the name of the external system from which the plug-in receives messages. Additionally, you can define whether the plug-in filters for specific properties on the incoming message header or body. Finally, you can provide specific connection information if the messages are retrieved from an external system. See "Properties View External Event Receiver Tab" for more information.

8. (For external event receivers only) Click the **Correlation** tab.

   Map messages from external systems to specific OSM tasks. You can use the JMSCorrelationID or enter an XPath expression to filter for a specific element in the XML body of the message. See "Properties View Correlation Tab" for more information.

9. (For XSLT types only) Click the **XSLT** tab.

   Define where the XSLT style sheet is located, caching properties for the style sheet, and the exit status that the plug-in should use if it throws an exception.

   See "Properties View XSLT Tab" for more information.

10. (For XQuery types only) Click the **XQuery** tab.

    Define where the XQuery file is located, caching properties for the file, and the exit status that the plug-in should use if it throws an exception.

    See "Properties View XQuery Tab" for more information.

> **Note:**
>
> XQuery automation types cannot be implemented when using releases prior to OSM 7.0.

11. (For Sender types only) Click the **Routing** tab.

    Define where the automation plug-in should send messages, and where the external systems should send responses. See "Properties View Routing Tab " for more information.

12. Click the **Notes** tab.

    Document the intended use of the automation plug-in.

13. Click **Save**.

Example: Modeling a Basic Automator Plug-In

This example demonstrates how to configure an Automator plug-in that receives data from an internal OSM JMS queue and updates order data using an XSLT style sheet. In the example, assume that the XSLT style sheet includes conditional logic to apply a level 1 priority to the order if the order is from a specific customer.

This example demonstrates how to:

1. Create an automated task and add the relevant task data.
2. Add an automation plug-in to the automated task.
3. Configure the automation plug-in properties.

> **Note:**
>
> An automated plug-in exists within the context of a Design Studio cartridge project, order, process, and automated task. For purposes of demonstration, this example assumes the existence of multiple Design Studio entities. For example, it assumes the existence of a cartridge project called DSLCartridge, an order called DSLOrder, a process called DSLProcess, and an XSLT style sheet called check_customer.xslt that populates default values in the order data. It assumes that the Data Dictionary includes the two data nodes, customer_name and order_priority. It also assumes that the new automated task will be added to the DSLProcess entity. The naming conventions used in this example are for illustrative purposes only.

Step 1: Creating the automated task

1. From the **Studio** menu, select **New**, then select **Order and Service Management**, and then select **Automated Task**.

   The Automated Task wizard is displayed.
2. In the Automated Task wizard, enter or select the following values:
   - In the **Project** field, enter **DSLCartridge**.
   - In the **Order** list, select **DSLOrder**.
   - In the **Name** field, enter **Check_Customer**.

   See "Creating New Tasks" for more information.
3. Click **Finish**.

   The new automated task is displayed in the Automated Task editor.
4. Click the **Task Data** tab.

   In this example, you will update the order_priority field with a default value of 1 if the order is from a specific customer.

> **Note:**
>
> Normally, the task data includes all of the data that the task requires to complete. To simplify the example, this task includes only the two pertinent fields: customer_name and order_priority. See "Defining Task Data" for more information.

5. Right-click in the Task Data area and select **Select from Data Schema.**

   The Select Data Elements dialog box is displayed.

6. Select the data nodes **customer_name** and **order_priority**.

7. Click **OK**.

   The two data nodes are displayed in the Task Data area.

8. Click the **Permissions** tab.

   On the **Permissions** tab, you can ensure that only the automation role has permissions for automated tasks. See the note in "Assigning Task Permissions" for more information.

You are now ready to add a plug-in to the automated task.

Step 2: Adding the automation plug-in to the automated task

1. In the Automated Task editor, click the **Automation** tab.
2. Click **Add**.

   The Add Automation dialog box is displayed.

3. In the **Name** field, enter **Check_Customer**.
4. In the **Automation Type** field, select **XSLT Automator**.
5. Click **OK**.

   The Check_Customer plug-in is displayed in the **Automation** list.

6. From the **Automation** list, select the **Check_Customer** plug-in.
7. Click **Properties**.

   The Properties view is displayed showing the automation plug-in properties tabs.

   You are now ready to define the automation plug-in properties.

Step 3: Defining automation plug-in properties

1. In the Automated Task editor Properties view **Details** tab, accept the default value in the **EJB Name** field.
2. Ensure that the model variable that defaults to the **Run As** field points to a user name set up in the WebLogic Server Administration console. When you deploy the cartridge, the user in the **Run As** field is added automatically to the OSM_automation group.

   For more information about users and groups, see the discussion of setting up security in *OSM System Administrator's Guide*. For more information about model variables, see "Project Editor Model Variables Tab".

3. Click the **XSLT** tab.

   On the **XSLT** tab, you define where the XSLT style sheet is located and the status to set if the automation fails. In this example, you define a location on your local machine where the XSLT file is stored.

4. Select **Absolute Path**.
5. In the **XSLT** field, enter the location of the XSLT file.

   For this example, enter **C:\oracle\user_projects\domains\osmdomain\xslt\DSLCartridge\1.0.0\check_customer.xslt**.
6. In the **Exception** field, select **Failure**.

   This field represents the exit status that the plug-in should use if it throws an exception. The options available in this field include any status values you assigned to the task.
7. Select **Update Order**.

   This option ensures that the default values obtained from the XSLT style sheet are saved to the order data.
8. Click **Save**.

   You have completed the basic configuration for an Automator plug-in defined as an internal event receiver.

> **Note:**
>
> Successful automation requires a complete automation build file in the cartridge. If no automation build file exists, Quick Fix generates one.

Related Topics

Configuring Automation Plug-In Properties

Working with Automated Tasks

Working with Manual Tasks

A manual task is a task that must be performed by a person. For example, a manual task could involve a technician who travels to a customer's home to install a phone line. The system displays the description of the manual task in the Task web client worklist and query list.

After you have created a manual task entity using the Manual Task wizard or the Create Task wizard, you can start modeling the task data and assigning other attributes. When modeling manual tasks, see "Defining Manual Task Behaviors" for more information.

Related Topics

About Tasks

Task Editor

Defining Manual Task Behaviors

The Manual Task editor **Task Data** tab enables you to define behaviors at the task level. Behaviors provide a way to extend the functionality and appearance of task data. Each behavior type performs an action; for example, calculating or validating data, or displaying fields in read-only or read-write modes. When you define a behavior at the task level, the behavior applies only to the task.

When defining behaviors at the task level, you can use the Task editor **Task Data** tab to create the behavior, the Properties view for the behavior to refine the behavior information, and the Task editor **Behaviors** tab to view all of the behaviors defined for a task.

To define a behavior at the task level:

1. From the **Studio** menu, select **Show Design Perspective**.
2. Use the Studio Projects view.
3. Double-click the manual task entity for which you are defining the behavior.

   Design Studio displays the task in the Manual Task editor.
4. In the Task Data area, select the data node upon which to model the behavior.

   See "Task Editor Task Data Tab" for more information about the fields on this tab.
5. Right-click in the Behaviors area and select **Add Behavior**.
6. Select a behavior from the list.

> **Note:**
>
> The Calculation, Event, and Lookup behavior types cannot be defined for structured data elements. These behaviors are not relevant because structured data elements do not represent actual data and cannot be acted upon in this way.

   Design Studio adds the behavior to the Behaviors area. Each behavior type enables you to dynamically control a specific aspect of your order data model.
7. In the Behaviors area, click the behavior to open the Properties view.

   The Properties view is displayed with the set of properties that you must define for the corresponding behavior type. See "Working with Behaviors" for more information.
8. Click **Save**.

> **Note:**
>
> After you define the behavior properties, you can click the Task editor **Behaviors** tab to review all of the behavior properties information defined for the task. See "Task Editor Behaviors Tab" for more information.

Working with Automated Tasks

An automated task is completed by an external OSM agent or by automation plug-ins. You can create an automated task to connect to a database, transform some data, or communicate with an external system.

When you create an automated task you must also configure at least one automation plug-in to perform the intended operation. Design Studio provides several built-in automation plug-ins, or you can develop your own plug-in using a custom template. 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).

After you have created an Automated Task entity using the Automated Task wizard, you can start modeling the task and configuring the automation plug-ins.

See *OSM Developer's Guide* for more information about automation.

When modeling automated tasks, see the following topics:

- Adding Automation Plug-ins to Automated Tasks

Related Topics

Working with Automation Plug-Ins

Defining Automated Task Behaviors

The Automated Task editor **Task Data** tab enables you to define behaviors at the task level. Behaviors provide a way to extend the functionality and appearance of task data. Each behavior type performs an action; for example, calculating or validating data, or displaying fields in read-only or read-write modes. When you define a behavior at the task level, the behavior applies only to the task.

When defining behaviors at the task level, you can use the Task editor **Task Data** tab to create the behavior, the Properties view for the behavior to refine the behavior information, and the Task editor **Behaviors** tab to view all of the behaviors defined for a task.

To define a behavior at the task level:

1. From the **Studio** menu, select **Show Design Perspective**.
2. Use the Studio Projects view.
3. Double-click the automated task entity for which you are defining the behavior.

   Design Studio displays the task in the Automated Task editor.
4. In the Task Data area, select the data node upon which to model the behavior.

   See "Task Editor Task Data Tab" for more information about the fields on this tab.
5. Right-click in the Behaviors area and select **Add Behavior**.
6. Select a behavior from the list.

   > **Note:**
   >
   > The Calculation, Event, and Lookup behavior types cannot be defined for structured data elements. These behaviors are not relevant because structured data elements do not represent actual data and cannot be acted upon in this way.

   Design Studio adds the behavior to the Behaviors area. Each behavior type enables you to dynamically control a specific aspect of your order data model.
7. In the Behaviors area, click the behavior to open the Properties view.

The Properties view is displayed with the set of properties that you must define for the corresponding behavior type. See "Working with Behaviors" for more information.

8. Click **Save**.

> **Note:**
>
> After you define the behavior properties, you can click the Task editor **Behaviors** tab to review all of the behavior properties information defined for the task. See "Task Editor Behaviors Tab" for more information.

Adding Automation Plug-ins to Automated Tasks

When using automated tasks in a process, you must create the automation plug-ins that perform the processing for the task. Automation plug-ins enable the system integrator to extend OSM behavior by running specific business logic when events occur or by sending data to and receiving data from external systems. Design Studio supports two types of built-in automation plug-ins: Sender and Automator. See "Working with Automation Plug-Ins" for information about built-in automation types.

To add an automation plug-in to an automated task:

1. From the **Studio** menu, select **Show Design Perspective**.
2. Use the Studio Projects view.
3. Double-click an automated task.

   Design Studio displays the corresponding automated task in the Automated Task editor.
4. Click the **Automation** tab.
5. In the **Automation** tab of the Automated Task editor, click **Add**.

   The Add Automation dialog box is displayed.
6. In the **Name** field, enter a name for the plug-in.

   The name must be unique among the plug-in entity types within the same namespace.
7. Select the plug-in type.

   For example, select the Automator type if the plug-in receives data and performs some work. Select the Sender type if the plug-in receives data, performs some work, and then sends data to external systems.
8. In the **Event Type** field, do one of the following:
   - If the plug-in receives data from external systems via topics or queues, select **External Event Receiver**.

     Automations defined as external event receivers receive incoming JMS or XML messages from external systems and can automatically convert and correlate message data. Additionally, Sender plug-ins defined as external event receivers can generate new outbound messages based on received messages.
   - If the plug-in receives data from the OSM order, select **Internal Event Receiver**.

     Automations defined as internal event receivers receive messages from an internal queue to which the automation framework subscribes. Messages sent by OSM to the internal queue are triggered by internal events.

9. Click **OK**.

   The newly created plug-in is displayed in the **Automation** list.

10. Select the plug-in from the list and click **Properties**.

    The Properties view is displayed with information that varies by plug-in type (Automator or Sender) and by event receiver type (Internal or External). See "Configuring Automation Plug-In Properties" for more information.

Related Topics

Working with Automated Tasks

Working with Activation Tasks

Activation tasks provide integration between OSM and either ASAP or IP Service Activator. You can model a process flow that includes one or more tasks that activate services in a network using those systems.

Before modeling activation tasks, you must import at least one ASAP or IP Service Activator service cartridge, which you use to define relationships between tasks and service actions. See "Importing Activation Cartridge Projects" for more information.

When you create a first activation task in a workspace, Design Studio creates a data dictionary project called ActivationOSMIntegrationDataDictionary containing the data schema structure for ASAP and IP Service Activator. Data elements are annotated on the **Notes** tab of the Data Schema editor. The ASAP data schema is called **ASAP_OSM**. The IP Service Activator data schema is called **IPSA**.

When modeling activation tasks, see the following topics:

- About Activation Tasks
- Modeling Activation Tasks

About Activation Tasks

The interaction between OSM and the activation system is established through a service request and response, which you configure by mapping OSM task data to system parameters.

For ASAP, the OSM data is transformed to an OSS/J or web services order and sent to ASAP to activate the specified services.

For IP Service Activator, the OSM data is transformed to a web services order that is sent to IP Service Activator to activate the specified services.

The activation system returns event responses or exceptions, depending on the result of the activation.

Using the activation task, you can:

- Update OSM orders with data from all events and exceptions returned by the activation system (either ASAP or IP Service Activator).
- Map OSM data to Activation order headers, global parameters, and service action parameters.
- Automatically map OSM data to global or service action parameters with the same name.
- Define conditional transition states and statuses for completion events and completion exceptions returned by the activation system.

- Enter Map and Key security credentials for web services orders.

Mapping OSM data to ASAP and IP Service Activator parameters ensures:

- That OSM sends the data that the activation system requires for service actions (the request)
- That OSM orders are updated with the right information returned from the activation system (the response)
- That the OSM activation task transitions properly when the integration is complete

See the following topics for more information on service action requests:

- About Service Action Request Mapping
- About Service Action Response Mapping
- About State and Status Transition Mapping

About Service Action Request Mapping

The service action request is made up of the following data:

- OSM header data: information that applies to the customer or to all line items on the order
- OSM task data: information that is available to the task and necessary for the task to complete

The service action is made up of the following data:

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

You define parameter values in the Design Studio Properties view. For service action and global parameters, you can define default mapping information. For activation order header parameters, you can define either default mapping information or default actions (for example, the default value for the srqAction parameter is ADD).

Some parameters in the activation order headers are prepopulated with default values, indicated by a check mark when you first expand the **Activation Order Headers** in the **Service Actions** tree.

Related Topics

Configuring Service Action Requests

Task Editor Request Data Tab

Modeling Activation Tasks

About Service Action Response Mapping

You create data structures in OSM to contain the response information returned from ASAP and IP Service Activator. For each event and exception returned by the activation system, you select the parameter values you want to retain, then identify the OSM data structure to which these parameters are added. When the activation system returns an event or exception, OSM updates the order data with the parameter values that you selected.

The list of events and exceptions and activation response parameters are different for each activation system.

The activation response parameters for ASAP can be customized.

The activation response parameters for IP Service Activator cannot be modified and conform to how IP Service Activator expects to receive response data from OSM.

The infoParm parameter is significantly more complex for IP Service Activator than it is for ASAP.

When an activation task is configured with IP Service Activator service actions that require an IP Service Activator transaction to fulfill the activation request, the transaction structure in the InfoParms structure contains information relating to the IP Service Activator transaction.

When an activation task is configured with IP Service Activator service actions that look up data (such as for navigation service actions), the return data conforms to the infoParms structure in the activation response, with one infoParm per service action.

Related Topics

Configuring Service Action Responses

Task Editor Response Data Tab

Modeling Activation Tasks

About State and Status Transition Mapping

Using the Task editor **Response Data** tab, you can configure state and status transitions for completion events and exceptions returned by the activation system. You can define multiple transitions (each with an XPath expression) to model different scenarios for variations in the response data. For example, if an ASAP 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**. You define the condition in the Properties views. At run time, OSM evaluates the conditions in the order you have defined them and stops evaluating when a condition evaluates to **true**.

Completion events and exceptions must include a default transition in the event that all specified conditions fail. You can change or delete the predefined default values, or you can create your own. You can change or delete the predefined default values or you can create your own. If you define no default conditions for the required completion events and exceptions (no condition is defined with XPath expression **true()**) Design Studio creates a problem marker.

The following table lists completion events and exceptions that require a default transition configuration:

Name	Activation System	Type
orderCompleteEvent	ASAP and IP Service Activator	Event
orderCreateEvent	ASAP only	Event
orderEstimateEvent	ASAP only	Event
orderFailEvent	ASAP and IP Service Activator	Event
orderNEUnknownEvent	ASAP only	Event
orderRollbackEvent	ASAP only	Event

Name	Activation System	Type
orderSoftErrorEvent	ASAP only	Event
orderStartupEvent	ASAP only	Event
orderTimeoutEvent	ASAP only	Event
orderTimeoutWarningEvent	ASAP and IP Service Activator	Event
createOrderByValueException	ASAP and IP Service Activator	Exception
getOrderByKeyException	ASAP only	Exception
queryManagedEntitiesException	ASAP only	Exception

Related Topics

Configuring Service Action Response State and Status Transitions

Task Editor Response Data Tab

Modeling Activation Tasks

Modeling Activation Tasks

You model activation tasks to integrate OSM with either and ASAP or IP Service Activator. You can model a process flow that includes one or more tasks that activate services in a network.

> **Note:**
>
> Before modeling activation tasks, ensure that you have defined the Design Studio preferences for the OSM SDK and WebLogic Server installations. See "Defining Order and Service Management General Preferences" for more information.

To model an activation task:

1. Import an ASAP or IP Service Activator service cartridge.

   See "Importing Activation Cartridge Projects" for more information.

2. Create an Activation Task entity.

   You create an Activation Task entity to hold all of the information necessary to send a request to the activation system, receive the response, update the order, and transition the task. See "Creating New Tasks" for information about creating tasks, and see "Designing Tasks and Activities" for information about creating tasks from the Process editor.

   > **Note:**
   >
   > The first time that you create an Activation Task entity in a workspace, Design Studio automatically creates a new project to contain data elements necessary for integration between OSM and the activation system. This project is sealed and the data within should not be changed.

3. Model the activation task data.

You select the data that the activation task requires from the order data or from a data dictionary. See "Defining Task Data" for more information.

4. Configure the mapping information needed to make the service action request to the activation system.

   See "Configuring Service Action Requests" for more information.

5. Configure the mapping information needed to update OSM orders with the response data returned by the activation system.

   See "Configuring Service Action Responses" for more information.

6. Configure state and status transitions for completion events and completion exceptions returned by the activation system.

   See "Configuring Service Action Response State and Status Transitions" for more information.

7. Configure activation task details.

   You define the attributes that enable the activation task to process properly in the Activation environment. See "Task Editor Details Tab" and "Task Editor Activation Task Details Tab" for more information.

8. Define activation task compensation strategies.

   You specify how to compensate an activation task if the task is affected by amendment processing. See "Task Editor Redo Tab" and "Task Editor Undo Tab" for more information.

9. Configure activation task states and statuses.

   A task state determines the milestone of a task in a process. A task status describes how a task was completed and determines the next task in the process. See "About Task States and Statuses" and "Assigning Task States and Statuses" for more information.

10. Configure activation task permissions.

    You assign execution modes to roles for each task to specify which roles can perform the execution mode. See "Assigning Task Permissions" for more information.

11. (Optional) Configure activation task jeopardies.

    You can configure conditional jeopardy notifications to alert users or systems that the activation task may be at risk. See "Task Editor Jeopardy Tab" for more information.

12. (Optional) Configure task state automation events.

    You configure state-based event notifications to alert users or systems of changes to the activation task state. See "Task Editor Events Tab" for more information.

Configuring Service Action Requests

You configure service action requests by mapping OSM order header and task data to Activation order header, service action, and global parameters. Additionally, you can define conditional logic to determine when service actions should be added to a work order request.

> **Note:**
>
> See "Modeling ASAP Services" and "Defining Service Action Properties" for more information about creating and configuring service actions.

To configure service action requests:

1. Associate service actions with the activation task:
   a. On the **Request Data** tab, right-click in the Service Actions area and select **Add Service Action**.

      > **Note:**
      >
      > Service actions for IP Service Activator are available in reference Design Studio IP Service Activator projects or are generated by Design Studio when a CTM template is imported into an Activation IPSA project. See "About CTM Templates" for more information.

   b. Right-click the service action to either change the sequence of service actions, open them in the Service Action editor, or remove them from the task.
2. (Optional) Define new global parameters.

   You can define service actions as global parameters to avoid mapping the parameter multiple times:
   a. In the **Request Data** tab, right-click in the Service Actions area and select **Add Global Parameters**.

      The Add Global Parameter dialog box is displayed.
   b. In the **Name** field, select the desired parameter, enter the value, and click **OK**.
3. Map OSM data to Activation order header, service action, and global scalar parameters:

   > **Note:**
   >
   > For details about mapping and optionally transforming OSM data to service action XML parameters, see "Mapping OSM Data to Service Action XML Parameters".

   a. In the **Request Data** tab Task Data area, select **Order Header** or **Task Data**.

      Depending on your selection, the fixed order header data or the task data defined for the activation task is displayed in the Task Data area.
   b. In the Service Actions area, expand the **Activation Order Headers** folder, the **Global Parameters** folder, or any service action folder to display the parameters.

      Required service action parameters are displayed with an asterisk after the parameter name.
   c. Drag an OSM order header or a task data node onto an Activation order header, service action, or global parameter.

      A check mark appears next to the parameter to indicate that it is mapped to OSM data. Right-click the parameter and select **Properties** to review the mapping information, default value, and condition expression, depending on the parameter.

> **Note:**
>
> To automatically build XPath expressions, press and hold the **Alt** key, then drag OSM data from the Task Data area to the Properties view **Binding** field. Constants or default values must be enclosed within apostrophes (' ').

4. (Optional) Automatically map task data to global parameters:
   a. In the **Request Data** tab Task Data area, select **Task Data**.
   b. In the Task Data area, right-click a data structure and select **Auto map parameter**.

   Design Studio automatically maps the data structure, including its child elements, to a global parameter of the same name (case sensitive).
5. Define conditional logic for service actions and parameters.

   You define conditional logic (as an XPath expression) to determine when to include a service action on a work order request. See "Properties Service Action Binding View" for more information.
6. Add service action parameters to OSM task data.

   You can add all parameters of a service action to a selected OSM data structure. Service action parameters are not added to the structure if it contains a child element with the same name as the parameter. See "Task Editor Request Data Tab" for more information.

> **Note:**
>
> Design Studio limits the maximum length of service action parameters to 1000 characters when adding them to a structure. If you create data elements for service action parameter fields manually (using the Data Schema editor), ensure that you set the maximum length of the new data element equal to the maximum length defined for the service action parameter.

Related Topics

About Service Action Request Mapping

Task Editor Request Data Tab

Mapping OSM Data to Service Action XML Parameters

Modeling Activation Tasks

Mapping OSM Data to Service Action XML Parameters

The Activation Task editor supports the following ways of mapping OSM data to ASAP and IP Service Activator service action XML data types:

- Mapping OSM XML data to an ASAP or IP Service Activator XML parameter without modeling the XML structure using an XPath expression. See "Mapping OSM Data to Service Action XML Parameters Using XPath" for more information.
- Mapping and optionally transforming OSM XML data structures with child elements to an ASAP or IP Service Activator XML parameter using an XSLT snippet. See "Mapping OSM Data to Service Action XML Parameters Using XSLT" for more information.

Related Topics

About Service Action Request Mapping

Task Editor Request Data Tab

Modeling Activation Tasks

Mapping OSM Data to Service Action XML Parameters Using XPath

To map OSM XML data to ASAP or IP Service Activator parameters using an XPath expression:

1. Open a data schema associated to an OSM project.

   The Data Schema editor is displayed.

2. Right-click in the parameter area and select **Add Structure (CTRL + ALT + S)**.

   The Create Data Schema Structure dialog box is displayed.

3. Do the following:

   a. In the **Name** field, enter a name for the structure. For example, **XMLTypeNoChildren**.

   b. Click **Finish**.

      The empty structure is displayed in the Data Schema editor parameter area.

4. Open an OSM order that you want to associate the structure to.

   The Order editor is displayed.

5. Right-click in the Order Template area and select **Select from Dictionary**.

   The Select Data Elements dialog box is displayed.

6. Select the empty structure.

7. Click **OK**.

   The structure is displayed in the Order Template area.

8. Open an Activation Task that you want to associate the structure to.

   The Activation Task editor is displayed.

9. In the **Request Data** tab Task Data area, select **Task Data**.

10. Right-click in the Task Data area and select **Select from Order Template**.

    The Select an Order Template Node dialog box is displayed.

11. Select the empty structure.

12. Click **OK**.

    The structure is displayed in the Task Data area.

13. In the Service Actions area, right-click and select **Add Service Action**.

    The Select a Service Action dialog box is displayed.

14. Select a service action that includes the XML parameter you want to map to the empty OSM structure.

15. Click **OK**.

    The service action is displayed in the Service Actions area.

16. Expand the newly added service action.

    All parameters associated to the service action appear.

17. From the Task Data area, drag the empty structure to the corresponding service action XML parameter in the Service Action area.

    A check mark appears next to the parameter to indicate that it is mapped to the OSM structure.

18. Click the service action parameter. Verify the following fields in the **Properties** tab:

    a. In the **Binding Type** field, ensure that the **XPath expression** field is selected.

    b. In the **Binding** field, ensure that the XPath expression references the OSM structure. For example: **osm:XMLTypeNoChildren**

Related Topics

Mapping OSM Data to Service Action XML Parameters

Mapping OSM Data to Service Action XML Parameters Using XSLT

To map and optionally transform OSM XML data structures with child elements to an ASAP or IP Service Activator XML parameter using XSLT snippets:

1. Open a data schema associated to an OSM project.

   The Data Schema editor is displayed.

2. Create an OSM structure with any number of child elements and child structures with child elements.

3. Open an OSM order that you want to associate the structure to.

   The Order editor is displayed.

4. Right-click in the Order Template area and select **Select from Dictionary**.

   The Select Data Elements dialog box is displayed.

5. Select the structure.

6. Click **OK**.

   The structure is displayed in the Order Template area.

7. Open an Activation Task that you want to associate the structure to.

   The Activation Task editor is displayed.

8. In the **Request Data** tab Task Data area, select **Task Data**.

9. Right-click in the Task Data area and select **Select from Order Template**.

   The Select an Order Template Node dialog box is displayed.

10. Select the empty structure and click **OK**.

    The structure is displayed in the Task Data area.

11. Right-click in the Service Actions area and select **Add Service Action**.

    The Select a Service Action dialog box is displayed.

12. Select a service action that includes the XML parameter you want to map to the OSM structure.

13. Click **OK**.

The service action is displayed in the Service Actions area.

14. Expand the newly added service action.

    All parameters associated with the service action are displayed.

15. From the Task Data area, drag the structure to the corresponding service action XML parameter in the Service Action area.

    A check mark appears next to the parameter to indicate that it is mapped to the OSM structure.

> **Note:**
>
> The OSM structure and children do not map to structures in the Service Actions area, only to individual parameters.

16. Click the service action parameter. Verify the following fields in the **Properties** tab:

    a. In the **Binding Type** field, ensure that the **XSLT snippet** field is selected.

    b. In the **Binding** field, ensure that the XSLT snippet maps the OSM structure to the ASAP or IP Service Activator structure.

    For example:

```
<mslv-sa:serviceValue xsi:type="mslv-sa:ASAPServiceValue">
    <mslv-sa:name>stuff1</mslv-sa:name>
    <mslv-sa:xmlValue>
        <ASAPproj:XMLData xmlns:ASAPproj="http://xmlns.oracle.com/
communications/sce/dictionary/ASAPproj/ASAPproj">
            <ASAPproj:data>
                <ASAPproj:Title>
                    <xsl:value-of select="osm:XMLData/osm:data/osm:Title"/>
                </ASAPproj:Title>
                <ASAPproj:LineItem>
                    <xsl:value-of select="osm:XMLData/osm:data/osm:LineItem"/>
                </ASAPproj:LineItem>
            </ASAPproj:data>
            <ASAPproj:XMLId>
                <xsl:value-of select="osm:XMLData/osm:XMLId"/>
            </ASAPproj:XMLId>
            <ASAPproj:XMLType>
                <xsl:value-of select="osm:XMLData/osm:XMLType"/>
            </ASAPproj:XMLType>
        </ASAPproj:XMLData>
    </mslv-sa:xmlValue>
    <mslv-sa:type>OPTIONAL_XML</mslv-sa:type>
</mslv-sa:serviceValue>
```

> **Note:**
>
> By default, Design Studio assumes that the OSM and Activation parameters structures are identical. If the parameters are different, modify the default mapping as described in the next step.

    c. (Optional) If the default XSLT expression mapping does not correspond to the ASAP or IP Service Activator parameter structure, change the parameter mappings within the **<mslv-sa:xmlValue>** element.

For example:

```
<mslv-sa:serviceValue xsi:type="mslv-sa:ASAPServiceValue">
    <mslv-sa:name>stuff1</mslv-sa:name>
    <mslv-sa:xmlValue>
        <ASAPproj:NewValue1 xmlns:ASAPproj="http://xmlns.oracle.com/
communications/sce/dictionary/ASAPproj/ASAPproj">
            <ASAPproj:NewValue2>
                <ASAPproj:NewValue3>
                    <xsl:value-of select="osm:XMLData/osm:data/osm:Title"/>
                </ASAPproj:NewValue3>
                <ASAPproj:NewValue4>
                    <xsl:value-of select="osm:XMLData/osm:data/osm:LineItem"/>
                </ASAPproj:NewValue4>
            </ASAPproj:NewValue2>
            <ASAPproj:NewValue5>
                <xsl:value-of select="osm:XMLData/osm:XMLId"/>
            </ASAPproj:NewValue5>
            <ASAPproj:NewValue6>
                <xsl:value-of select="osm:XMLData/osm:XMLType"/>
            </ASAPproj:NewValue6>
        </ASAPproj:NewValue1>
    </mslv-sa:xmlValue>
    <mslv-sa:type>OPTIONAL_XML</mslv-sa:type>
</mslv-sa:serviceValue>
```

where *NewValue1* to *NewValue6* correspond to the ASAP or IP Service Activator XML parameters.

Related Topics

Mapping OSM Data to Service Action XML Parameters

Configuring Service Action Responses

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 the activation system, you select which parameter values to retain, then identify the data structure to which these parameters are added. When the activation system returns an event or exception, OSM updates the order data with the selected response parameter values.

To configure service action responses:

1. In the Activation Task editor **Response Data** tab, select an event or exception from the Event/Exception area.

   All of the data an event or exception returns appears in the Activation Response area.

2. Add an OSM data structure from the order template or from a data dictionary to contain the information returned by the service action response:

   a. Right-click in the Response Data Location area or the OSM Data Binding area and select **Select from Order Template** or **Select from Data Dictionary**.

      If you select a structure from a data dictionary, Design Studio automatically adds that structure to the order template.

   b. Select a structure from the dialog box and click **OK**.

      The structure is displayed in the Response Data Location and OSM Data Binding area.

> **Note:**
>
> Do not add a data structure that uses a distributed order template. Attempting to map a response value to a data element in a distributed order template will cause an error. For more information about distributed order templates, see *OSM Concepts*.

3. Do at least one of the following:
   - Map fixed activation response structures to task data structures:
     a. Click the **Set Data Location** subtab.
     b. In the Activation Response area, check the activation response elements or structures that you want to map to task data structures.
     c. In the Response Data Location area, right-click a task data structure and select **Set as Data Location**.

        Design Studio adds the data you selected for the event or exception in the Activation Response area to the structure in the Response Data Location area. When the event or exception is returned from the activation system, OSM adds the data values to the OSM structure.
   - Bind activation response elements to arbitrary task data elements:
     a. Click the **Response Data Mapping** subtab.
     b. In the Activation Response area, drag an activation response data element onto a task data element in the OSM Data Binding area.

        The task data element is checked.

   If you select no data in the Activation Response area for an event or exception, OSM ignores that event or exception when it is returned by the activation system.
4. (Optional; ASAP only) Define conditional mappings for service action response parameters. See "Filtering ASAP Response Data".

   This step enables you to reduce the response data returned to OSM.

   > **Note:**
   >
   > The information returned by IP Service Activator infoparms is structured to enable a detailed mapping of return data back to OSM order data. Therefore, there is no need to filter the amount of response data.
   >
   > The ASAP Infoparm is less structured.

5. Right-click the OSM structure and select **Open Properties View**.

   The Properties view for the task data node is displayed.
6. Click the **Identification** tab.

   See "Task Data Node Properties View Identification Tab" for more information.
7. Select **Override Data Dictionary Minimum/Maximum**.
8. In the **Maximum** field, select **Unbounded**.
9. Click **Save**.

Related Topics

Task Editor Response Data Tab

About Service Action Response Mapping

Modeling Activation Tasks

Filtering ASAP Response Data

The amount of response data returned by an activation system can be very large, while the needed data might be quite small. Parsing large amounts of response data can affect OSM performance. If you notice a reduction in OSM performance due to large amounts of response data, you can specify a condition on specific parameters to limit the response data. You can create response data limitations for as many events or exceptions as you choose by binding the activation response data to the desired OSM data.

To limit response data:

1. In the Task editor **Response Data** tab, select an event or exception from the **Event/ Exception** field.
2. In the Response Mapping area, click the **Set Data Location** tab.
3. In the Activation Response area, right-click **InfoParm**.

   The Response Filter Properties view is displayed.
4. Drag a parameter into the Response Filter **Condition** field.

   The XPath representation of the parameter is created in the **Condition** field.
5. Add the desired condition to the XPath representation.

   For example, consider that you only want to update infoParm data in OSM if the serviceId infoParm parameter from orderCompleteEvent is equal to 2. First, select **orderCompleteEvent** in the **Event/Exception** field. Then click **Detailed Parameters** and **infoParm** in the Activation Response field. Drag **serviceId** into the **Condition** field. The XPath representation of serviceID is as follows:

   ```
   mslv-sa:serviceId
   ```

   Now set the desired condition by adding **='2'**:

   ```
   mslv-sa:serviceId='2'
   ```
6. Click **Save**.

Related Topics

Configuring Service Action Responses

About Service Action Response Mapping

Modeling Activation Tasks

Response Filter Area

Configuring Service Action Response State and Status Transitions

For completion events and exceptions returned by the activation system, you can configure state and status transitions.

To configure state and status transitions:

1. In the Activation Task editor **Response Data** tab, select an event or exception from the **Event/Exception** field.

   Predefined default states and status transitions appear in the Transition to State and Status area.

2. Click the **Add** button.

   The State/Status Selection dialog box is displayed.

3. In the **Condition Name** field, enter a name for the transition.

4. Select the **State** or **Status** option.

   If you select **State**, select a user-defined state. If you select **Status**, select a predefined task status. See "About Task States and Statuses" and "Assigning Task States and Statuses" for more information.

5. Click **OK**.

   Design Studio adds the new transition to the table.

6. (Optional) Select a transition row and click **Move Up** or **Move Down** to change the order of the transitions.

   The order in which they appear in the table determines the order in which OSM evaluates the conditions at run time.

7. Select a transition row and click **Properties**.

   The Properties view for the state/status transition is displayed.

8. In the **Condition** field, define an XPath expression to define the conditions under which the transition occurs.

   See "Properties State/Status Transition View" for more information about defining conditions.

9. Click **Save**.

Related Topics

About State and Status Transition Mapping

Task Editor Response Data Tab

Modeling Activation Tasks

Working with Transformation Tasks

Transformation tasks enable you to access a transformation manager from an OSM process. These tasks are similar to automated tasks, except that the automation plug-in is prepopulated when the task is created. However, you are not restricted to using the provided automation plug-in. All of the plug-in configuration options that are available with automated tasks are also available with transformation tasks.

Model your order transformation manager entities before you model the transformation task. For more information about modeling order transformation, see *OSM Concepts*.

Related Topics

Working with Automated Tasks

Working with Order Item Parameter Bindings

Working with Transformation Sequences

Working with Transformation Managers

Working with Mapping Rules

Task Editor

Use the Task editor to model the tasks you use in your processes to offer a specific service to a customer. Each task type has its own set of tabs in the Task editor. The following table lists the Task editor tabs that appear for each task type.

Tab Name	Manual Task	Automated Task	Activation Task	Transformation Task
Task Editor Activation Task Details Tab	No	No	Yes	No
Task Editor Automation Tab	No	Yes	No	Yes
Task Editor Behaviors Tab	Yes	Yes	No	No
Task Editor Compensation Tab	Yes	Yes	No	Yes
Task Editor Details Tab	Yes	Yes	Yes	Yes
Task Editor Events Tab	Yes	Yes	Yes	Yes
Task Editor Fallouts Tab	Yes	Yes	No	Yes
Task Editor Jeopardy Tab	Yes	Yes	Yes	Yes
Task Editor Permissions Tab	Yes	Yes	Yes	Yes
Task Editor Redo Tab	No	No	Yes	No
Task Editor Request Data Tab	No	No	Yes	No
Task Editor Response Data Tab	No	No	Yes	No
Task Editor Composite Data View Tab	Yes	Yes	No	Yes
Task Editor States/Statuses Tab	Yes	Yes	Yes	Yes
Task Editor Task Data Tab	Yes	Yes	No	Yes
Task Editor Undo Tab	No	No	Yes	No

The following field is common to multiple tabs in the Task editor:

Field	Use
Description	Edit the display name of the task.

Task Editor Activation Task Details Tab

The Task editor **Activation Task Details** tab appears for activation task types.

Use the **Activation Task Details** tab to define the attributes that enable the activation task to run properly in the Activation environment. For example, you can associate an activation task with an order, specify the user to run this task, and provide the details for the response queue name, maximum cache size, and cache time-out.

Field	Use
Activation System	Displays the activation system against which the activation task is registered.
Run As	Enter the OSM user name (security principal) that can run this task. A password is not necessary to authenticate this user because only an administrator has the authority to deploy components into the server. Ensure that the user is set up in the WebLogic Server console. For more information about setting up users and groups, see *OSM System Administrator's Guide*. **Note:** Oracle recommends using the DEFAULT_AUTOMATION_USER cartridge model variable in the **Run As** field. See "Project Editor Model Variables Tab" for more information.
Maximum Number in Cache	Specify the maximum number of entries in the cache that are maintained at any one time.
Cache Timeout	Specify the number of seconds for which the cache is valid.
Exception	Select the exit status to use when an exception occurs during the activation task processing. Status options include any status values you assigned to the task.
Activation Order ID Node	Select a data node to store the order ID. The activation order ID is used by the defined compensation strategy when a task is affected by amendment processing. See "Task Editor Redo Tab" and "Task Editor Undo Tab" for more information.
OSSJ	Enter the location to which OSM sends OSSJ service action requests. Configure this attribute if you want to connect to an ASAP instance using Java message service (JMS). Depending on how you set up your cartridge project model variables, you can use the default values or define your own queue location names. See "Project Editor Model Variables Tab" for more information.
Web services	Enter the location to which OSM sends web services service action requests. Configure this attribute if you want to connect to an activation system instance using web services messages. You must also configure the **Map** and **Key** fields to secure the web services messages. Depending on how you set up your cartridge project model variables, you can use the default values or define your own queue location names. See "Project Editor Model Variables Tab" for more information.
Map and **Key** for Activation Credential	ASAP and IP Service Activator secure web services service actions with a web services user name and password located in the activation system WebLogic server. You must store this user name and password within the Fusion Middleware Credential Store Framework (CSF) using the **credStoreAdmin.bat** tool located in the *OSM_home*/**SDK/XMLImportExport** folder, where *OSM_home* is the location where the OSM software is installed. The **credStoreAdmin.bat** tool creates a map and a key that corresponds to the ASAP Web Services user name and password. For more information about this tool, see the *OSM System Administrator's Guide*.
Environment ID	Enter the activation system environment ID to which the service action requests are sent. **Note:** When you create cartridges, some of the variable information to define may depend on a specific environment. If you do not have environment specific values for variables that you will need at run time, you can create tokens for the variables and later define specific variable values for each environment in which you will use the variable. Tokens are placeholders for environment-specific values that can be defined at the time of deployment. See "Project Editor Model Variables Tab" for more information.
Response Queue	Enter the JNDI name of the response queue on which this automator listens. If you do not enter a value, the system uses a default value. Values must be defined in WebLogic Server. Design Studio populates this field with a default value if service action requests are configured to be submitted using OSSJ. When service action requests are configured to be submitted using web services, you must define a response queue.

Field	Use
JMS topic for events	Enter the location to which service action responses are sent. Depending on how you set up your cartridge project model variables, you can use the default values or define your own topic location names. See "Project Editor Model Variables Tab" for more information. If you are using store and forward (SAF) to communicate to an ASAP instance located on a different WebLogic server, Oracle recommends selecting the **Use a queue** check box to use a queue instead of a topic. For more information about SAF, see the *OSM Installation Guide*.

Related Topics

Modeling Activation Tasks

About Activation Tasks

Task Editor Automation Tab

The Task editor **Automation** tab appears for automation and transformation task types.

The **Automation** tab displays a list of automation plug-ins registered against the automated task. When the OSM server encounters the task during run-time it triggers the plug-ins to begin the work that they are designed to perform.

When modeling automated tasks in the **Automation** tab, see the following topics for more information:

- Properties View Details Tab
- Properties View External Event Receiver Tab
- Properties View Compensation Tab
- Properties View Correlation Tab
- Properties View XQuery Tab
- Properties View XSLT Tab
- Properties View Routing Tab
- Properties View Custom Plug-in Tab
- Properties View System Interaction Tab
- Properties View Notes Tab

Field	Use
Name	Displays the name of the automation plug-in. Click the value in the Name column to change or edit the name. **Note:** Design Studio displays the automation plug-ins in the **Automation** tab alphabetically in ascending or descending order. The order in which the plug-ins appear does not indicate or affect the order in which the plug-ins will perform during run time. When you have multiple automation plug-ins registered against an automated task, you must ensure that the plug-ins are consuming only those messages that are pertinent to the plug-in operation. See "Properties View External Event Receiver Tab" for information about how to filter for specific message properties.
Type	Displays the automation plug-in type. There are two basic types of built-in automation plug-ins: Sender and Automator. Each type can be implemented by an XSLT style sheet or by XQuery. Additionally, you can create your own custom plug-in using the customer plug-in template. See "Working with Automation Plug-Ins" for more information. **Note:** XQuery automation types cannot be implemented when using releases prior to OSM 7.0.

Field	Use
Event Type	Displays whether an automation plug-in receives data from an external system queue or from an internal OSM queue. See "Configuring Automation Plug-In Properties" for more information.
Properties	Select an automation plug-in from the Automation table and click **Properties** to access the Automation Plug-in Properties tabs. See "Configuring Automation Plug-In Properties" for more information.
Remove	Select an automation plug-in from the Automation table and click **Remove** to delete the automation from the list of plug-ins registered against the task.
Add	Click to add an automation plug-in to the list of plug-ins registered against the task. See "Adding Automation Plug-ins to Automated Tasks" for more information.

Properties View System Interaction Tab

Use the Properties view **System Interaction** tab to view and update automation properties for System Interaction specifications.

Field	Use
Target System	Displays the target system defined in the associated Order Component specification.
Response Message	(Displayed for External Automators only). Lists Event Operation IDs and Endpoint Operation IDs defined in System Interaction Entity in the associated Order Component Specification.
OpenAPI Operation	(Displayed for Senders only). Lists Endpoint Operation IDs defined in System Interaction Entity in the associated Order Component Specification.

For Endpoint Operation IDs, there must be both an internal and an external automation plugin. That is, when an Endpoint Operation ID is selected on a sender automation, the same Endpoint Operation ID must also be defined for an external automator.

For Event Operation IDs, only external automator is needed as they are unsolicited events.

System Interaction cannot be defined for internal automators in Automated tasks.

Related Topics

Example: Modeling a Basic Automator Plug-In

Configuring Automation Plug-In Properties

Working with Automated Tasks

Properties View Details Tab

Use the Properties view **Detail** tab to define information that is common to all types of automation plug-in and event receiver types.

Field	Use
Name	Enter a plug-in name. The name must be unique among plug-in entities in the same namespace. **Note:** While plug-in names can be any arbitrary name that you assign to the automation, Oracle recommends that you use a consistent naming pattern for all of the automations that you create.

Field	Use
EJB Name	Edit the system-provided default EJB name. The default value with which Design Studio initially populates this field depends on where the automation is defined: • If the automation is defined for a task (automated task, task state-based event notification, task jeopardy notification), the EJB name defaults to *TaskName.AutomationName*, where *TaskName* is the name you provided when defining the task, and *AutomationName* is the name you provided when defining the automation. • If the automation is defined for an order (order milestone-based event notification, order data changed event notification, order jeopardy notification), the EJB name defaults to *OrderName.AutomationName*, where *OrderName* is the name you provided when defining the order, and *AutomationName* is the name you provided when defining the automation. • If the automation is defined for a process (task state-based event notification, task status-based event notification), the EJB name defaults to *ProcessName.AutomationName*, where *ProcessName* is the name you provided when defining the task, and *AutomationName* is the name you provided when defining the automation. **Note:** The EJB name must be unique per OSM server. However, as there is no method for predicting to which OSM server the cartridge will be deployed, you should ensure uniqueness across all automation plug-ins defined for a cartridge. The default EJB name guarantees this uniqueness; therefore, Oracle recommends that you do not change the defaulted EJB name. See "Working with Jeopardy and Event Notifications" for more information jeopardy and event notifications.
Run As	Enter the OSM user name (security principal) whose credentials are used to run this automation plug-in. A password is not necessary to authenticate this user because only an administrator has the authority to deploy components into the server. The value of this field must reflect the user ID that is used to run the automation: • The user ID must be set up in the WebLogic Server Administration console. See the discussion of setting up security in *OSM System Administrator's Guide* for more information. • The user ID must be defined in the OSM Administration area of the Order Management web client (a workgroup in OSM Administration is a role in Design Studio). See "Working with Roles" for more information. • The user ID must be assigned to the workgroup in the OSM Administration area of the Order Management web client that corresponds to the role defined on the **Permissions** tab of the Design Studio task, order, or process that defines the automation. **Note:** Oracle recommends using the DEFAULT_AUTOMATION_USER cartridge model variable in the **Run As** field. See "Defining Model Variables" for more information.
Event Type	This field is enabled only for automations defined for automated tasks. Select one of the following: • **External Event Receiver** if the plug-in will receive data from external systems via topics or queues. Automations defined as external event receivers receive incoming JMS or XML messages from external systems, and can automatically convert and correlate message data. Additionally, Sender plug-ins defined as external event receivers can generate new outbound messages based on received messages. • **Internal Event Receiver** if the plug-in receives data from the OSM order data. Automations defined as internal event receivers receive messages from an internal queue to which the automation framework subscribes. Messages sent by OSM to the internal queue are triggered by internal events. **Note:** If you intend to secure automations such that different user IDs have access to run different automations, Oracle recommends that you incorporate these changes after you ensure that the automations are working successfully. Changing a plug-in's automation event type may result in the loss of any data that is not common between the event receiver types.
Fail Task on Automation Exception	Select this check-box to fail the task if an exception occurs when running the automation plug-in. The plug-in can throw any exception and OSM retries the plug-in as many times as configured on the JMS destination retry. On the last retry attempt OSM fails the task and logs the exception message as the failure reason.

Related Topics

Example: Modeling a Basic Automator Plug-In

Working with Automated Tasks

Configuring Automation Plug-In Properties

Properties View External Event Receiver Tab

Use the Properties view **External Event Receiver** tab to define how OSM retrieves and processes messages placed on the queue by external systems.

The Properties view **External Event Receiver** tab appears only for Automator and Sender plug-in types defined as external event receivers. External event receivers listen to external system queues or topics for JMS messages. To define a plug-in as an external event receiver, select the **External Event Receiver** option on the Add Automation dialog box when creating a new automation entry.

Field	Use
JNDI Name	Enter the name of the external system from which the plug-in receives messages. JNDI Name is mandatory and contains a system-supplied default value which you must change to reflect your own system topology. The JNDI Name must be unique in the workspace. The default value with which Design Studio initially populates this field depends on where the automation is defined: • If automation is defined for a task (automated task, task state-based event notification, task jeopardy notification), the JNDI name defaults to *TaskName.AutomationName.jndiName*, where *TaskName* is the name you provided when defining the task, and *AutomationName* is the name you provided when defining the automation. • If automation is defined for an order (order milestone-based event notification, order data changed event notification, order jeopardy notification), the JNDI name defaults to *OrderName.AutomationName.jndiName*, where *OrderName* is the name you provided when defining the order, and *AutomationName.jndiName* is the name you provided when defining the automation. • If automation is defined for a process (task state-based event notification, task status-based event notification), the JNDI name defaults to *ProcessName.AutomationName.jndiName*, where *ProcessName* is the name you provided when defining the task, and *AutomationName* is the name you provided when defining the automation.
Destination Type	Select the type of the response destination. A JMS destination is either a javax.jms.Queue or a javax.jms.Topic. Topics are generally used when messages are published for general availability to multiple external systems. Queues are generally used if the sender wants only a single external system to consume the message.
URL, **Initial Context Factory**, **Connection Factory**	(Optional) Enter this information to connect to an external application server. Specify the URL and the InitialContextFactory class for the JNDI provider, and specify the ConnectionFactory class for the JMS server.

Field	Use
Message Property Selector	Enter a selector based on the message properties applied to the external queue. Using an XPath expression or query statement in this field enables you to filter incoming messages to only those messages with specific properties defined at the header level. Using a message property enables you to interrogate the message on the external queue and determine which plug-in should perform the processing of this task instance. When you have multiple plug-ins with identical event receiver types defined for a task, the OSM server will select the first plug-in whose message property selector evaluates to true. A message on an external JMS queue can be consumed only once. Consequently, it is critical to ensure that your plug-ins consume the appropriate messages. Useful properties for selection include source, type, process, process status, or priority. For example, for a single task, you might create multiple external event receiver Automator plug-ins, each defined with a mutually exclusive message property to distinguish between task priority levels, where one plug-in processes tasks defined as high priority, and a different plug-in processes tasks defined as low priority. **Note:** See the JMS specification for the syntax of this selector expression on the Oracle Technology Network website at: `http://www.oracle.com/technetwork/java/jms/index.html` **Note**: When you define only one automation plug-in external event receiver for each automated task, and you use the Optimized build-and-deploy mode to build and deploy automation plug-ins, you are not required to enter a selector in the **Message Property Selector** field. (For OSM 7.3 servers and later, Optimized is the only build-and-deploy mode available.) In this case, automated tasks can share the same JMS queue without a message property selector being set. You must set a message property selector when you do either of the following: • Define multiple automation plug-in external event receivers for the same automated task. • Use the Legacy build-and-deploy mode to build and deploy cartridges with automation plug-ins. • Use both (Allow server preference to decide) build-and-deploy mode to build and deploy cartridges with automation plug-ins and configure the OSM server dispatch mode for the Internal mode. For information on build-and-deploy modes, see "Defining Build-and-Deploy Modes for Automation Plug-ins." **Note:** XPath and XQuery fields are limited to 4000 characters.

Field	Use

XML Message Body Selector	In the **Select** field, enter an XPath expression to select an element from the XML body content. In the **Compare** field, enter the string value of the element to determine the match. Using an XPath expression in this field enables you to filter incoming messages to only those messages defined with specific properties in the body of incoming messages.
	The XML Message Body Selector function is deprecated, but it is supported for backward compatibility. Oracle recommends that you use an alternate way of filtering messages such as message property selector.
	For example, the following sample response from the external system includes a <typeOrder> element that defines the order type:
	`<orderResponse xmlns="http://xmlns.oracle.com/communications/sce/dictionary/OsmCentralOMExample/interactionResponse" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">`
	`<numSalesOrder>1</numSalesOrder>`
	`<numOrder>3</numOrder>`
	`<typeOrder>NEW</typeOrder>`
	`<errorCode>0</errorCode>`
	`<message>OK</message>`
	`<status>A</status>`
	`</orderResponse>`
	For this example response, to consume only the NEW type of order, you can use either of the following in the **Select** field:
	• For a non-namespace-aware query:
	`/*[local-name() ='orderResponse']/*[local-name()='typeOrder']/text()`
	• For a namespace-aware query:
	`/*[local-name() ='orderResponse' and namespace-uri()=' ://xmlns.oracle.com/communications/sce/dictionary/OsmCentralOMExample/interactionResponse']/*[local-name()='typeOrder' and namespace-uri()='://xmlns.oracle.com/communications/sce/dictionary/OsmCentralOMExample/interactionResponse']/text()`
	In both cases, you would set the **Compare** field to **NEW**. All selected messages that contain a typeOrder parameter value of **NEW** (for example, as opposed to Revision, or Follow-On) would be directed to the automation plug-in.
	Note: You can use the **XML Message Body Selector** with response messages of type XMLMessage. OSM ignores the selector for other message types, such as TextMessage. If you need to use the message body selector ensure that messages are of type XMLMessage message. For non-XMLMessage messages you are restricted to the **Message Property Selector**.
	Note: XPath and XQuery fields are limited to 4000 characters.

Related Topics

Example: Modeling a Basic Automator Plug-In

Configuring Automation Plug-In Properties

Working with Automated Tasks

Properties View Compensation Tab

Use the **Properties view Compensation** tab to define the execution modes that automation plug-in can process. This tab appears only for Automator and Sender plug-in types defined as an Internal Event Receiver.

Field	Use
Execution Mode	Specify the execution modes that this plug-in can run when invoked. For more information about execution modes, see "About Task Compensation." Select one of the following execution modes from the **Normal** column for execution modes that occur when the task is processing normally: • Select **Do** to indicate that the automation should run during normal order processing. Deselect this check box and select **Redo** or **Undo** to restrict the automation to compensation processing only. **Note:** If you select no check boxes on this tab, automation processing defaults to the **Do** execution mode. • Select **Redo** to indicate whether the automation should run again if the automation processed once but the order has changed since then. • Select **Undo** to indicate whether the action taken by an automation should be undone when the automation processed but the order has changed since then. You can also exclude certain data structure and elements from being undone on a task by setting the Ignore rollback during undo check box in the Order editor, Order Template, Properties View Order Data tab (see "Properties View Order Data Tab"). Select one of the following execution modes from the **Fallout** column for execution modes that occur when the task has failed: • Select **Do** so that if the task fails when running in the normal Do execution mode, the automation task can still run plug-ins configured with the **Do** in fallout mode. Deselect this check box and select **Redo** or **Undo** to restrict the automation to compensation processing only. • Select **Redo** so that if the task fails when running in the normal Redo execution mode, the automation task can still run plug-ins configured with the **Redo** in fallout mode. • Select **Undo** so that if the task fails when running in the normal Redo execution mode, the automation task can still run plug-ins configured with the **Undo** in fallout mode. You can also exclude certain data structure and elements from being undone on a task by setting the Ignore rollback during undo check box in the Order editor, Order Template, Properties View Order Data tab (see "Properties View Order Data Tab"). **Note:** Design Studio has validations in place that prevent you from defining more than one internal event receiver automation with the same execution mode per automated task or automated notification. For example, an automated task can not define two internal event receiver automations that both have **Do** selected for the execution mode. However, an automated task can define three internal event receiver automations if each defines a different execution mode (**Do**, **Redo**, and **Undo**).

Related Topics

Example: Modeling a Basic Automator Plug-In

Configuring Automation Plug-In Properties

Working with Automated Tasks

Properties View Correlation Tab

Use the **Correlation** tab to specify how an in-bound response from an external system correlates back to the original outbound message that initiated the communication with the external system.

The Properties view **Correlation** tab appears only for Automator and Sender plug-in types defined as external event receivers. The Correlation parameter is mandatory and defaults to **Message Property**, which in turn defaults to **JMSCorrelationID**.

Field	Use
Correlation	Select the method for correlating responses from external systems with the originating response: • Based on a **Message Property**. Message property correlation provides the ability to correlate the incoming message with the appropriate automated task, based on a message property defined in the message header. Message property correlation is the simplest and most commonly used form of correlation. The default message property is **JMSCorrelationID**, which is a JMS-generated number that is placed in the header of all OSM outbound messages. You have the option to change the message property to something other than **JMSCorrelationID**, and be responsible for setting the value on all outbound messages that expect an in-bound response. • Based on an element in the **XML Body.** XML body correlation provides the ability to correlate a response message based on a message property defined in the message body, such as particular data field. If used, the **XML Body** field is defined as an XPath. The **Correlation** parameter is mandatory and defaults to **Message Property**, and the system sets the corresponding default value for the **Message Property** field to **JMSCorrelationID**.
Message Property or **XML Body**	This field is conditional to the selection that you make in the **Correlation** field. If you selected **Message Property** in the **Correlation** field, the system supplies the standard JMS correlation property **JMSCorrelationID.** The OSM order correlates with incoming events from external systems when the **JMSCorrelationID** on the order and on the message matches. If you selected **XML Body** in the **Correlation** field, you can enter an XPath expression to point to an element in the XML body of the message. The value for this element in the OSM order must match the value for the same element on the incoming message. **Note:** XPath and XQuery fields are limited to 4000 characters.

Related Topics

About Automation Message Correlation

Example: Modeling a Basic Automator Plug-In

Configuring Automation Plug-In Properties

Properties View XQuery Tab

Use the **XQuery** tab to specify your XQuery file so the predefined automation plug-in can access it. This tab appears for XQuery type plug-ins only.

> **Note:**
>
> XQuery automation types cannot be implemented when using releases prior to OSM 7.0.

Field	Use
Script	Specify which method to use to retrieve the XQuery file. Select from the following options: • **Bundle in**: Select this option, then click the **XQuery** field **Select** button to identify the file from the **resources** directory. Design Studio will bundle this file with the PAR file during the build. • **Absolute path**: Select this option and enter the physical location of the file. At run time, OSM retrieves the file from the server. • **URL**: Specify a URL to access the file. **Note:** Oracle recommends that you select **Bundle in** for production mode, as this mode pulls the files into the OSM PAR file. As a result, you can deploy the EAR file (which contains the PAR file) to any server and, at run time, the application can locate the files. If you select **Absolute Path** or **URL** for production mode, you can deploy the EAR file (which contains the PAR file) to any server but are responsible for ensuring the files reside in specified location on the server. Conversely, **Absolute Path** or **URL** are optimal for testing mode because they do not require a rebuild and redeploy to pick up changes to the XQuery.
Maximum Number in Cache	Specify the maximum number of XQuery files that can be maintained in the cache at any one time.
Cache Timeout	Enter the number of seconds before the OSM server refreshes the cache.
Exception	Select the exit status that the plug-in should use if it throws an exception. Status options include any status values you assigned to the task. **Note:** This field does not apply to an automation plug-in for an order milestone automation event notification (at the order level); it applies to setting up an automation at the task level.
Update Order	Select this option if you want to update (add, change, or delete) the OSM order data with the data retrieved from an external system. This field appears for Automator automation plug-ins only.

Related Topics

About Automation Message Correlation

Example: Modeling a Basic Automator Plug-In

Configuring Automation Plug-In Properties

Properties View XSLT Tab

Use the Properties view **XSLT** tab to specify the location of your XSLT file so the predefined automation plug-in can access it. This tab appears for all plug-ins except for custom plug-ins.

You can use XSLT to perform some business logic against a task, and determine the exit status based on the processing results. XSLT enables you to model complex calculations, such as date-based calculations, mathematical expressions, calls to external systems, and so forth. Additionally, you can update the order data with the results from your XSLT calculations. Finally, you can use the XSLT style sheet to transform data when sending and receiving data from external systems.

Field	Use
Script	Specify which XSLT style sheet you want to use to transform documents. Select from the following options: • **Bundle in**: Select this option, then click the **Select** button to identify a style sheet from the **resources** directory. Design Studio will bundle this XSLT file with the PAR file during the build. • **Absolute path**: Select this option and enter the physical location of the XSLT file. At run time, OSM retrieves the file from the server. • **URL**: Specify a URL to access the file. **Note:** Oracle recommends that you select **Bundle in** for production mode, as this mode pulls the XSLT files into the OSM PAR file. As a result, you can deploy the EAR file (which contains the PAR file) to any server and, at run time, the application can locate the XSLT files. If you select **Absolute Path** or **URL** for production mode, you can deploy the EAR file (which contains the PAR file) to any server but are responsible for ensuring the XSLT files reside in specified location on the server. **Absolute Path** or **URL** are optimal for testing mode because they do not require a rebuild and redeploy to pick up changes to the XSLT.
Maximum Number in Cache	Specify the maximum number of XSLT style sheets that can be maintained in the cache at any one time.
Cache Timeout	Enter the number of seconds before the OSM server refreshes the cache.
Exception	Select the exit status that the plug-in should use if it throws an exception. Status options include any status values you assigned to the task. **Note:** This field does not apply to an automation plug-in for an order milestone automation event notification (at the order level); it applies to setting up an automation at the task level.
Transformer Factory	(Optional) If you have developed a custom TransformerFactory for XSLT transformation, specify the location. Design Studio provides a default TransformerFactory.
Update Order	Select this option if you want to update (add, change, or delete) the OSM order data with the data retrieved from an external system. This field appears for Automator automation plug-ins only.

Related Topics

Example: Modeling a Basic Automator Plug-In

Configuring Automation Plug-In Properties

Working with Automated Tasks

Properties View Routing Tab

Use the Properties view **Routing** tab to specify where to send XML messages and where external systems can deliver responses. The Properties view **Routing** tab appears for all non-custom XSLT Sender plug-ins.

On the **Routing to** subtab you can specify where to send the XML Request message (JMS Destination). On the **Routing Reply to** subtab you can specify where the external system can deliver the XML Response or Exception message.

Field	Use
JNDI Name	Enter the name of the queue to which the automation plug-in sends messages (on the **To** tab) or to which external systems send response (on the **Reply To** tab). **JNDI Name** is mandatory. Edit the system-supplied default value to reflect your own system topology. The JNDI name must be unique in the workspace.

Field	Use
Destination Type	Select the type of the message destination. A JMS destination is either a javax.jms.Queue or a javax.jms.Topic. You might use a topic, for example, if you want to publish messages for general availability to multiple external systems (on the **To** tab) or subscribe to a queue with multiple listeners (on the **Reply To** tab). You might use queues if you want only a single external system to consume the message.
URL, **Initial Context Factory**, and **Connection Factory**	(Optional) Enter this information to connect to an external application server. Specify the URL and the InitialContextFactory class for the JNDI provider, and specify the ConnectionFactory class for the JMS server.
Send Null Message	Select this option if you want to send a JMS message to an external system even if the message body is empty.

Related Topics

Example: Modeling a Basic Automator Plug-In

Configuring Automation Plug-In Properties

Working with Automated Tasks

Properties View Custom Plug-in Tab

Use the Properties view **Custom Plug-in** tab to define a custom automation plug-in entity in Design Studio. The Custom Automation Plug-in editor associates a Java class representing the custom automation plug-in to the custom automation plug-in entity.

This tab appears when the selected plug-in is a custom automation plug-in. The value is initialized with the value of the XML template for the type of custom automation plug-in.

See *OSM Developer's Guide* for more information about defining the custom automation plug-in.

Related Topics

Example: Modeling a Basic Automator Plug-In

Configuring Automation Plug-In Properties

Working with Automated Tasks

Properties View Notes Tab

Use the Properties view **Notes** tab to describe the intended use for the plug-in. The Properties view **Notes** tab is common to all types of automation plug-ins and event receiver types.

Related Topics

Example: Modeling a Basic Automator Plug-In

Configuring Automation Plug-In Properties

Working with Automated Tasks

Task Editor Behaviors Tab

The Task editor **Behaviors** tab appears for manual and automated task types.

Use the **Behaviors** tab to view all of the behaviors defined for the data nodes in a manual task and an automated task.

The Behaviors table displays the name and type of the task, whether the behavior is enabled, the inheritance properties, the path name of the data node on which the behavior is defined, and the task where the behavior was originally defined.

The information on the **Behaviors** tab is read-only. To change the information that appears on this tab, select a behavior from the table and click the **Properties** button to access the **Behaviors Properties** tabs. See "Working with Behaviors" for more information about the defining behavior properties.

Task Editor Compensation Tab

The Task editor **Compensation** tab appears for manual, automated, and transformation task types.

Use the **Compensation** tab to define your compensation strategy for manual and automated tasks.

> **Note:**
>
> Although compensation strategies are defined on an individual task basis, they must be analyzed within the context of the workflow.

Field	Use
When this task needs to be re-evaluated, compensate by:	A task is re-evaluated by the system if it has visibility to order data (data that is defined as significant) that has changed as a result of an order amendment or as a result of amendment compensation performed on another task to which the task has visibility. The default option, which is to redo the task, applies to tasks that are linear in nature and have the same completion status (no branching). Select **Redo (one single operation)** to instruct the system to perform *Undo* and *Do* operations in a single operation. This option is recommended, when possible, as it performs the fewest number of Undo and Do operations necessary for compensation. Select **Undo then do (two separate operations)** to undo this task and all successor tasks and roll back all order changes, then perform the Do operation again. Use this option to rollback all order changes and re-perform the task from the beginning. Select **Do Nothing** to instruct the system to bypass updating the affected task. For example, you might select this option if a similar task downstream in the process will be compensated, thereby optimizing the compensation plan. Select **Compensation Expression** to create an XQuery expression in the **Compensation Expression** field that dynamically selects a compensation strategy (Redo, Undo then do, or Do nothing) based on revision order data. See "Compensation XQuery Expressions" for more information about compensation strategy XQuery expressions.

Field	Use
When this task is no longer required, compensate by:	A task or subprocess is no longer required when: • The order is canceled. When an order is canceled, the system processes an undo on all of the completed tasks and subprocesses and returns the order to the creation task. The tasks and subprocesses associated with the order are no longer required because the order has been canceled. • A branch becomes obsolete. A branch becomes obsolete when the redo processing of a particular task or subprocesses causes that task or subprocess to a) exit with a different completion status and b) start a new branch. Because the tasks and subprocesses in the obsolete branch are no longer required, they are undone, one task or subprocess at a time, starting with the last completed task or subprocess in the branch. • When the task re-evaluate compensation strategy is selected as **Undo then do** and if this option is selected as **Undo**, then all the tasks are undone based on their corresponding compensation strategy. If this option is selected as **Do Nothing**, the corresponding tasks are not undone. In both scenarios, the system rolls back the order changes. The difference between them is the creation of a compensation undo task. Undoing the task and rolling back order changes creates an undo task; automatically rolling back order changes does not create an undo task. Undo compensation tasks created for manual tasks appear in the Task web client Worklists and must be manually acknowledged in order to be rolled back. This also applies to a task which has an **Undo then do** compensation strategy for when the task needs to be re-evaluated. When the task needs to be undone (as a part of **Undo then do**), it follows the compensation strategy for when the task is no longer required (either **Do nothing** or **Undo**). When the task needs to be compensated, it gets undone first and then done. The task does not come into Undo in the worklist, but only goes to Do (in amending) or Do (in progress). Select **Undo** to create an undo read-only task in the Task web client. Select **Do Nothing** to instruct the system that no compensation is necessary. Select **Compensation Expression** to create an XQuery expression in the **Compensation Expression** field that dynamically selects a compensation strategy (Undo or Do nothing) based on revision order data. See "Compensation XQuery Expressions" for more information about compensation strategy XQuery expressions.
When an amendment occurs this task will be compensated if it is:	Most tasks should only be included in amendment processing after the task has completed. However, you may want to include in progress tasks in amendment processing when the tasks are long running, for example when interacting with a workforce management system where task fulfillment can take hours or even days to complete. Select **Completed** to instruct the system to include the task in amendment processing only when the task is completed. Select **Completed or in progress** to instruct the system to include the task in amendment processing when the task is completed or when the task is in progress. A task is considered to be in progress when the task is in the Accepted state or in any user-defined state. You can also further refine when an in progress task is included into amendment processing by specifying the **In Progress Compensation Include Expression** and the **In Progress Compensation Complete Expression** XQuery expressions. Select **In Progress Compensation Include Expression** to create an XQuery expression that further specifies when instances of this in progress task can be included in amendment processing based on revision order data. For example, the expression may determine that the task only be included in amendment processing when it includes product A rather than product B. See "Compensation XQuery Expressions" for more information about compensation strategy XQuery expressions. Select **In Progress Compensation Complete Expression** to create an XQuery expression in the **Compensation Expression** field that runs whenever data is updated on the task that checks when compensation has completed based on the order data changes. For example, the order amendment could specify that the task run in redo mode. The task re-sends a request for a customer service with changes to an external fulfillment system that returns an acknowledgement response that the XQuery expression recognizes as completing the compensation for the in progress redo task. The task can then return to the normal do execution mode and waits for the external system to functionally complete the task and respond so that the task can be completed. See "Compensation XQuery Expressions" for more information about compensation strategy XQuery expressions.

Field	Use
When an amendment occurs if this task is in progress it will:	If amendment processing occurs while a task is in progress, you can specify what kind of grace period should be enforced before the task can run in the compensation execution mode. Select **Wait for the grace period** to instruct the task to run in the compensation execution mode when the grace period specified on the order-life cycle for the Process Amendment transition. Select **Be excluded from the grace period** to instruct the task to run immediately regardless of the grace period specified on the order-life cycle for the Process Amendment transition. Select **Wait for specified duration** to statically configure the grace period for the task by seconds, minutes, hours, or days. Select **Dynamic Expression** to create an XQuery expression that dynamically specified the wait duration based on revision order data. This expression runs regardless of what option is specified from the above list. See "Compensation XQuery Expressions" for more information about compensation strategy XQuery expressions.

Note:

If an amendment is received while a task is in a fallout execution mode, the following will happen:

- If the task is not configured to be compensated if it is in progress, the execution mode of the task will not change as a result of the amendment order.
- If the task is configured to be compensated if it is in progress, and the amendment contains changes to significant data:
  - If the task is still needed after the changes to the order from the amendment are considered, it will transition automatically to (normal) Redo mode.
  - If the task is no longer needed after the changes to the order from the amendment are considered, it will transition automatically to (normal) Undo mode.

  In both of these cases, your automation code (for either Redo or Undo execution mode) should contain both a check to see if the task has been in a fallout execution mode, and also any code that is needed to resolve any actions that have been taken in the fallout execution mode. For example, if your automation for Do in Fallout mode opens a trouble ticket, your Redo automation should check to see whether it needs to close a trouble ticket.
- If the amendment order contains no changes to significant data, the execution mode of the task will not change as a result of the amendment order.

Related Topics

About Task Compensation

Working with Tasks

Task Editor Details Tab

The Task editor **Details** tab appears for manual, automated, activation, and transformation task types.

Use the **Details** tab to define attributes that you can use to extend the task definition.

For all tasks in a process, there are properties that you define in Design Studio that the OSM server requires to properly run the task. These properties include the order with which the task is associated, the amount of time in which you expect the task to complete, the group responsible for completing the task, and the manner in which the tasks are assigned. You configure these details on the Task editor **Details** tab. The **Details** tab also contains properties that enable you to add or remove a parent task (and its inherited data), and to model the data node on which a multi-instance process relies to create multiple instances of the task.

Field	Use
Extends	Select an existing or create a new task to extend this task (the task's data is inherited) by clicking the **Select** button. To create a new task that this task would be an extension of, click **New**. After you have selected or created an task, click **Open** to access the Task editor. Click **Clear** (red X) to clear the selected value from the field. Using task inheritance, you can leverage existing task data when building new, similar tasks. See "About Task Extensions and Inheritance" for more information.
Order	The order associated with a task determines the overall data set that will be available to the task when you model the task data. **Note:** If you are planning to use the task for an order (OrderA) and also an order (OrderB) that is extended from that order, you must select the parent order (OrderA) here.
Pivot Node	(Optional) Select the Pivot node for this task. When OSM runs the corresponding task at run-time, the system generates a separate task instance for each separate value of the pivot node in the order. For example, if the pivot node is an address field, and three addresses are included in the order, the system generates three separate task instances when this task occurs in a process. **Note**: OSM compensation processing does not support task pivot nodes.
Expected Duration and **Calculate using Workgroup Calendar**	Specify the length of time expected to complete the task. By default, the expected duration of a task is set to 1 day (system time). You can select any value up to 999 in weeks, days, hours, minutes, or seconds. You can also calculate the duration based on your workgroup calendar by selecting **Calculate using Workgroup Calendar**. 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. Expected duration can be useful during reporting and jeopardy processing.
Order Priority Offset	Select a value between 9 and -9 to differentiate this task's priority within 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 other tasks in the order 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.
Responsibility	Select which department or team is responsible for this task. The default value is System. You can select **System** or enter a value that is meaningful within the context of your system topology. This field is only visible to the reporting API.
Namespace Based on Task Name	(Automated, Activation, and Transformation tasks only) Select this option to use a namespace for the task that is based on the task name.
Transport	Select **System Interaction** to use system interaction in automated tasks. The default value is Direct JMS.

Field	Use
Assignment Algorithm	(Manual tasks only) (Optional) Select the algorithm to use when automatically assigning tasks to users. OSM provides two default algorithms: *Load Balancing* and *Round Robin*. The **Load Balancing** algorithm attempts to distribute tasks based on a user's current workload. The OSM server assigns tasks after determining which user in the workgroup has the fewest number of assigned tasks. The **Round Robin** algorithm assigns tasks in a predefined order among the users in the workgroup. You can add custom assignment algorithms to OSM, using OSM's cartridge management tools. For custom algorithms, you must manually enter the algorithm name in the **Assignment Algorithm** field. If you do not specify an algorithm in this field, you must manually assign tasks.
JNDI Name	(Manual tasks only) Enter the JNDI name for custom assignment algorithms.
Transformation Manager	(Transformation tasks only) Enter the name of the transformation manager to be called when this transformation task is reached. Do any of the following: • Click **Select** to select an existing transformation manager. • Click **New** to create a new transformation manager. See "Creating New Transformation Managers" for more information. • Click **Open** to open the selected transformation manager in the Transformation Manager editor.
Order Component	(Transformation tasks only) Enter the name of the order component that provides context for this transformation task and assists in order item selection. Do any of the following: • Click **Select** to select an existing order component. • Click **New** to create a new order component. See "Creating New Order Component Specifications" for more information. • Click **Open** to open the selected order component in the Order Component editor. If you are not using the default provided automation plug-in for the transformation task, this field may be optional, depending on the way your automation is written.
Update Order with Transformation Response	(Transformation tasks only) Select this option to enable OSM to persist the transformed order items on the order.

> **Note:**
>
> You cannot use pivot nodes to model multiple instances of activation tasks. To model multiple activation task instances, create a multi-instance subprocess that contains only the activation task.

Related Topics

Defining Task Data

Working with Tasks

Task Editor Events Tab

The Task editor **Events** tab appears for manual, automated, activation, and transformation task types.

Use the **Events** tab to create task state automation event notifications. You select the task state that triggers the automation, and then configure the automation plug-in that will perform the work.

Field	Use
State	The **State** column displays the states for which you have defined automation events. When the task reaches the corresponding state, the OSM server triggers the automation event plug-in.
Name	In the Automation column, the **Name** field displays the name of automation plug-in.
Automation Type	Displays the automation plug-in type. See "Working with Automation Plug-Ins" for more information.
Add	Click the State column **Add** button to add a predefined task state to the list. Click **Add** in the Automation column to define a new automation plug-in for the corresponding task state.
Remove	Select a state or an automation plug-in and click **Remove** to delete the entity from the list of events.
Properties	Select an automation plug-in and click **Properties** to configure the properties of the new plug-in. See "Configuring Automation Plug-In Properties" for more information. The **Properties** button appears only after you have added at least one automation plug-in to the table.

Related Topics

Creating Order Milestone and Task State Automation Event Notifications

Working with Event Notifications

Task Editor Fallouts Tab

The Task editor **Fallouts** tab appears for manual, automated, and transformation task types.

Use the **Fallouts** tab to specify the types of fallout that can occur for a task.

Click **Add** to open the Select Fallouts dialog box, where you can select fallouts previously defined on the Order editor **Fallouts** tab.

Select any fallout defined in the **Name** column and click **Remove** to delete the fallout from the list.

Select any fallout defined in the **Name** column and **click** Open to open the fallout in the Order editor **Fallouts** tab.

Related Topics

About Task Fallout

Task Editor Jeopardy Tab

The Task editor **Jeopardy** tab appears for manual, automated, activation, and transformation task types.

Use the **Jeopardy** tab to create jeopardy notifications when certain conditions arise in a task and you want to alert users or systems of processes, orders, or tasks that may be at risk.

The **Jeopardy** tab has the following subtabs:

- Task Editor Jeopardy Details Tab
- Task Editor Jeopardy Conditions Tab
- Task Editor Jeopardy Notify Roles Tab
- Task Editor Jeopardy Polling Tab
- Task Editor Jeopardy Automation Tab
- Task Editor Jeopardy Notes Tab

Task Editor Jeopardy Details Tab

Use the **Jeopardy Details** tab to name the jeopardy, select the notification rule, set the priority level, enable or disable the notification, and specify whether to send the notification by email.

Field	Use
Name	Enter a name to identify the jeopardy.
Rule	Select the rule the system should evaluate before generating this jeopardy. This field defaults to the system-based *null_rule*. If you do not change the default value, OSM will always trigger this notification at the specified polling interval. See "Defining Order Rules" for more information about setting up new rules.
Priority	Enter a priority from 1 to 255 (1 is the highest priority). The notification with the highest priority is evaluated first.
Enabled	Select this option to enable this jeopardy notification, or deselect the option if you intend to implement the notification at a later time.
Email	Select this option to send email notifications to all users in the workgroup associated with the specified role. By default, notifications appear in the Notifications page of the Task web client. However, you can specify that notifications be sent by email by selecting the **Email** check box. When you assign users to a workgroup in the OSM Administration area of the Order Management web client, you can set up OSM to notify users by email. When a notification occurs, the system sends a notification ID number through email. See *OSM Order Management Web Client User's Guide* for information about configuring email notification properties for user roles. See *OSM Installation Guide* for information about configuring the outgoing email server.

Related Topics

Creating Jeopardy Notifications in the Task or Order Editor

Working with Jeopardy Notifications

Working with Tasks

Task Editor Jeopardy Conditions Tab

Use the **Jeopardy Conditions** tab to select the conditions under which the jeopardy should be raised. For example, you can raise a jeopardy when this task exceeds the expected or a given duration.

Field	Use
Raise a Jeopardy when Process Duration Exceeds	Raise a jeopardy if the process to which the task is associated has exceeded the **Expected Duration** of the order (defined on the Order editor **Details** tab) or **Given Duration**, specified by the time interval defined in the adjacent field.
Raise a Jeopardy when Task Duration Exceeds	Raise a jeopardy if the task has exceeded the **Expected Duration** (defined on the Task editor **Details** tab) or **Given Duration**, specified by the time interval defined in the adjacent field.
Raise a Jeopardy when the order is received within	Raise a jeopardy if the order has been received and the time interval defined in the adjacent field has been exceeded.
Multiple events per Task instance	When a task has multiple instances, select this option if, when a jeopardy notification is triggered, you want a notification triggered for every task instance.

Related Topics

Creating Jeopardy Notifications in the Task or Order Editor

Working with Jeopardy Notifications

Working with Tasks

Task Editor Jeopardy Notify Roles Tab

Use the **Jeopardy Notify Roles** tab to select the roles to be notified when the jeopardy occurs.

Select a predefined jeopardy from the list in the left column to activate a list of available roles. See "Working with Roles" for information about defining roles. Using the directional arrow buttons, move the roles (those groups to whom you want the notification sent) into the Selected Column.

If the jeopardy notification is sent to an external system via an automation plug-in, ensure that you include the role whose credentials are used when running the automation plug-in. See "Configuring Automation Plug-In Properties" for more information.

Related Topics

Creating Jeopardy Notifications in the Task or Order Editor

Working with Jeopardy Notifications

Working with Tasks

Task Editor Jeopardy Polling Tab

Use the **Jeopardy Polling** tab to select the interval at which the OSM server evaluates the condition that triggers the jeopardy notification. You can define the polling so that the system checks for the condition only once, or you can define the polling at hourly, daily, weekly, or monthly intervals.

Field	Use
Interval	Select the interval at which the OSM server evaluates the condition that triggers the jeopardy notification. Select **Once** if you want the system to check for the condition only once when the order is received. When you select **Once**, the system disregards the **Next Start** field. Use the **Hours**, **Days**, and **Months** fields to define a specific interval at which the OSM server evaluates the condition that triggers the jeopardy notification. For example, if you want the system to check for the condition every two days, select the **Day(s)** option and from the drop-down list select 2.

Field	Use
Next Start	Select the date and time that you want the notification to begin checking. You can specify a date for any polling interval. The system uses the current date and time as the default value.

Related Topics

Creating Jeopardy Notifications in the Task or Order Editor

Working with Jeopardy Notifications

Working with Orders

Working with Tasks

Task Editor Jeopardy Automation Tab

Use the **Jeopardy Automation** tab to configure an automation plug-in that performs the work or sends data to an external system when the jeopardy notification is triggered. OSM supports one automation plug-in per Jeopardy.

Field	Use
Add	Click **Add** to open the Add Automation dialog box is displayed, where you can define a new automation plug-in for the jeopardy notification.
Name	Enter a name for the automation entry.
Automation Type	Select the automation plug-in type from the available list. Click **OK** to add the automation entry to the Jeopardy Automation table.
Properties	Select any entry in the table and click to define the automation properties. See "Configuring Automation Plug-In Properties" for more information about defining automation properties in the Properties view.

Related Topics

Creating Jeopardy Notifications in the Task or Order Editor

Working with Jeopardy Notifications

Working with Tasks

Task Editor Jeopardy Notes Tab

Use the jeopardy **Notes** tab to denote the intended use of the notification or any additional information that you want to append to the jeopardy data.

Related Topics

Creating Jeopardy Notifications in the Task or Order Editor

Working with Jeopardy Notifications

Working with Tasks

Task Editor Permissions Tab

The Task editor **Permissions** tab appears for manual, automated, activation, and transformation task types.

Use the **Permissions** tab to assign roles to each of the three possible task execution modes.

Field	Use
Do, **Redo**, **Undo**, **Do in Fallout**, **Redo in Fallout**, and **Undo in Fallout**	For each role listed in the Role Name column, select or deselect, as appropriate, the **Do**, **Undo**, **Redo**, **Do in Fallout**, **Redo in Fallout**, and **Undo in Fallout** check boxes to enable or disable access to the task execution modes. These options represent the three possible task execution modes: • Do is the default mode for a task that runs under normal processing. • Undo reverses the effects of the associated Do operation. • Redo combines both Undo and Do operations in a single operation. • Do in Fallout is the mode for a task that runs when the task fails while running in Do mode. • Undo in Fallout is the mode for a task that runs when the task fails while running in Undo mode. • Redo in Fallout is the mode for a task that runs when the task fails while running in Redo mode.
Select	Click **Select** to select a predefined role to add to the permissions list. If no roles have been previously defined, click **New** to create a role. You must define at least one role in the permissions list for every task.
New	Click **New** to open the Role wizard and create a new role to assign to the task. To select a role that was previously defined, click **Select**. You must define at least one role in the permissions list for every task.
Open	Select any role in the Role Name column and click **Open** to open the role in the Role editor.
Remove	Select any role in the Role Name column and click **Remove** to delete the role from the task permissions list.

Related Topics

Assigning Task Permissions

Task Editor Redo Tab

The Task editor **Redo** tab appears for activation task types.

Use the **Redo** tab to define part of your compensation strategy for activation tasks: to redo tasks that are affected by amendments. Complete the compensation strategy for activation tasks on the **Undo** tab. See "Task Editor Undo Tab" for more information.

Field	Use
Compensation Strategy	Specify the compensation strategy to redo a task when it is affected by an amendment: • Select **Manual** if manual intervention is required at run time. • Select **Ignore** to instruct OSM to skip this task. • Select **Undo then do** to instruct OSM to undo the task and redo the task as two separate transactions. The task is redone using the same request mapping defined on the **Request Data** tab. • Select **Redo (amend existing order)** to instruct OSM to undo the task and redo the task as a single transaction, sending the oderByValueRequest parameter with the replace command, replacing the original order with the new command. The task is redone using the same request mapping defined on the **Request Data** tab. • Select **Redo (new order)** to instruct OSM to send a new order to the activation system. The new order can be configured with new request mappings.
Use existing request mapping	When **Compensation Strategy** is set to **Redo (new order)**, the Redo operation uses the same request mapping settings as the original order.
Re-configure request mapping	When **Compensation Strategy** is set to **Redo (new order)**, you can specify new request mapping settings for the Redo operation. The Task Data area and Service Actions area behave as they do on the **Request Data** tab. See "Task Editor Request Data Tab" for more information.

Related Topics

Modeling Activation Tasks

About Activation Tasks

Task Editor Request Data Tab

The Task editor **Request Data** tab appears for activation task types.

Use the **Request Data** tab to configure service action requests by mapping OSM order header and task data to Activation order header, service action, and global parameters.

See the following topics for more information:

- Properties Activation Order Header Binding View
- Properties Global Parameter Binding View
- Properties Service Action Binding View
- Properties Parameter Binding View

Field	Use
Task Data area	Select one of the following values: • Select **Order Header** to display the standard order header fields request parameters. • Select **Task Data** to display the task data request parameters. Right-click a data element in this view and select **Auto map parameter** to automatically map the element to a service action parameter that shares the same name (case insensitive). See "Configuring Service Action Requests" for information about mapping. See "Defining Task Data" for information about adding OSM data to the activation task.

Field	Use
Service Action area	Displays the request parameters to which you can map OSM data. Expand the **Activation Order Headers** folder, the **Global Parameters** folder, or any service action to review the parameters available for mapping. Check marks indicate which parameters are mapped to OSM data. **Note:** Some Activation order header parameters require default values. Before mapping OSM data to Activation order header parameters, note which parameters are prepopulated with a check mark to determine those that require default values. Right-click in the Service Actions area to access the context menu. The Service Actions area context menu enables you to add service data to a task, define new global parameters, associate additional service actions to the tasks, remove parameters, and remove mapping information. **Note:** When adding service action parameters to OSM task data, you can add all parameters of a service action to a selected OSM data structure. Service action parameters are not added to the structure if it contains a child element with the same name as the parameter. Design Studio limits the maximum length of service action parameters to 1000 when adding them to a structure. If you create data elements for service action parameter fields manually (using the Data Schema editor), ensure that you set the maximum length of the new data element equal to the maximum length defined for service action parameter.

Related Topics

Configuring Service Action Requests

Task Editor Request Data Tab

About Service Action Request Mapping

Modeling Activation Tasks

Properties Activation Order Header Binding View

Use the **Activation Order Header Binding** view to review and edit mapping information between OSM data and Activation order header parameters.

Field	Use
Order Header	Displays the parameter label.
Condition	Enter the condition that determines whether the parameter is included in the request. If the condition evaluates to **true**, the parameter is sent.
Binding Type	Select to define the expression path as an **XPath Expression** or as an **XSLT Snippet**. For example, you might define the expression path as an XSLT snippet if you are mapping OSM data to a compound order header parameter.
Binding	Displays the mapping information for an order header parameter. **Note:** If you are mapping OSM data to a compound parameter, you can reference the **CreateOrderByValueRequest_generated.xsl** file to ensure that all XPath expressions are defined correctly. To review the **CreateOrderByValueRequest_generated.xsl** file, switch to the Java perspective and click the **Package Explorer** tab. Each activation task is listed in the **Activation** directory in the project **resources** folder.

Related Topics

Configuring Service Action Requests

Task Editor Request Data Tab

About Service Action Request Mapping

Modeling Activation Tasks

Properties Global Parameter Binding View

Use the **Global Parameter Binding** view to review and edit mapping information between OSM data and global parameters.

Field	Use
Parameter	Displays the parameter label.
Condition	Enter the condition that determines whether the parameter is included in the request. If the condition evaluates to **true**, the parameter is sent.
Binding Type	Select this option to define the expression path as an **XPath Expression** or as an **XSLT Snippet**. For example, you might define the expression path as an XSLT snippet if you are mapping OSM data to a compound global parameter.
Binding	Displays the mapping information for a global parameter. **Note:** If you are mapping OSM data to a compound parameter, you can reference the **CreateOrderByValueRequest_generated.xsl** file to ensure that all XPath expressions are defined correctly. To review the **CreateOrderByValueRequest_generated.xsl** file, switch to the Java perspective and click the **Package Explorer** tab. Each activation task is listed in the **Activation** directory in the project **resources** folder.

Related Topics

Configuring Service Action Requests

Task Editor Request Data Tab

About Service Action Request Mapping

Modeling Activation Tasks

Properties Service Action Binding View

Use the **Service Action Binding** view to review and edit mapping information between OSM data and service action parameters and to define the conditions under which the service is added to the request.

Field	Use
Service Action	Displays the service action to which the selected parameter is associated.
View Node	Displays the activation system parameter name.
Condition	Enter the condition that determines whether the service is included in the request. If the condition evaluates to **true**, the parameter is sent.

Related Topics

Configuring Service Action Requests

Task Editor Request Data Tab

About Service Action Request Mapping

Modeling Activation Tasks

Properties Parameter Binding View

Use the **Parameter Binding** view to review and edit mapping information between OSM data and service action parameters and to review the default information defined for the service action.

Field	Use
Service Action	Displays the service action to which the selected parameter is associated.
Parameter	Displays the parameter name.
Default Value	Displays the default value defined for a parameter. You define service action parameters in the Service Action editor.
Condition	Enter the condition that determines whether the parameter is included in the request. If the condition evaluates to **true**, the parameter is sent.
Binding Type	Select this option to define the expression path as an **XPath Expression** or as an **XSLT Snippet**. For example, you might define the expression path as an XSLT snippet if you are mapping OSM data to an ASAP compound parameter.

Binding	Displays the mapping information for a parameter in a service action folder. Consider the following example, which demonstrates a mapping of OSM data elements **dsl**, **VoIP**, and **tv** to an ASAP compound parameter. In this example, you would select **XSLT Snippet** in the **Binding Type** field and enter the following:
	`<xsl:if test="osm:feature/osm:dsl='true'">`
	`<mslv-sa:serviceValue xsi:type="mslv-sa:ASAPServiceValue">`
	`<mslv-sa:name>OLD_SERVICE</mslv-sa:name>`
	`<mslv-sa:value>DSL</mslv-sa:value>`
	`</mslv-sa:serviceValue>`
	`</xsl:if>`
	
	`<xsl:if test="osm:feature/osm:VoIP='true'">`
	`<mslv-sa:serviceValue xsi:type="mslv-sa:ASAPServiceValue">`
	`<mslv-sa:name>OLD_SERVICE</mslv-sa:name>`
	`<mslv-sa:value>VOIP</mslv-sa:value>`
	`</mslv-sa:serviceValue>`
	`</xsl:if>`
	
	`<xsl:if test="osm:feature/osm:tv='true'">`
	`<mslv-sa:serviceValue xsi:type="mslv-sa:ASAPServiceValue">`
	`<mslv-sa:name>OLD_SERVICE</mslv-sa:name>`
	`<mslv-sa:value>TV</mslv-sa:value>`
	`</mslv-sa:serviceValue>`
	`</xsl:if>`
	
	Note: If you are mapping OSM data to a compound parameter, you can reference the **CreateOrderByValueRequest_generated.xsl** file to ensure that all XPath expressions are defined correctly. To review the **CreateOrderByValueRequest_generated.xsl** file, switch to the Java perspective and click the **Package Explorer** tab. Each activation task is listed in the **Activation** directory in the project **resources** folder.

Related Topics

Configuring Service Action Requests

Task Editor Request Data Tab

About Service Action Request Mapping

Modeling Activation Tasks

Task Editor Response Data Tab

The Task editor **Response Data** tab appears for activation task types.

Use the **Response Data** tab to map responses to OSM data structures and configure state and status transitions for completion events and exceptions returned by the activation system.

Field	Use
Event/Exception	Select an event or an exception. For each event or exception, you define: • The response data to use to update the OSM order • The location in OSM where the response data is to be copied • A state or status transition Events and exceptions that include any mapping or transition configuration are represented by a shaded (green) flag icon. Events and exceptions with no configuration defined are represented by an empty (or gray) flag icon.
Activation Response	Select the data returned from the activation system that updates the OSM order. If you do not select a data field for an event or exception, OSM ignores that event or exception.
Response Data Location	Define the OSM data structure to contain the data returned from an event or exception. For each event or exception, you select an existing data structure from the order template or from the data dictionary and right-click that data structure to define the data location. See "Configuring Service Action Responses" for more information.
OSM Data Binding	Bind activation response elements to arbitrary task data elements by dragging elements from the Activation Response area onto OSM data structures. **Note:** Do not add an OSM data structure in this field that uses a distributed order template. Attempting to map a response value to a data element in a distributed order template will cause an error. For more information about distributed order templates, see *OSM Concepts*.
Transition to State or Status	Displays default transitions defined for completion events and exceptions. You can define additional state transitions for user-defined states and additional transitions for predefined task statuses using the **Add** button.
Move Up and **Move Down**	Select a state or status transition row in the **Transition to State or Status** table and click **Move Up** and **Move Down** to change the order in which OSM evaluates the transition conditions at run time.
Properties	Select a state or status transition row in the **Transition to State or Status** table and click **Properties** to define the condition against which OSM evaluates the transition. See "Properties State/Status Transition View" for more information.
Response Filter	Enables you to limit the amount of response data sent to update OSM order data. See "Response Filter Area" and "Filtering ASAP Response Data" for more information.
Remove	Select a state or status transition row in the **Transition to State or Status** table and click **Remove** to delete the transition.
Add	Click **Add** to define a new user-defined state or status transition for a selected event or exception.

Related Topics

Configuring Service Action Responses

About Service Action Response Mapping

Modeling Activation Tasks

Response Filter Area

Properties State/Status Transition View

Use the **Properties State/Status Transition** view to define the condition against which OSM evaluates (at run time) a state or status transition for a service action response.

Field	Use
Condition Name	Displays the condition name as defined on the **Response Data** tab in the **Transition to State or Status** table.
State/Status	Displays a user-defined state or status transition as defined on the **Response Data** tab in the **Transition to State or Status** table. Click **Select** to change the state or status.
Condition	Define the condition against which OSM evaluates the transition. At run time, OSM evaluates the conditions in an order you define and stops evaluating when a condition evaluates to **true**. For example, consider that you want to define a condition for the **orderFailEvent** to transition the task to a suspended state (**NEUnknown**) because of a network element error. You can define the condition in the following manner: `contains($osmOrderDocument/osm:GetOrder.Response/osm:_root/osm:ASAPResponse/osm:EventData/osm:reason[starts-with(.,'orderFailEvent')], 'SARM_MSG:Routing Error')` **Note**: The `$osmOrderDocument` is a variable that represents the OSM order data. Completion events and exceptions must include a default transition should all specified conditions fail. You can change or delete the predefined default values, or you can create your own. However, if you define no default conditions for ASAP completion events and exceptions (no condition is defined with XPath expression **`true()`**) Design Studio creates a problem marker.

Related Topics

Configuring Service Action Responses

About Service Action Response Mapping

Task Editor Response Data Tab

Modeling Activation Tasks

Response Filter Area

Use the Response Filter area to display and define conditional mappings for service-action response parameters and value items. For example, you may want to update order parameters with response data only when certain infoParms have a certain value. In the Response Filter area, you can specify an XPath condition which determines whether or not to update OSM order data with response data.

Field	Use
Event/Exception Name	Name of the event or exception on which to filter parameters or value items.
Filter on	Select the parameters on which to filter. Select from either **infoParm** or **Command History value item**.

Field	Use

Condition	Define the conditional mappings of the infoParm or value item parameters by dragging and dropping the desired parameters or items from the Activation Response area into the **Condition** field. When the XPath representation of a parameter is displayed, set the desired condition by entering an XPath operator.
	For example, if you only want to update an order when the serviceId infoParm parameter from orderCompleteEvent is equal to two. First, select **orderCompleteEvent** in the **Event/Exception** field. Then, in the Activation Response area, click **Detailed Parameters** and **infoParm**. Drag and drop **serviceId** into the **Condition** field. The XPath representation of **serviceId** will appear as follows:
	`mslv-sa:serviceId`
	Now set the desired condition by adding ='2'
	`mslv-sa:serviceId='2'`

Related Topics

Filtering ASAP Response Data

Configuring Service Action Responses

About Service Action Response Mapping

Task Editor Response Data Tab

Modeling Activation Tasks

Task Editor Composite Data View Tab

The Task editor **Composite Data View** tab appears for manual, automated, and transformation task types.

Use the **Composite Data View** tab to display all of the data that is available to a task within the context of an OSM solution. For example, if you added a new fulfillment function to extend a solution, you would see the additional data nodes required by the function as well as any new control data. The task data in the **Composite Data View** tab is read-only. You model the data in the **Task Data** tab of the Composite Cartridge View editor.

> **Tip:**
>
> The composite data view has at least the same number or more data nodes than its corresponding task view.

Field	Use
Solution	Select the solution to display all of the task data associated with the solution.
Task Data	Displays all of the data that is available to a task, including additional data that has been contributed within the context of a solution. You cannot modify any data that appears in the Task Data area.
Behaviors	Displays all of the behaviors defined for each data node. Select a data node in the Task Data area to view the behaviors defined for the node. You cannot modify any behaviors that appear in the Behaviors area.

Related Topics

Working with Composite Cartridge Views

Working with Composite Cartridge Projects

Task Editor States/Statuses Tab

The Task editor **States/Statuses** tab appears for manual, automated, activation, and transformation task types.

Use the **States/Statuses** tab to add, remove, and assign predefined states and statuses to tasks, and to define status severity levels.

Field	Use
Name	Displays the database name of the entity. Select the value in the column to rename.
Display Name	Displays the name of the entity as it will appear in the Task web client. Select the value in the column to rename. **Note:** Design Studio automatically capitalizes display names.
Constraint	Sets the Constraint severity level, which determines the transition behavior of a task when a constraint violation is encountered. The Constraint value represents the highest allowable Constraint behavior violation value with which the task transition will be allowed to occur. Select one of the following: • **Critical:** The transition is allowed for all constraint violations. • **Error:** The transition allowed for all constraint violations except Critical. • **Warning:** The transition is allowed for all constraint violations except Critical and Error (this is the default). • **Valid:** The transition is allowed only for a Valid constraint violation. • **None:** The transition is not allowed for any constraint violations. See "Defining Constraint Behavior Properties" for more information.

Related Topics

Assigning Task States and Statuses

About Task States and Statuses

Task Editor Task Data Tab

The Task editor **Task Data** tab appears for manual, automated, and transformation task types.

Use the **Task Data** tab to define which data is necessary to complete the task. You can drag data from the Data Element view into the Task Data area, or right-click in the Task Data area to select data from the Order Template or Data Dictionary dialog boxes.

When modeling task data using the **Task Data** tab, see the following topics for more information:

- Task Data Node Properties View Identification Tab
- Task Data Node Properties View Dictionary Tab

Field	Use
Task Data	Displays the data that the task requires to complete. The order in which the data appears in the Task Data area is the order in which it appears in the Task web client (or the order in which the data appears in the XML API if this task is an automated task intended to integrate with an external system). Select a data node, right-click and select **Move Up** or **Move Down** to reposition the node in the task view. See "About the Task Editor Task Data Context Menu" for descriptions of other actions you can perform in the Task Data context menu.

Field	Use
Behaviors	Displays all of the behaviors defined for each data node. Select a data node in the Task Data area to view the behaviors defined for the node, or to create new behaviors. When defining behaviors at the task level, you can use the Task editor **Task Data** tab to create the behavior, the Behavior Properties tabs to refine the behavior information, and the Task editor **Behaviors** tab to quickly view all of the behaviors defined for a task. See the following topics for information about defining behaviors: • Defining Manual Task Behaviors • Defining Automated Task Behaviors

Related Topics

Defining Task Data

Working with Tasks

Task Data Node Properties View Identification Tab

Use the Task Data Node Properties View **Identification** tab to edit the information defined for the corresponding data element at the task level.

Right-click any attribute in the Task editor **Task Data** tab and select **Open Properties View** to edit the data element properties at the task level.

Field	Use
Name	Displays the name of the element as defined in the Data Dictionary. The name of the node is not available for edit. You can edit the value in the **Display Name** field on the **Data Schema editor Details** subtab to edit the manner in which the element displays.
Path	Displays an XPath expression to define the location of the node in the Data Dictionary.
Default Value	Select this option and enter a value to initially populate the field associated with this data node in the Task web client.
Read Only	Select this option to make the field read-only field (for this task only) in the Task web client.
Significance	By default, a node inherits significance from its parent. At the task level, you can define the significance as **Not Significant** if you do not want to use the node during amendment processing. During amendment processing, the OSM system compensates only for task instances that use significant data elements as inputs. If an element is not specified as significant, the system updates the order only with the changed data (no compensation is required). Data significance is supported at the data dictionary, order template, and task view levels.
Override Data Dictionary Minimum/Maximum	You can define the Minimum and Maximum values at the task level: In the **Minimum** field, select the number of times the global element referenced can appear in an instance document. Select **0** if you want the element to be optional. By default, nodes are optional at the task level. In the **Maximum** field, select the maximum number of times the global element referenced can appear. Select *unbounded* to indicate there is no maximum number of occurrences.
Apply To Children	Select this option to propagate the **Read-Only**, **Significance**, and **Override Data Dictionary** fields to all direct children of the selected element. The Confirm Change dialog box appears. Select **Recursive** in the Confirm Change dialog box to apply the changes to all other children of the selected element. All changes are saved immediately upon confirmation.

Field	Use
Contributor	Identifies the task that contributes the data element. For example, consider that you have 2 tasks, Task1 and Task2. Task2 extends Task1 and also contains 1 additional data element, *billing_start_date*. The contributor for all of the data elements (except for *billing_start_date*) appears as Task1. The contributor for *billing_start_date* appears as Task2.

Related Topics

Defining Task Data

Working with Tasks

Task Data Node Properties View Dictionary Tab

Use the Task Data Node Properties View **Dictionary** tab to edit the information defined for the corresponding data element at the task level.

Right-click any attribute in the Task editor **Task Data** tab and select **Open Properties View** to edit the data element properties at the task level.

Field	Use
Name	Displays the name of the element as defined in the Data Dictionary. The name of the node is not available for edit. You can edit the value in the **Display Name** field on the **Data Schema editor Details** subtab to edit the manner in which the element displays.
Display Name	Displays the name of the element as it will appear in the Task web client. You can define different display names for the languages that you support in the Task web client. Only those languages defined on the Windows, Preferences, Oracle Design Studio dialog box appear as options. See "Defining OSM Preferences" for more information about defining languages for use in OSM.
Type	Displays the data element type. This field is read-only.
Max Length	Maximum number of units of length for a string element type. This field is read-only.
Minimum or **Maximum**	Displays the **Minimum** and **Maximum** field values as defined in the Data Dictionary. See "Task Data Node Properties View Dictionary Tab" for information about overriding this value.
Path	Displays an XPath expression to define the location of the node in the Data Dictionary. This field is read-only.
Namespace	Identifies the namespace in which this cartridge exists, and identifies cartridge version within the namespace, if applicable.

Related Topics

Defining Task Data

Working with Tasks

Task Editor Undo Tab

The Task editor **Undo** tab appears for IP Service Activator and ASAP activation tasks. (A new activation task is designated to be either an IP Service Activator or ASAP activation task when you choose the Activation System value on the Activation Task Wizard.)

Use the **Undo** tab to define part of your compensation strategy for activation tasks: to undo tasks that are affected by amendments. Complete the compensation strategy for activation tasks on the **Redo** tab. See "Task Editor Redo Tab" for more information.

Field	Use
Compensation Strategy (ASAP only)	Specify the compensation strategy to undo a task when it is affected by an amendment: • Select **Manual** if manual intervention is required at run time. • Select **Ignore** to instruct OSM to skip this task.
Compensation Strategy (IP Service Activator only)	Specify the compensation strategy to undo a task when it is affected by an amendment: • Select **Manual** if manual intervention is required at run time. • Select **Ignore** to instruct OSM to skip this task. • Select **Undo** to instruct OSM to cancel the original task, or to cancel another task.
Cancel original order (IP Service Activator only)	When **Compensation Strategy** is set to **Undo**, this option cancels the original order id.
Create a new order to undo (IP Service Activator only)	When **Compensation Strategy** is set to **Undo**, you can configure a new task to undo, by specifying the data node that contains the Activation order ID. The Activation order ID is configured on the **Activation Task Details** tab. See "Task Editor Activation Task Details Tab" for more information.

Related Topics

Modeling Activation Tasks

About Activation Tasks

8

Working with Order Lifecycle Policies

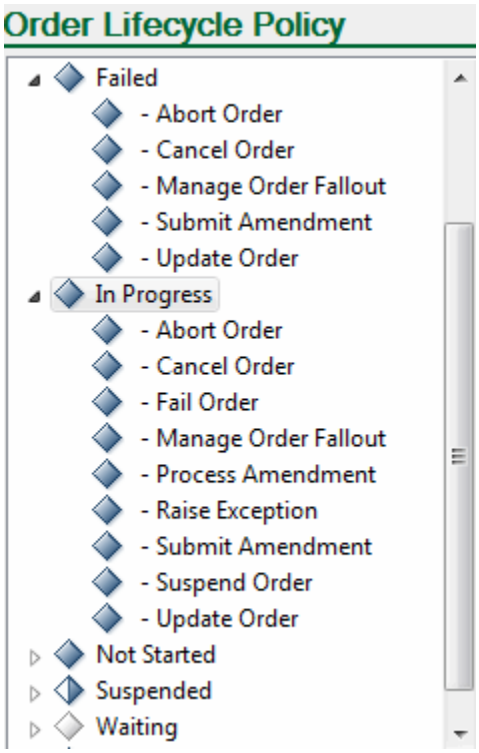
Every order you model within Design Studio must be associated with an order lifecycle policy. An order lifecycle policy controls which transactions a role can perform while the order is in a particular order state. For example, while an order is in the *In Progress* state, you might want your Customer Service role to perform the *Update Order*, *Cancel Order*, and *Suspend Order* transactions, while your Fallout role performs *Raise Exception*.

When working with order lifecycle policies, see the following topics:

About Order States and Transactions

An order's progress in an Oracle Communications Order and Service Management (OSM) run-time environment is tracked by its *state* at various stages of its life cycle. Transitions from one order state to another are achieved through transactions. Each order state is associated with a set of transactions that can be performed while the order is in that particular state. See *OSM Concepts* for information about order states and transactions.

Transactions are not enabled until roles are assigned to them. In the Order Lifecycle Policy editor, enabled transactions are represented by a fully-shaded diamond-shaped icon. Disabled (unassigned) transactions are represented with a diamond icon that contains no shading. If all transactions for a particular state are enabled, the state is represented with a fully-shaded diamond icon; partial-enabling is represented with a half shaded diamond. The following graphic demonstrates the use of differently shaded icons in the Order Lifecycle Policy editor left-column state and transaction menu tree, using the delivered default order lifecycle policy, which contains a minimum set of order state and transaction combinations assigned to all roles:



You can create a custom policy with no default transactions and no role assignments, and then model it using the Order Lifecycle Policy editor. Depending upon your business processes, you may configure one general policy that supports many different order types, or you may need to configure a unique policy for each order type.

Related Topics

Working with Order Lifecycle Policies

Working with Orders

Creating New Order Lifecycle Policies

You create new order lifecycle policies to control which transactions a role can perform while the order is in a particular order state.

To create an order lifecycle policy:

1. From the **Studio** menu, select **New**, select **Order and Service Management**, select **Order Management**, then select **Order Lifecycle Policy**.
2. In the **Project** field, select the OSM project in which to save this entity.
3. In the **Name** field, enter a name for the policy.

   The name must be unique among order lifecycle policy entity types in the same namespace.
4. (Optional) Select a location for the order lifecycle policy.

   By default, Design Studio saves the order lifecycle policy to your default workspace location. You can enter a folder name in the **Folder** field, or select a location different from the system-provided default. To select a different location:

a. Click the Folder field **Browse** button.

b. Navigate to the directory in which to save the entity.

c. Click **OK**.

5. Click **Next**.

6. (Optional) Create the policy with a custom configuration.

   You can create custom configuration using the **Create default order lifecycle policy for the selected roles** check box. Do one of the following:

   - To create the policy with no default transactions and no role assignment, deselect the check box.
   - To create the policy with the default set of transactions but modify the role assignment, leave the check box selected and move the selected roles to the available roles as appropriate.

7. Click **Finish**.

   The newly created policy is displayed under the selected project in the Studio Projects view.

Related Topics

Configuring Order Lifecycle Policies

Working with Order Lifecycle Policies

Working with Orders

Configuring Order Lifecycle Policies

You configure order lifecycle policies to control which transactions a role can perform while the order is in a particular order state.

To configure lifecycle order policies:

1. From the **Studio** menu, select **Show Design Perspective**.
2. Click the **Studio Projects** tab.
3. Double-click any order lifecycle policy entity.

   The Order Lifecycle Policy editor opens and displays the lifecycle policy.

4. (Optional) Create permissions for multiple state and transaction combinations.

   When creating permissions for multiple state and transaction combinations, do the following:

   a. Click **Grant Permission**.

      The Add Permissions to Transactions dialog box opens.

   b. Press and hold the Shift key to select multiple consecutive state and transaction pairs. Or, press and hold the Control key to select multiple non-consecutive pairs.

   c. In the Roles list, select the roles to which you want to add permissions.

   d. Click the arrow buttons to move the selected roles from the Available list to the Selected list.

   e. Click **OK**.

5. In the left-column state and transaction menu tree, expand an order state to see the related transactions.
6. Select a transaction in the left-column state and transaction menu tree.

   If you previously defined permissions for the transaction, the permitted roles are displayed in the Permissions tab Selected area. The transaction you select here is the transaction for which you want to define permissions. For example, select the Suspend Order transaction to permit a specific group of users to suspend orders.
7. Click the **Permissions** tab **Add** button.

   Design Studio adds a default display name to the Permissions area.
8. Select the default name and rename the permission, as appropriate.
9. Ensure that the new permission is actively selected.
10. In the Roles Available area, select the roles to which you will give transaction permissions.
11. Click the arrow keys to move the selected roles into the Selected area.
12. (Optional) With the new permission actively selected, click the **Add** button in the Condition area.

    Design Studio adds a condition with a default name **Condition** and a corresponding XPath expression with the default value **true()**.
13. (Optional) Modify the default XPath expression for the permission condition.

    The XPath expression must evaluate to true before the selected roles are permitted access to the transaction. See "Order Lifecycle Policy Permissions Tab" for more information.
14. (Optional) Click the **Transition Condition** tab.

    Use the **Transition Condition** tab to define the conditions that control whether the order can transition to the transaction. When adding transaction transition conditions:

    a. Click the **Transition Condition** tab **Add** button.

       Design Studio adds a condition with a default name **Condition** and a corresponding XPath expression with the default value **true()**.

    b. Modify the default XPath expression for the transition condition.

       If the condition for a particular transaction transition evaluates to false, then the transaction is disabled while the order is in the surrounding order state.

       See "Order Lifecycle Policy Transition Conditions Tab" for more information.
15. (Optional) Click the **Grace Period** tab.

    The **Grace Period** tab appears for the Suspend Order, Process Amendment, and Cancel Order transactions only. You can define grace periods by wait duration and by event frequency.
16. Click **Save**.

Order Lifecycle Policy Editor

Use the Order Lifecycle Policy editor to add permissions to order transactions. If you create a policy based on the default configuration, any roles you have defined within Design Studio are automatically preselected for the default transactions. You can add permissions to a group of transactions, or to a single transaction.

The following fields are common among multiple Order Lifecycle Policy editor subtabs.

Field	Use
Display Name	Edit the display name for the order lifecycle policy.
State and Transaction menu tree	This menu tree (at the left side of the Order Lifecycle Policy editor) contains a list of the order states and the transactions that can occur for each order state. Expand an order state folder to reveal the related transactions. Select a transaction in this column to configure permissions for the transaction. Click the **Add Permissions** button to configure permissions for multiple transactions simultaneously.

See the following topics when using the Order Lifecycle Policy editor:

- Order Lifecycle Policy Permissions Tab
- Order Lifecycle Policy Transition Conditions Tab
- Order Lifecycle Policy Editor Grace Periods Tab

Order Lifecycle Policy Permissions Tab

Use the Order Lifecycle Policy editor to add permissions to order transactions.

Field	Use
Permissions	Select a transaction from the left-column order state and transaction menu tree and click the **Permissions** field **Add** button to add a new permission for the selected transaction. Select any permission and click **Remove** to delete the permission from the list. Select the default name and rename the permission, as appropriate. See "Configuring Order Lifecycle Policies" for information about assigning roles to permissions.
Roles	Select a permission in the **Permissions** field to view the roles assigned to the permission. All of the roles defined in the workspace in the Available Column. To permit a role to perform the transaction associated with the permission, select a role and click the arrow keys to move roles into the Selected area. Click the **Create Role** button to create and add a new role. See "Creating New Roles" for more information.

Field	Use
Conditions	Define conditions for the permissions. Select a permission and click the **Add** button in the Condition area. Select the default name **condition** and the default XPath expression **true()** to modify the values. Select any condition and click **Remove** to delete the condition from the list. OSM evaluates the condition for a permission when the transaction is attempted. If it evaluates to true, the assigned roles are able to perform the transaction. If it evaluates to false, the assigned roles are unable to perform the transaction. **Note**: XPath uses path expressions to select data nodes in XML documents. A path expression with a single dot (.) represents the current node. Two dots (..) represents the parent of the current node. A slash (/) represents the root node. XPath and XQuery fields are limited to 4000 characters.

Related Topics

Configuring Order Lifecycle Policies

Working with Order Lifecycle Policies

Working with Orders

Order Lifecycle Policy Transition Conditions Tab

Use the Order Lifecycle Policy Transition tab to define conditions for transaction transitions.

Field	Use
Conditions	Define conditions for the transaction transition. Click the **Add** button in the Condition area to add a condition for the selected transaction in the left-column states and transactions menu tree. Select the default name **condition** and the default XPath expression **true()** to modify the values. Select any condition and click **Remove** to delete the condition from the list. OSM evaluates the condition when the order transitions to the selected transaction. If the condition evaluates to false, then the transaction is disabled while the order is in the surrounding order state.
Expression	When you add a condition, the default XPath expression **true()** is automatically added. **Note**: XPath uses path expressions to select data nodes in XML documents. A path expression with a single dot (.) represents the current node. Two dots (..) represents the parent of the current node. A slash (/) represents the root node. XPath and XQuery fields are limited to 4000 characters.
Message	Add a human readable message to display in error logs if the condition evaluates to false.

Field	Use
Display as	Add an error severity to associate with a condition that evaluates to false. Select one of the following: • VALID: • WARNING: • ERROR: • CRITICAL:

Related Topics

Configuring Order Lifecycle Policies

Working with Order Lifecycle Policies

Working with Orders

Transition Condition for Checking a Hard Point of No Return

The following XQuery can be used to check whether a hard point of no return has been reached, so that an amendment can be rejected if it is received after a hard point of no return. This XQuery checks to see whether there have been any revisions to significant data for order items that have reached a hard point of no return. Business considerations will determine what state/transition combinations will need to check for the point of no return, but at a minimum it should be defined in the **In Progress** state for the **Submit Amendment** transition.

To use this XQuery, follow the standard procedure for updating the lifecycle policy, creating a new transition condition and using the XQuery below in the Expression box for that condition. See "Configuring Order Lifecycle Policies" for more information about updating the lifecycle policy.

```
declare variable $PONR_NOT_YET := "NOT YET";

(: Checks for Hard Point Of No Return, return = true means no PONR
   has been reached. Raise an error if PONR has been reached. :)
declare function local:allowRevision(
    $taskData as element()) as xs:boolean {
    let $rootData := $taskData/_root
    let $changes := $taskData/RevisionPerspective/Changes
    return
        if (fn:exists($rootData) and fn:exists($changes))
        then (
            let $changedOrderItems as element()* :=
                    local:getChangedOrderItems($rootData, $changes)
            let $revisionOrderItemsPastHardPONR as xs:string* :=
                for $orderItem in $changedOrderItems
                    return local:getOrderItemsPastHardPONR($orderItem)
            return fn:not(fn:exists($revisionOrderItemsPastHardPONR)))
        else fn:true() };

declare function local:getChangedOrderItems(
    $root as element(),
    $changes as element()) as element()* {
    let $indices := local:getOrderItemIndicesForChecking($changes)
    let $distinctIndicies := fn:distinct-values($indices)
    for $index in $distinctIndicies
        return local:getOrderItem($root, $index) };

declare function local:getOrderItemIndicesForChecking(
```

```
    $changes as element()) as xs:string* {
    for $change in $changes/*[@significant = "true"]
        return local:getOrderItemIndex($change) };

declare function local:getOrderItemIndex(
    $changeNode as element()) as xs:string* {
    let $changeType := local-name($changeNode)
    let $tokens := fn:tokenize($changeNode/@path, "/")
    let $t1 := $tokens[position() = 2]
    let $t2 := $tokens[position() = 3]
    let $t3 := $tokens[position() = 4]
    let $t4 := $tokens[position() = 5]
    return
        if (fn:starts-with($t1, "ControlData")
            and fn:starts-with($t2, "Functions")) then
            (: /ControlData/Functions/*Function/orderItem/... :)
            local:getOrderItemIndexInFunction(
                fn:root($changeNode)/GetOrder.Response/_root,
                (: Functions/@index, if exists :)
                fn:substring-before(fn:substring-after($t2,"'"), "'"),
                (: e.g. SyncCustomerFunction/@index :)
                fn:substring-before(fn:substring-after($t3,"'"), "'"),
                (: e.g. orderItem/@index :)
                fn:substring-before(fn:substring-after($t4,"'"), "'"))
        else
                ""   };

declare function local:getOrderItemIndexInFunction(
    $root as element(),
    $functionsIndex as xs:string,
    $functionIndex as xs:string,
    $orderItemIndex as xs:string) as xs:string* {
    if (fn:boolean($functionsIndex)) then
        $root/ControlData/Functions[@index = $functionsIndex]/*[@index =
            $functionIndex]/orderItem[@index =
            $orderItemIndex]/orderItemRef/@referencedIndex
    else
        $root/ControlData/Functions/*[@index = $functionIndex]/orderItem[@index =
            $orderItemIndex]/orderItemRef/@referencedIndex    };

declare function local:getOrderItem(
    $root as element(),
    $orderItemIndex as xs:string) as element()* {
    $root/ControlData/OrderItem[@index = $orderItemIndex]    };

declare function local:getOrderItemsPastHardPONR(
    $orderItem as element()) as xs:string* {
    let $lineId as xs:string := local:getLineId($orderItem)
    let $pointOfNoReturn as xs:string := local:getPointOfNoReturn($orderItem)
    let $isHardPONRReached := if ($pointOfNoReturn = "HARD")
                              then true()
                              else false()
    return
        if ($isHardPONRReached)
        then $lineId
        else () };

declare function local:getLineId(
    $orderItem as element()) as xs:string {
    fn:normalize-space($orderItem/LineID/text()) };

declare function local:getPointOfNoReturn(
```

```
    $orderItem as element()) as xs:string {
    let $ponrData := fn:normalize-space($orderItem/PoNR/text())
    let $ponrCode :=
        if (fn:empty($ponrData))
        then $PONR_NOT_YET
        else (
            let $lastPonrValue :=
                fn:normalize-space($orderItem/PoNR[last()]/text())
            return
                (: We are looking for strings with either [xxxx]xxxx or
                   xxxx format. Return what is in the [] or the whole string
                   if no brackets. :)
                let $hard1 := fn:tokenize($lastPonrValue, "\[|\]")
                return fn:concat( $hard1[1] , $hard1[2] )
        )
    return
        $ponrCode   };

(: Detect false revision order. return = true means
   there are significant data changes in the revision order :)
declare function local:doSignificantChangesExist(
    $taskData as element()) as xs:boolean {
    let $dataChanges :=
        $taskData/RevisionPerspective/Changes/*[@significant='true']
    return
        if (fn:exists($dataChanges))
        then true()
        else false() };

(: Only do the complex calculation for a valid revision.:)
let $taskData := fn:root(.)/GetOrder.Response
let $isValidRevision := local:doSignificantChangesExist($taskData)
return if ($isValidRevision)
then
    local:allowRevision($taskData)
else
    fn:true()
```

Order Lifecycle Policy Editor Grace Periods Tab

Use the Order Lifecycle Policy Editor Grace Periods tab to specify a period of time that the system should wait before suspending, amending, or canceling an order.

A grace period specifies a period of time to wait for all accepted tasks to complete before transitioning an order. You can specify a grace period for the Suspend Order, Process Amendment, and Cancel Order transactions. Grace periods are defined by a wait duration and an event frequency.

Field	Use
Wait Duration	Select **Indefinitely** (the default setting) or specify a time frame using the minimum and maximum times that the system waits before forcing the transition.
Event Frequency	Specify the frequency at which the system should generate a jeopardy notification (defined as every hour, by default) while the wait duration remains unsatisfied.

Related Topics

Configuring Order Lifecycle Policies

9

Working with Data Providers

You use data providers in Oracle Communications Order and Service Management (OSM) in conjunction with Data Instance behaviors to augment order information by retrieving information from external systems. When modeling data providers, see the following topics:

- About Data Providers
- Creating New Data Providers
- Data Provider Editor

Related Topics

Working with Behaviors

About Data Providers

You use data providers in conjunction with Data Instance behaviors to augment order information by retrieving information from external systems. After you've defined a data provider, you can reuse or extend the configuration for multiple Data Instance behaviors.

For example, consider that you have a task that requires information that is not included in an order, such as a customer name and address. To obtain this information, you can define in the Task editor a Data Instance behavior called *Customer ID*. When you define the properties for the data instance, you can specify an existing data provider or create a new data provider that will describe the configuration necessary to retrieve the information from the external CRM system. If, to attach to the external CRM system, you know that you will need to include a host value and a password, you can use the Data Provider editor to add *host* and *password* as input parameters and define default values for these parameters, written as an XPath or XQuery expression.

Related Topics

Understanding Built-in Data Provider Types

Data Provider Editor

Data Provider Editor Settings Tab

Working with Data Providers

Defining Data Instance Behavior Properties

Understanding Built-in Data Provider Types

Design Studio provides several built-in data provider types intended to retrieve external XML instances from the following sources:

Data Provider	Description
Objectel	Use to invoke an Objectel server extension. The returned XML document is used as the external instance. This adapter provides a reliable transport call into Objectel. Although JMS is an asynchronous protocol, the adapter itself is not. While JMS simplifies transaction management, recovery, offline capabilities, and security, these benefits are not really of relevance when considered within the context of a Data Instance rule. The JMS adapter utilizes additional resources in the application server in the form of temporary JMS destinations to which Objectel sends the response. These can be expensive if an order has many adapters being called concurrently. Oracle does not recommend this adapter in this situation. Objectel is an inventory tracking application designed to assist telecommunication and network engineers with the documentation of the equipment used in providing data and voice communications, with the creation of facilities, and with the assignment of customer circuits.
Order	Use order data from any OSM order as an external instance.
Property File	Reads the data instance data values from a property file.
SOAP	Invoke SOAP web services using HTTP protocol and utilize the responses.
XML Attachment	Use an XML file that has been attached to any OSM order as an external instance.
XML File	Use an XML file accessible from any standard URL as an external instance. This built-in data provider is useful for integrating external XML data located in a file system, FTP site, from HTTP, or in a Java .jar file.
XML Validation	Use to validate any XML document using a schema. Both the document and the schema can be either elements or URLs.
JDBC	Query any JDBC database, then use the results within a behavior. This built-in data provider is useful for acquiring information stored in an external database.
Web Service	Use to invoke OSM Web Service operations GetOrder and FindOrder. This built-in data provider acts as a wrapper around the OSM Web Service API allowing these operations to be invoked from external instances.

For more information about the built-in data provider parameters, and examples, see *OSM Modeling Guide.*

Related Topics

About Data Providers

Working with Data Providers

Creating New Data Providers

You create data providers to use in conjunction with Data Instance behaviors to augment order information by retrieving information from external systems.

To create a new data provider:

1. From the **Studio** menu, select **New**, select **Order and Service Management**, select **Order Management**, then select **Data Provider**.
2. In the **Project** field, select the project in which to save this entity.
3. In the **Name** field, enter a name for the data provider.

The name must be unique among the data provider entity types in the same namespace.

4. (Optional) Select a location for the data provider.

   By default, Design Studio saves the data provider to your default workspace location. You can enter a folder name in the **Folder** field, or select a location different from the system-provided default. To select a different location:

   a. Click the Folder field **Browse** button.

   b. Navigate to the directory in which to save the entity.

   c. Click **OK**.

5. In the **Provider Type** field, select the provider type for the data provider.

   Design Studio provides a list of built-in data providers that you can configure to retrieve external XML instances. The SOAP provider type is the default setting.

6. Click **Finish**.

   Design Studio adds the data provider to the appropriate project in the Studio Projects view.

Related Topics

About Data Providers

Configuring Data Providers

You configure data providers to define the input and output parameters.

1. From the **Studio** menu, select **Show Design Perspective**.
2. Click the **Studio Projects** tab.
3. Double-click any data provider entity.

   The Data Provider editor opens and displays the data provider.

4. In the **Settings** tab, configure any values that you would like to change for your implementation.

   If you are configuring a custom data provider, you must enter a value in the **Provider Class** field.

5. Click the **Interface** tab.

   If you are using a built-in data provider type, the required parameters for your type have been included automatically in the **Parameter** field. Parameters with an asterisk after the name must be configured with values.

6. For each of the provided parameters, click on the parameter name and do the following:

   a. Select either **XPATH** or **XQUERY** in the **Default Value** drop-down list, depending on the format of the value you are going to provide.

   b. Enter the value of the parameter in the **Default Value** field.

   See *OSM Modeling Guide* for more information about the required parameters for each data provider type.

7. (Optional) Specify a value in the **Results Documents** field. If you do not provide an XML structure, the system will not display the parameters on the Data Instance Behavior Properties tab.

   If you do not provide an XML structure, the system will not display the parameters on the Data Instance Behavior Properties tab.

Related Topics

About Data Providers

About Data Instance Behaviors

Data Provider Editor

Use the Data Provider editor to configure the system settings and interface parameters necessary to retrieve information from external systems. You use data providers in conjunction with Data Instance behaviors to augment order information by retrieving information from external systems.

When configuring system settings and interface parameters in the Data Provider editor, see the following topics:

- Data Provider Editor Settings Tab
- Data Provider Editor Interface Tab

Data Provider Editor Settings Tab

Use the Data Provider editor Settings tab to configure the external system settings for data providers.

Field	Use
Provider Type	Select a built-in data provider or a custom data provider that you will create. You can switch between one provider type and another. If at least one parameter value already exists for the provider type you are changing, a warning message appears indicating that the parameters of the new provider type will replace the existing provider type parameters.
Provider Class	If you select *Custom* in the Provider Type field, you must provide a class name.
Scope	Specify how OSM should cache external data instances. Select one of the following cache levels from the **Scope** field: • *System (the default):* The system caches and reuses external data instances system-wide. Use this scope level if retrieving the external instance is expensive and performed frequently. The system reuses the cached instance results only if the actual resolved values of all parameters are identical and the lookup adapter class is the same. • *Node*: The system caches external data instances at the node level. This level of cache is specific to the user, session, and task. For example, the system retrieves any given external instance when a view node on an order is instantiated. The system reuses the external instance across all instances of the node regardless of how many instances of that view node exist in the order. Use this setting if it is moderately expensive to retrieve the external instance and the field referencing the external instance is a multi-instance node. The system only re-uses cached instances across multi-instance nodes if the actual resolved values of all parameters are identical and the lookup adapter class is the same. • *None*: The system retrieves external data instances for each instance of the field on the order and they are not cached.

Field	Use
Maximum Time, Maximum number cached	If you select **System** or **Node** in the **Scope** field, specify the following cache settings: • In **Maximum Time**, specify the maximum time (in milliseconds) for which a cached external instance is valid. For example, enter *5000* to define the <timeout> as 5 seconds. • In **Maximum number cached**, specify the maximum number of actual entries in the cache that is maintained at any one time for this defined external instance.

For information about building a custom Data Provider, see the *OSM Modeling Guide*.

Related Topics

Working with Data Providers

Data Provider Editor

Defining Data Instance Behavior Properties

Data Provider Editor Interface Tab

Use the Data Provider editor Interface tab to define the input parameters and default settings for the external system and specify the provider class and cache settings.

Field	Use
Parameters	When you create a new data provider, Design Studio displays all of the mandatory and optional parameters based on the selected provider type. Mandatory parameters are shown with an asterisk (*) to differentiate them from optional parameters. Click **Add** to add an input parameter, and select the new parameter to rename it. Input parameters specify named parameters whose values are used when retrieving an external instance. The value is determined at run time and is based on the XPath or XQuery expression you define in the **Default Value** field.
Default Value	Define the content of the associated parameter element as an XPath to a node or as an XQuery expression. • XPath supports functions in expressions and provides for a core library of functions dealing with strings, numbers, Booleans, and node sets. In addition to the core XPath functions defined by the XPath standard, a number of extended functions are supported with OSM. These extended functions provide additional functionality that is useful to create behaviors but does not conform to the XPath standard. For more information about XPath functions, see *OSM Developer's Guide*. • XQuery enables the use of sophisticated expressions and XML transformations. XQuery syntax is backwards compatible with XPath 1.0 and contains additional syntax elements. You can use XQuery in situations where a more expressive language or transformation abilities are needed.
Result Document	(Optional) Specify the structure of the XML document. Though this field is optional, if you do not provide an XML structure, the system will not display the parameters on the Data Instance Behavior Properties tab.

> **Note:**
>
> XPath and XQuery fields are limited to 4000 characters.

Related Topics

Working with Data Providers

Data Provider Editor

Defining Data Instance Behavior Properties

10

Working with Orders

When you create an Oracle Communications Order and Service Management (OSM) project, an order entity is automatically generated and placed in your project directory. You can create additional order entities using the Order wizard.

You model various aspects of the order using the tabs in the Order editor; for example, the order data, behaviors, rules, properties, and permissions. Every order you create must also be associated with an order lifecycle policy, which you configure using the Order Lifecycle Policy editor. See "Working with Order Lifecycle Policies" for more information.

When modeling orders, see the following topics:

- About Order Extensions and Inheritance
- About Reference Nodes
- Creating New Orders
- Defining Order Data
- Defining Order Behaviors
- Defining Order Details
- Enabling Order Amendment Processing
- Defining Order Details
- Defining Order Fallout
- Defining Order Data Changed Notifications
- Assigning Order Permissions
- Defining Order Jeopardy Notifications
- Defining Order Event Notifications
- Order Editor

Related Topics

Modeling Data

About Order Extensions and Inheritance

During order creation, you can base new orders on the functionality of an existing order by using the extend feature. When you extend an order, the extended order inherits all of the data, tasks, rules, and behaviors of the parent order. For example, if you have multiple order types that all require the same subset of processes and tasks, you can create a base order that contains this data, then extend from this order to create as many new orders as necessary. You can add new data and behaviors to each of the new orders to create unique order templates and behavior functionality. To implement changes to the inherited data, you edit the data in the parent order and Design Studio automatically implements those changes among all of the extended orders.

You cannot edit order data inherited from a parent order. For example, if you are working in an order that includes data inherited from a parent order, you cannot remove, rename, or reposition data elements inherited from the parent order, make changes to inherited behaviors, and so forth.

The child order does not inherit any configuration details specified in the parent order **Details**, **Amendable**, **Notifications**, **Permissions**, **Jeopardies**, **Events**, or **Composite Data View** tabs. You must manually set these configuration details for each child order.

> **Note:**
>
> Design Studio does not permit cyclic referencing. For example, if order **O2** extends from order **O1**, and order **O3** extends from order **O2**, then you cannot extend order **O1** from order **O3**.

About Reference Nodes

A reference node is a data node that is created by referencing another data node within the order template. The reference data node has the same data typing and structure of the node that it is referencing. However, the reference data node is a distinct instance of the data structure that it references.

Reference data nodes enable you to create information once and reuse it in multiple locations in your data model. A reference node points back to a single data node location and ensures that you can efficiently manage and update a node when it is used in multiple locations.

> **Note:**
>
> This feature is not available in releases prior to OSM 7.0.

For example, imagine that you create a data structure called **customer** that includes all of the information required for a customer profile: the data element **customerName**, the structure **address**, the element **phoneNumber**, and so forth. Another data structure, called **devices**, contains a list of devices, and each device requires the customer profile information. Rather than remodeling the customer profile data for each device, you can create a reference node to the **customer** structure. If the customer information changes (for example, they require a new type of address), you are not required to change the information at every instance where the customer profile information is referenced, but only once in the **customer** structure.

You must set up reference nodes at order creation time as part of coding the automation plug-ins that call the CreateOrderBySpecification web service operation. For an example of how you set up reference nodes when you create an order using the CreateOrderBySpecification web service operation, see the discussion on setting up reference nodes in *OSM Developer's Guide*.

You must create a reference node association in the order template. See "Adding Reference Data Nodes" for information about adding a reference node to an order template.

Creating New Orders

You create orders to configure the data and properties of incoming orders.

To create orders:

1. From the **Studio** menu, select **New**, select **Order and Service Management**, select **Order Management**, then select **Order**.
2. In the **Project** field, select the OSM project in which to save this entity.
3. (Optional) In the **Extends** field, select an existing order to leverage the order data and extend the functionality of that existing order.

   Click **Select** and select an order for the **Extends** field. If a suitable order does not yet exist, click **New** to create the order. When finished, click **OK**. Your selection populates the corresponding **Extends** field in the Order wizard. See "About Order Extensions and Inheritance" for more information.
4. In the **Name** field, enter a name for the order.

   The name must be unique among order entity types in the same namespace.
5. (Optional) Select a location for the order.

   By default, Design Studio saves the order to your default workspace location. You can enter a folder name in the **Folder** field or select a location different from the default. To select a different location:

   a. Click the **Folder** field **Browse** button.

   b. Navigate to the directory in which to save the entity.

   c. Click **OK**.
6. Click **Finish**.

   Design Studio creates the order entity and saves it to the selected project in the Studio Projects view.

Related Topics

Working with Orders

Defining Order Data

The data that you define for an order is available to the tasks included in the process associated with the order. When defining order data, see the following topics:

- Adding New Data to an Order
- Adding Existing Data to an Order
- Adding a New Data Structure Definition to an Order
- Adding an Existing Data Structure Definition to an Order
- Adding Reference Data Nodes
- Renaming Data Elements at the Order Level
- About Modeling Data in the Order Template

Related Topics

Order Editor

Adding New Data to an Order

You can create new data in the Data Dictionary and add it to the order.

To create new data:

1. From the **Studio** menu, select **Show Design Perspective**.
2. Click the **Studio Projects** tab.
3. Double-click an existing order.

   The Order editor opens with the **Order Template** tab active. See "Order Editor Order Template Tab" for more information.
4. Right-click inside the Order editor **Order Template** tab and select **Open Data Element** view.

   The Data Element view opens.
5. Right-click inside the Data Element view and select **Add Simple Schema Element** or **Add Structured Schema Element**.

   The Create Data Schema Element dialog box or the Create Data Schema Structure dialog box is displayed.
6. Complete the form and click **Finish**.
7. Drag the new data from the Data Element view into the Order editor **Order Template** tab.

> **Tip:**
>
> Press and hold the Shift key to select multiple consecutive elements. Press and hold the Control key to select multiple non-consecutive elements.

Related Topics

Order Editor

Adding Existing Data to an Order

You select data previously created in the data dictionary to add to an order.

To add data you have previously created:

1. From the **Studio** menu, select **Show Design Perspective**.
2. Click the **Studio Projects** tab.
3. Double-click an existing order.

   The Order editor opens with the **Order Template** tab active. See "Order Editor Order Template Tab" for more information.
4. Right-click inside the Order editor **Order Template** tab and select **Select from Data Dictionary**.

   The Select Data Elements dialog box is displayed.

> **Note:**
>
> You can alternatively select **Open Data Element view** and then drag and drop data elements from the Data Dictionary onto the **Order Template** tab.

5. Select the data you want to add to the order.

> **Tips:**
>
> When selecting data to add to the order template:
>
> - Press and hold the Shift key to select multiple consecutive elements. Press and hold the Control key to select multiple non-consecutive elements.
> - Select a parent node to add all data elements (simple and structured data elements) in its hierarchy.
> - Select a child node to add only the child node and its parent nodes. Design Studio automatically adds parent nodes associated to the child node up to the root of the data schema.

6. Click **OK**.

   Design Studio adds the data to the Order editor **Order Template** tab.

7. Click **Save**.

Related Topics

Order Editor

About Modeling Data in the Order Template

Adding Reference Data Nodes

When modeling data in the order template, you can add reference nodes. A reference data node is a data node that is created by referencing another data node within the order template. The reference data node has the same data typing and structure of the node that it is referencing. However, the reference data node is a distinct instance of the data structure that it references.

For example, **OrderItemRef** is a reference data node for an order component that references the **orderItem** data structure in the order template. The **OrderItemRef** reference data node is a distinct instance of **orderItem** in the order component, but shares the structure of **orderItem** - a data node that is already defined and contributed by the Order Item Specification.

> **Note:**
>
> This feature is not available in releases prior to OSM 7.0.

To add a reference node:

1. From the **Studio** menu, select **Show Design Perspective**.

2. Click the **Studio Projects** tab.
3. Double-click an existing order.

   The Order editor opens with the **Order Template** tab active. See "Order Editor Order Template Tab" for more information.
4. Add a reference node to the root-level of the order template or to a data structure by doing one of the following:

   a. To add a reference node to the root-level of the order template, in the Order editor **Order Template** tab, right-click *in the tab area* and select **Add Reference Node**.

   b. To add a reference node to a data structure, in the Order editor **Order Template** tab, right-click *on the data structure* and select **Add Reference Node**.

      The Reference Node Creation dialog box is displayed.
5. Select the data node to which the reference node will point.

   At run time, the reference node will obtain its value from the data node that you select.
6. Click **OK**.
7. Click **Save**.

You must set up reference nodes at order creation time in addition to creating the reference node association in the order template; otherwise, the reference node will be empty at run time. For an example of how to set up reference nodes when you create an order using the CreateOrderBySpecification web service operation, see the discussion on setting up reference nodes in *OSM Developer's Guide*.

Related Topics

About Reference Nodes

Order Editor

Adding a New Data Structure Definition to an Order

You can create a data structure definition and add it to an order.

To create a data structure definition:

1. From the **Studio** menu, select **Show Design Perspective**.
2. Click the **Studio Projects** tab.
3. Double-click the existing order to which you want to add a data structure definition.

   The Order editor opens with the **Order Template** tab active. See "Order Editor Order Template Tab" for more information.
4. In the Order editor **Order Template** tab, right-click on the data element, select **Select Data Structure Definition** and click **New**.

   The Data Structure Definition wizard is displayed.
5. Enter a name for the data structure definition.
6. Click **Finish**.
7. Click **Save**.

Related Topics

About Modeling Data in the Order Template

Order Editor

Adding an Existing Data Structure Definition to an Order

You can add an existing data structure definition to an order.

To add a data structure definition to an order:

1. From the **Studio** menu, select **Show Design Perspective**.
2. Click the **Studio Projects** tab.
3. Double-click the existing order to which you want to add a data structure definition.

   The Order editor opens with the **Order Template** tab active. See "Order Editor Order Template Tab" for more information.
4. In the Order editor **Order Template** tab, right-click on the data element, select **Select Data Structure Definition**.
5. In Matching items, select a data structure definition.

> **Note:**
>
> If no data structure definitions are displayed in the Matching items area, you must define the dependency of the data structure definition to the model project before you add it to the order. For more information, see "Managing Project Dependencies".

6. Click **OK**.

   The data structure definition is added to the order, and all child data elements and structures of the data structure definition are also added and displayed.

> **Note:**
>
> Derived data structure definitions are not displayed in the order template.

7. Click **Save**.

Related Topics

About Modeling Data in the Order Template

Order Editor

Renaming Data Elements at the Order Level

You rename a data element at the order level by providing an alias for the data element in the Order editor **Order Template** tab. When you rename data elements at the order level, Design Studio automatically updates that data element name in all associated tasks and extended orders. However, the data element instance in the Data Dictionary is not affected by the change.

For example, consider that you have a data model 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 six-digit number). To avoid data type collisions in the run-time environment, you can rename one instance of the **EmployeeID** data element at the order level.

To rename data elements at the order level:

1. From the **Studio** menu, select **Show Design Perspective**.
2. Click the **Studio Projects** tab.
3. Double-click the existing order that contains the data element to rename.

   The Order editor opens with the **Order Template** tab active. See "Order Editor Order Template Tab" for more information.
4. In the Order editor **Order Template** tab, right-click on the data element, select **Refactoring**, and then select **Rename**.

   The Rename Order Template Node dialog box is displayed.

   The dialog box displays the current data element node name and the data element node name as defined in the Data Dictionary. The rename that you make here does not affect the node name at the Data Dictionary level.
5. In the **Name** field, enter the new name for the data element.
6. (Optional) Click the **Preview** button.

   The Rename Order Template Node dialog box shows all instances of the data element defined in related tasks and extended orders that will change after you rename it at the order level.

   If the rename is not allowed, a problem error is displayed. You can check the error log for information on why the rename failed.

   Click **Continue** to proceed.
7. Click **OK**.

   Design Studio implements the change immediately in the project.

Related Topics

Order Editor

Defining Order Behaviors

Behaviors provide a way to extend the functionality and appearance of order data. Each behavior type performs an action; for example, calculating or validating data or displaying fields in read-only or read-write modes. When you define a behavior at the order level, the behavior applies to all manual tasks in the order model.

To define a behavior at the order level:

1. From the **Studio** menu, select **Show Design Perspective**.
2. Click the **Studio Projects** tab.
3. Double-click an existing order.

   The Order editor opens with the **Order Template** tab active. See "Order Editor Order Template Tab" for more information.
4. In the Order editor **Order Template** tab area, select the data node upon which to model the behavior.

5. Right-click in the Behaviors area and select **Add Behavior**.

   Each behavior type enables you to dynamically control a specific aspect of your order data model.

6. Select a behavior type from the list.

   > **Note:**
   >
   > You cannot define Calculation, Event, and Lookup behaviors for structured data elements, because structured data elements do not represent actual data.

   Design Studio adds the behavior to the Behaviors area.

7. In the Behaviors area, click the new behavior.

   The Behaviors Properties view opens, which includes a set of properties that you must define for the corresponding behavior type. See "Working with Behaviors" for more information about defining behavior properties.

8. (Optional) In the Order editor, click the **Behaviors** tab.

   Use the Order editor **Behaviors** tab to quickly view all of the behaviors defined for the data nodes in an order. See "Order Editor Behaviors Tab" for more information.

Related Topics

Order Editor

Defining Order Details

Order details define the process, order lifecycle policy, and creation task associated with the order type. The details also include an execution priority and whether the order type inherits from and extends another order type.

To define order details:

1. From the **Studio** menu, select **Show Design Perspective**.
2. Click the **Studio Projects** tab.
3. Double-click an existing order.

   The Order editor opens with the **Order Template** tab active. See "Order Editor Order Template Tab" for more information.

4. Click the Order editor **Details** tab.

   See "Order Editor Details Tab" for more information about the fields on this tab.

5. In the **Extends** field, determine whether to inherit order attributes from another order.

   Orders can inherit data from other orders, which enables you to leverage order data when building new, similar orders. See "About Order Extensions and Inheritance" for more information.

6. In the **Subject** field, select an order subject.

   The order subject can be used to filter the orders in the OSM web clients.

7. In the **Lifecycle Policy** field, select a lifecycle policy to control which order state and transaction combinations a role can perform for this order type.

See "Working with Order Lifecycle Policies" for more information.

8. In the **Default Process** field, select the process to which this order is submitted. See "Working with Processes" for more information.
9. In the **Creation Task** field, select the task that creates and submits the order before the workflow begins. The creation task defines the data that is required to be present when the order is created.
10. In the **Priority Range** field, specify a minimum and maximum priority for the order to process within.

    For example, if you specify a range of 5 to 7 and the order is created with a priority of less than 5, the priority value will be rounded up to 5. If the order is created with a priority of more than 7, the priority value will be rounded down to 7.
11. Click **Save**.

Related Topics

Order Editor

Enabling Order Amendment Processing

To enable OSM to amend in-flight orders, you must configure the order to allow amendment processing. By default, orders are not amendable.

To enable order amendment processing:

1. From the **Studio** menu, select **Show Design Perspective**.
2. Double-click any order in the Solution view or Studio Projects view.

   The order opens in the Order editor.
3. Click the **Amendable** tab.
4. Select the **Amendable** option.
5. Click the **Add** button in the Key area.

   Design Studio adds a key with a default name **key** and a corresponding XPath expression with the default value **true()**.
6. In the **Expression** field, specify an order key as an XPath to a node that will uniquely match an amended order to its corresponding OSM order.

   For example, you might specify a customer reference ID as an XPath. You can select a data node from the Data Element view and drag the selected data node into the **XPath Expressions** field to define the XPath expression. See "Order Editor Amendable Tab " for more information about defining order keys.
7. Select one or more events to be published at run time for this order type.

   If you select no events, the system publishes no events. See "Order Editor Amendable Tab " for more information.
8. Specify the version as an XPath to a node that will return a numeric value representing the version of an amended order.

   Amendments with higher versions are considered to be more recent than amendments with lower versions. If there are multiple queued amended orders for the same original order, OSM processes only the most recent amendment version.
9. Click **Save**.

Related Topics

Working with Event Notifications

Working with Orders

Defining Order Rules

You define rules for orders to evaluate the order contents. Rules are used in process flow decisions, conditional transitions, subprocess logic, delay activities, jeopardies, and events and enable you to evaluate against the content of an order by comparing data node to data node or data node to a fixed value. When you compare data to data, you compare the contents of two data nodes (of the same type); for example, you might compare a due date with a payment date, based on some condition. When you compare data to a value, you compare a data node to a fixed value.

When you first create an order, the system automatically assigns to the order a system-based **null_rule** which always evaluates to true. This default configuration ensures that the order will be submitted to a process. You cannot remove the **null_rule** or modify its definition; however, you can define any number of your own custom rules.

To define rules for orders:

1. From the **Studio** menu, select **Show Design Perspective**.
2. Click the **Studio Projects** tab.
3. Double-click an order.

   The order displays in the Order editor.
4. Click the **Rules** tab.

   See "Order Editor Rules Tab" for more information about the fields on this tab.
5. In the Rules area, click the corresponding **Add** button.

   The Add Rule dialog box is displayed.
6. In the **Name** field, enter a name for the new rule.

   The name must be unique among rule entity types in the same namespace.
7. Click **OK**.

   The new rule is displayed in the Name column. You can select the rule entity in the Name column at any time to edit the rule name.
8. In the Name column, select the new rule entity.
9. In the **Definition** tab, click the **Add** button.

   The Order Template Selection dialog box is displayed.
10. Select the node against which the rule will evaluate.

    You can select one node for the rule or select multiple nodes to create multiple rules.

    You can right-click a data structure definition node to specify a derived complex type.
11. Click **OK**.
12. Select the node you just added.
13. Click the **Properties** button.

The Properties view **Rules Expressions** tab is displayed, where you can define values for the fields in the remaining steps. See "Properties View Rules Expression Tab" for more information.

14. In the **Data** field, enter the XPath expression to identify the location of the data node.

    You can also select a data node from the Data Element view and drag the selected data node into the **Data** field to define the XPath expression. To drag a data node into the Properties view **Rules Expressions** tab, press and hold the Alt key before you select and drag the data node to the field.

    Additionally, you can click the corresponding **Select** button to select another data node.

15. In the **Operator** field, select an operator from the list.

    The options available in the **Operator** field depend on the data type used in the **Data** field.

16. In the **Data/Value** field, enter an XPath expression or enter a fixed value.

    You can select a data node from the **Order Template** tab and drag the selected data node into the **Data/Value** field to define the XPath expression. To drag a data node into the Properties view **Rules Expressions** tab, press and hold the Alt key before you select and drag the data node to the field.

    Additionally, you can click the corresponding **Select** button to select another data node.

17. In the Order editor **Definition** tab, click **Add** to add another expression to the rule.

    Each condition is separated by either **And** or **Or** (**And** is the default).

Related Topics

Order Editor

Modeling Data

Defining Order Fallout

Fallout refers to orders that encounter problems during fulfillment and therefore fall out of normal processing. OSM places these orders in Failed state (you can also manually fail orders in the Order Management web client).

When defining order fallouts, see the following topics:

- Associating Order Fallouts with Data Nodes
- Associating Order Fallouts with Fallout Groups

Associating Order Fallouts with Data Nodes

In Design Studio, you associate a fallout name with one or multiple data nodes whose values you will want to review (in the Customer Management web client) when the corresponding type of fallout occurs.

To associate order fallouts with data nodes:

1. From the **Studio** menu, select **Show Design Perspective**.
2. Click the **Studio Projects** tab.
3. Double-click an order.

   The order displays in the Order editor.

4. Click the **Fallout** tab.

   See "Order Editor Fallouts Tab" for more information about the fields on this tab.

5. In the Name area, click the corresponding **Add** button.

   The Add Fallout dialog box is displayed.

6. In the **Name** field, enter a name for the new order fallout.

   The name must be unique among fallout types in the same namespace.

7. Click **OK**.

   The new order fallout is displayed in the Name column. You can select the value in the Name column at any time to edit the name.

8. Select the new order fallout in the Name column.

9. Enter a display name for the order fallout.

   You can associate the display name that appears in the Task web client with a specific language by using the optional language attribute. Only those languages defined appear as options. See "Defining Language Preferences" for information about defining languages for use in OSM.

10. In the Nodes area, click the **Add** button.

    The Order Template Node Selection dialog box is displayed.

11. Select one or multiple data nodes whose values you will want to review (in the Order Management web client) when this fallout occurs.

    You can right-click a data structure definition node to specify a derived complex type.

12. (Optional) In the Nodes area, click the **Remove** button to delete the association with the data node.

13. Click **Save**.

Related Topics

Associating Order Fallouts with Fallout Groups

Order Editor

Associating Order Fallouts with Fallout Groups

You can group similar types of fallouts into groups, enabling you to review multiple fallouts together in the Order Management web client when the corresponding types of fallout occur.

To associate order fallouts with fallout groups:

1. From the **Studio** menu, select **Show Design Perspective**.

2. Click the **Studio Projects** tab.

3. Double-click an order.

   The order displays in the Order editor.

4. Click the **Fallout Groups** tab.

   See "Order Editor Fallout Groups Tab" for more information about the fields on this tab.

5. In the Name area, click the corresponding **Add** button.

   The Add Fallout Group dialog box is displayed.

6. In the **Name** field, enter a name for the new fallout group.

   The name must be unique among fallout group types in the same namespace.

7. Click **OK**.

   The new fallout group is displayed in the Name column. You can select the value in the Name column at any time to edit the name.

8. In the Name column, select the new fallout group.

9. Enter a display name for the fallout group.

   You can associate the display name that appears in the Task web client with a specific language by using the optional language attribute. Only those languages defined appear as options. See "Defining Language Preferences" for information about defining languages for use in OSM.

10. In the Fallouts area, click the **Add** button.

    The Select Fallouts dialog box is displayed.

11. Select one or multiple fallouts to group together.

12. (Optional) In the Fallouts area, click the **Remove** button to delete the association with the data node.

13. Click **Save**.

Related Topics

Associating Order Fallouts with Data Nodes

Order Editor

Defining Order Data Changed Notifications

You define order data changed notifications to update external systems with status updates when a specific data node in the order data is updated with a new value. Data change notifications are triggered by changes to order data.

See "Creating Order Data Changed Notifications" for information about creating data change notifications at the order level.

> **Note:**
>
> This feature is not available in releases prior to OSM 7.0.

Related Topics

Working with Orders

Working with Jeopardy and Event Notifications

Assigning Order Permissions

When you assign permissions to orders, you define how specified roles can search for orders in the Task web client, which fields of data they can see, whether the roles can add additional

columns of data to their Worklist, Notification, and Query pages, and whether they can create orders of the associated type.

To assign order permissions:

1. From the **Studio** menu, select **Show Design Perspective**.
2. Click the **Studio Projects** tab.
3. Double-click any order entity.

   The order displays in the Order editor.
4. Click the **Permissions** tab.

   See "Order Editor Permissions Tab" for more information about the fields on this tab.
5. Do one of the following:
   - To select from existing roles, click **Select**.
   - To create a new role, click **New**.

   See "Creating New Roles" for more information.
6. (Optional) To view permissions for existing roles, select the role and click **Open**.

   The system displays the role in the Role editor, where you can view the permissions assigned to the role. You assign permissions to a role to give the users in that role access to related functions in the Task web client. See "Role Editor Role Tab" for more information.
7. (Optional) Select a role and click **Remove** if you want to delete an associated role from the task.
8. (Optional) Click the **Details** tab.

   See "Order Editor Permissions Details Tab" for more information about the fields on this tab. On the **Details** tab, you can:
   - Enable the associated role to create this order type in the Task web client by selecting **Create Orders**.
   - Define a set of flexible headers for Task web client users.

     Flexible headers are additional columns of data that Task web client users can add (through the preferences settings) to their Worklist, Notification, and Query pages. Click **Add** to select the data nodes that represent the flexible header columns. After adding a data node, select the name or description to edit those values.

     > **Note:**
     >
     > If you change the flexible headers and re-deploy the cartridges while users are logged in to the OSM web clients, users may have to log out and log back in to see the changes.
9. (Optional) Limit the orders a role can view.

   To limit the orders a role can view:

   a. Click the **Filters** tab.

      See "Order Editor Permissions Filters Tab" for more information about the fields on this tab.

   b. Click **Add**.

The Order Template Node Selection dialog box is displayed.

c. Select the data node on which to define the condition that limits the orders the role can view.

> **Note:**
>
> If you apply a filter to a multi-instance data element, the filter will always be evaluated based on the first instance of the data. It is not possible to specify another instance of the data element to use.

d. Click the **Properties** button.

The **Filter Expressions** tab is displayed, where you can define values for the fields on the **Filters** tab. See "Properties View Filter Expression Tab" for more information.

e. In the **Data** field, enter the XPath expression to identify the location of the data node.

You can also select a data node from the Data Element view and drag the selected data node into the **Data** field to define the XPath expression. To drag a data node into the Filter Expressions tab, press and hold the Alt key before you select and drag the data node to the field. Click the corresponding **Select** button to select a different data node.

f. In the **Operator** field, select an operator from the list.

The options available in the **Operator** field depend on the data type used in the **Data** field.

g. Select either the **Data** option button or the **Value** option button to specify the expression that the condition evaluates against.

h. In the Order editor **Filters** tab, click **Add** to add another expression to the rule.

Each condition is separated by either **And** or **Or** (**And** is the default).

10. Click the **Query Task** tab.

Use the **Query Task** tab to select the task that will generate the query view used by Task web client users.

When selecting query tasks:

- Click **New** to create a new query task and add the task to the order.
- Click **Add** to add an existing task to the order.
- Select any task and click **Open** to review the task in the Task editor.

You can associate multiple query tasks with an order, and define each task as the Summary view, the Detail view, or the Default view. See "Order Editor Permissions Query Tasks Tab" for more information about the fields on this tab.

11. Click **Save**.

Related Topics

Order Editor Permissions Details Tab

Order Editor Permissions Filters Tab

Working with Orders

Defining Order Jeopardy Notifications

You define order jeopardy notifications when you want to alert users or systems that an order may be at risk. Jeopardy notifications are based on rules that you configure in Design Studio and which the OSM server evaluates at regular intervals. A jeopardy notification can be sent to a user group or may be consumed by an automation plug-in.

See "Creating Jeopardy Notifications in the Task or Order Editor" for information about defining jeopardy notifications at the order level.

Related Topics

Working with Orders

Working with Jeopardy and Event Notifications

Defining Order Event Notifications

You define order event notifications to generate a milestone-based event that works with automation plug-ins. You select the order milestone that triggers the automation and then configure the automation plug-in that will perform the work.

See "Creating Order Milestone and Task State Automation Event Notifications" for information about creating order event notifications.

Related Topics

Working with Orders

Working with Jeopardy and Event Notifications

Order Editor

Use the Order editor to model order attributes, such as the order data, behaviors, rules, properties, and permissions. Every order you create must also be associated with an order lifecycle policy. See "Working with Order Lifecycle Policies" for more information.

When working with the Order editor, see the following topics:

- Order Editor Order Template Tab
- Order Editor Behaviors Tab
- Order Editor Details Tab
- Order Editor Amendable Tab
- Order Editor Rules Tab
- Order Editor Fallouts Tab
- Order Editor Fallout Groups Tab
- Order Editor Notification Tab
- Order Editor Permissions Tab
- Order Editor Jeopardy Tab
- Order Editor Events Tab

- Order Editor Composite Data View Tab

Order Editor Order Template Tab

Use the Order editor **Order Template** tab to model all of the data necessary to provision the order. You can drag data from the Dictionary view into the Order Template area, or right-click in the Order Template area to select data from the Data Dictionary dialog box. The following table describes the fields on the Order Editor Order Template tab.

Field	Use
Order Template	Contains all of the data necessary to fulfill or provision an order. This area represents a template from which you can select the data nodes that tasks require during the fulfillment or provisioning process. To hide all data elements related to control data (the reserved ControlData area that OSM uses for executing orchestration), deselect the **Show Control Data** check box. See "About the Order Template Context Menu" for descriptions of other actions you can perform in the Order Template context menu.
Behaviors	Displays all of the behaviors defined for each data node in the order template. Select a data node in the Order Template area to view the behaviors defined by the node, or to create new behaviors. When defining behaviors at the order level, you can use the Order editor **Order Template** tab to create the behavior, the **Behavior Properties** tabs to refine the behavior information, and the Order editor **Behaviors** tab to quickly view all of the behaviors defined for a task. See "Defining Order Behaviors" for more information.

When modeling order data, see the following topics for additional information:

- Properties View Order Data Tab
- Properties View Dictionary Tab
- Properties View Key Tab
- Properties View Usage Tab

Related Topics

Defining Order Data

Properties View Order Data Tab

Use the Properties view **Order Data** tab to access and edit the information defined for the corresponding data element at the order template level. You can right-click any attribute in the Order editor **Order Template** field and select **Open Properties View** to open the Properties view **Order Data** tab. The following table describes the fields on the Properties view **Order Data** tab.

Field	Use
Name	The system displays the name of the node as defined in the Data Dictionary. The name of the node is not available for edit on this tab. You can edit the value in the **Display Name** field in the Data Schema editor **Details** subtab to edit the manner in which the element displays in the Task web client.
Path	The system displays an XPath expression to define the location of the node in the Data Dictionary.

Field	Use
Contributing Template	Displays the parent structure when the selected structure has been extended from a base structure. During order creation, you can base new orders on the functionality of an existing order by using the extend feature. When you extend an order, the extended order inherits all of the data, tasks, rules, and behaviors of the parent order. You can add new data and behaviors to each of the new orders to create unique order templates and behavior functionality. To implement changes to the inherited data, you edit the data in the parent order, and Design Studio automatically implements those changes among all of the extended orders.See "About Order Extensions and Inheritance" for more information. Additionally, the contributing template can reflect that the data node was contributed by an order component.
Data Dictionary	The system displays the name of the data schema (within the Data Dictionary) in which the node is defined.
XML Type	Select to signify that the structure is an XML data type. Structures defined as XML data types in the Data Dictionary can contain XML documents. **Note**: Before you use XML data types, copy all relevant schema (XSD) files into the cartridge project. Use the Java perspective Package Explorer view to copy the schema files into the **dataDictionary** folder. **Note:** This feature is not available in releases prior to OSM 7.0. See the Eclipse *Java Development User Guide* for more information about the Java perspective.
Significance	By default, a node inherits significance from its parent. At the order level, you can define the significance as **Not Significant** if you do not want to use the node during amendment processing. During amendment processing, the OSM system compensates only for order instances that use significant data elements as inputs. If an element is not specified as significant, the system updates the order only with the changed data (no compensation is required). Data significance is supported at the Data Dictionary, the order template, and the task levels.
Ignore rollback during undo	There may be data on an order or task that you want to exclude from rollback in cases when the order or task is running in undo mode. For example, you may want to retain data related to status messages when a task rolls back during a fallout scenario where the status messages may contain important troubleshooting information. If you set this value for a structure, all child structures and elements also ignore rollback during the undo execution mode. You typically set this value for data generated during order processing, for example, with external fulfillment state updates, external processing states updates, or status nodes for external response messages. Oracle recommends that you not set this value for data from the upstream system, for example, from the original order. Oracle recommends that when you set this value, you also set the significance value to **Not Significant**. **Note**: Do not set this value for ControlData fulfillment state and processing state elements because OSM calculates these elements based on data received from external system where **Ignore rollback during undo** is appropriate to set.

Related Topics

Defining Order Data

Order Editor Order Template Tab

Properties View Dictionary Tab

Use the Properties view **Dictionary** tab to access and edit the information defined for the corresponding data element at the order template level. You can right-click any attribute in the Order Template area and select **Open Properties View** to access and edit the information defined for the corresponding data element at the order template level. The following table describes the fields on the Properties view **Dictionary** tab.

Field	Use
Name	The system displays the name of the node as defined in the Data Dictionary. The name of the node is not available for edit. You can edit the value in the **Display Name** field in the Data Schema editor **Details** subtab to edit the manner in which the element displays in the Task web client.
Display Name	You can associate the display name with a specific language by using the optional language attribute. Only those languages defined appear as options. See "Defining OSM Preferences" for information about defining languages for use in OSM.
Type	Displays the data element type. This field is read only.
Max Length	Specify the maximum number of units of length for a string element type. You must define the maximum length with a non-negative integer.
Minimum	Select the number of times the global element referenced can appear in an instance document. Select **0** if you want the element to be optional.
Maximum	Select the maximum number of times the global element referenced can appear. Select **unbounded** to indicate there is no maximum number of occurrences.
Path	The system displays an XPath expression to define the location of the node in the Data Dictionary.
Namespace	Identifies the namespace in which this cartridge exists, and identifies the cartridge version within the namespace, if applicable.

Related Topics

Defining Order Data

Order Editor Order Template Tab

Properties View Key Tab

Use the Properties view **Key** tab to access and edit the key information defined for the corresponding data element at the order template level. You can right-click any attribute in the Order Template area and select **Open Properties View** to access and edit the information defined for the corresponding data element at the order template level. The following table describes the fields on the Properties view **Key** tab.

Field	Use
Key XPath Expression	If this node is a multi-instance data node, you can specify one or more order data keys to uniquely match the data instance from an revision order to a data instance on the current order data. The order data key of a node must be an XPath that points to data within its scope. If the node is a group node, the XPath expression must point to its children nodes; if the node is a value node, it can only point to itself. If no keys are defined, OSM uses the relative position of the changed data when comparing the revision order data with the current order data. You can select a data element from the Order editor **Order Template** tab and drag the selected data node into the **XPath Expression** field to define the XPath expression. **Note:** XPath uses path expressions to select data nodes in XML documents. A path expression with a single dot (.) represents the current node. Two dots (..) represents the parent of the current node. A slash (/) represents the root node. XPath and XQuery fields are limited to 4000 characters.

Related Topics

Defining Order Data

Order Editor Order Template Tab

Properties View Usage Tab

Use the Properties View Usage tab to view in which tasks and cartridges the corresponding data element is defined. You can right-click any attribute in the Order Template area and select **Open Properties View** to access and edit the information defined for the corresponding data element at the order template level.

Select any row in the table and click **Open** to open the task in the appropriate editor.

Related Topics

Defining Order Data

Order Editor Order Template Tab

Order Editor Behaviors Tab

Click the Order editor **Behaviors** tab to quickly view all of the behaviors defined for the data nodes in an order. The Behaviors table displays the name and type of the behavior, whether the behavior is enabled, the inheritance properties, the path name of the data node on which the behavior is defined, and the order where the behavior was originally defined.

The information on the Order editor **Behaviors** tab is read only. To change the information that appears on this tab, select a behavior from the table and click the **Properties** button to access the **Behaviors Properties** tabs. See "Working with Behaviors" for more information about defining behavior properties.

Related Topics

Defining Order Behaviors

Working with Orders

Order Editor Details Tab

Use the Order editor **Details** tab to define the order attributes that you use to associate the order with other entities, enabling the order to process correctly in the OSM run-time environment. The following table describes the fields on the Order editor **Details** tab.

> **Note:**
>
> The options in the **Amendable** tab are disabled if you select **TMF Order** in the **Details** tab.

Field	Use
Extends	You can select an existing or create a new order to extend this order (the order's data is inherited) by clicking the **Select** button. To create a new order that this order would be an extension of, click **New**. After you have selected or created an order, click **Open** to access the Order editor. Click **Clear** (red X) to clear the selected value from the field. Order data extensibility enables you to leverage order data when building new, similar orders.
Lifecycle Policy	Select an existing or create a new lifecycle policy to control which order state/transaction combinations a role can perform for this order type. Every order you create within Design Studio must be associated with an order lifecycle policy.

Field	Use
Default Process	Select an existing or create a new process to which the order is submitted. When the selected default process is an orchestration process, Design Studio looks for the Data Dictionary project **OracleComms_OSM_CommonDataDictionary**, which contains definitions of common OSM structures such as control data, base order item data elements, base function data elements, and so on. If the Data Dictionary does not exist, you will be prompted to import it. After the Data Dictionary is imported, Design Studio automatically attaches base control data such as **ControlData/Functions** and **ControlData/OrderItem** from the imported Data Dictionary to the order.
Order Item	Identifies whether an order item specification is associated with the order through the following relationship: Order > Orchestration Process > Orchestration Sequence > Order Item Specification. If no process (or a provisioning process) is associated with the order, **None** is displayed. If an orchestration process is associated with the order but there is no association with an order item through the relationship path, **No Order Item Configured** is displayed.
Creation Task	Select an existing or create a new task to create and submit the order before the workflow begins. The creation task defines which subset of data is required to create the order. When at the creation task, an order has not been submitted to a process and has had no work completed. The creation task has two associated states, **submit** and **cancel**. Additionally, you can define statuses for the creation task on the Task editor **States/Statuses** tab. You need a creation task for any order creation (manual, automated, etc.). If you want to enable behaviors when creating an order, select manual tasks as creation tasks for an order. If the order associated with the creation task is defined as amendable (on the Order editor **Amendable** tab), do not include optional fields in the creation task as this can cause unexpected results. When including optional fields in the creation task, the original order is submitted with all optional fields left empty. The optional fields are later populated during task execution. When a revision order is submitted with the optional fields now populated, the system treats the optional fields on the revision as different instances of the fields from the ones populated on the original order and OSM triggers compensation. **Note:** You can automate order creation using the XML API or the web service interface. See *OSM Developer's Guide* for more information.
Order Source	Enter an order source for the order if you would like it to be different from the order name. In the order structure, there are separate fields for order source and order type. If you leave this field blank, both fields on the order will default to the order name when the cartridge is built. If you enter a value in this field, it will be used for the order source field, and the order name will continue to be used for the order type.
Order Source Description	Enter a description of the order source if desired. If this value is not entered, it will be defaulted to the order name when the cartridge is built. This description will be displayed in the Task web client.
Priority Range	Specify a minimum and maximum priority for the order to process within. For example, if you specify a range of 5-7 and the order is created with a priority of less than 5, the priority value will be rounded up to 5. If the order is created with a priority of more than 7, the priority value will be rounded down to 7.
Realizes	If this order is a concrete implementation of a Functional Area from the PSR model, click **Select** to select the Functional Area. If a Functional Area has been defined for the order, you can click **Open** to open the Functional Area. This association can also be defined in the Functional Area. See "About Functional Areas" for more information.
TMF Order	Select to specify that this order supports TMF.
Hosted Specification	Select a hosted specification that you want to use.

Related Topics

Defining Order Details

Working with Orders

Order Editor Amendable Tab

Use the Order editor **Amendable** tab to configure the order to allow amendment processing.

> **Note:**
>
> The options in the **Amendable** tab are disabled for TMF orders. That is, if you select **TMF Order** in the **Details** tab.

The following table describes the options on the Order editor **Amendable** tab.

Options	Use
Not Amendable	Select to indicate that there can be no amendment processing against this order.
Amendable	Select to allow amendment processing against this order.
Key	Specify an order key as an XPath to a node that will uniquely match an amended order to its corresponding OSM order. For example, you might specify a customer reference ID as an XPath using the following expression: **_root/Cust_Ref_ID** Alternatively, you can select a data node from the Data Element view and drag the selected data node into the **XPath Expressions** field to define the XPath expression. To drag a data node into the **XPath Expressions** field, press and hold the Alt key before you select and drag the data node to the field. **Note:** XPath uses path expressions to select data nodes in XML documents. A path expression with a single dot (.) represents the current node. Two dots (..) represents the parent of the current node. A slash (/) represents the root node. XPath and XQuery fields are limited to 4000 characters.
Version	Specify the version as an XPath to a node that will return a numeric value representing the version of an amended order. Amendments with higher versions are considered to be more recent than amendments with lower versions. If there are multiple queued amended orders for the same original order, OSM processes only the most recent amendment version. You can select and drag a data node from the Data Element view into the **XPath Expressions** field to define the XPath expression. To drag a data node into the **XPath Expressions** field, press and hold the Alt key before you select and drag the data node to the field. **Note:** XPath uses path expressions to select data nodes in XML documents. A path expression with a single dot (.) represents the current node. Two dots (..) represents the parent of the current node. A slash (/) represents the root node. XPath and XQuery fields are limited to 4000 characters.
Events	Select one or more events to be published at run time for this order type. OSM events are sent to the JMS destination **OrderStateChange.Event** queue and are published as topics. External systems can subscribe to this queue and retrieve the published events.
Amendment Abandoned	Select to publish this event when multiple amendments have been sent to OSM and an amendment has rendered an earlier version of amendment unnecessary. If an amendment is in progress, OSM puts any subsequent amendments in a queue for processing. If multiple amendments have been sent to OSM, the server processes the next amendment in the queue by selecting the highest version (optionally defined in the **Version** field) or the amendment with the most recent timestamp if no version has been defined. When multiple amendments are queued, the OSM server processes only the most recent amendment.
Amendment Completed	Select to publish this event when amendment processing has completed.

Options	Use
Amendment Started	Select to publish this event when amendment processing begins for any revision order.
Amendment Queued	Select to publish this event when amendment processing for a revision order is queued.
Amendment Terminating	Select to publish this event when amendment processing for a revision order is in the process of getting terminated.
Amendment Terminated	Select to publish this event when amendment processing for a revision order is terminated.
State Change	Select to publish this event when the order transitions from one state to another.
Order Created	Select to publish this event when the order is created in OSM.
Order Removed	Select to publish this event when the order has been deleted.

Order Editor Rules Tab

Use the Order editor **Rules** tab to create rule definitions at the order level. When modeling rules, see "Properties View Rules Expression Tab" for more information. The following table describes the fields on the Order editor **Rules Definition** tab. The Order editor **Rules Comments** tab and the Order editor **Rules Notes** tab are blank fields.

Field	Use
Condition	When defining multiple rule expressions, each rule expression is separated by an **And** or an **Or**. **And** is the default value, and indicates that both the expression before and the expression after **And** must evaluate to true if the rule is to evaluate to true. Use **Or** to indicate that either the expression before or the expression after **Or** can evaluate to true if the rule is to evaluate to true.
Data	Enter the XPath expression to identify the location of the data node. You can also select a data node from the Data Element view and drag the selected data node into the **Data** field to define the XPath expression. To drag the selected data node into the **Data** field, press and hold the Alt key before you select and drag the data node to the field. **Note:** XPath uses path expressions to select data nodes in XML documents. A path expression with a single dot (.) represents the current node. Two dots (..) represents the parent of the current node. A slash (/) represents the root node. XPath and XQuery fields are limited to 4000 characters.
Operator	Select an operator from the list. **Note**: When selecting an operator from the **Operator** field of the **Order** editor **Rules** tab, all possible operators are displayed, whether or not they are valid. To ensure only valid operators are displayed, choose an operator from the **Properties** view **Rules Expression** tab. The options available in the **Properties** view **Rules Expression** tab in the **Operator** field depend on the data type used in the **Data** field. See "Properties View Rules Expression Tab" for more information.
Data/Value	Enter an XPath expression or enter a fixed value. You can select a data node from the **Order Template** tab and drag the selected data node into the **Data/Value** field to define the XPath expression. To drag the selected data node into the **Data/Value** field, press and hold the Alt key before you select and drag the data node to the field. **Note:** XPath uses path expressions to select data nodes in XML documents. A path expression with a single dot (.) represents the current node. Two dots (..) represents the parent of the current node. A slash (/) represents the root node. XPath and XQuery fields are limited to 4000 characters.

Properties View Rules Expression Tab

Use the Properties view **Rules Expression** tab to define rule expressions. To access the **Rules Expression** tab, select a rule attribute on the Order editor **Rules** tab **Definition** tab and click **Properties**.

The fields on the **Rules Expression** tab are identical to those on the **Definition** tab. However, the options that are available for the **Value** field and the **Operator** lookup list on the **Rules Expression** tab change depending on the element type. For example, if you select an element that is a lookup type, the values that you defined in the Data Dictionary for this element appear as available options in the list. If you define a datetime element, the options available enable you to define a system datetime or a calendar datetime.

Related Topics

Defining Order Details

Working with Orders

Order Editor Fallouts Tab

Use the Order editor **Fallouts** tab to create new order fallouts. You associate data nodes with the fallout and review the values for those data nodes in the Order Management web client when the corresponding fallout occurs for an order. The following table describes the fields on the Order editor **Fallouts** tab.

Field	Use
Display Name	You can associate the fallout display name at the order template level to a specific language by using the optional language attribute. Only those languages defined on the Oracle Design Studio dialog box appear as options. See "Defining OSM Preferences" for information about defining languages for use in OSM.
Name	Click **Add** to open the Add Fallout dialog box, where you can create a new fallout category to associate with the order. Select any fallout category defined in the Name column and click **Rename** to specify a different fallout name, or click **Remove** to delete the fallout category from the list.
Nodes	Associate the data nodes whose values you will want to review (in the Order Management web client) when this fallout occurs. Click the corresponding **Add** button to open the Order Template Node Selection dialog box, where you can select one or multiple data nodes to associate with the fallout. Select any data node and click **Remove** to delete the node from the list.

Related Topics

Defining Order Fallout

Order Editor Fallout Groups Tab

Order Editor Fallout Groups Tab

Use the Order editor **Fallout Groups** tab to link similar types of fallouts together. You associate data nodes with the fallout and review the values for those data nodes in the Order Management web client when the corresponding fallout occurs for an order. The following table describes the fields on the Order editor **Fallout Groups** tab.

Field	Use
Display Name	You can associate the fallout group display name at the order template level to a specific language by using the optional language attribute. Only those languages defined on the Oracle Design Studio dialog box appear as options. See "Defining OSM Preferences" for information about defining languages for use in OSM.
Name	Click **Add** to open the Add Fallout Group dialog box, where you can create a new fallout group to associate with the order. Select any fallout group defined in the Name column and click **Rename** to specify a different fallout group name, or click **Remove** to delete the fallout group from the list.
Fallouts	Associate the fallout groups whose values you will want to review (in the Order Management web client) when this fallout occurs. Click the corresponding **Add** button to open the Order Select Fallouts dialog box, where you can select one or multiple data fallouts to associate with the fallout group. Select any fallout and click **Remove** to delete the fallout from the list.

Related Topics

Defining Order Fallout

Order Editor Fallouts Tab

Order Editor Notification Tab

Use the Order editor **Notification** tab to create order data changed notifications. Order data changed notifications are triggered by changes to order data.

> **Note:**
>
> This feature is not available in releases prior to OSM 7.0.

When modeling order data changed notifications, see the following topics:

- Order Editor Notification Details Tab
- Order Editor Notification Notify Roles Tab
- Order Editor Notification Data Changed Tab
- Order Editor Notification Automation Tab
- Order Editor Notification Notes Tab

The above tabs apply to existing event notifications in the list. Until you add an event notification (by clicking **Add**), they are not accessible.

Order Editor Notification Details Tab

Use the Order editor **Notification Details** tab to name the notification, set the priority level, enable or disable the notification, and specify whether to send the notification by email. The following table describes the fields on the Order editor **Notification** tab.

> **Note:**
>
> This feature is not available in releases prior to OSM 7.0.

Field	Use
Name	Enter a name to identify the notification.
Priority	Enter a priority from 1 to 255 (1 is the highest priority). The notification with the highest priority is evaluated first.
Enabled	Select to enable this notification, or deselect the option if you intend to implement the notification at a later time.
Email	Select to send email notifications to all users in the workgroup associated with the specified role. When you assign users to a workgroup in the OSM Administration area of the Order Management web client, you can set up OSM to notify users by email when a notification occurs with the notification ID number. See *OSM Order Management Web Client User's Guide* for information about configuring email notification properties for user roles. **Note:** Order-data-changed notifications are intended to update external systems with status updates when a specific data node in the order data is updated with a new value. The OSM server does not send order-data-changed event notifications to Task web client Notifications pages. When notifying users, the server sends these notifications to email addresses only.

Related Topics

Creating Order Data Changed Notifications

Working with Event Notifications

Working with Orders

Order Editor Notification Notify Roles Tab

Use the Order editor **Notification Notify Roles** tab to select the roles to be notified when the notification occurs.

> **Note:**
>
> This feature is not available in releases prior to OSM 7.0.

Select a predefined notification from the list in the Available column to activate a list of available roles. See "Working with Roles" for information about defining roles. Using the directional arrow buttons, move the roles (those groups to whom you want the notification sent) into the Selected column.

If the notification is sent to an external system via an automation plug-in, ensure that you include the role whose credentials are used when running the automation plug-in. See "Working with Automated Tasks" for more information.

Related Topics

Creating Order Data Changed Notifications

Working with Jeopardy Notifications

Working with Orders

Order Editor Notification Data Changed Tab

Use the Order editor **Notification Data Changed** tab to identify the data node for which changes to the value triggers the data change notification. All of the data nodes visible in the order template (defined on the Order editor **Order Template** tab) are available as options.

Click **Add** to open the Order Template Node Selection dialog box, where you can select the data node. Select any node defined in the Nodes column and click **Remove** to delete the node from the list.

> **Note:**
>
> This feature is not available in releases prior to OSM 7.0.

Related Topics

Creating Order Data Changed Notifications

Working with Event Notifications

Working with Orders

Order Editor Notification Automation Tab

Use the **Notification Automation** tab to configure an automation plug-in that performs the work or sends data to an external system when the notification is triggered. The following table describes the fields on the Order editor **Notification Automation** tab.

> **Note:**
>
> This feature is not available in releases prior to OSM 7.0.

Field	Use
Name	Enter a name for the automation plug-in.
Automation Type	Select the automation plug-in type from the available list. Click **OK** to add the automation entry to the Automation table.
View	Click an automation, and click **Select** in the **View** field to choose a query task to use with the automation. You must define an OSM user in the automation plug-in **Run As** field to run the automation plug-in and configure one or more roles and default query tasks using the Order editor **Permissions** tab. Associate the roles with the OSM user using the OSM Administration area of the Order Management web client. If the **Run As** OSM user has more than one role, each with a different default query task, then multiple query task views are available to run the automation plug-in. You can select a query task to allow OSM to predictably use one query view to run the automation plug-in. If the query task view has already been selected, click **Open** to view the query task. To create a new query task, click **New** to start the New Studio Entity wizard. **Note:** You must configure the **View** field if you are creating an OSM 7.2 (or later) cartridge. You can deploy older cartridges with the OSM 7.2 (or later) server, but random selection of query task views may occur if an OSM user has more than one role, each with a different default query task.

> **Note:**
>
> See "Configuring Automation Plug-In Properties" for information about defining automation properties on the **Properties** tab.

Related Topics

Creating Order Data Changed Notifications

Working with Event Notifications

Working with Orders

Order Editor Notification Notes Tab

Use the Order editor Notification **Notes** tab to denote the intended use of the notification or any additional information that you want to append to the notification data.

> **Note:**
>
> This feature is not available in releases prior to OSM 7.0.

Related Topics

Creating Order Data Changed Notifications

Working with Event Notifications

Working with Orders

Order Editor Permissions Tab

Use the Order editor **Permissions** tab to assign roles to the order and to customize the role settings. The following table describes the fields on the Order editor **Permissions** tab.

Field	Use
Roles	Add the roles that will have access to this order type in the Task web client. Click **Select** to select from existing roles or **New** to create a new role. To view permissions for existing roles, select the role and click **Open**. The system displays the role in the Role editor, where you can view the permissions assigned to the role. Select a role and click **Remove** if you want to delete an associated role from the task.

When modeling order permissions, see the following topics for more information:

- Order Editor Permissions Details Tab
- Order Editor Permissions Filters Tab
- Order Editor Permissions Query Tasks Tab
- Properties View Filter Expression Tab

Related Topics

Working with Roles

Order Editor Permissions Tab

Assigning Order Permissions

Order Editor Permissions Details Tab

You use the Order editor **Permissions Details** tab to enable roles to create orders of this type and to define the flexible headers available to Task web client users. The following table describes the fields on the Order editor **Permissions Details** tab.

Field	Use
Create Orders	Select to enable the associated role to create this order type.
Flexible Header	Flexible headers are additional columns of data that Task web client users can add (through the preferences settings) to their Worklist, Notification, and Query view lists. You define which data nodes the users can add in the **Flexible Header** field. The roles associated with the order can add these data nodes to their view lists so that they can view the data without having to access the corresponding editor. Click the **Add** button to access a list of data elements defined in the order template. The **Description** name you enter appears in the column header of the Task web client Worklist, Notifications, and Query views. **Note:** If you change the flexible headers and re-deploy the cartridges while users are logged in to the OSM web clients, users may have to log out and log back in to see the changes.

> **Note:**
>
> Flexible headers are displayed as lookup lists or as range fields. If flexible headers are enumerated by the designer, they are displayed as lookup lists. If they are not enumerated, they are displayed as range fields.
>
> When entering data into range fields, either enter data only in the From field or enter data in both the **From** and **To** fields. Filling only the **From** field queries only that exact data; filling both the **From** and **To** fields queries the range entered.
>
> See "Enumerations Tab" and "Settings Tab" for more information about creating lookup lists and range fields.

Related Topics

Assigning Order Permissions

Order Editor Permissions Tab

Order Editor Permissions Filters Tab

Use the Order editor **Permissions Filters** tab to limit the orders a role can view.

Click **Add** to open the Order Template Node Selection dialog box, where you can select the data node to filter on. To remove a filter, select any filter node defined in the Filters table and click **Remove** to delete the filter node from the list.

To specify filter values for a filter node, select it and click **Properties**. Modify values in the Filter Expression editor to configure your filter. For example, if you want a role to view orders only from Paris, select the data element **city** from the order template, and click **Properties**. In the Filter Expression editor, select the **=** operator, and enter **Paris** in the **Value** field. You can also

filter using And/Or combinations. For example, if you want a role to view orders from either Paris or London, add another similar line separated by **Or** and specify **London**. The following table describes the fields on the Order editor **Permissions Filters** tab.

Field	Use
Condition	When defining multiple rule expressions, each rule expression is separated by an **And** or an **Or**. **Or** is the default value, and indicates that either the expression before or the expression after **Or** can evaluate to true if the rule is to evaluate to true. Use **And** to indicate that both the expression before and the expression after **And** must evaluate to true if the rule is to evaluate to true.
Data	Enter the XPath expression to identify the location of the data node. You can also select a data node from the Data Element view and drag the selected data node into the **Data** field to define the XPath expression. To drag the selected data node into the **Data** field, press and hold the Alt key before you select and drag the data node to the field. **Note:** XPath uses path expressions to select data nodes in XML documents. A path expression with a single dot (.) represents the current node. Two dots (..) represents the parent of the current node. A slash (/) represents the root node. XPath and XQuery fields are limited to 4000 characters.
Operator	Select an operator from the list. The options available in the **Operator** field depend on the data type used in the **Data** field.
Data/Value	Enter an XPath expression or enter a fixed value. You can select a data node from the **Order Template** tab and drag the selected data node into the **Data/Value** field to define the XPath expression. To drag the selected data node into the **Data/Value** fields, press and hold the Alt key before you select and drag the data node to the field. **Note:** XPath uses path expressions to select data nodes in XML documents. A path expression with a single dot (.) represents the current node. Two dots (..) represents the parent of the current node. A slash (/) represents the root node. XPath and XQuery fields are limited to 4000 characters.

Related Topics

Assigning Order Permissions

Order Editor Permissions Tab

Order Editor Permissions Query Tasks Tab

Use the Order editor **Permissions Query Tasks** tab to select the task that will generate the query view used by Task web client users. You can select any manual, automated, or activation task already defined, or create a new task specifically for the run-time query. The following table describes the fields on the Order editor **Permissions Query** tab.

Field	Use
Name	Select the task that will generate the query view used by Task web client users. At run time, the OSM server returns a specific set of data when you use the search query functionality in the Task web client. You determine which data set the OSM server returns by creating or selecting a query task. The data associated with the task that you select here will be the data returned to you from the run-time query. You can select a task that you use elsewhere in processes, or you can create a task that is used only for run-time queries. You can associate multiple query tasks with each order and define each task as the Summary view, the Detail view, or the Default view.

Field	Use
Summary	Select to display the corresponding task data set in the Order Management web client **Summary** tab. The **Summary** tab provides a selection of the most important information about the selected order, component, or item and appears when you open the Order Details page. You can include data from multiple query tasks in the **Summary** tab. The Order Management web client displays on the **Summary** tab all of the data from all of the tasks for which you specify the Summary option.
Details	Select to display the task data set in the Order Management web client **Data** tab. The tasks for which you select this option appear as choices in the Order Management web client **Data** tab **View** field. You can specify that multiple tasks appear as options in the **View** field; each option will present the web client user with a different view, each containing a specific set of data.
Default	Select to specify that the OSM server displays this task data set when returning search queries in the Task web client. Select this option when configuring query tasks for cartridges intended for OSM 6.3.1 environments. You can select only one query task as the default option. **Note**: To see an attachment that is created in a previous task, you must have a role that has a query task with the **Default** option selected.

Related Topics

Assigning Order Permissions

Order Editor Permissions Tab

Properties View Filter Expression Tab

Use the Order editor Properties view **Filter Expression** tab to define rule expressions. To access the **Filter Expression** tab, select a conditional expression on the Order editor **Permissions Filter** tab and click **Properties**.

The fields on the **Filter Expression** tab are identical to those on the Order editor **Permissions Filters** tab. When you select values on this tab, they appear in the Order editor **Permissions Filter** tab. See "Order Editor Permissions Filters Tab" for more information.

Related Topics

Assigning Order Permissions

Order Editor Permissions Tab

Order Editor Jeopardy Tab

Use the Order editor **Jeopardy** tab to create jeopardy notifications when certain conditions arise in an order and you want to alert users or systems of processes, orders, or tasks that may be at risk.

When modeling jeopardy notifications, see the following topics:

- Order Editor Jeopardy Details Tab
- Order Editor Jeopardy Conditions Tab
- Order Editor Jeopardy Notify Roles Tab
- Order Editor Jeopardy Polling Tab
- Order Editor Jeopardy Automation Tab
- Order Editor Jeopardy Notes Tab

Order Editor Jeopardy Details Tab

Use the Order editor **Jeopardy Details** tab to name the jeopardy, select the notification rule, set the priority level, enable or disable the notification, and specify whether to send the notification by email. The following table describes the fields on the Order editor **Jeopardy Details** tab.

Field	Use
Name	Enter a name to identify the jeopardy.
Rule	Select the rule the system should evaluate before generating this jeopardy. This field defaults to the system-based **null_rule**. If you do not change the default value, OSM will always trigger this notification at the specified polling interval. See "Defining Order Rules" for more information about setting up new rules.
Priority	Enter a priority from 1 to 255 (1 is the highest priority). The notification with the highest priority is evaluated first.
Enabled	Select to enable this jeopardy notification, or deselect the option if you intend to implement the notification at a later time.
Email	Select to send email notifications to all users in the workgroup associated with the specified role. By default, notifications appear in the Notifications page of the Task web client. However, you can specify that notifications be sent by email by selecting the **Email** check box. When you assign users to a workgroup in the OSM Administration area of the Order Management web client, you can set up OSM to notify users by email. When a notification occurs, the system sends a notification ID number through email. See *OSM Order Management Web Client User's Guide* for information about configuring email notification properties for user roles. See *OSM Installation Guide* for information about configuring the outgoing email server.

Related Topics

Creating Jeopardy Notifications in the Task or Order Editor

Working with Jeopardy Notifications

Working with Orders

Order Editor Jeopardy Conditions Tab

Use the Order editor **Jeopardy Conditions** tab to select the conditions under which the jeopardy should be raised. For example, you can raise a jeopardy when this order exceeds the expected or a given duration or when the order is received within a certain number of days. The following table describes the fields on the Order editor **Jeopardy Conditions** tab.

Field	Use
Order State	Select the state that the order must be in before the jeopardy notification is triggered: **In Progress** or **Completed**.
Raise a Jeopardy when the order is received within	This field is available only when you have selected **In Progress** in the **Order State** field. For orders that are in progress, you can raise a jeopardy if the order has been received and has exceeded the time interval defined in the adjacent field.
Raise a Jeopardy when the order is completed within	This field is available only when you have selected **Completed** in the **Order State** field. For orders that are completed, you can raise a jeopardy if the order has been completed and has exceeded the time interval defined in the adjacent field.

Field	Use
Raise a Jeopardy when Process Duration Exceeds	After selecting this field, select either **Expected Duration** or **Given Duration** to raise a jeopardy if the process to which the order is associated has exceeded the expected duration of the order (defined on the Order editor **Details** tab) or given duration, specified by the time interval defined in the adjacent field.

Order Editor Jeopardy Notify Roles Tab

Use the Order editor **Jeopardy Notify Roles** tab to select the roles to be notified when the jeopardy occurs.

Select a predefined jeopardy from the list in the left column to activate a list of available roles. See "Working with Roles" for information about defining roles. Using the directional arrow buttons, move the roles (those groups to whom you want the notification sent) from the Available column into the Selected column.

If the jeopardy notification is sent to an external system via an automation plug-in, ensure that you include the role whose credentials are used when running the automation plug-in. See "Configuring Automation Plug-In Properties" for more information.

Related Topics

Creating Jeopardy Notifications in the Task or Order Editor

Working with Jeopardy Notifications

Working with Orders

Order Editor Jeopardy Polling Tab

Use the Order editor **Jeopardy Polling** tab to select the interval at which the OSM server evaluates the condition that triggers the jeopardy notification. You can define the polling so that the system checks for the condition only once, or you can define the polling at hourly, daily, weekly, or monthly intervals. The following table describes the fields on the Order editor **Jeopardy Polling** tab.

Field	Use
Interval	Select the interval at which the OSM server evaluates the condition that triggers the jeopardy notification. Select **Once** if you want the system to check for the condition only once when the order is received. When you select **Once**, the system disregards the **Next Start** field. Use the **Hours**, **Days**, and **Months** fields to define a specific interval at which the OSM server evaluates the condition that triggers the jeopardy notification. For example, if you want the system to check for the condition every two days, select the **Day(s)** option and, from the list, select **2**.
Next Start	Select the date and time that you want the notification to begin checking. You can specify a date for any polling interval. The system uses the current date and time as the default value.

Related Topics

Creating Jeopardy Notifications in the Task or Order Editor

Working with Jeopardy Notifications

Working with Orders

Order Editor Jeopardy Automation Tab

Use the Order editor **Jeopardy Automation** tab to configure an automation plug-in that performs the work or sends data to an external system when the jeopardy notification is triggered. OSM supports one automation plug-in per jeopardy.

You can also modify the properties of automation plug-ins. See "Configuring Automation Plug-In Properties" for more information about defining automation properties on the **Properties** tab.

The following table describes the fields on the Order editor **Jeopardy Automation** tab.

Field	Use
Name	Enter a name for the automation entry.
Automation Type	Select the automation plug-in type from the available list. Click **OK** to add the automation entry to the Jeopardy Automation table.
View	Click an automation, and click **Select** in the **View** field to choose a query task to use with the automation. You must define an OSM user in the automation plug-in **Run As** Property field to run the automation plug-in and configure one or more roles and default query tasks using the Order editor **Permissions** tab. Associate the roles with the OSM user using the OSM Administration area of the Order Management web client. If the **Run As** OSM user has more than one role, each with a different default query task, then multiple query task views are available to run the automation plug-in. You can select a query task to allow OSM to predictably use one query view to run the automation plug-in. If the query task view has already been selected, click **Open** to view the query task. To create a new query task, click **New** to start the New Studio Entity wizard. See "Order Editor Permissions Query Tasks Tab" for more information about query tasks. **Note:** You must configure the **View** field if you are creating an OSM 7.2 (or later) cartridge. You can deploy older cartridges with the OSM 7.2 (or later) server, but random selection of query task views may occur if an OSM user has more than one role, each with a different default query task.

Related Topics

Creating Jeopardy Notifications in the Task or Order Editor

Working with Jeopardy Notifications

Working with Orders

Order Editor Jeopardy Notes Tab

Use the Order editor **Notes** tab to denote the intended use of the notification or any additional information that you want to append to the jeopardy data.

Related Topics

Creating Jeopardy Notifications in the Task or Order Editor

Working with Jeopardy Notifications

Working with Orders

Order Editor Events Tab

Use the Order editor **Events** tab to create order milestone event notifications. You select the order milestone that triggers the automation and then configure the automation plug-in that will perform the work.

You can configure the properties of automations. See "Configuring Automation Plug-In Properties" for more information.

The following table describes the fields on the Order editor **Events** tab.

Field	Use
Milestone	The Milestone column displays the milestones for which you have defined automation events. When the order reaches the corresponding milestone, the OSM server triggers the automation event plug-in.
Name	In the Automation column, the **Name** field displays the name of automation plug-in.
Automation Type	Displays the automation plug-in type. See "Working with Automation Plug-Ins" for more information.
View	Click an automation, and click **Select** in the **View** field to choose a query task to use with the automation. You must define an OSM user in the automation plug-in **Run As** Property field to run the automation plug-in and configure one or more roles and default query tasks using the Order editor **Permissions** tab. Associate the roles with the OSM user using the OSM Administration area of the Order Management web client. If the **Run As** OSM user has more than one role, each with a different default query task, then multiple query task views are available to run the automation plug-in. You can select a query task to allow OSM to predictably use one query view to run the automation plug-in. If only one default query task is available in the Order Editor Query Task tab, then this query task is automatically added to the **View** field when you create a new automation. See "Order Editor Permissions Query Tasks Tab" for more information. If the query task view has already been selected, click **Open** to view the query task. To create a new query task, click **New** to start the New Studio Entity wizard. **Note:** You must configure the **View** field if you are creating an OSM 7.2 (or later) cartridge. You can deploy older cartridges with the OSM 7.2 (or later) server, but random selection of query task views may occur if an OSM user has more than one role, each with a different default query task.

Related Topics

Creating Order Milestone and Task State Automation Event Notifications

Working with Event Notifications

Working with Orders

Order Editor Composite Data View Tab

Use the Order editor **Composite Data View** tab to display all of the data that is available to the order within the context of an OSM solution. The data in the **Composite Data View** tab is read only. The following table describes the fields on the Order editor **Composite Data View** tab.

> **Tip:**
>
> The composite data view may have fewer data nodes than its corresponding order template view. For example, if a particular function is not included in the solution, its **/ControlData/Functions/***Order_Component_Name* structure will not be in the composite data view.

Field	Use
Solution	Select the solution to display all of the order data associated with the solution.

Field	Use
Order Template	Displays all of the data necessary to fulfill or provision an order within the context of a solution. You cannot modify any data that appears in the Order Template area.
Show Control Data	By default, the check box is selected and all control data is shown. If the default process in the **Details** tab is an orchestration process, control data elements are created automatically and populated to the Order Template and Task Data areas. Deselect the check box if you do not want to show control data.
Behaviors	Displays all of the behaviors assigned to each data node. Select a data node in the Order Template area to view the behaviors assigned to the node. You cannot modify any behaviors that appear in the Behaviors area.

Related Topics

Working with Composite Cartridge Views

Working with Composite Cartridge Projects

11

Working with Behaviors

Behaviors provide a way to exercise greater control over validation and presentation of order data to Oracle Communications Order and Service Management (OSM) web client users. Each behavior type lets you dynamically control a specific aspect of your order data model.

Behaviors can be created for manual tasks only. They can be created at the data element level (most general), the order level (more specific), or the task level (most specific). After the behavior is created, you can model the actions you want it to perform through its properties settings.

When modeling behaviors, see the following topics for more information:

- About Web Client Behavior Support
- Creating New Behaviors
- Defining Behavior Detail Properties
- Defining Behavior Condition Properties
- Defining Behavior Notes Properties
- Defining Calculate Behavior Properties
- Defining Constraint Behavior Properties
- Defining Data Instance Behavior Properties
- Defining Event Behavior Properties
- Defining Information Behavior Properties
- Defining Lookup Behavior Properties
- Defining Read Only Behavior Properties
- Defining Relevant Behavior Properties
- Defining Style Behavior Properties

> **Note:**
>
> See *OSM Concepts* for more information about behavior default values, inheritance, and declarative syntax.

About Web Client Behavior Support

The following table identifies whether the Task web client or the Order Management web client can display OSM data behavior information.

Behavior Name	Task Web Client Support	Order Management Web Client Support
Calculate Behavior	Yes	Yes
Constraint Behavior	Yes	No
Data Instance Behavior	N/A	N/A
Event Behavior	Yes	No
Information Behavior	Yes	Yes
Lookup Behavior	Yes	Yes, partial
Read Only Behavior	Yes	Yes
Relevant Behavior	Yes	Yes
Style Behavior	Yes	Yes, partial

See *OSM Concepts* for more information about how the web clients use and display OSM data behavior information.

Related Topics

Working with Behaviors

Creating New Behaviors

Creating New Behaviors

Behaviors can be created for manual and automated tasks. They can be created at the data element level (most general), the order level (more specific), or the task level (most specific). See the following topics for information about creating new behaviors:

- See "Defining Behaviors at the Data Schema Level" for information about creating behaviors at the data schema level.
- See "Defining Manual Task Behaviors" for information about creating manual behaviors at the task level.
- See "Defining Automated Task Behaviors", for information about creating automated behaviors at the task level.
- See "Defining Order Behaviors" for information about creating behaviors at the order level.

Defining Behavior Detail Properties

Behavior detail properties are common to all behaviors. You can disable behaviors temporarily, override the manner in which behaviors are inherited, determine where a behavior was initially defined, and so forth.

> **Note:**
>
> The level at which you create a behavior (at the data element level, task level, or order level) determines where you access and configure the behavior's properties. See "Creating New Behaviors" for more information.

To define behavior property details:

1. From the Design perspective, right-click the behavior and select **Open Properties View**.

   The Behaviors Properties view is displayed.

2. Click the **Details** tab.

   The **Behaviors Properties view Details** tab is displayed. The **Name**, **Type**, and **Path** field values are read-only, and cannot be modified on this tab.

Related Topics

Behaviors Properties View Details Tab

Working with Behaviors

Behaviors Properties View Details Tab

Use the Behaviors Properties View Details tab to enable behaviors and to force local, specific exceptions to the way behaviors are evaluated for a given node.

The **Properties view Details** tab is common to all behaviors.

Field	Use
Name	Displays the name of the behavior. To rename a behavior, from the Behaviors area right-click the behavior and select **Rename**. **Note:** The name of the behavior can only be changed in the location at which the behavior is defined.
Type	Displays the type of behavior selected.
Path	Displays the node context on which the behavior is defined.
Origin	Displays where the behavior is defined. The behavior's inheritance properties are determined by the definition location.
Enabled	Deselect to disable the behavior in the run-time environment. If you disable a behavior and deploy the cartridge, the OSM server will ignore this behavior. For example, you can disable behaviors during testing. By default, this check box is selected.
Final	Select to prevent another behavior of the same type, for the same node, at the same or more specific level from overriding that behavior.
Override	Select to indicate that the behavior takes precedence over any other behavior of the same type, for the same node, at the same or more general (order) level. **Note**: Override does not function if the behavior that you are trying to override has the Final attribute enabled.

Related Topics

Defining Behavior Detail Properties

Working with Behaviors

Defining Behavior Condition Properties

You can apply conditions to a behavior that determine if it is applied, based on the view data.

> **Note:**
>
> The level at which you create a behavior (at the data element level, task level, or order level) determines where you access and configure the behavior's properties. See "Creating New Behaviors" for more information.

To define behavior conditions:

1. From the Design perspective, right-click a behavior and select **Open Properties View**.

   The Behaviors Properties view is displayed.
2. Click the **Conditions** tab.

   The **Behaviors Properties view Conditions** tab is displayed.
3. Click **Add**.

   A new condition is displayed in the **Condition** field with the default name **Condition**. The default XPath expression **true()** appears in the **XPath Expression** field.
4. Select the default condition name to change the default name.
5. Select the default XPath expression to replace it or modify it.
6. Click **Remove** to delete a selected condition.

Related Topics

About Behavior Condition Properties

Behaviors Properties View Conditions Tab

Working with Behaviors

About Behavior Condition Properties

You can apply conditions to any behavior to determine the conditions under which the behavior should apply. You can add conditions as XPath expressions against which the behavior can run a Boolean compare. If the Boolean compare returns true, the behavior is applied.

For example, to associate a behavior with a postal code field in a web client to target all customers in a specific region, you might apply the condition:

```
../postal_code = '95419'
```

You can select a data node from the **Order Template** tab (when working in the Order editor) or from the **Task Data** tab (when working in a Task editor) and drag the selected data node into the **XPath Expressions** field to define the XPath expression. To drag a data node into the Properties view Conditions tab, press and hold the Alt key before you select and drag the data node to the **XPath Expressions** field.

> **Note:**
>
> XPath uses path expressions to select data nodes in XML documents. A path expression with a single dot (.) represents the current node. Two dots (..) represents the parent of the current node. A slash (/) represents the root node.
>
> XPath and XQuery fields are limited to 4000 characters.

When no condition is defined for the behavior, the OSM server will always apply the behavior. When you define multiple conditions for the behavior, all conditions must evaluate to true for the OSM server to apply the behavior.

Defining Constraint Behavior Condition Properties

When defining conditions for Constraint behaviors, you specify the conditions that must be satisfied to avoid triggering the behavior. If any one of the conditions defined for the Constraint behavior are not met (those that evaluate to false), the OSM server triggers the constraint and displays the appropriate message to the user, based on the severity level. If no conditions are specified, the constraint will not be triggered.

> **Note:**
>
> See *OSM Concepts* for more information about behavior default values, inheritance, and declarative syntax.

Related Topics

Defining Behavior Condition Properties

Behaviors Properties View Conditions Tab

Working with Behaviors

Behaviors Properties View Conditions Tab

Use the Behaviors Properties View **Conditions** tab to apply conditions to a behavior to determine when the behavior should apply.

The **Properties view Conditions** tab is common to all behaviors.

Field	Use
Add	Click to apply a new condition to the behavior.
Conditions	Displays the list of conditions defined for the corresponding behavior. Select a condition to rename it.
XPath Expression	Displays the XPath Expression that defines the logic of the corresponding condition. Select the expression to modify or remove it. To drag a data node into the Properties view Conditions tab, press and hold the Alt key before you select and drag the data node to the XPath Expressions field. XPath and XQuery fields are limited to 4000 characters.

Field	Use
Remove	Click to remove the highlighted condition.

Related Topics

Defining Behavior Condition Properties

About Behavior Condition Properties

Working with Behaviors

Defining Behavior Notes Properties

On the Properties view **Notes** tab, you can describe the intended use of the behavior. For example, you might describe the functionality of a complex behavior, or provide instructions for implementation or testing.

Related Topics

Working with Behaviors

Defining Calculate Behavior Properties

When editing order and task data in an editor, you can right-click data node behaviors and select **Open Properties View** to access the behavior properties.You use the Properties view tabs to model Calculate behaviors.

> **Note:**
>
> The level at which you create a behavior (at the data element level, task level, or order level) determines where you access and configure the behavior's properties. See "Creating New Behaviors" for more information.

To define Calculate behavior properties:

1. From the Design perspective, right-click the behavior and select **Open Properties View**.

   The Behaviors Properties view is displayed.
2. Click the **Calculation** tab.
3. In the **XPath Expression** field, enter the calculation as a mathematical expression or as an XPath expression.

   See "Calculate Behavior Properties View Calculation Tab" for more information.
4. Click the **Details** tab.

   The **Behaviors Properties view Details** tab is displayed. The **Name**, **Type**, and **Path** field values are read-only, and cannot be modified on this tab. See "Defining Behavior Detail Properties" for more information about the options that you can define on this page.
5. (Optional) Click the **Conditions** tab.

Use the **Conditions** tab to add conditional logic to the Calculate behavior. See "Defining Behavior Condition Properties" for more information about defining conditions for behaviors.

6. (Optional) Click the **Notes** tab.

   Use the **Notes** tab to describe the functionality or include internal documentation about the Calculate behavior.

Related Topics

Calculate Behavior Properties View Calculation Tab

About Calculate Behaviors

Working with Behaviors

About Web Client Behavior Support

About Calculate Behaviors

The Calculate behavior enables you to calculate a field's value based on a formula that references other field values. When defining Calculate behaviors on the Calculation Behaviors Properties tabs, you can use XPath expressions to support numeric operations and string concatenations. For example:

XPath Expression	Result
`../loopback`	Set the current field equal to the value found in the **../loopback** field.
`concat('S',instance('interfacedetail')/Port)`	Set the current field equal to a concatenation between the letter *S* and the value found in the **Port** field that is returned by the **peinterfacedetails** data provider.
`instance('interfacedetail')/portType`	Set the current field equal to the **portType** returned by the **interfacedetail** data instance provider.

See *OSM Concepts* for more information about behavior default values, inheritance, and declarative syntax. For more information about XPath specifications, see the World Wide Web Consortium (W3C) website at:

`http://www.w3.org/TR/xpath20/`

> **Note:**
>
> XPath uses path expressions to select data nodes in XML documents. A path expression with a single dot (.) represents the current node. Two dots (..) represents the parent of the current node. A slash (/) represents the root node.
>
> XPath and XQuery fields are limited to 4000 characters.

Related Topics

Calculate Behavior Properties View Calculation Tab

Defining Calculate Behavior Properties

Working with Behaviors

About Web Client Behavior Support

Calculate Behavior Properties View Calculation Tab

On the **Properties view Calculations** tab, you can define the expression that will produce the calculation.

Field	Use
XPath Expression	Enter a mathematical expression or an XPath Expression. You can select a data node from the **Order Template** tab (when working in the Order editor), or from the **Task Data** tab (when working in a Task editor) and drag the selected data node into the **XPath Expression** field to define the XPath expression. To drag a data node into the **Properties view Calculation** tab, press and hold the Alt key before you select and drag the data node to the **XPath Expressions** field. **Note:** XPath uses path expressions to select data nodes in XML documents. A path expression with a single dot (.) represents the current node. Two dots (..) represents the parent of the current node. A slash (/) represents the root node. XPath and XQuery fields are limited to 4000 characters.

Related Topics

Defining Calculate Behavior Properties

About Calculate Behaviors

Working with Behaviors

Defining Constraint Behavior Properties

The Constraint behavior enables you to specify conditions that must be satisfied for a given data node to be considered valid. If the condition is not satisfied (that is, if it evaluates to false), messages are displayed to the user.

When editing order and task data in an editor, you can right-click data node behaviors and select **Open Properties View** to access the behavior properties.You use the Properties view tabs to model Constraint behaviors.

> **Note:**
>
> The level at which you create a behavior (at the data element level, task level, or order level) determines where you access and configure the behavior's properties. See "Creating New Behaviors" for more information.

To define Constraint behavior properties:

1. From the Design perspective, right-click the behavior and select **Open Properties View**.

   The Behaviors Properties view is displayed.
2. Click the **Message** tab.

3. In the **Language** field, select a predefined language in which to display the message to the web client user.

   See "Defining OSM Preferences" for more information.

4. In the **Message** field, enter the text that you want to display to the web client user when the OSM server applies a Constraint behavior.

5. In the **Display As** field, define the level of severity of the message.

   The level of severity in conjunction with the task status severity setting affects how the OSM server proceeds after a Constraint behavior has been triggered. See "Constraint Behavior Properties View Message Tab" and "Task Editor States/Statuses Tab" for more information.

6. Click the **Details** tab.

   The **Behaviors Properties view Details** tab is displayed. The **Name**, **Type**, and **Path** field values are read-only, and cannot be modified on this tab. See "Defining Behavior Detail Properties" for more information about the options that you can define on this page.

7. Click the **Conditions** tab.

   Use the **Conditions** tab to add conditional logic to the Constraint behavior.

   > **Note:**
   >
   > The OSM server applies Constraint behaviors when any Constraint behavior conditions evaluate to false. If you define no conditions for the Constraint behavior, the OSM server will never apply the behavior in the web client.

   See "Defining Behavior Condition Properties" for more information about defining conditions for behaviors.

8. Click the **Notes** tab.

   Use the **Notes** tab to describe the functionality or include internal documentation about the Constraint behavior.

> **Note:**
>
> See *OSM Concepts* for more information about behavior default values, inheritance, and declarative syntax.

Related Topics

Constraint Behavior Properties View Message Tab

Working with Behaviors

About Web Client Behavior Support

Constraint Behavior Properties View Message Tab

Use the **Properties view Message** tab to define the language, content, and severity level of the message. For data element level Constraint behaviors, the severity level, in conjunction with the task status severity level, affects whether or not the task is allowed to transition.

Field	Use
Language	Select the display language for the message.
Message	Enter one or more messages to display when a condition is not satisfied.
Display as	Select the severity level as follows: • Critical: On save, OSM does not save the order data and displays the message in bold red text, with the "ERROR" label. • Error: On save, OSM saves the order data and displays the message in red text, with the "ERROR" label. • Warning: On save, OSM saves the order data and displays the message in yellow text, with the "WARNING" label. • Valid: On save, OSM saves the order data and displays the message in green text, with the "INFO" label.

Related Topics

Defining Constraint Behavior Properties

Working with Behaviors

About Web Client Behavior Support

Task Editor States/Statuses Tab

Defining Data Instance Behavior Properties

When editing order and task data in an editor, you can right-click data node behaviors and select **Open Properties View** to access the behavior properties.You use the Properties view tabs to model Data Instance behaviors.

> **Note:**
>
> The level at which you create a behavior (at the data element level, task level, or order level) determines where you access and configure the behavior's properties. See "Creating New Behaviors" for more information.

To define data instance behavior properties:

1. From the Design perspective, right-click the behavior and select **Open Properties View**.

   The Behaviors Properties view is displayed.
2. Click the **Data** tab.
3. In the **Language** field, select a predefined language in which to display the message to the web client user.

See "Defining OSM Preferences" for more information.

4. Specify whether to use a data provider to retrieve the data from an external system, or to statically define the data inline.
   - To use a data provider to retrieve the information, proceed to step 5.
   - To statically define the data inline, select **Inline** enter the XML information into the **XML** field, then skip to step 11.
5. In the **Data Provider** field, click **Select**.

   The Select Data Provider dialog box is displayed.

   Alternatively, you can click **New** to create a new data provider. See "Creating New Data Providers " for more information about creating new data providers for Data Instance behaviors.
6. Select a data provider from the list.
7. Click **OK**.

   The input parameters defined for the data provider are displayed in the **Parameters** field.
8. In the **Parameters** field, select an input parameter.
9. In the **Expression** field, define the value for the input parameter that the data provider requires when retrieving the data from the external system.

   The value is evaluated at run-time and is based on the XPath or XQuery expression you define in the **Expression** field.
10. (Optional) Select **Use Default Expression** to use the default expression data defined for the parameters.

    For example, if you were creating a Data Instance behavior to obtain a list of available ports from inventory, you might define an end point parameter to provide the server connection information required to connect to the inventory system. You can define (in the Data Provider editor) the default information defined for the end point parameter, and use that information for the corresponding Data Instance behavior.
11. Click the **Details** tab.

    The **Behaviors Properties view Details** tab is displayed. The **Name**, **Type**, and **Path** field values are read-only, and cannot be modified on this tab. See "Defining Behavior Detail Properties" for more information about the options that you can define on this page.
12. Click the **Conditions** tab.

    Use the **Conditions** tab to add conditional logic to the Data Instance behavior. See "Defining Behavior Condition Properties" for more information about defining conditions for behaviors.
13. Click the **Notes** tab.

    Use the **Notes** tab to describe the functionality or include internal documentation about the Data Instance behavior.

Related Topics

Data Instance Behavior Properties View Data Tab

About Data Instance Behaviors

Working with Behaviors

Working with Data Providers

About Data Instance Behaviors

The Data Instance behavior differs significantly from all other behavior types in that it does not define any behavior. All other behavior types define some sort of action to be performed; for example, a calculation or a lookup. You can use a Data Instance behavior to obtain data that is not included in the order data and make that data available to other behaviors. There are two methods for obtaining the data for the Data Instance behavior:

- You can use a data provider, which is an adapter that can retrieve data in an XML format from external systems. Design Studio delivers several built-in data provider types intended to retrieve external XML instances from specific sources, such as an Objectel server extension or a SOAP web service. Additionally, you can create your own custom data provider. See "Working with Data Providers" for more information.
- You can statically define data inline in XML format (or create an XQuery statement to retrieve an XML document) on the Data Instance Behavior Properties Data tab. For example, consider that you are creating a data instance behavior that will eventually retrieve a list of available ports from your inventory system. Early in the development cycle, the API required to connect to the inventory system may not be implemented correctly or completely. You can use the inline feature to statically define a dummy payload that represents the data that you anticipate will be returned from the inventory system to test the behavior functionality.

> **Note:**
>
> See *OSM Concepts* for more information about behavior default values, inheritance, and declarative syntax.

Related Topics

Data Instance Behavior Properties View Data Tab

Defining Data Instance Behavior Properties

Working with Behaviors

Working with Data Providers

About Web Client Behavior Support

Data Instance Behavior Properties View Data Tab

Use the **Properties view Data** tab to define the data provider configuration that will interface with the external system.

Field	Use

Language	You can declare Data Instance behaviors specific to a given language by using the optional **Language** attribute. If this attribute is set, OSM automatically selects the appropriate instance using the user's language preferences set in the web browser.
	Note: If you declare multiple language-based instances, consider the following:
	• To appear in the **Language** drop-down list, languages other than the default must be selected from the Oracle Design Studio languages group. See "Defining Language Preferences".
	• One Data Instance behavior is created per language selected.
	• In order to differentiate among language-based behaviors, the system appends (internally) a language code to the data instance name using the "name of the data instance"_"language code" pattern. For example:
	`DataInstanceX_en-ca`
	The language codes used are from the Oracle Design Studio languages group.
	• If you need to refer to this instance from any other behavior (for example, Lookup), you must specify the full value in the behavior's XPath expression. For example:
	`instance('DataInstanceX_en-ca')/lookupEntry`

Data Provider	You can reuse an existing data provider configuration for this Data Instance behavior by clicking the **Select** button. To create a new data provider configuration, click **New**. After you have selected or created a data provider, click **Open** to access the Data Provider editor, where you can define input parameters, result documents, and cache settings. Click **Clear** (red X) to clear the selected value from the field.
Parameters, Expression, Use Default Expression	Select a parameter to define the value for the input parameter that the data provider requires when retrieving the data from the external system. The value is evaluated at runtime and is based on the XPath or XQuery expression you define in the **Expression** field. Select the **Use Default Expression** option if you want to use the default values for the input parameters that you defined using the **Data Provider editor Interface** tab. You can select a data node from the **Order Template** tab (when working in the Order editor) or from the **Task Data** tab (when working in a Task editor) and drag the selected data node into the **Parameters** field to define the XPath expression. **Note:** To drag a data node into the **Properties view Data** tab, press and hold the Alt key before you select and drag the data node to the **XPath Expressions** field. XPath uses path expressions to select data nodes in XML documents. A path expression with a single dot (.) represents the current node. Two dots (..) represents the parent of the current node. A slash (/) represents the root node. XPath and XQuery fields are limited to 4000 characters.

Field	Use
Inline	Select if you want to make static information available to the task. This option assumes that the information is not located on an external system. When you select this option, you can declare an XML document statically within the XML field. **Note:** You can use the **Inline** option early in development cycles when you know you want to employ a data provider to an external system, but you haven't yet built it. Using the **Inline** option, you can create a dummy information structure for testing purposes. You can remove the inline XML static information and clear the inline option later in the cycle after you build the data provider configuration. When using the **Inline** option, consider that there exists no XML validation in the static document field.

Related Topics

Defining Data Instance Behavior Properties

About Data Instance Behaviors

Working with Behaviors

Defining Event Behavior Properties

When editing order and task data in an editor, you can right-click data node behaviors and select **Open Properties View** to access the behavior properties.You use the Properties view tabs to model Event behaviors.

> **Note:**
>
> The level at which you create a behavior (at the data element level, task level, or order level) determines where you access and configure the behavior's properties. See "Creating New Behaviors" for more information.

To define Event behavior properties:

1. From the Design perspective, right-click the behavior and select **Open Properties View**.

   The Behaviors Properties view is displayed.
2. Click the **Events** tab.
3. Specify when the OSM server should apply the Event behavior.

   Select:

   - **Save** to apply the event when the user clicks the Task web client **Save** button.
   - **Refresh** to apply the event immediately after the user leaves the field associated with the Event behavior.
4. Click the **Details** tab.

   The **Behaviors Properties view Details** tab is displayed. The **Name**, **Type**, and **Path** field values are read-only, and cannot be modified on this tab. See "Defining Behavior Detail Properties" for more information about the options that you can define on this page.

5. Click the **Conditions** tab.

   Use the **Conditions** tab to add conditional logic to the Event behavior.

   See "Defining Behavior Condition Properties" for more information about defining conditions for behaviors.

6. Click the **Notes** tab.

   Use the **Notes** tab to describe the functionality or include internal documentation about the Event behavior.

Related Topics

About Event Behaviors

Event Behavior Properties View Event Tab

Working with Behaviors

About Event Behaviors

The Event behavior specifies an action to perform when a given event occurs. Currently, there is one supported event: *value-changed*. When data associated with the node for which an event rule is defined is changed, the event rule signals the OSM server to re-render the view and return the new results to the Task web client.

For example, you can combine the Event behavior with Relevant Behaviors to display certain fields in an Task web client based on user selection. Consider that you use a Payment Type field with Cash and Credit Card as options. You can create an Event behavior for the Payment Type to re-render the view after the user tabs out of the field. You can create Relevant rules for Credit Card Number, Expiration, and so forth, so that fields relevant to a credit card payment appear in an Task web client when a user selects the Payment Type of Credit Card.

> **Note:**
>
> See *OSM Concepts* for more information about behavior default values, inheritance, and declarative syntax.

Related Topics

Defining Event Behavior Properties

Event Behavior Properties View Event Tab

Working with Behaviors

About Web Client Behavior Support

Event Behavior Properties View Event Tab

On the Properties view Event tab, you can specify how the OSM server should re-render an Task web client view when an event behavior occurs.

Field	Use
Save	Select to signal the OSM server to re-render the Task web client view only after the user clicks the Task web client **Save** button.
Refresh	Select to signal the OSM server to re-render the Task web client view immediately after the user moves out of the field associated with the behavior.

Related Topics

Defining Event Behavior Properties

About Event Behaviors

Working with Behaviors

Defining Information Behavior Properties

The Information behavior enables you to create labels, hints (tool tips), and help information for data nodes that appear in a web client. Before you define a behavior's properties, you must first create the behavior at either the data element level, task level, or order level.

When editing order and task data in an editor, you can right-click a behavior and select **Open Properties View** to access the behavior properties. You use the Properties view tabs to model Information behaviors.

> **Note:**
>
> The level at which you create a behavior (at the data element level, task level, or order level) determines where you access and configure the behavior's properties. See "Creating New Behaviors" for more information.

To define Information behavior properties:

1. From the Design perspective, right-click the behavior and select **Open Properties View**.

   The Behaviors Properties view is displayed.
2. Click the **Labels** tab.
3. In the **Language** field, select a predefined language in which to display the label to the web client user.

   See "Defining Language Preferences" for more information about defining and using languages in Design Studio. See "Defining Information Behaviors in Multiple Languages" for information about how to change the language in which data fields appear in the web client.
4. In the **XPath Expression** field, enter an XPath Expression or enter a literal (enclosed by single quotes) to describe the label that you want to display to the web client user.

   See "Information Behavior Properties View Labels Tab" for more information about entering XPath expressions.
5. Click the **Hints** tab.

6. In the **Language** field, select a predefined language in which to display the hint to the web client user when they scroll over the associated field.

   See "Defining OSM Preferences" for more information about defining and using languages in Design Studio. See "Defining Information Behaviors in Multiple Languages" for information about how to change the language in which hints are displayed in the web client.

7. In the **XPath Expression** field, enter an XPath Expression or enter a literal (enclosed by single quotes) to describe the hint that you want to display to the web client user.

   See "Information Behavior Properties View Hints Tab" for more information about entering XPath expressions.

8. Click the **Help** tab.

9. In the **Language** field, select a predefined language in which to display the help text to the web client user when they click Help button for the associated field.

   See "Defining OSM Preferences" for more information about defining and using languages in Design Studio.

10. In the **Topic** field, enter the topic name for this help message.

11. In the **Message** field, enter the help documentation that will be displayed in the web client when the user clicks the associated field Help button.

    Use valid HTML to enter the help message.

12. Click the **Details** tab.

    The **Behaviors Properties view Details** tab is displayed. The **Name**, **Type**, and **Path** field values are read-only, and cannot be modified on this tab. See "Defining Behavior Detail Properties" for more information about the options that you can define on this page.

13. Click the **Conditions** tab.

    Use the **Conditions** tab to add conditional logic to the Information behavior. See "Defining Behavior Condition Properties" for more information about defining conditions for behaviors.

14. Click the **Notes** tab.

    Use the **Notes** tab to describe the functionality or include internal documentation about the Information behavior.

> **Note:**
>
> When you define Information behavior properties for a data element with a range where the minimum is zero, the Task web client applies the behavior only when you add a data node. If there are no data nodes, the help or hint does not appear, and the field uses the Display Name from the Data Dictionary.

See *OSM Concepts* for more information about behavior default values, inheritance, and declarative syntax.

Related Topics

Defining Information Behaviors in Multiple Languages

Information Behavior Properties View Labels Tab

Information Behavior Properties View Hints Tab

Information Behavior Properties View Help Tab

Working with Behaviors

About Web Client Behavior Support

Defining Information Behaviors in Multiple Languages

You can use Information behaviors to configure web client labels and field level tool tips (called hints in Design Studio) in multiple languages. The configuration that you complete in Design Studio can enable a web client to display field labels and hints in a preferred language when a web client detects changes to Internet browser language preferences.

The following example demonstrates how to define labels in a preferred language. The steps for defining hints are identical, except that you define the language and XPath expression in the **Information Properties view Hints** tab.

To create labels and hints in multiple languages:

1. From the **Window** menu, select **Preferences**.

   The Preferences dialog box is displayed.

2. In the Preferences dialog box, click **Oracle Design Studio**.

   The Design Studio Language Preferences options are displayed.

3. Add the language to the group of languages with which you intend to work.

   See "Defining OSM Preferences" for more information about adding languages to the Preferences dialog box.

4. Click **OK**.

   Design Studio closes the Preference dialog box.

5. Determine at which level you will create the Information behavior.

   You can create behaviors at the data, order, and task levels. For example, behaviors defined at the data level apply to the data node in all orders and tasks; behaviors applied at the order level apply to all tasks in the order; and behaviors defined at the task level apply to a single task in the order.

   - See "Defining Behaviors at the Data Schema Level" for information about creating behaviors at the data schema level.
   - See "Defining Manual Task Behaviors" for information about creating behaviors at the task level.
   - See "Defining Order Behaviors" for information about creating behaviors at the order level.

6. In the Data Schema, Order, or Task editor, select a data node and create the Information behavior.

7. Right-click the Information behavior icon and select **Open Properties View**.

   The **Information Properties view Labels** tab is displayed.

8. In the **Information Properties view Labels** tab **Language** field, select the language for which you want to define a label.

   No changes to **Language** field are necessary if you are defining labels in the default language.

> **Note:**
>
> By default, the system uses the Display Name defined for the data node and the default language defined in the Design Studio Preferences dialog box when labeling fields in the web client.

9. In the **XPath Expression** field, enter the text for the new label.

   Use single quotes to wrap the text when entering literals in an XPath expression. See "Information Behavior Properties View Labels Tab" for more information about entering XPath expressions.

10. Click **Save**.

11. Clean, build, and deploy your cartridge to the run-time environment.

    See "Packaging and Deploying OSM Cartridges" for more information and cleaning, building, and deploying cartridges.

12. In the web client, navigate to the browser language preferences to select the language in which you defined your labels and hints.

    For example, in Internet Explorer:

    a. Select **Tools, Internet Options**.

    b. Click **Language**.

       The Language Preferences dialog box is displayed.

    c. Click **Add**.

       The Add Language dialog box is displayed.

    d. Select the language in which you created new field labels or hints in Design Studio.

    e. Click **OK**.

    f. Select the language, and click **Move Up** to move the language to the first position.

    g. Click **OK**.

       The system displays the new labels in the web client.

Related Topics

Information Behavior Properties View Labels Tab

Information Behavior Properties View Hints Tab

Information Behavior Properties View Help Tab

Working with Behaviors

Information Behavior Properties View Labels Tab

On the **Properties view Labels** tab, you can create labels by selecting a language and defining an expression.

Field	Use
Language	You can declare Information behaviors specific to a given language by using the optional **Language** attribute. If this attribute is set, OSM automatically selects the appropriate instance using the user's language preferences set in the web browser.
XPath Expression	Enter a mathematical expression or an XPath Expression. You can select a data node from the **Order Template** tab (when working in the Order editor) or from the **Task Data** tab (when working in a Task editor) and drag the selected data node into the **XPath Expression** field to define the XPath expression. To drag a data node into the **Properties view Labels** tab, press and hold the Alt key before you select and drag the data node to the **XPath Expressions** field. XPath uses path expressions to select data nodes in XML documents. A path expression with a single dot (.) represents the current node. Two dots (..) represents the parent of the current node. A slash (/) represents the root node. XPath and XQuery fields are limited to 4000 characters.

Related Topics

Defining Information Behavior Properties

Working with Behaviors

Information Behavior Properties View Hints Tab

On the **Properties view Hints** tab, you can create hints (tooltips) in the web client by selecting a language and defining a message.

Field	Use
Language	You can declare Information behaviors specific to a given language by using the optional **Language** attribute. If this attribute is set, OSM automatically selects the appropriate instance using the user's language preferences set in the web browser.
XPath Expression	Enter a mathematical expression or an XPath Expression. You can select a data node from the **Order Template** tab (when working in the Order editor), or from the **Task Data** tab (when working in a Task editor) and drag the selected data node into the **XPath Expression** field to define the XPath expression. To drag a data node into the **Properties view Hint** tab, press and hold the Alt key before you select and drag the data node to the **XPath Expressions** field. XPath uses path expressions to select data nodes in XML documents. A path expression with a single dot (.) represents the current node. Two dots (..) represents the parent of the current node. A slash (/) represents the root node. XPath and XQuery fields are limited to 4000 characters.

Related Topics

Defining Information Behavior Properties

Working with Behaviors

Information Behavior Properties View Help Tab

On the **Properties view Help** tab, you can create context-sensitive, online help in the web client by selecting a language and defining the help topic and text.

Field	Use
Language	You can declare Information behaviors specific to a given language by using the optional **Language** attribute. If this attribute is set, OSM automatically selects the appropriate instance using the user's language preferences set in the web browser.
Topic	Enter the topic name for this help message.
Message	Enter the help documentation that will be displayed in the web client when the user clicks the associated field Help button. **Note:** Use valid HTML to enter the help message.

Related Topics

Defining Information Behavior Properties

Working with Behaviors

Defining Lookup Behavior Properties

When editing order and task data in an editor, you can right-click data node behaviors and select **Open Properties View** to access the behavior properties.You use the Properties view tabs to model Lookup behaviors.

> **Note:**
>
> The level at which you create a behavior (at the data element level, task level, or order level) determines where you access and configure the behavior's properties. See "Creating New Behaviors" for more information.

To define Lookup behavior properties:

1. From the Design perspective, right-click the behavior and select **Open Properties View**.

   The Behaviors Properties view is displayed.
2. Click the **Nodeset** tab.
3. In the **XPath Expression** field, enter the XPath expression that selects the set of data nodes that will comprise the lookup results.
4. Click the **Value/Name** tab.

   A default column, called ValueColumn, is displayed in the Value Name table. The OSM server uses this value to populate the data node to which the Lookup behavior is attached.
5. (Optional) Click **Add**.

   Design Studio creates an additional column in the drop down list. For example, you can add a second column to create a label for the data retrieved.
6. Specify the value that OSM server should use for the corresponding data node.

If you have multiple columns defined in the Value Name table, select the column whose value you want the OSM server to use for the data node, and click **Set as Value**.

For example, if you have two columns, a code column and description column, you might elect to have the user make a selection based on the description field, but identify the code column as the value that the OSM server should use to update the task.

7. In the **Name** field, enter a name for the selected column value.
8. In the **Node** field, enter an XPath expression to define the relative path to the node in the nodeset.

   For example, if the nodeset contains 2 data nodes, you must identify which of the two data nodes the OSM server should retrieve for the select column.
9. Click the **Details** tab.

   The **Behaviors Properties view Details** tab is displayed. The **Name**, **Type**, and **Path** field values are read-only, and cannot be modified on this tab. See "Defining Behavior Detail Properties" for more information about the options that you can define on this page.
10. Click the **Conditions** tab.

    Use the **Conditions** tab to add conditional logic to the Lookup behavior. See "Defining Behavior Condition Properties" for more information about defining conditions for behaviors.
11. Click the **Notes**.

    Use the **Notes** tab to describe the functionality or include internal documentation about the Lookup behavior.

Related Topics

Lookup Behavior Properties View Nodeset Tab

Lookup Behavior Properties View Value/Name Tab

About Lookup Behaviors

Working with Behaviors

About Lookup Behaviors

Use the Lookup behavior to specify a set of dynamically generated field value choices to be included in a drop down list. The Lookup behavior can retrieve data of any type, and can obtain the data from the task view data or from an external system, (such as Objectel) using a Data Instance behavior. You can also define the behavior to display multiple columns to the end user.

You attach Lookup behaviors to simple type data nodes, and define the location of the data using an XPath expression. When the Lookup behavior retrieves the data, it expects a repeating XML data structure that it will use to build the list of options.

When configuring Lookup behavior properties, you define where the OSM server should obtain the information, how much information to present in the drop down list, the order in which the options should be presented to the user, and the value that the OSM server uses for the data node (for which the Lookup behavior is defined) when a user makes a selection. For example, consider that you want a Lookup behavior to retrieve a code and a description of mobile phone handset color options. You can create a Data Instance behavior to retrieve the available handset color options from inventory, and display those options to the user with a Lookup behavior:

Code Column Value	Description Column Value
12HS00B	Blue Handset
12HS00S	Silver Handset
12HS00R	Red Handset

You might elect to display only the description to the user (in a single column) but update the task data with the code value.

> **Note:**
>
> See *OSM Concepts* for more information about behavior default values, inheritance, and declarative syntax.

Related Topics

Lookup Behavior Properties View Nodeset Tab

Lookup Behavior Properties View Value/Name Tab

Defining Lookup Behavior Properties

Working with Behaviors

About Web Client Behavior Support

Lookup Behavior Properties View Nodeset Tab

Use the **Properties view Nodeset** tab to define the XPath expression that selects the set of nodes that comprise the lookup results. You can enter the expression or drag the desired nodes into the **XPath Expression** field.

Field	Use
XPath Expression	Enter an XPath expression. You can select a data node from the **Order Template** tab (when working in the Order editor), or from the **Task Data** tab (when working in a Task editor) and drag the selected data node into the **XPath Expression** field to define the XPath expression. To drag a data node into the **Properties view Nodeset** tab, press and hold the Alt key before you select and drag the data node to the **XPath Expressions** field. XPath uses path expressions to select data nodes in XML documents. A path expression with a single dot (.) represents the current node. Two dots (..) represents the parent of the current node. A slash (/) represents the root node. XPath and XQuery fields are limited to 4000 characters.

Related Topics

Defining Lookup Behavior Properties

Working with Behaviors

About Lookup Behaviors

Lookup Behavior Properties View Value/Name Tab

Use the Lookup Behavior Properties View Value/Name tab to specify the value that the OSM server should use for a data node, and to create multiple columns to display in the web client.

> **Note:**
>
> You can convert a label to a value by selecting the **Set as Value** check box. When you do this, the previous value automatically becomes a label.

Field	Use
Set as Value	If you have multiple columns defined in the Value Name table, select the column whose value you want the OSM server to use for the data node, and click **Set as Value**. This value will be stored in the order. For example, if you have two columns, a code column and description column, you might elect to have the user make a selection based on the description field, but identify the code column as the value that the OSM server should use to update the task. Only one column in a lookup behavior can be used for Set as Value. Use the **Hidden** field to determine whether the column used as Set as Value is also used as a display column.
Name	Enter a name for the lookup value result.
Hidden	Specify whether the column is displayed or not. By default, the column is displayed. If more than one column is displayed, the value displayed in the field will be the value in the first visible column.
Position	Define the column's position in the lookup result.
Sort Direction	Specify how the column is sorted. Indicate either **Ascending** or **Descending**.
Sort Order	Determine how the column is used to sort the lookup. For example, if this value is **1**, it is the primary sort key.
Selection Severity	Select how open the field is to users entering data that is not on the list of values defined in the lookup. Valid values are: • **VALID**: A value that is not on the list is valid. • **WARNING**: A value that is not on the list generates a warning but can be saved to the order and the task transitioned. • **ERROR**: A value that is not on the list generates an error but can be saved to the order. The task cannot be transitioned until the value that is not on the lookup list is removed. • **CRITICAL**: (default) A value that is not on the list generates an error and cannot be saved to the order. The task cannot be transitioned until the value that is not on the lookup list is removed.
Node	(Mandatory) Enter an XPath expression to define the relative path to the node in the nodeset.

Field	Use
Language	Select the display language.
Description	Enter a brief description of the layout.

Related Topics

Defining Lookup Behavior Properties

Working with Behaviors

About Lookup Behaviors

Defining Read Only Behavior Properties

When editing order and task data in an editor, you can right-click data node behaviors and select **Open Properties View** to access the behavior properties.You use the Properties view tabs to model Read Only behaviors.

> **Note:**
>
> The level at which you create a behavior (at the data element level, task level, or order level) determines where you access and configure the behavior's properties. See "Creating New Behaviors" for more information.

To define Read Only behavior properties:

1. From the Design perspective, right-click the behavior and select **Open Properties View**.

   The Behaviors Properties view is displayed.

2. Click the **Details** tab.

   The **Behaviors Properties view Details** tab is displayed. The **Name**, **Type**, and **Path** field values are read-only, and cannot be modified on this tab. See "Defining Behavior Detail Properties" for more information about the options that you can define on this page.

3. Click the **Conditions** tab.

   Use the **Conditions** tab to add conditional logic to the Read Only behavior. See "Defining Behavior Condition Properties" for more information about defining conditions for behaviors.

4. Click the **Notes** tab.

   Use the **Notes** tab to describe the functionality or include internal documentation about the Read Only behavior.

Related Topics

About Read Only Behaviors

Working with Behaviors

About Read Only Behaviors

The Read Only behavior enables you to target specific conditions when the OSM server should make a field read-only in the Task web client. Information is always read-only in the Order Management web client. You can use the Read Only behavior to make fields read only or read/write, depending on the content of other fields.

For example, assume that a view has an IP Allocation {static, DHCP}, an IP Address, a Subnet Mask, and a Default Gateway field. You can create a read-only rule on IP address, Subnet Mask, and Default Gateway that evaluates to *True* when the value *DHCP* is selected for the IP Allocation field.

See *OSM Concepts* for more information about behavior default values, inheritance, and declarative syntax.

> **Note:**
>
> If a data node's Read Only behavior evaluates to true, all children of that data node are read-only.

About Read Only Behaviors and Read Only Task View Options

In Design Studio, you can define a data node as read-only by creating a Read Only behavior, and by defining the data node as read-only at the task view level:

- Defining a Read Only behavior for a data node enables you to apply specific conditions to determine when the OSM server should make the associated field in the Task web client read-only.
- Defining a data node as read-only at the task view level ensures that a field always is displayed as read-only in the Task web client (in the context of the associated task). See "Task Data Node Properties View Identification Tab" for information about defining data nodes as read-only at the task view level.

> **Note:**
>
> If a Read Only behavior and a task view read-only option conflict for the same data node, the OSM server defaults to the Read Only behavior. For example, consider that you have identified a data node as read-only on the Task Data Node Properties View Identification tab, and you've also defined a Read Only behavior for that same data node. If the behavior evaluates to false, the Task web client field associated with the data node will be editable.

Related Topics

Defining Read Only Behavior Properties

Working with Behaviors

About Web Client Behavior Support

Defining Relevant Behavior Properties

When editing order and task data in an editor, you can right-click data node behaviors and select **Open Properties View** to access the behavior properties.You use the Properties view tabs to model Relevant behaviors.

> **Note:**
>
> The level at which you create a behavior (at the data element level, task level, or order level) determines where you access and configure the behavior's properties. See "Creating New Behaviors" for more information.

To define Relevant behavior properties:

1. From the Design perspective, right-click the behavior and select **Open Properties View**.

   The Behaviors Properties view is displayed.

2. Click the **Details** tab.

   The **Behaviors Properties view Details** tab is displayed. The **Name**, **Type**, and **Path** field values are read-only, and cannot be modified on this tab. See "Defining Behavior Detail Properties" for more information about the options that you can define on this page.

3. Click the **Conditions** tab.

   Use the **Conditions** tab to add conditional logic to the Relevant behavior. See "Defining Behavior Condition Properties" for more information about defining conditions for behaviors.

4. Click the **Notes** tab.

   Use the **Notes** tab to describe the functionality or include internal documentation about the Relevant behavior.

Related Topics

About Relevant Behaviors

Working with Behaviors

About Relevant Behaviors

Use the Relevant behavior to make fields hidden or visible, depending on the content of other fields. You can apply a Relevant behavior to a group node to hide all of its children when the behavior evaluates to false.

The Relevant behavior is usually combined with an Event behavior. When you associate a Relevant behavior to a field, the fields in the web client can only be hidden or revealed upon an OSM server refresh.

For example, if a user selects the value Check in the Payment Method field, the Credit Card Number and Type fields become non-relevant, and should be hidden. In this example, you define an Event behavior to refresh the web client window when the user tabs out of the Payment Method field. Additionally, you define Relevant behaviors for the Credit Card Number and Type fields to evaluate to false when the user selects Check as the payment method.

> **Note:**
>
> See *OSM Concepts* for more information about behavior default values, inheritance, and declarative syntax.

Related Topics

Defining Relevant Behavior Properties

Working with Behaviors

About Web Client Behavior Support

Defining Style Behavior Properties

Use the Style behavior to control the appearance of a given node in the web client. You can apply multiple Style behaviors to a data node, each defined with specific conditions such that when one condition evaluates to true, the OSM server will apply the corresponding style behavior. For example, you can specify that the font color for a field in the web client appear black unless a user enters an invalid value, in which case you specify the font color to change to red.

OSM applies style behaviors to all compensation strategies: **Redo**, **Undo**, and **Do Nothing**.

When editing order and task data in an editor, you can right-click data node behaviors and select **Open Properties View** to access the behavior properties.You use the Properties view tabs to model Style behaviors.

> **Note:**
>
> The level at which you create a behavior (at the data element level, task level, or order level) determines where you access and configure the behavior's properties. See "Creating New Behaviors" for more information.

To define Style behavior properties:

1. From the Design perspective, right-click the behavior and select **Open Properties View**.

   The Behaviors Properties view is displayed.

2. Click the **Appearance** tab.

   Use the **Appearance** tab to specify how field options are displayed, whether to include line breaks after fields, and whether to hide sensitive information in field contents. See "Style Behavior Properties View Appearance Tab" for more information.

3. Click the **Layout** tab.

   Use the **Layout** tab to group child nodes into tabs or table columns. These options are available only for structures that contain child nodes. See "Style Behavior Properties View Layout Tab" for more information.

4. Click the **CSS Style** tab.

   On the **CSS Style** tab, you can add HTML formatting to field values and labels in the web client. Enter HTML directly into the CSS Style Attribute fields for values and labels, or enter

a class name to control the formatting for values and labels. See "Style Behavior Properties View CSS Style Tab" for more information.

5. Click the **Details** tab.

   The **Behaviors Properties view Details** tab is displayed. The **Name**, **Type**, and **Path** field values are read-only, and cannot be modified on this tab. See "Defining Behavior Detail Properties" for more information about the options that you can define on this page.

6. Click the **Conditions** tab.

   Use the **Conditions** tab to add conditional logic to the Style behavior. See "Defining Behavior Condition Properties" for more information about defining conditions for behaviors.

7. Click the **Notes** tab.

   Use the **Notes** tab to describe the functionality or include internal documentation about the Style behavior.

> **Note:**
>
> See *OSM Concepts* for more information about behavior default values, inheritance, and declarative syntax.

Related Topics

Style Behavior Properties View Appearance Tab

Style Behavior Properties View Layout Tab

Style Behavior Properties View CSS Style Tab

Working with Behaviors

About Web Client Behavior Support

Style Behavior Properties View Appearance Tab

Use the Properties view **Appearance** tab to define attributes that control the appearance of a given node in the web client.

The **Appearance** tab is rendered differently based on whether the behavior is defined for an element or a structure. For an element, all fields are enabled except **Locate in**, which deals with page layouts for structures. For a structure, all fields are disabled except **Locate in**.

Field	Use
Appearance	Select a value to control the appearance of the node based on one of the following selections: • Default: Uses the default node appearance • Compact: A fixed number of choices are rendered, with scrolling facilities, as needed • Full: All choices are rendered at all times • Minimal: A minimum number of choices are rendered, with the ability to temporarily render additional choices
Locate in	Specifies the page on which the group is placed. *New Page* locates the group in a newly created page, without a title.

Field	Use
Place on New Line	Select to place the node at the start of a new line and to define a condition under which the line break is inserted.
Make it a Password Field	Select to protect the contents of nodes containing sensitive information and to define a condition under which the field is secret.

Related Topics

Defining Style Behavior Properties

Working with Behaviors

Style Behavior Properties View Layout Tab

Use the Properties view **Layout** tab to define attributes that control the appearance of a given node in the web client.

The **Layout** tab is rendered differently based on whether the behavior is defined for an element or a structure. For an element, all fields are disabled because the tab deals with page layouts for structures. For a structure with child elements, all fields are enabled.

Field	Use
None	Select to specify no layout. This is the default.
Page Layout	Select to specify how to organize a group's child nodes into tabbed pages. When you select this option, you can also enter a name and language for the layout, as well as a brief description. To use this layout, the node must be a complex type. A complex type element is an element that contains other elements or attributes.
Table Layout	Select to place the group in a table format. Child nodes within the group are represented by columns, and instances of the group are represented by rows. Columns are displayed from left to right in the same order that they appear from top to bottom in a view that does not use the table layout. You can choose to hide a child node so that it does not appear in the table. To use this layout, the node must be a complex type and have children. A complex type element is an element that contains other elements or attributes.

Related Topics

Defining Style Behavior Properties

Working with Behaviors

Style Behavior Properties View CSS Style Tab

Use the Properties view **CSS Style** tab to define attributes that control the appearance of a given node in the web client.

> **Note:**
>
> If you define a CSS Class for a Style behavior, the CSS class must exist when deploying OSM. Otherwise, WebLogic Server throws an error and the default class is used.

Field	Use
CSS Style Attribute	Enter a CSS style attribute to change the field value format of the data node associated with the style behavior. For example, to customize the color and background color of the field value, you might enter: `color:#EE7500;BACKGROUND-COLOR: #FFFFDD`
CSS Class Name	Enter a CSS class name defined in a **customPrint.css** or **customScreen.css** file to change the field value format of the data node associated with the style behavior.
Label CSS Style Attribute	Enter a CSS style attribute to change the field label format of the data node associated with the style behavior. For example, to customize the color and background color of the field label, you might enter: `color:#EE7500;BACKGROUND-COLOR: #FFFFDD`
Label CSS Class Name	Enter a CSS class name defined in a **customPrint.css** or **customScreen.css** file to change the field label format of the data node associated with the style behavior.

Related Topics

Defining Style Behavior Properties

Working with Behaviors

12

Working with Jeopardy and Event Notifications

There are two basic types of notifications that you can configure in Design Studio: jeopardy notifications and event notifications:

- Use jeopardy notifications when certain conditions arise in an order or task and you want to alert users or systems of processes, orders, or tasks that may be at risk.
- Use event notifications to alert users of changes to order milestones or task states.

When modeling notifications, see the following topics:

- Working with Jeopardy Notifications
- Working with Event Notifications
- Order Jeopardy Editor

Working with Jeopardy Notifications

A jeopardy notification is a message that you can configure in Design Studio to occur under specific conditions, and to be sent to specific users or systems. You can configure jeopardy notifications to be sent once, periodically, or when certain conditions arise in an order or task to alert users or systems of processes, orders, or tasks that may be at risk.

Jeopardies are rule-based. When you create a new jeopardy notification, you select a predefined rule that must evaluate to true before OSM can trigger the notification.

When working with jeopardy notifications, see the following topics:

- Creating Jeopardy Notifications in the Order Jeopardy Editor
- Creating Jeopardy Notifications in the Task or Order Editor

Creating Jeopardy Notifications in the Order Jeopardy Editor

Create jeopardy notifications using the Order Jeopardy editor when you want the most flexibility in defining the jeopardy conditions to alert users or systems of orders that may be at risk.

To create a jeopardy notification:

1. From the **Studio** menu, select **New**, select **Order and Service Management**, select **Order Management**, then select **Order Jeopardy**.

   The Order Jeopardy wizard is displayed.
2. In the **Project** field, select the project in which to save the order jeopardy.
3. In the **Name** field, enter a name for the order jeopardy.

   The name must be unique among order jeopardy entity types within the same namespace.
4. In the **Namespace** field, select an existing namespace or enter a unique namespace in which to include the order jeopardy.

   Design Studio uses the last saved namespace as the default.

5. (Optional) Select a location for the order jeopardy.

   By default, Design Studio saves the entity to your default workspace location. You can enter a folder name in the **Folder** field or select a location different from the system-provided default. To select a different location:

   a. Click the **Folder** field **Browse** button.

   b. Navigate to the directory in which to save the entity.

   c. Click **OK**.

6. Click **Finish**.

   Design Studio adds the new order jeopardy to the Studio Projects view and opens the new entity in the Order Jeopardy editor.

7. In the **Details** tab, configure the required fields, which are **Target Order** and **Roles**, as well as any optional fields appropriate for your situation.

   For more information about the fields in this tab, see "Order Jeopardy Editor Details Tab".

8. In the **Policy** tab, configure the conditions under which the jeopardy should be raised.

   For more information about the fields in this tab, see "Order Jeopardy Editor Policy Tab".

9. (Optional) In the **Automation** tab, configure the information about any automation that should be triggered when the jeopardy notification is raised.

   For more information about the fields in this tab, see "Order Jeopardy Editor Automation Tab".

Related Topics

Order Jeopardy Editor

Creating Jeopardy Notifications in the Task or Order Editor

Create jeopardy notifications when certain conditions arise in an order or task and you want to alert users or systems of processes, orders, or tasks that may be at risk.

You can also associate rules to jeopardy notifications that can trigger automations.

> **Note:**
>
> For order jeopardy notifications, the Order Jeopardy editor provides more functionality than jeopardy notifications configured in the Order editor. For more information, see "Creating Jeopardy Notifications in the Order Jeopardy Editor".

To create a jeopardy notification:

1. In the Order editor or Task editor, click the **Jeopardy** tab.
2. Under the Jeopardy column, click **Add**.

   The Jeopardy wizard is displayed.

3. In the **Name** field, enter a name for the jeopardy.

   The name must be unique among notification entities in the same namespace.

4. In the **Rule** field, select the rule that must evaluate to true before OSM can trigger this jeopardy notification.

   Design Studio uses the **null_rule** as the default value. Unless you select a different rule, OSM triggers this notification at run-time at the specified polling interval whenever the specified jeopardy conditions are met. See "Defining Order Rules" for more information about setting up new rules.

5. In the **Priority** field, select a priority for the notification.

   1 is the highest priority. OSM evaluates jeopardies with the highest priority first. For notifications that are sent to external systems, this field represents the JMS queue priority.

6. Select **Enabled**.

   You can deselect the Enable option if you want to include the rule in the cartridge but disable the rule at run-time. For example, you might use this feature during design or testing phases, or if you intend to implement the notification at a later time.

7. Specify whether to send the notification to specific email accounts.

   By default, notifications appear in the Notifications page of the Task web client. However, you can specify that notifications be sent to a user's email account by selecting the **Email** check box. You associate users with email accounts in Administrator.

   See *OSM Order Management Web Client User's Guide* for information about configuring email notification properties for user roles.

8. Click **Next**.

9. Specify the conditions under which the jeopardy notification should be triggered.

10. Click **Next**.

11. Select the roles to be notified when the jeopardy is triggered.

    See "Working with Roles" for more information.

12. Click **Next**.

13. Specify how often the OSM server should re-evaluate the jeopardy condition.

14. Click **Finish**.

    The jeopardy notification is added to the order or task, as appropriate.

    You can edit or add any jeopardy notification attributes at any time by navigating to the Jeopardy subtabs.

15. In the order Jeopardy column, select the jeopardy for which you want to add an automation plug-in.

16. In the **Automation** column, click **Add**.

    The Add Automation dialog box is displayed, which enables you to create the automation plug-in.

17. Enter a name for the automation plug-in, select the appropriate Automation Type from the available list, and click **OK**.

    The newly created plug-in is displayed in the **Automation** column.

18. Select the automation plug-in.

    When you select the automation plug-in, Design Studio displays its Properties tab. Use the subtabs to configure the plug-in. The tabs will vary depending on the type of plug-in selected.

    See "Configuring Automation Plug-In Properties" for more information.

19. For Order jeopardy, if the automation plug-in has an OSM user in the automation plug-in **Run As** property field with more than one role, click **Select** from the **View** field to choose a query task view to use for the automation plug-in. If only one default query task exists in the Order editor Permissions tab, then Design Studio automatically associates it with new automations.

Related Topics

Order Editor Jeopardy Tab

Task Editor Jeopardy Tab

Working with Orders

Working with Tasks

Working with Event Notifications

Event notifications are based on changes to order milestones or task states. The type of the event notification determines where you configure the notification, whether it can be sent to a work group, or whether it will be automatically consumed by an automation plug-in. There are three types of event notifications:

- Task status-based event notifications are triggered by rules that evaluate when a task transitions to a specific status or exception path within a process. You define task status event notifications using the Properties Events tab in the Process editor. See "Creating Task Status-Based Event Notifications" for more information.
- Order milestone and task state automation event notifications are triggered when an order transitions to a specific milestone or when a task transitions to a specific state. When you configure a milestone-based or state-based event notification for an order in the Order editor or for a task in the Task editor, the notification triggers automatically upon reaching the specified order milestone or task state, and the notification is consumed by an automation plug-in that performs the work. When you configure a task state-based event notification for a specific task in the Process editor, you can also include an additional rule that must evaluate to true before OSM triggers the notification, and you can elect to send the notification to a work group.

  See "Creating Order Milestone and Task State Automation Event Notifications" and "Creating Process-specific Task Event Notifications" for more information.
- order data changed notifications are triggered by changes made to the order data. You can configure order data changed notifications to update external systems (such as CRM) with status updates when a specific data node in the order data is updated with a new value. You configure order data changed notifications in the Order editor Notifications tab. See "Creating Order Data Changed Notifications" for more information.

> **Note:**
>
> Order data changed notifications are not available when using releases prior to OSM 7.0.

Related Topics

Working with Jeopardy and Event Notifications

Working with Orders

Creating Order Milestone and Task State Automation Event Notifications

You create order milestone and task state automation event notifications at the order or task level. You select the order milestone (in the Order editor Events tab) or the task state (in the Task editor Events tab) that triggers the automation, and then configure the automation plug-in that will perform the work.

For example, when a task reaches the Assigned state, you can automate an external lookup before allowing the workflow to continue.

To create an order milestone or task state automation event notification:

1. Determine the level at which to create the event notification.

   For example, if you want the notification to trigger when the order reaches the completion milestone, define the notification at the order level. To create an event at the order level, navigate to the **Order editor Events** tab. To create an event at the task level, navigate to the **Task editor Events** tab.

2. From the Order editor or the Task editor **Events** tab, click **Add** in the Milestone or State column, respectively.

   A Selection dialog box opens, which lists all of the milestones or states that have been defined for this order or task.

3. Select a milestone or task from the list of options.

   For order events, the milestones include **Completion**, **Creation**, **Deletion**, **Exception**. For task events, all of the states that you have defined on the **Task editor States/Statuses** tab appear in the list of options.

4. Click **OK**.

5. In the order event Milestones column or the task event States column, select the milestone or state for which you want to add an automation plug-in.

6. In the **Automation** column, click **Add**.

   The Add Automation dialog box is displayed, which enables you to create the automation plug-in.

7. Enter a name for the automation plug-in, select the appropriate action from the available list, and click **OK**.

   The newly created plug-in is displayed in the **Automation** column.

8. Select the automation plug-in.

   When you select the automation plug-in, Design Studio displays its Properties tab. Use the subtabs to configure the plug-in. The tabs will vary depending on the type of plug-in selected.

   See "Configuring Automation Plug-In Properties" for more information.

9. For Order Events, if the automation plug-in has an OSM user in the automation plug-in **Run As** property field with more than one role, click **Select** from the **View** field to choose a query task view to use for the automation plug-in. If only one default query task exists in the Order editor Permissions tab, then Design Studio automatically associates it with new automations.

Related Topics

Task Editor Events Tab

Order Editor Events Tab

Working with Event Notifications

Working with Orders

Working with Tasks

Creating Process-specific Task Event Notifications

You can configure task state-based event notifications for all instances of task in a specific process or for a single instance of a task in the process. When you configure a state-based event notification for a specific task in the Process editor, you can include rules which must evaluate to true before the notification is triggered. When the task reaches the specified state, OSM evaluates the rule to determine whether the event notification is triggered.

To create process-specific task event notifications:

1. From the Process editor, select the task to which the event applies.

   The Properties tab for the selected task opens.
2. Click the **Events** tab.
3. Click **Add**.

   The Event wizard is displayed.
4. In the **Name** field, enter the mnemonic for the task event.

   The name must unique among the notification types within the same namespace.
5. In the **Display Name** field, enter the name of the task event that should be displayed to users.
6. In the **Rule** field, select the rule that must evaluate to true before OSM can trigger this event notification.

   The null_rule is the default value for this field. If you do not change the default value, the OSM server will always trigger this notification when the corresponding task reaches the specific state. See "Defining Order Rules" for information about setting up new rules.
7. In the **Priority** field, select a priority for the notification.

   1 is the highest priority. For notifications that are sent to external systems, this field represents the JMS queue priority.
8. Select **Enabled**.

   Deselect this option if you intend to implement the task event notification at a later time.
9. Specify whether to send the notification to specific email accounts.

   By default, notifications appear in the Notifications page of the Task web client. However, you can specify that notifications be sent to a user's email account by selecting the **Email** check box. You associate users with email accounts in Administrator. See *OSM Order Management Web Client User's Guide* for information about configuring email notification properties for user roles. See *OSM Installation Guide* for information about configuring the outgoing email server.
10. In the **State** field, specify the state that the task must be in before OSM evaluates the rule associated with the event.

    The three mandatory states (accepted, completed, received) and all custom states that you defined on the Task editor States/Statuses tab appear as values.

11. Click **Next**.
12. Select the roles to be notified when the event is triggered.

    See "Working with Roles" for more information.
13. Click **Finish**.

    The event notification is added to the event table.
14. Select the event.

    When you select the event, Design Studio activates the Event subtabs. You can add any undefined elements at any time by using these subtabs. See the following topics for defining the values in the Events subtabs:

    - Properties Events Detail Tab
    - Properties Events Notify Roles Tab
    - Properties Events Automation Tab
    - Event Properties Notes Tab

Related Topics

Working with Event Notifications

Working with Processes

Properties Events Detail Tab

Use the **Properties Events Details** tab to name task state-based and task status-based event notifications, specify the rule that triggers the event, set the priority level, enable or disable the event, and specify whether to send the notification by email.

Field	Use
Name	Enter the mnemonic for the notification.
Display Name	Enter the name to be displayed to the user.
Rule	Select the rule the system should evaluate before generating this notification. This field defaults to the system-based *null_rule*. If you do not change the default value, OSM will always trigger this notification when the task reaches the specified state or when the task transitions to the specified status. See "Defining Order Rules" for more information about setting up new rules.
Priority	Enter a priority from 1 to 255 (1 is the highest priority). The notification with the highest priority is evaluated first.
Enabled	Select to enable this notification, or deselect the option if you intend to implement the notification at a later time.

Field	Use
Email	Select to send email notifications to all users in the workgroup associated with the specified role. By default, notifications appear in the Notifications page of the Task web client. However, you can specify that notifications be sent by email by selecting the **Email** check box. When you assign users to a workgroup in the OSM Administration area of the Order Management web client, you can set up OSM to notify users by email. When a notification occurs, the system sends a notification ID number through email. See *OSM Order Management Web Client User's Guide* for information about configuring email notification properties for user roles.
State	Appears for task entities only. Select the state to which the task must be in before OSM evaluates the rule associated with the event. The three mandatory states (accepted, completed, received) and all custom states that you defined on the Task editor States/Statuses tab appear as values.

Related Topics

Creating Task Status-Based Event Notifications

Creating Process-specific Task Event Notifications

Working with Processes

Working with Event Notifications

Properties Events Notify Roles Tab

Use the Properties Events Notify Roles tab to select the roles to be notified when the event occurs.

Select a predefined notification from the list in the left column to activate a list of available roles. See "Working with Roles" for information about defining roles. Using the directional arrow buttons, move the roles (those groups to whom you want the notification sent) into the Selected Column.

If the notification message will be consumed by an automation plug-in, ensure that you include the role whose credentials are used when running the automation plug-in. See "Working with Automated Tasks" for more information.

Related Topics

Creating Task Status-Based Event Notifications

Creating Process-specific Task Event Notifications

Working with Processes

Working with Event Notifications

Properties Events Automation Tab

Use the **Properties Events Automation** tab to configure an automation plug-in that performs the work or sends data to an external system when the notification is triggered.

Field	Use
Add	Click to open the Add Automation dialog box opens.
Name	Enter a name for the automation entry.
Action	Select the automation plug-in type from the available list. Click **OK** to add the automation entry to the Automation table.
Properties	Select any entry in the table and click to define the automation properties. See "Configuring Automation Plug-In Properties" for information about defining automation properties on the Properties tab.

Related Topics

Creating Task Status-Based Event Notifications

Creating Process-specific Task Event Notifications

Working with Processes

Working with Event Notifications

Event Properties Notes Tab

Use the **Event Properties Notes** tab to denote the intended use of the event, or any additional information that you want to append to the event configuration.

Related Topics

Creating Task Status-Based Event Notifications

Creating Process-specific Task Event Notifications

Working with Processes

Working with Event Notifications

Creating Task Status-Based Event Notifications

Task status-based event notifications are triggered by rules that evaluate when a task moves to a specific status or exception within a process. For example, you might define a failure status that prompts an evaluation against a rule that, when evaluating to true, generates a notification to your fallout specialist. See "Defining Order Fallout" for more information.

You can define task status-based event notifications using the Properties Events tab in the Process editor. When you create a task status-based event notification, the notification applies to a single task flow or exception path.

To create a task status-based event notification:

1. From the Process editor, select the flow or exception path to which the event applies.

   The Properties tab for the selected transition opens.
2. Click the **Events** tab.
3. Click **Add**.

   The Event wizard is displayed.

4. In the **Name** field, enter a name for the task status-based event.

   Ensure that the name is unique among the notification entity types. Two notifications cannot share the same name.

5. In the **Rule** field, select the rule that must evaluate to true before OSM can trigger this event notification.

   The null_rule is the default value for this field. If you do not change the default value, the OSM server will always trigger this notification when the task transitions to the specified status or exception path. See "Defining Order Rules" for more information about setting up new rules.

6. In the **Priority** field, select a priority for the notification.

   1 is the highest priority. For notifications that are sent to external systems, this field represents the JMS queue priority.

7. Select **Enabled**.

   Deselect this option if you intend to implement the task status-based event notification at a later time.

8. Specify whether to send the notification to specific email accounts.

   By default, notifications appear in the Notifications page of the Task web client. However, you can specify that notifications be sent to a user's email account by selecting the **Email** check box. You associate users with email accounts in the Administration area of the Order Management web client. See *OSM Order Management Web Client User's Guide* for information about configuring email notification properties for user roles.

9. Click **Next**.

10. Select the roles to be notified when the event is triggered.

    See "Working with Roles" for more information.

11. Click **Finish**.

    The event notification is added to the event table.

12. Select the event.

    When you select the event, Design Studio activates the Event subtabs. You can add any undefined elements at any time by using these subtabs. See the following topics for defining the values in the Events subtabs:

    - Properties Events Detail Tab
    - Properties Events Notify Roles Tab
    - Properties Events Automation Tab
    - Event Properties Notes Tab

Related Topics

Working with Processes

Working with Event Notifications

Creating Order Data Changed Notifications

Order data changed notifications are triggered by changes to the order data. You can configure order data changed notifications to update external systems (such as CRM) with status updates when a specific data node in the order data is updated with a new value.

> **Note:**
>
> This feature is not available when using releases prior to OSM 7.0.

You can create order data changed notifications at the order level from the **Order editor Notifications** tab.

To create order data changed notifications:

1. In the Order editor, click the **Notifications** tab.
2. Under the Event Notifications column, click **Add**.

   The Event wizard is displayed, where you can select conditions for the notification and the roles to be notified. You can define this information in the wizard, or later by using the Notifications subtabs.
3. In the **Name** field, enter a name for the notification.

   Ensure that the name is unique among the notification entity types. Two notifications cannot share the same name.
4. In the **Priority** field, select a priority for the notification.

   1 is the highest priority. For notifications that are sent to external systems, this field represents the JMS queue priority.
5. Select **Enabled**.

   Deselect this option if you intend to implement the notification at a later time.
6. Specify whether to send the notification to specific email accounts.

   You can specify that notifications be sent to a user's email account by selecting the **Email** check box. You associate users with email accounts in the Administration area of the OSM Order Management web client application. See *OSM Order Management Web Client User's Guide* for information about configuring email notification properties for user roles.

   > **Note:**
   >
   > Order data changed notifications are intended to update external systems with status updates when a specific data node in the order data is updated with a new value. The OSM server does not send order data changed event notifications to Task web client Notifications pages. When notifying users, the server sends these notifications to email addresses only.
7. Click **Next**.
8. Select the roles to be notified when the notification is triggered.

   See "Working with Roles" for more information.
9. Click **Finish**.

   The notification is added to the order.

   You can edit or add any notification attributes at any time by navigating to the Notifications subtabs.
10. In the **Automation** column, click **Add**.

The Add Automation dialog box is displayed, which enables you to create the automation plug-in.

11. Enter a name for the automation plug-in, select the appropriate Automation Type from the available list, and click **OK**.

    The newly created plug-in is displayed in the **Automation** column.

12. Select the automation plug-in.

    When you select the automation plug-in, Design Studio displays its Properties tab. Use the subtabs to configure the plug-in. The tabs will vary depending on the type of plug-in selected.

    See "Configuring Automation Plug-In Properties" for more information.

13. For Order data change notifications, if the automation plug-in has an OSM user in the automation plug-in **Run As** property field with more than one role, click **Select** from the **View** field to choose a query task view to use for the automation plug-in. If only one default query task exists in the Order editor Permissions tab, then Design Studio automatically associates it with new automations.

14. Click the **Data Changed** subtab.

15. Click the **Add**.

    The Order Template Node Selection dialog box is displayed.

16. Select a data node that, when updated with a new value, will trigger the notification.

17. Click **OK**.

    The data node is added to the Nodes table.

18. Click **Save**.

Related Topics

Order Editor Notification Tab

Working with Orders

Working with Automated Tasks

Order Jeopardy Editor

Use the Order Jeopardy editor to model jeopardy conditions.

The following fields are common to all Order Jeopardy editor tabs:

Field	Use
Description	Edit the display name of the order jeopardy.
Namespace	Select an existing namespace or enter a unique namespace in which to include the order jeopardy. Design Studio uses the last saved namespace as the default.

When working with the Order Jeopardy editor, see the following topics:

- Order Jeopardy Editor Details Tab
- Order Jeopardy Editor Policy Tab
- Order Jeopardy Editor Automation Tab

Order Jeopardy Editor Details Tab

Use the Order Jeopardy editor **Details** tab to define the conditions under which the jeopardy will be evaluated. The following table describes the fields on the Order Jeopardy editor **Details** tab.

> **Note:**
>
> See "Order Jeopardy Editor" for information about fields that appear on all of the Order Jeopardy editor tabs.

Field	Use
Operational	Select this option to indicate that the primary configuration for this jeopardy is to be in a file in the system where OSM is running. When you select this option, you should configure the jeopardy in the normal manner. However, only the automation information for this configuration (and the automation plug-in, if any) is deployed to OSM with your cartridge. For the rest of the configuration, the next time the cartridge is built, a sample configuration file corresponding to your configuration of the jeopardy will be generated and placed in the **samples/orderJeopardy** folder for your cartridge. You can see this folder in the Package Explorer view. You can then copy or move the file to the server where OSM is running. You use the **oracle.communications.ordermanagement.order.OperationalOverrideFileURLs** parameter in the **oms-config.xml** file to indicate the location of the file you have configured. For more information about configuring operational jeopardy files on the OSM server, see *OSM System Administrator's Guide*.
Target Order	Click **Select** next to this field to select the order to which this order jeopardy will apply.
View	Click **Select** next to this field to select the task view the order jeopardy will use to obtain data to be used in the configuration.
Rule	Click **Select** next to this field to select a rule to use to limit when this order jeopardy will be evaluated.
Priority	Enter a priority from 1 to 255 (1 is the highest priority). The notification with the highest priority is evaluated first.
Enabled	Select to enable this jeopardy notification, or deselect if you intend to implement the notification at a later time.
Roles	Select the roles that have permission to respond to this notification. Do any of the following: • Click **Select** to select an existing role. • Select a role and click **Remove** to remove the role from the list for this order jeopardy. • Click **Add** to create a new role. See "Creating New Roles" for more information. • Select a role and click **Open** to open the role in the Role editor.

Related Topics

Creating Jeopardy Notifications in the Order Jeopardy Editor

Order Jeopardy Editor

Order Jeopardy Editor Policy Tab

Use the Order Jeopardy editor **Policy** tab to define the order states and duration for the timer. The following table describes the fields on the Order Jeopardy editor **Policy** tab.

> **Note:**
>
> See "Order Jeopardy Editor" for information about fields that appear on all of the Order Jeopardy editor tabs.

When configuring order jeopardy policy timer duration, see also the following topics:

- Order Jeopardy Editor Policy Tab Duration Value Subtab
- Order Jeopardy Editor Policy Tab Offset Subtab
- Order Jeopardy Editor Policy Tab XQuery Expression Subtab
- Order Jeopardy Editor Policy Tab Unit Type and Default Value Subtab
- Order Jeopardy Editor Policy Tab Data Path Expression Subtab

Field	Use
Start Condition	Specify the order states that, when the order enters one of them, should start the jeopardy timer. This should be set if the value being returned in the Timer Duration area is a duration, rather than a date/time. If this value is not set, OSM will expect a date/time value to be returned from the configuration in the Timer Duration area. Since the **Specify a Duration Value** and **Use the Order Expected Duration** options always return a duration, this value should always be set if either of those options is selected. Do any of the following: • Click **Select** to add one of the available order states to the list of states. • Select one of the states in the list and click **Remove** to remove the state from the list of states. • Select one of the states in the list and click **Open** to open the Order State editor for that order state.
Pause Timer and Block Jeopardy	Specify the order states that, when the order enters one of them, should pause the jeopardy timer. This should not be set if **Start Condition** does not have a value. If **Start Condition** has a value, this field is optional. Do any of the following: • Click **Select** to add one of the available order states to the list of states. • Select one of the states in the list and click **Remove** to remove the state from the list of states. • Select one of the states in the list and click **Open** to open the Order State editor for that order state.

Field	Use
Stop Condition	Specify the order states that, when the order enters one of them, should stop the jeopardy timer. This value should always be set. Do any of the following: • Click **Select** to add one of the available order states to the list of states. • Select one of the states in the list and click **Remove** to remove the state from the list of states. • Select one of the states in the list and click **Open** to open the Order State editor for that order state.
Timer Duration	Sets the time after which a jeopardy will be raised. Do one of the following: • Select **Specify a Duration Value** to specify a specific duration before a jeopardy notification is raised. • Select **Use the Order Expected Duration** to use the expected duration of the order to determine when a jeopardy notification is raised. • Select **Specify an XQuery Expression to return a duration or date/time** to provide an XQuery expression to determine the duration before a jeopardy notification is raised. The XQuery expression can return either a duration or a date/time value. • Select **Specify a Data Path Expression to contain a duration or date/time** to provide the path to a data element on the order that will contain the duration before a jeopardy notification is raised. The data path expression can return either a duration or a date/time value.

Related Topics

Creating Jeopardy Notifications in the Order Jeopardy Editor

Order Jeopardy Editor

Order Jeopardy Editor Policy Tab Duration Value Subtab

Use the Order Jeopardy editor **Policy** tab, **Duration Value** subtab to set the details of the duration timer. This subtab is available if the **Specify a Duration Value** option is selected in the Order Jeopardy editor **Policy** tab, Timer Duration area.

Field	Use
Duration Amount	Enter the number of duration units to wait before raising a jeopardy on the order.
Duration Unit	Select the appropriate units for the duration value from the list.

Related Topics

Order Jeopardy Editor Policy Tab

Order Jeopardy Editor Policy Tab Offset Subtab

Use the Order Jeopardy editor **Policy** tab, **Offset** subtab to set the details of an offset to the calculated duration. This subtab is available for all timer duration options.

Field	Use
Apply an offset or use a percentage of the duration	Select this option to provide an offset using the other fields in this subtab.
Percentage	Select this option to configure the offset in terms of the percentage of the duration. For example, you could raise a jeopardy when 90% of the order's expected duration has passed.
Add	Select this option to add a fixed amount of time to the duration.
Subtract	Select this option to subtract a fixed amount of time from the duration.
Offset/Percentage Amount	If you have selected **Percentage**, enter a number between 1 and 100 to indicate the percentage of the total duration that should elapse before the jeopardy is raised. If you have selected **Add** or **Subtract**, enter the number of offset units to add or subtract from the duration.
Offset Unit	Set the units for the offset amount. This field is not available when the **Percentage** option is selected.

Related Topics

Order Jeopardy Editor Policy Tab

Order Jeopardy Editor Policy Tab XQuery Expression Subtab

Use the Order Jeopardy editor **Policy** tab, **XQuery Expression** subtab to use an XQuery expression to determine the duration. This subtab is available if the **Specify an XQuery Expression to evaluate the duration** option is selected in the Order Jeopardy editor **Policy** tab, Timer Duration area.

Field	Use
Expression	Enter the XQuery expression to use to determine the duration.
Data Changed area	Select **Once when the timer starts** to evaluate the expression once only. Select **When any of the following nodes change** to evaluate the expression when the timer starts and also when any of the specified order nodes change. When this option is selected, you should also provide a list of data nodes by doing any of the following: • Click **Select** to select the data node from a list of available nodes. • Select a node in the list and click **Remove** to remove the data node from the list. • Select a node in the list and click **Open** to open the **Order Template** tab of the Order editor with the data node selected. If no data nodes are provided and **When any of the following nodes change** option is selected, the expression is only evaluated when the timer starts.

Related Topics

Order Jeopardy Editor Policy Tab

Order Jeopardy Editor Policy Tab Unit Type and Default Value Subtab

Use the Order Jeopardy editor **Policy** tab, **Unit Type and Default Value** subtab to set the unit type of the duration and a default value if the value is not found. This subtab is available if the

Specify an Xquery Expression to evaluate the duration option or the **Specify a Data Path Expression to evaluate the duration** option is selected in the Order Jeopardy editor **Policy** tab, Timer Duration area.

Field	Use
Expression Units	Select the units of measure used in the returned duration value.
Default	Enter the default value to be used if the XQuery expression or data path do not return a valid value. The value in this field is expressed in the units selected in the **Expression Units** field.

Related Topics

Order Jeopardy Editor Policy Tab

Order Jeopardy Editor Policy Tab Data Path Expression Subtab

Use the Order Jeopardy editor **Policy** tab, **Data Path Expression** subtab to set the details of the duration jeopardy timer. This subtab is available if the **Specify a Data Path Expression to evaluate the duration** option is selected in the Order Jeopardy editor **Policy** tab, Timer Duration area.

Field	Use
Data Path	Select the data node that contains the duration. Do any of the following: • Click **Select** to select the data node from a list of available nodes. • Click **Remove** to remove the data node from the field. • Click **Open** to open the **Order Template** tab of the Order editor with the data node selected.

Related Topics

Order Jeopardy Editor Policy Tab

Order Jeopardy Editor Automation Tab

Use the Order Jeopardy editor **Automation** tab to configure an automation that is triggered by the jeopardy. The following table describes the fields on the Order Jeopardy editor **Automation** tab.

> **Note:**
>
> See "Order Jeopardy Editor" for information about fields that appear on all of the Order Jeopardy editor tabs.

When configuring order jeopardy automations, see also the following topics:

- Order Jeopardy Editor Automation Tab Details Subtab
- Order Jeopardy Editor Automation Tab Script Subtab
- Order Jeopardy Editor Automation Tab Routing Subtab
- Order Jeopardy Editor Automation Tab Notes Subtab

Field	Use
Trigger Automation when Jeopardy is Raised	Select this option to cause an automation plug-in to be called when the order jeopardy is raised.
Automation Type	Select the automation plug-in type from the available list. See "Working with Automation Plug-Ins" for more information about the different automation types.
Custom Automation Plugin	If you selected Custom Automation in the Automation Type field, enter the name of the custom plug-in.

Related Topics

Order Jeopardy Editor

Working with Automation Plug-Ins

Order Jeopardy Editor Automation Tab Details Subtab

Use the Order Jeopardy editor **Automation** tab, **Details** subtab to provide information about the automation to trigger if the order jeopardy is raised.

Field	Use
Name	Enter a plug-in name. The name must be unique among plug-in entities in the same namespace. **Note:** While plug-in names can be any arbitrary name that you assign to the automation, Oracle recommends that you use a consistent naming pattern for all of the automations that you create.
Run As	Enter the OSM user name (security principal) whose credentials are used to process this automation plug-in. A password is not necessary to authenticate this user because only an administrator has the authority to deploy components into the server. The value of this field must reflect the user ID that is used to run the automation: • The user ID must be set up in the WebLogic Server Administration console. See the discussion of setting up security in *OSM System Administrator's Guide* for more information. • The user ID must be defined in the OSM Administration area of the Order Management web client (a workgroup in OSM Administration is a role in Design Studio). See "Working with Roles" for more information. • The user ID must be assigned to the workgroup in the OSM Administration area of the Order Management web client that corresponds to the role defined on the **Permissions** tab of the Design Studio task, order, or process that defines the automation. **Note:** Oracle recommends using the DEFAULT_AUTOMATION_USER cartridge model variable in the **Run As** field. See "Defining Model Variables" for more information.

Related Topics

Order Jeopardy Editor Automation Tab

Order Jeopardy Editor Automation Tab Script Subtab

Use the Order Jeopardy editor **Automation** tab, **Script** subtab, to provide information about an XQuery script to make available to the automation.

Field	Use
Script	Specify which method to use to retrieve the XQuery file. Select from the following options: • **Absolute path**: Select this option and enter the physical location of the file. At run time, OSM retrieves the file from the server. • **URL**: Specify a URL to access the file. • **Bundle in**: Select this option, then click the **XQuery** field **Select** button to identify the file from the **resources** directory. Design Studio will bundle this file with the PAR file during the build. **Note:** Oracle recommends that you select **Bundle in** for production mode, as this mode pulls the files into the OSM PAR file. As a result, you can deploy the EAR file (which contains the PAR file) to any server and, at run time, the application can locate the files. If you select **Absolute Path** or **URL** for production mode, you can deploy the EAR file (which contains the PAR file) to any server but are responsible for ensuring the files reside in specified location on the server. Conversely, **Absolute Path** or **URL** are optimal for testing mode because they do not require a rebuild and redeploy to pick up changes to the XQuery. For information about the XQuery file referenced here, see "Order Jeopardy Automation XQuery Plug-ins".
Maximum Number of Stylesheets in Cache	Specify the maximum number of XQuery style sheets that can be maintained in the cache at any one time.
Cache Timeout in Seconds	Enter the number of seconds before the OSM server refreshes the cache.
Transformer Factory	If you have developed a custom TransformerFactory for XSLT transformation, specify the location. Design Studio provides a default TransformerFactory.
Update Order	Select this option if you want to update (add, change, or delete) the OSM order data with the data retrieved from an external system. This field appears for Automator automation plug-ins only.

Related Topics

Order Jeopardy Editor Automation Tab

Order Jeopardy Editor Automation Tab Routing Subtab

Use the Order Jeopardy editor **Automation** tab, **Routing** subtab, to specify where to send XML messages and where external systems can deliver responses.

In the **To** area, you can specify where to send the request message. In the **Reply To** area, you can specify where the external system can deliver the response or exception message.

Field	Use
JNDI Name	Enter the name of the queue to which the automation plug-in sends messages (on the **To** tab) or to which external systems send response (on the **Reply To** tab). **JNDI Name** is mandatory. Edit the system-supplied default value to reflect your own system topology. The JNDI name must be unique in the workspace.
Destination Type	Select the type of the message destination. A JMS destination is either a javax.jms.Queue or a javax.jms.Topic. You might use a topic, for example, if you want to publish messages for general availability to multiple external systems (on the **To** tab) or subscribe to a queue with multiple listeners (on the **Reply To** tab). You might use queues if you want only a single external system to consume the message.
Initial Context Factory, **Connection Factory**, and **URL**	Enter this information to connect to an external application server. Specify the URL and the InitialContextFactory class for the JNDI provider, and specify the ConnectionFactory class for the JMS server.
Send Null Message	Select this option if you want to send a JMS message to an external system even if the message body is empty.

Related Topics

Order Jeopardy Editor Automation Tab

Order Jeopardy Editor Automation Tab Notes Subtab

Use the Order Jeopardy editor **Automation** tab, **Notes** subtab, to provide information to other Design Studio users about the automation.

Enter the information you wish to provide in the field on this subtab.

Related Topics

Order Jeopardy Editor Automation Tab

13

Packaging and Deploying OSM Cartridges

You package Oracle Communications Order and Service Management (OSM) cartridge projects to control which entities, libraries, and resources to include in the cartridge PAR file. After packaging cartridge projects, you can deploy them to OSM run-time environments.

When packaging and deploying cartridge projects, see the following topics:

- Packaging Order and Service Management Cartridges
- Defining Build-and-Deploy Modes for Automation Plug-ins
- Deploying Cartridge Projects
- Testing OSM Cartridge Models
- Managing Changes to Deployed Cartridges

Packaging Order and Service Management Cartridges

Before you can deploy to the OSM run-time environment, you must determine which entities, libraries, and resources to include in the cartridge. Design Studio enables you to model multiple order types within a single project and deploy those order types to an OSM run-time environment within the context of a single project. For example, if you have defined a project with data to support the DSL services Add, Delete, and Modify for orders that come from 2 different sources (Siebel and Portal, for example), you can deploy the entire configuration to a run-time environment in one deployment cycle.

When packaging OSM cartridge projects, see the following topics:

- Multiple Order Data Inconsistencies
- Building and Packaging Projects

Multiple Order Data Inconsistencies

When you combine multiple orders into a single project and deploy the cartridge to an OSM run-time environment, the OSM server combines the order data from each order into single master order template. Consequently, when packaging cartridges that contain multiple order types, the system automatically detects order data inconsistencies across order types and creates problem markers to identify the conflict. You must resolve all problem marker errors before you can deploy the cartridge.

For example, consider that two orders packaged in the same project both contain the ID data element. In Order 1, the ID data element is defined as a string (intended to be populated by a customer name and set of digits). In Order 2, the ID data element is defined as an integer (intended to be populated by a unique set of digits). In the run time environment, the OSM server has no ability to discern whether to treat the ID data element as a *string* data type or as an *int* data type. In this example, Design Studio would create a problem marker which you must resolve before deploying the cartridge.

Additionally, it is possible to introduce data inconsistencies when you have included the same data element in multiple orders, each defined with conflicting behaviors. For example, it is possible to model in Order 1 a Read Only behavior for the ID data element that evaluates to

true when the field value equals *10001*, while in Order 2, model a Read Only behavior for the ID data element that evaluates to true when the field value does not equal *10001*. Because the OSM server is not able to resolve these types of conflicts, Design Studio detects them and forces you to resolve them prior to deployment.

Related Topics

Packaging and Deploying OSM Cartridges

Defining Build-and-Deploy Modes for Automation Plug-ins

> **Note:**
>
> The information in this topic applies between the OSM 7.0.3 and 7.2.4.x releases. Design Studio uses only the Legacy build-and-deploy mode for releases before OSM 7.0.3, and uses only the Optimized build-and-deploy mode starting in OSM 7.3.

When you deploy cartridges with automation plug-ins to your OSM run-time environment (automation plug-ins for automation tasks as well as for activation tasks), you can define a configuration to have Design Studio build and deploy automation plug-ins in a particular way. Prior to OSM 7.0.3, when you built and deployed a cartridge that included automation plug-ins, OSM ran each automation plug-in in that cartridge in its own separate EAR file; this method of building and deploying automation plug-ins is now referred to as the Legacy build-and-deploy mode. Legacy mode simply refers to the manner in which automation plug-ins were deployed and processed prior to OSM 7.0.3. As of release OSM 7.0.3, you can build and deploy a cartridge in Design Studio using the Optimized build-and-deploy mode, the default mode; this mode improves the performance of processing of automated or activation tasks and improves the performance of build and deployment of cartridges with such tasks.

> **Note:**
>
> The Legacy build-and-deploy mode is deprecated, but it is supported for backward compatibility in OSM server versions before OSM 7.3.

> **Note:**
>
> XML Catalog support is required to be enabled for all cartridges as of release OSM 7.0.3.

Related Topics

About Build-and-Deploy Modes for Automation Plug-ins

Setting Automation Plug-in Build-and-Deploy Modes for Individual Cartridges

Setting Automation Plug-in Build-and-Deploy Modes for All Cartridges

About Build-and-Deploy Modes for Automation Plug-ins

> **Note:**
>
> The information in this topic applies between the OSM 7.0.3 and 7.2.4.x releases. Design Studio uses only the Legacy build-and-deploy mode for releases before OSM 7.0.3, and uses only the Optimized build-and-deploy mode starting in OSM 7.3.

Build-and-deploy modes for automation plug-ins affect how the plug-ins are processed at run time. See "Defining Build-and-Deploy Modes for Automation Plug-ins" for introductory information on build-and-deploy modes. A dispatch mode setting on the OSM server controls the automation plug-in dispatch mode of the OSM system, which is directly related to the build-and-deploy mode you configure in Design Studio. The build-and-deploy mode configured in Design Studio controls building and deploying the automation components required for Legacy mode or Optimized mode; it can also be configured to build and deploy the automation components required for both modes.

- The Optimized mode enables the automation plug-ins to be deployed and processed more efficiently (automation plug-ins can run in a common EAR file).
- The Legacy mode deploys and processes the automation plug-in in a manner consistent with previous OSM releases (each automation plug-in runs in its own EAR file).
- The mode entitled **Both (Allow server preference to decide)** indicates the automation plug-in can run in either Optimized or Legacy mode (Design Studio builds the cartridge with the automation components required for either mode).

If you build and deploy the automation components required for both modes, OSM processes the automation plug-in at run time in the mode specified by the dispatch mode setting on the OSM server. See the discussion on automation plug-in dispatch modes in *OSM Developer's Guide* for information on how to define the dispatch mode setting on the OSM server.

The following table summarizes the effective mode OSM uses at run time based on how the automation plug-in build-and-deploy mode and the server dispatch mode are set as of OSM 7.0.3:

Automation Plug-in Build-and-Deploy Mode (set in Design Studio)	OSM Server Dispatch Mode	Effective Mode Used at Run Time
Optimized	Legacy	Optimized
Optimized	Internal	Optimized
Legacy	Legacy	Legacy
Legacy	Internal	Legacy
Both	Legacy	Legacy
Both	Internal	Optimized

You can set the automation plug-in build-and-deploy mode in two levels:

- Global preference: Set the mode for all cartridges in the same workspace by setting a global preference. To specify a build-and-deploy mode for all cartridges in the same workspace, see "Setting Automation Plug-in Build-and-Deploy Modes for All Cartridges".

- Cartridge-level preference: Set the mode for individual cartridges by setting a cartridge management variable. To specify a build-and-deploy mode for individual cartridges, see "Setting Automation Plug-in Build-and-Deploy Modes for Individual Cartridges".

> **Note:**
>
> The cartridge-level preference overrides the global preference.

Design Studio uses the following build-and-deploy modes when a cartridge is deployed:

- If the **Target Version** field is to **7.3** or later, the Design Studio build-and-deploy mode is set to Optimized and cannot be changed.
- If the **Target Version** field is set to an OSM release between OSM 7.0.3 and OSM 7.2.4; for example, set to **7.0.3**, Design Studio uses the build-and-deploy mode that you set for the individual cartridge (the default mode is **Optimized**) or the mode you set as the global preference.
- If the **Target Version** field is set to **7.0.1** or below, the Design Studio build-and-deploy mode is set to Legacy and cannot be changed.

> **Note:**
>
> Optimized mode is not available when using OSM server releases 7.0.2 or earlier. Legacy mode is not available when using OSM server releases 7.3 or later.

Setting Automation Plug-in Build-and-Deploy Modes for All Cartridges

> **Note:**
>
> The information in this topic applies between the OSM 7.0.3 and 7.2.4.x releases. Design Studio uses only the Legacy build-and-deploy mode for releases before OSM 7.0.3, and uses only the Optimized build-and-deploy mode starting in OSM 7.3.

You can set the build-and-deploy mode of automation plug-ins for all cartridges in the same workspace as a global preference.

To set the automation plug-in build-and-deploy mode for all cartridges:

1. From the **Window** menu, select **Preferences**, then select **Oracle Design Studio**, and then select **Order and Service Management Preferences**.
2. In the **Build and Deploy Mode** field, do one of the following:
   - To build and deploy cartridges so that automation plug-ins are run in a common EAR file (deploys and processes automation plug-ins more efficiently), select **Optimized**.
   - To build and deploy cartridges so that each automation plug-in is run in its own EAR file (deploys and processes the automation plug-in in a manner consistent with previous OSM releases), select **Legacy**.

- To build and deploy the automation components for both Optimized and Legacy modes, select **Both (Allow server preference to decide)**.

  If you set this option, the automation plug-in can process at run time in either Optimized mode or Legacy mode because the automation components required for both are built and deployed. In this case, OSM uses the automation plug-in dispatch mode defined on the OSM server at run time.

Design Studio builds and deploys all cartridges with the mode you specify as a global preference.

> **Note:**
>
> Cartridges for which you have a set a different build-and-deploy mode at the cartridge level will build and deploy in that mode.

> **Note:**
>
> Oracle recommends using Optimized mode because this mode improves the performance of building and deploying cartridges that include automation plug-ins.

3. Click **OK**.

   Design Studio saves your deployment preferences and closes the Preferences dialog box.

Related Topics

Defining Build-and-Deploy Modes for Automation Plug-ins

About Build-and-Deploy Modes for Automation Plug-ins

Setting Automation Plug-in Build-and-Deploy Modes for Individual Cartridges

Defining Order and Service Management General Preferences

Setting Automation Plug-in Build-and-Deploy Modes for Individual Cartridges

> **Note:**
>
> The information in this topic applies between the OSM 7.0.3 and 7.2.4.x releases. Design Studio uses only the Legacy build-and-deploy mode for releases before OSM 7.0.3, and uses only the Optimized build-and-deploy mode starting in OSM 7.3.

If your OSM server is a version between OSM 7.0.3 and 7.2.4.x, you can set the build-and-deploy mode of automation plug-ins for each cartridge in your workspace. Setting the build-and-deploy mode for an individual cartridge overrides the build-and-deploy mode set as a global preference; that is, OSM uses the mode you set at the cartridge level at run time. For OSM 7.3 and later, all cartridges use **optimized** mode.

To set automation plug-in build-and-deploy mode preferences for a specific cartridge in your workspace:

1. From the **Studio** menu, select **Show Design Perspective**.
2. In the Studio Projects view, double-click the Project entity for which you want to set the build-and-deploy mode preference.

   The project opens in the Project editor.
3. Click the **Cartridge Management Variables** tab.
4. Add the **BUILD_DEPLOY_MODE** variable.
5. In the **Default Value** column, do one of the following:
   - To build and deploy this cartridge so that automation plug-ins are run in a common EAR file (deploys and processes automation plug-ins more efficiently), enter **optimized**.
   - To build and deploy this cartridge so that each automation plug-in is run in its own EAR file (deploys and processes the automation plug-in in a manner consistent with previous OSM releases), enter **legacy**.
   - To build and deploy the automation components for both Optimized and Legacy modes, enter **both**.

     If you set this option, OSM uses the automation plug-in dispatch mode defined on the OSM server at run time.
6. Click **Save**.

Related Topics

Setting Automation Plug-in Build-and-Deploy Modes for All Cartridges

Defining Build-and-Deploy Modes for Automation Plug-ins

About Build-and-Deploy Modes for Automation Plug-ins

Project Editor Cartridge Management Variables Tab

Testing OSM Cartridge Models

Design Studio enables you to make changes to OSM cartridges, deploy them to an environment, and review the changes without leaving the Design Studio user interface. Using the Submit Test feature, you can submit sample XML orders to run-time environments for the purpose of reviewing the cartridge model behavior. Once you have deployed the full cartridge to an environment, you can use the Optimize Deploy feature in conjunction with Submit Test to model and test your cartridges efficiently.

When testing OSM cartridge models, see the following topics:

- About Submit Test
- Submitting Test Orders to Run-time Environments

About Submit Test

The Design Studio Submit Test feature enables you to submit a sample XML order to a run-time environment for the purpose of testing your cartridge model. For example, if you were creating recognition rules for an orchestration cartridge, you could submit sample orders to ensure that the OSM server was properly recognizing the input messages for each recognition

rule and directing the order to the right cartridge. Additionally, you could target the sample order to a specific version of a cartridge, test OSM behaviors, and so forth.

When submitting sample orders to run-time environments, the root level of the sample order XML document must be either the CreateOrder or the CreateOrderBySpec XML API request. For example:

```
<?xml version="1.0" encoding="UTF-16"?>
<ord:CreateOrder xmlns:ord="http://xmlns.oracle.com/communications/ordermanagement">
<zeb:order xmlns:zeb="http://www.example.org/zebra" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance">
     <zeb:sampleLine>
          <zeb:lineName>newPhoneLine</zeb:lineName>
          <zeb:class>potsService</zeb:class>
     </zeb:sampleLine>
</zeb:order>
</ord:CreateOrder>
```

You save all sample XML orders in a project **samples** directory, accessible from the Package Explorer view.

Submitting Test Orders to Run-time Environments

The Design Studio Submit Test feature enables you to submit a sample XML order to a run-time environment for the purpose of testing your cartridge model.

To submit test orders to run-time environments:

1. From the Package Explorer view, copy a sample XML order into the project **samples** directory.
2. Connect to an environment.

   See "Testing Run-Time Environment Connectivity" for more information.

   > **Note:**
   >
   > When connecting to the environment, ensure that the account that you use to log-in is set up properly in WebLogic. To submit tests from Design Studio, user accounts must be assigned to the OMS_client and OMS_ws_api groups in WebLogic.

3. If necessary, deploy the full cartridge to the run-time environment.

   See "Deploying Cartridge Projects" for more information.
4. From the **Studio** menu, select **Show Design Perspective**.
5. In the Studio Projects view, right-click a cartridge and select **Submit Test**.

   If you have successfully connected to an environment, and your project contains a valid sample order in the project **samples** directory, Design Studio displays a list of environments and files available to submit.
6. Select an environment and order combination.

   The Console view is displayed, indicating the status of the submitted order. If the connection is successful, the Console view displays the OSM server response, which includes the order ID, version number, the cartridge name, and so forth. Design Studio

opens a browser window in the editor area that points to the log-in window for the environment.

> **Note:**
>
> For the browser window in the editor area of Design Studio to work with OSM, you must configure Eclipse to use a supported external browser. See "Defining Web Browser Preferences" for more information.

> **Note:**
>
> Starting with OSM 7.2, order IDs are allocated in blocks. For OSM running on a standalone database, there is no visible impact. However, if OSM is running on an Oracle RAC database, Order IDs are assigned from different blocks, one for each Oracle RAC instance. This means that when orders are submitted, the Order IDs may not be sequential.

7. Log into the environment.

   The web client displays a list of orders submitted to the environment.

8. Locate the table row that contains your sample order.

   For example, you might locate the row using the order ID displayed in the Console view.

9. Double-click the sample order to open the order and review the order data in the web client tabs.

   For example, if you were submitting a sample order for an orchestration cartridge, you could review the information in the Summary tab, Data tab, Orchestration Plan tab, and so forth.

   For information about the fields in the Order Management web client, see *OSM Order Management Web Client User's Guide.* For information about the fields in the Task web client, see *OSM Task Web Client User's Guide*.

10. (Optional) Update the cartridge with additional changes, save the changes, and use the optimize deploy feature to deploy only the cartridge metadata changes.

    See "Deploying Cartridge Projects with Optimize Deploy" for more information.

11. (Optional) Connect to another environment to test the cartridge model in multiple environments.

    For each environment that you submit tests to, Design Studio opens a separate browser window. See "Testing Run-Time Environment Connectivity" for more information.

12. (Optional) Clear the environment connection information if you want to submit a sample order as a different user.

    From the Studio Projects view, right-click a cartridge and select **Submit Test**, then select an environment, then select **Clear Environment Credentials** to clear the environment connection information.

Managing Changes to Deployed Cartridges

You cannot deploy modified versions of existing deployed cartridges with the same name and version - Design Studio always increments the version number by default. However, if you have small changes to make that do not include changes to the order template or data dictionary, you can override the existing cartridge.

For larger changes to a cartridge, including introducing new tasks, changing data types, or adding new orders, you should deploy a new version of the cartridge. Otherwise, you might encounter errors such as missing or extra nodes when running orders.

If orders are originating in an upstream system, they won't automatically be routed to the new cartridge. Upstream systems must be modified so that new orders specifically target the new version of the cartridge.

OSM performance can be impacted if you have many versions of a cartridge deployed at the same time. For example, you might experience slower Worklist response time. Factors which contribute to possible performance issues when you have multiple cartridge versions include the number of versions, the design complexity of your cartridges including the number of tasks in your orders, the number of OSM users, your hardware and so on.

Managing Orders for Multiple Cartridge Versions

In-progress orders for an existing cartridge are not impacted by the deployment of a new version of that cartridge, nor are they automatically migrated to the new version of the cartridge. For new orders to be created by default against the new version of the cartridge, the Default option (in the Project editor Properties tab) for the cartridge should be selected before it is deployed. Orders can be targeted to previous versions of a cartridge by specifying the version numbers in the order. See the discussion of the Project editor Properties Tab in the OSM Modeling Processes Online help for more details.

It is possible to send a revision order from a newer version of a cartridge against an in-flight order from an older version of the cartridge. The order will process only data that was contained in the original cartridge metadata.

XMLIE provides scripts to migrate orders from one cartridge version to another. For details, see the discussion on migrating orders in *OSM System Administrator's Guide*.

When an older cartridge version is no longer needed, consider removing it from OSM. You may wish to back up completed orders associated with the cartridge before undeploying it because the completed orders may be purged from the system.

To avoid purging orders from an old cartridge, you can create a different cartridge version for modeling changes, then create and process new orders with the new cartridge namespace and version. The old orders are processed with the old cartridge version. You can also disable the creation tasks of the old cartridge to ensure that no new orders are created with the old cartridge version.

Modifying Cartridges After Upgrading OSM Versions

If the OSM software is upgraded to a new version, all cartridges in use must be rebuilt using the updated SDK and redeployed. See the discussion on Upgrading OSM in *OSM Installation Guide* for details on upgrading cartridges when you upgrade to a new version of OSM.

Studio Environment Editor

Use the Studio Environment editor to define the run-time environment connection information, to define the Secure Socket Layer (SSL) keystore file location, and to review and edit the cartridge and model variables defined for the cartridge.

When defining run-time environment connection information, see the following topics:

- Studio Environment Editor Connection Tab
- Studio Environment Editor SSL Tab
- Studio Environment Editor Properties Tab

Studio Environment Editor Connection Tab

Use the Studio Environment editor **Connection** tab to define the connection parameter necessary to connect to the run-time environment.

Field	Use
Address	Enter the WebLogic IP address (or the fully qualified domain name if DNS is enabled) and port necessary to connect to the OSM run-time environment. During initial OSM installation, the OSM installer program connects to a running Oracle WebLogic server to automatically deploy the **cartridge_management_ws.ear** file, which contains the Cartridge Management Web Service that enables you to connect to OSM from Design Studio. In the Studio Environment editor **Address** field, Design Studio displays a default destination URL for this Oracle WebLogic server. However, you must edit the IP address/server name and port number to match your own server address configuration: `http://IPAddressOrQualifiedDomanName:port/cartridge/wsapi` where *`IPAddressOrQualifiedDomanName`* is the IP address or server name of the Oracle WebLogic server that you connected to during installation and *`port`* is the Oracle WebLogic server port number configured to receive web requests. **Note:** If you are deploying to a clustered environment, specify the proxy server for *`IPAddressOrQualifiedDomanName`*. See *OSM Installation Guide* for more information about installing OSM and connecting to Oracle WebLogic servers.

Related Topics

Testing Run-Time Environment Connectivity

Studio Environment Editor

Studio Environment Editor SSL Tab

Use the Studio Environment editor SSL tab to encrypt your cartridge data prior to deployment.

> **Note:**
>
> Before you deploy cartridges from Design Studio using an SSL connection, you must enable SSL in the WebLogic server to ensure that the Cartridge Management Web Service accepts the SSL connection. See *Design Studio System Administrator's Guide* for more information.

Field	Use
Keystore	Identify the location of your keystore file. The keystore is a file (encrypted with a password) that contains private keys and trusted certificates.

Studio Environment Editor Properties Tab

Use the Studio Environment editor **Properties** tab to review and edit the model and cartridge default variables defined for all of the cartridges in the workspace.

Column	Use
Name	Displays the name of the variable. Design Studio displays all cartridge model and cartridge management variables that are defined in the workspace.
Environment Value	Displays the default value defined for the variable. You can select this value to change the default value to an environment-specific value. Default values are represented by blue diamond-shaped icons.
Source Cartridge	Displays the name of the cartridge from which Design Studio has retrieved the variable and the default value. When a variable is used in multiple cartridges, the name that appears in this column is the name of the first cartridge in which Design Studio encounters the variable. **Note:** When defining default values for variables, employ the same default value for a variable across all cartridges in a workspace. If a variable defined in multiple cartridges does not share the same variable value, a warning appears in the Problems view.

Related Topics

Project Editor Model Variables Tab

Studio Environment Editor

Studio Environment Editor Order and Service Management Test Submission URL Area

Use the Studio Environment Editor Order and Service Management Test Submission URL Area in the **Connection Information** tab on the Studio Environment Editor to specify URLs to submit test orders and to connect to the OSM Order Management web client.

Field	Use
Order Submission URL	Use this to specify the URL for order submission if different from the default URL specified in Address in the **Connection** area.

Field	Use
Source Cartridge	Use this to specify the URL to log in to the Order Management web client if different from the default Cluster/Server URL specified in the **Connection** area.

Related Topics

Testing Run-Time Environment Connectivity

Studio Environment Editor

Studio Environment Editor Connection Tab

A

Automation and Compensation Examples

You need to create automation plug-ins to use the Oracle Communications Order and Service Management (OSM) automation task and automated notification functionality. For information about the code required for the automation plug-ins, refer to the following topics:

- Predefined Automation Plug-ins
- Custom Java Automation Plug-ins
- Compensation XQuery Expressions
- Order Jeopardy Automation XQuery Plug-ins

Predefined Automation Plug-ins

The following topics provide automation plug-in examples for the predefined automation plug-in implementations that support XQuery and XSLT automations:

- Message Example
- Automation Plug-in XQuery Examples
- Automation Plug-in XSLT Examples
- Automation Plug-in Examples for Events, Jeopardies, and Notifications

Message Example

The predefined automation plug-in examples presuppose the following sample order:

```
<?xml version="1.0" encoding="UTF-8"?>
<ws:CreateOrder xmlns:ws="http://xmlns.oracle.com/communications/ordermanagement">
  <ProcessSalesOrderFulfillmentEBM xmlns="http://xmlns.oracle.com/EnterpriseObjects/
Core/EBO/SalesOrder/V2" xmlns:sord="http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/
SalesOrder/V2" xmlns:aia="http://www.oracle.com/XSL/Transform/java/
oracle.apps.aia.core.xpath.AIAFunctions" xmlns:xref="http://www.oracle.com/XSL/Transform/
java/oracle.tip.xref.xpath.XRefXPathFunctions">
     <corecom:EBMHeader xmlns:corecom="http://xmlns.oracle.com/EnterpriseObjects/Core/
Common/V2">
       <corecom:EBMID>2d32373630333234373636393035373 5</corecom:EBMID>
       <corecom:EBMName>{http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/
SalesOrder/V2} ProcessSalesOrderFulfillmentEBM</corecom:EBMName>
       <corecom:EBOName>{http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/
SalesOrder/V2} SalesOrderEBO</corecom:EBOName>
       <corecom:CreationDateTime>2009-03-09T18:46:36-07:00</corecom:CreationDateTime>
       <corecom:VerbCode>process</corecom:VerbCode>
       <corecom:MessageProcessingInstruction>
         <corecom:EnvironmentCode>PRODUCTION</corecom:EnvironmentCode>
       </corecom:MessageProcessingInstruction>
       <corecom:Sender>
       <!-- Information about the sender - for example, a Siebel CRM -->
       </corecom:Sender>
       <corecom:BusinessScope></corecom:BusinessScope>
       <corecom:EBMTracking></corecom:EBMTracking>
     </corecom:EBMHeader>
```

```
     <DataArea>
       <corecom:Process xmlns:corecom="http://xmlns.oracle.com/EnterpriseObjects/Core/
Common/V2"/>
       <ProcessSalesOrderFulfillment>
           <corecom:Identification xmlns:corecom="http://xmlns.oracle.com/
EnterpriseObjects/Core/Common/V2">
                <corecom:BusinessComponentID schemeID="SALESORDER_ID"
schemeAgencyID="COMMON">34333939373132333239373135353138</corecom:BusinessComponentID>
                <corecom:ID schemeID="SALESORDER_ID"
schemeAgencyID="SEBL_01">ScenarioA2</corecom:ID>
                <corecom:ApplicationObjectKey>
                     <corecom:ID schemeID="SALESORDER_ID"
schemeAgencyID="SEBL_01">88-2SGSG</corecom:ID>
                </corecom:ApplicationObjectKey>
                <corecom:Revision>
                    <corecom:Number>1</corecom:Number>
                </corecom:Revision>
           </corecom:Identification>
           <OrderDateTime>2009-03-09T18:40:21Z</OrderDateTime>
           <RequestedDeliveryDateTime>2009-03-10T00:00:00Z</RequestedDeliveryDateTime>
           <TypeCode>SALES ORDER</TypeCode>
           <FulfillmentPriorityCode>9</FulfillmentPriorityCode>
           <FulfillmentSuccessCode>DEFAULT</FulfillmentSuccessCode>
           <FulfillmentModeCode>DELIVER</FulfillmentModeCode>
           <SalesChannelCode/>
           <ProcessingNumber/>
           <ProcessingTypeCode/>
           <corecom:Status xmlns:corecom="http://xmlns.oracle.com/EnterpriseObjects/Core/
Common/V2">
                <corecom:Code>OPEN</corecom:Code>
                <corecom:Description/>
           </corecom:Status>
           <corecom:BusinessUnitReference xmlns:corecom="http://xmlns.oracle.com/
EnterpriseObjects/Core/Common/V2">
                <corecom:BusinessUnitIdentification>
                     <corecom:ID schemeID="ORGANIZATION_ID" schemeAgencyID="SEBL_01">0-
R9NH</corecom:ID>
                </corecom:BusinessUnitIdentification>
           </corecom:BusinessUnitReference>
           <corecom:CustomerPartyReference xmlns:corecom="http://xmlns.oracle.com/
EnterpriseObjects/Core/Common/V2">
                <corecom:CustomerPartyAccountIdentification>
                     <corecom:BusinessComponentID schemeID="CUSTOMERPARTY_ACCOUNTID"
schemeAgencyID="COMMON">2d3535373331303532333303536343833</corecom:BusinessComponentID>
                     <corecom:ID schemeID="CUSTOMERPARTY_ACCOUNTID"
schemeAgencyID="SEBL_01">88-2PB18</corecom:ID>
                     <corecom:ApplicationObjectKey>
                          <corecom:ID schemeID="CUSTOMERPARTY_ACCOUNTID"
schemeAgencyID="SEBL_01">88-2PB18</corecom:ID>
                     </corecom:ApplicationObjectKey>
                </corecom:CustomerPartyAccountIdentification>
                <corecom:CustomerPartyAccountName>Adam,10000</
corecom:CustomerPartyAccountName>
                <corecom:CustomerPartyAccountContactIdentification>
                     <corecom:BusinessComponentID schemeID="CUSTOMERPARTY_CONTACTID"
schemeAgencyID="COMMON">2d353130393634353031313333353938</corecom:BusinessComponentID>
                     <corecom:ApplicationObjectKey>
                          <corecom:ID schemeID="CUSTOMERPARTY_CONTACTID"
schemeAgencyID="SEBL_01">88-2MKA1</corecom:ID>
                     </corecom:ApplicationObjectKey>
                </corecom:CustomerPartyAccountContactIdentification>
                <corecom:CustomerPartyAccountContactAddressCommunication>
```

```
                    <corecom:AddressCommunication>
                         <corecom:Address>
                              <!-- Enter Address Nodes -->
                         </corecom:Address>
                    </corecom:AddressCommunication>
               </corecom:CustomerPartyAccountContactAddressCommunication>
               <corecom:CustomerPartyAccountTypeCode>RESIDENTIAL </
corecom:CustomerPartyAccountTypeCode>
          </corecom:CustomerPartyReference>
          <corecom:PriceListReference xmlns:corecom="http://xmlns.oracle.com/
EnterpriseObjects/Core/Common/V2">
               <corecom:PriceListIdentification>
                    <corecom:ID>88-2D1YC</corecom:ID>
               </corecom:PriceListIdentification>
          </corecom:PriceListReference>
          <corecom:ShipToPartyReference xmlns:corecom="http://xmlns.oracle.com/
EnterpriseObjects/Core/Common/V2">
              <corecom:LocationReference>
                   <corecom:Address>
                       <!-- Enter Address Nodes -->
                   </corecom:Address>
              </corecom:LocationReference>
              <corecom:CustomerPartyAccountIdentification>
                   <corecom:BusinessComponentID schemeID="CUSTOMERPARTY_ACCOUNTID"
schemeAgencyID="COMMON"/>
              </corecom:CustomerPartyAccountIdentification>
              <corecom:CustomerPartyAccountContactIdentification>
                   <corecom:BusinessComponentID schemeID="CUSTOMERPARTY_CONTACTID"
schemeAgencyID="COMMON">2d353130393634353031313333353938</corecom:BusinessComponentID>
                   <corecom:ApplicationObjectKey>
                        <corecom:ID schemeID="CUSTOMERPARTY_CONTACTID"
schemeAgencyID="SEBL_01">88-2MKA1</corecom:ID>
                   </corecom:ApplicationObjectKey>
              </corecom:CustomerPartyAccountContactIdentification>
          </corecom:ShipToPartyReference>
          <corecom:ParentSalesOrderReference xmlns:corecom="http://xmlns.oracle.com/
EnterpriseObjects/Core/Common/V2">
               <corecom:SalesOrderIdentification>
                   <corecom:BusinessComponentID schemeID="SALESORDER_ID"
schemeAgencyID="COMMON"/>
               </corecom:SalesOrderIdentification>
          </corecom:ParentSalesOrderReference>
          <corecom:ProjectReference xmlns:corecom="http://xmlns.oracle.com/
EnterpriseObjects/Core/Common/V2">
              <corecom:ProjectIdentfication>
                  <corecom:ID schemeID="PROJECT_ID" schemeAgencyID="SEBL_01"/>
              </corecom:ProjectIdentfication>
          </corecom:ProjectReference>
          <corecom:SalespersonPartyReference xmlns:corecom="http://xmlns.oracle.com/
EnterpriseObjects/Core/Common/V2">
              <corecom:PartyIdentification>
                   <corecom:ID schemeID="SALESPERSON_PARTYID"
schemeAgencyID="SEBL_01">0-1</corecom:ID>
              </corecom:PartyIdentification>
          </corecom:SalespersonPartyReference>
          <!-- Enter order line items here -->
      </ProcessSalesOrderFulfillment>
      </DataArea>
  </ProcessSalesOrderFulfillmentEBM>
</ws:CreateOrder>
```

Automation Plug-in XQuery Examples

The following topics provide XQuery automation plug-in examples for automation tasks:

- Internal XQuery Sender
- External XQuery Automator
- External XQuery Sender
- Internal XQuery Automator

Internal XQuery Sender

The Automated Task editor internal XQuery automator receives task data from OSM and sends data to an external system. You can send a message to an external system using whatever protocol that system requires, such as Telnet, HTTP, CORBA, SOAP, or web services.

The XQuery has the following characteristics:

- XQuery context in prolog: The input document for any automated task automation plug-in is the order data defined in the Automation Task editor Task Data tab. You can access this data by declaring the TaskContext OSM Java class. Always declare this class along with the $context java binding. For example:

```
declare namespace context = "java:com.mslv.oms.automation.TaskContext";
...
declare variable $context external;
```

- Prolog: You must declare ScriptSenderContextInvocation in any internal XQuery automator which extends ScriptReceiverContextInvocation. Always declare this class along with the $automator java binding. For example:

```
declare namespace automator =
"java:oracle.communications.ordermanagement.automation.plugin.ScriptSenderContextInvo
cation";
...
declare variable $automator external;
```

  Oracle recommends that you use the standard Apache log class. Always declare this class along with the $log java binding.

```
declare namespace log = "java:org.apache.commons.logging.Log";
...
declare variable $log external;
```

  You must use the TextMessage class for sending JMS based messages. Always declare this class along with the $outboundMessage Java binding. You can use JMS text based messages to send OSM Web Service messages to other OSM systems, such as a service order from an OSM COM system to an OSM SOM system.

```
declare namespace outboundMessage = "java:javax.jms.TextMessage";
...
declare variable $outboundMessage external;
```

> **Note:**
>
> If you need to support any other protocol for sending messages, you can implement a custom Java automation plug-in for the protocol or import a helper function implementation that supports the protocol.

- Body: The body for an internal XQuery sender can contain the following elements:
  - Use outboundMessage to set up the standard WebLogic JMS message properties for web services:

    ```
    outboundMessage:setStringProperty($outboundMessage, '_wls_mimehdrContent_Type',
    'text/xml; charset=&quot;utf-8&quot;'),
    ```

  - Use outboundMessage to set up the OSM Web Service URI JMS message property:

    ```
    outboundMessage:setStringProperty($outboundMessage, 'URI', '/osm/wsapi'),
    ```

  - You can optionally use outboundMessage with the XML API to populate a JMS property value from order data. For example this code sets up an Ora_OSM_COM_OrderId parameter that is populated with the OSM order ID:

    ```
    outboundMessage:setStringProperty($outboundMessage, 'Ora_OSM_COM_OrderId', /
    oms:GetOrder.Response/oms:OrderID),
    ```

  - You can optionally use outboundMessage to set the JMS Correlation ID for the automation task before sending the message. This allows OSM to route a return message with the same corresponding JMS property value to an external XQuery automator on the same automation task as the original sender automation plug-in. For example, the following code sets the JMS correlation ID using the original OSM COM order:

    ```
    outboundMessage:setJMSCorrelationID($outboundMessage, concat($order/oms:_root/
    oms:messageXmlData/ebo:ProcessSalesOrderFulfillmentEBM/ebo:DataArea/
    ebo:ProcessSalesOrderFulfillment/corecom:Identification/corecom:ID/text(),'-
    COM')),
    ```

    If this code were applied to "Message Example," the return value would be a concatenation of ScenarioA2 and -COM: `ScenarioA2-COM`.

    > **Note:**
    >
    > Other correlation scenarios are possible. For example, you may send a message from an automation task without expecting any response to the same automation task. In this scenario, another automation task further down in the process may be dedicated to receiving the response message, in which case an automation plug-in would be required that would set the correlation ID expected from the return message for that automated task. See the chapter about using automation in *OSM Developer's Guide* for more information about asynchronous communication scenarios.

  - Access to the task level order data (the task view) using the XML API `GetOrder.Response` function call. For example, the following code provides access to all order data passed into the task as a variable that is then used in other variables to access different parts of the data:

    ```
    let $order := /oms:GetOrder.Response
    let $othervariable := $order/oms:_root/oms:orderid
    ```

- Any XQuery logic your plug-in requires, such as if-then or if-then-else statements that evaluate based on one or more parameters within the response message. For example, there could be a choice of two or more messages that could be sent depending on the order data values, or you might log a message.
- A completeTaskOnExit method statement that completes the plug-in and transitions the task to the next task based on the status selected if the plug-in is intended to end the task. Typically, an automated task would contain an internal XQuery sender plug-in for sending a message and an external XQuery receiver plug-in for receiving a message, but you can also create an automation that only sends an order with another automation that receives the order. This can be useful if the response message takes a long time to return. If you are expecting the system to respond that you sent the message to, you must configure the internal XQuery sender with a reply to queue that listens for a message acknowledgement, whether the response is returned to an external automator on the same automation task or on another automation task.

The following example provides the code for an XQuery that sends a message from an OSM system in the COM role to an OSM system in the SOM role using the OSM Web Service interface and assumes JMS communication over T3S.

```
declare namespace automator =
"java:oracle.communications.ordermanagement.automation.plugin.ScriptSenderContextInvocati
on";
declare namespace context = "java:com.mslv.oms.automation.TaskContext";
declare namespace log = "java:org.apache.commons.logging.Log";
declare namespace outboundMessage = "java:javax.jms.TextMessage";
declare namespace oms="urn:com:metasolv:oms:xmlapi:1";
declare namespace to="http://TechnicalOrder";
declare namespace provord="http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/
ProvisioningOrder/V1";
declare namespace corecom="http://xmlns.oracle.com/EnterpriseObjects/Core/Common/V2";
declare namespace env="http://schemas.xmlsoap.org/soap/envelope/";
declare namespace cord="http://oracle.communications.c2a.model/internal/order";
declare namespace ebo="http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/SalesOrder/V2";

declare variable $automator external;
declare variable $context external;
declare variable $log external;
declare variable $outboundMessage external;

let $order := /oms:GetOrder.Response
let $technicalActions := $order/oms:_root/oms:TechnicalActions
let $ebm := $order/oms:_root/oms:messageXmlData
let $bi := $order/oms:_root/oms:CaptureInteractionResponse

return(
outboundMessage:setStringProperty($outboundMessage, '_wls_mimehdrContent_Type', 'text/
xml; charset=&amp;quot;utf-8&amp;quot;'),
outboundMessage:setStringProperty($outboundMessage, 'URI', '/osm/wsapi'),
outboundMessage:setStringProperty($outboundMessage, 'Ora_OSM_COM_OrderId', /
oms:GetOrder.Response/oms:OrderID),
outboundMessage:setJMSCorrelationID($outboundMessage, concat($order/oms:_root/
oms:messageXmlData/ebo:ProcessSalesOrderFulfillmentEBM/ebo:DataArea/
ebo:ProcessSalesOrderFulfillment/corecom:Identification/corecom:ID/text(),'-COM')),
log:info($log,concat('Sending Service Order for COM order: ', $order/oms:OrderID)),
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ord="http://xmlns.oracle.com/communications/ordermanagement">
        <soapenv:Header>
        <wsse:Security xmlns:wsse = "http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-wssecurity-secext-1.0.xsd" soapenv:mustUnderstand="1">
        <wsse:UsernameToken xmlns:wsu = "http://docs.oasis-open.org/wss/2004/01/
oasis-200401-wss-wssecurity-utility-1.0.xsd" wsu:Id="UsernameToken-4799946">
```

```
        <wsse:Username>demo</wsse:Username>
        <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
username-token-profile-1.0#PasswordText">passw0rd</wsse:Password>
        </wsse:UsernameToken>
        </wsse:Security>
        </soapenv:Header>
   <soapenv:Body>
      <ord:CreateOrder>
               <ebo:ProcessProvisioningOrderEBM xmlns:ebo="http://xmlns.oracle.com/
EnterpriseObjects/Core/EBO/ProvisioningOrder/V1">
<ebo:DataArea>
                           <corecom:Process xmlns="http://xmlns.oracle.com/
EnterpriseObjects/Core/EBO/ProvisioningOrder/V1" xmlns:corecom="http://xmlns.oracle.com/
EnterpriseObjects/Core/Common/V2" xmlns:aia="http://www.oracle.com/XSL/Transform/java/
oracle.apps.aia.core.xpath.AIAFunctions" xmlns:xref="http://www.oracle.com/XSL/Transform/
java/oracle.tip.xref.xpath.XRefXPathFunctions" xmlns:oms="urn:com:metasolv:oms:xmlapi:1"
xmlns:provord="http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/ProvisioningOrder/V1"/>
                           <provord:ProcessProvisioningOrder xmlns="http://
xmlns.oracle.com/EnterpriseObjects/Core/EBO/ProvisioningOrder/V1" xmlns:aia="http://
www.oracle.com/XSL/Transform/java/oracle.apps.aia.core.xpath.AIAFunctions"
xmlns:xref="http://www.oracle.com/XSL/Transform/java/
oracle.tip.xref.xpath.XRefXPathFunctions" xmlns:oms="urn:com:metasolv:oms:xmlapi:1"
xmlns:provord="http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/ProvisioningOrder/V1">
                           <corecom:SalesOrderReference>
                                <corecom:SalesOrderIdentification>
                                    {$order/oms:_root/oms:ServiceOrder/cord:Order/
cord:CustomerDetails/cord:OrderNumber/corecom:Identification/*}
                                </corecom:SalesOrderIdentification>
                           </corecom:SalesOrderReference>
                           <provord:RequestedDeliveryDateTime>2010-07-16T08:24:38Z </
provord:RequestedDeliveryDateTime>
                           <provord:TypeCode>SALES ORDER</provord:TypeCode>
                           <provord:FulfillmentPriorityCode>5</
provord:FulfillmentPriorityCode>
                           <provord:FulfillmentSuccessCode>DEFAULT </
provord:FulfillmentSuccessCode>
                           <provord:FulfillmentModeCode>DELIVER</
provord:FulfillmentModeCode>
                           <provord:ProcessingNumber/>
                           <provord:ProcessingTypeCode/>
                           <corecom:Status xmlns:corecom="http://xmlns.oracle.com/
EnterpriseObjects/Core/Common/V2">
                                <corecom:Code>IN PROGRESS</corecom:Code>
                           </corecom:Status>
                           <corecom:BusinessUnitReference xmlns:corecom="http://
xmlns.oracle.com/EnterpriseObjects/Core/Common/V2">
                                <corecom:BusinessUnitIdentification>
                                    <corecom:ID schemeID="ORGANIZATION_ID"
schemeAgencyID="SEBL_01">0-R9NH</corecom:ID>
                                </corecom:BusinessUnitIdentification>
                           </corecom:BusinessUnitReference>
                           {$order/oms:_root/oms:ServiceOrder/cord:Order/
cord:CustomerDetails/cord:CustomerParty/corecom:CustomerPartyReference}
                           <corecom:ParentProvisioningOrderReference
xmlns:corecom="http://xmlns.oracle.com/EnterpriseObjects/Core/Common/V2">
                                <corecom:ProvisioningOrderIdentification>
                                    <corecom:BusinessComponentID
schemeID="SALESORDER_ID" schemeAgencyID="COMMON"/>
                                </corecom:ProvisioningOrderIdentification>
                           </corecom:ParentProvisioningOrderReference>
                           {
                                for $x in $order/oms:_root/oms:ServiceOrder/cord:Order/
```

```
cord:ServiceOrderLine
                                return
                                <provord:ProvisioningOrderLine>
                                <corecom:Identification xmlns:corecom="http://
xmlns.oracle.com/EnterpriseObjects/Core/Common/V2">
                                     <corecom:BusinessComponentID>{concat($x/@id,'')} </
corecom:BusinessComponentID>
                                     <corecom:ID schemeID="SALESORDER_LINEID"
schemeAgencyID="SEBL_01">{concat($x/@id,'')}</corecom:ID>
                                     <corecom:ApplicationObjectKey>
                                          <corecom:ID schemeID="SALESORDER_LINEID"
schemeAgencyID="SEBL_01">{concat($x/@id,'')}</corecom:ID>
                                     </corecom:ApplicationObjectKey>
                                </corecom:Identification>
                                <provord:OrderQuantity>1</provord:OrderQuantity>
                                <provord:ServiceActionCode>{$x/cord:Action/text()} </
provord:ServiceActionCode>
                                <provord:ServicePointCode/>
                                <corecom:Status xmlns:corecom="http://xmlns.oracle.com/
EnterpriseObjects/Core/Common/V2">
                                    <corecom:Code>IN PROGRESS</corecom:Code>
                                </corecom:Status>
                                <corecom:ServiceAddress xmlns:corecom="http://
xmlns.oracle.com/EnterpriseObjects/Core/Common/V2">
                                     <corecom:Identification>
                                         <corecom:BusinessComponentID
schemeAgencyID="COMMON"
schemeID="CUSTOMERPARTY_ADDRESSID">2d32373332323131353131383631333 1</
corecom:BusinessComponentID>
                                         <corecom:ApplicationObjectKey>
                                              <corecom:ID schemeAgencyID="SEBL_01"
schemeID="CUSTOMERPARTY_ADDRESSID">88-2KKNH</corecom:ID>
                                         </corecom:ApplicationObjectKey>
                                     </corecom:Identification>
                                     <corecom:LineOne>{$x/cord:Address/cord:LineOne/
text()} </corecom:LineOne>
                                     <corecom:CityName>{$x/cord:Address/cord:CityName/
text()} </corecom:CityName>
                                     <corecom:StateName>{$x/cord:Address/cord:StateName/
text()} </corecom:StateName>
                                     <corecom:ProvinceName>{$x/cord:Address/
cord:ProvinceName/ text()}</corecom:ProvinceName>
                                     <corecom:CountryCode>{$x/cord:Address/
cord:CountryCode /text()}</corecom:CountryCode>
                                     <corecom:PostalCode>{$x/cord:Address/
cord:PostalCode /text()}</corecom:PostalCode>
                                </corecom:ServiceAddress>
                                <corecom:ItemReference xmlns:corecom="http://
xmlns.oracle.com/EnterpriseObjects/Core/Common/V2">
                                <corecom:ItemIdentification>
                                     <corecom:BusinessComponentID
schemeAgencyID="COMMON" schemeID="ITEM_ITEMID"/>
                                     <corecom:ApplicationObjectKey>
                                         <corecom:ID schemeID="ITEM_ITEMID"
schemeAgencyID="SEBL_01">{concat($x/cord:InstanceID/text(),'')}</corecom:ID>
                                     </corecom:ApplicationObjectKey>
                                     <corecom:AlternateObjectKey>
                                         <corecom:ContextID/>
                                     </corecom:AlternateObjectKey>
                                     <corecom:SupplierItemID/>
                                </corecom:ItemIdentification>
                                <corecom:Name>{concat($x/@name,'')}</corecom:Name>
```

```
                                        <corecom:ClassificationCode
listID="PermittedTypeCode"></corecom:ClassificationCode>
                                        <corecom:ClassificationCode
listID="BillingProductTypeCode"/>
                                        <corecom:ClassificationCode
listID="FulfillmentItemCode">{concat($x/@name,'')}</corecom:ClassificationCode>
                                        <corecom:ServiceIndicator>false</
corecom:ServiceIndicator>
                                        <corecom:TypeCode>SERVICE</corecom:TypeCode>
                                        <corecom:Description/>
                                        <corecom:SpecificationGroup>
                                            <corecom:Name>ExtensibleAttributes</
corecom:Name>
                                            {
                                                for $y in $x/cord:Attribute
                                                return
                                                <corecom:Specification>
                                                    <corecom:ServiceActionCode> </
corecom:ServiceActionCode>
                                                    <corecom:Name>{concat($y/@name,'')} </
corecom:Name>
                                                    <corecom:DataTypeCode>Text</
corecom:DataTypeCode>
                                                    <corecom:Value>{$y/cord:Value/
cord:value/text()} </corecom:Value>
                                                </corecom:Specification>
                                            }
                                        </corecom:SpecificationGroup>
                                        <corecom:PrimaryClassificationCode>{concat($x/
@name,'')} </corecom:PrimaryClassificationCode>
                                        <corecom:ServiceInstanceIndicator>true </
corecom:ServiceInstanceIndicator>
                                    </corecom:ItemReference>
                                    <provord:ProvisioningOrderLineSpecificationGroup>
                                        <corecom:SpecificationGroup>
                                            <corecom:Name>ExtensibleAttributes</
corecom:Name>
                                            <corecom:Specification>
                                                <corecom:Name>ParentSalesOrderLine</
corecom:Name>
                                                <corecom:Value>{$x/cord:primaryMapping/
text()} </corecom:Value>
                                            </corecom:Specification>
                                            {
                                                for $z in $x/cord:secondaryMapping
                                                return
                                                <corecom:Specification>
                                                    <corecom:Name>ParentSalesOrderLine</
corecom:Name>
                                                    <corecom:Value>{$z/text()}</corecom:Value>
                                                </corecom:Specification>
                                            }
                                        </corecom:SpecificationGroup>
                                    </provord:ProvisioningOrderLineSpecificationGroup>
                                    </provord:ProvisioningOrderLine>
                                }
                            </provord:ProcessProvisioningOrder>
                        </ebo:DataArea>
                    </ebo:ProcessProvisioningOrderEBM>
        </ord:CreateOrder>
    </soapenv:Body>
</soapenv:Envelope>
```

External XQuery Automator

The Automated Task editor external XQuery automator receives task data from an external system and optionally updates OSM order data. The XQuery has the following characteristics:

- XQuery context in prolog: The input document for any automated task automation plug-in is the order data defined in the Automation Task editor Task Data tab. You can access this data by declaring the TaskContext OSM Java class. Always declare this class along with the $context java binding. For example:

```
declare namespace context = "java:com.mslv.oms.automation.TaskContext";
...
declare variable $context external;
```

- Prolog: You must declare ScriptReceiverContextInvocation in any external XQuery automator. Typically, you can use the getOrderAsDOM method to receive external messages and the setUpdateOrder method to update the order data. Always declare this class along with the $automator java binding. For example:

```
declare namespace automator =
"java:oracle.communications.ordermanagement.automation.plugin.ScriptReceiverContextIn
vocation";
...
declare variable $automator external;
```

  Oracle recommends that you use the standard Apache log class. Always declare this class along with the $log java binding.

```
declare namespace log = "java:org.apache.commons.logging.Log";
...
declare variable $log external;
```

  Another necessary declaration includes the xmlapi namespace, that you can use with the ScriptReceiverContextInvocation getOrderAsDom method to retrieve the order data for the task as a variable. This task data variable can be used in an OrderDataUpdate to update the order data with the data values received in the response message, if an update to the order data is required. For example:

```
declare namespace oms="urn:com:metasolv:oms:xmlapi:1";
let $taskData := fn:root(automator:getOrderAsDOM($automator))/oms:GetOrder.Response
```

- Body: The body for an external XQuery automator can contain the following elements:
  - Any XQuery logic your plug-in requires, such as if-then or if-then-else statements that evaluate based on one or more parameters within the response message, or you might log a message.
  - A setUpdateOrder method statement that indicates whether there is an order data update. This method should be identical to what you selected in the Design Studio automation plug-in Properties View XQuery Tab Update Order check box.
  - A completeTaskOnExit method statement that completes the plug-in and transitions the task to the next task based on the status selected, if the plug-in is intended to end the task. Since there can be multiple plug-ins within a task, you would only need this method in the last plug-in listed. For example, the Failed status might transition to a fallout task, and the Succeed status may transition to the next task in the process.
  - An OrderDataUpdate statement that updates the order data based on the information returned in the response. See "Using OrderDataUpdate Elements to Pass Order Modification Data" for more information about structuring order update code.

- Indexing: Order data in OSM often includes multiple data instances. For example, an orchestration order must include the **ControlData/OrderItem** and **ControlData/Functions** multi-instance nodes. Multi-instance nodes in solution cartridges are possible for any data element where the maximum cardinality of the node is greater than 1. When updating a multi-instance data node using automations use the node index to reference the specific node instance you want to update. The node index is available in the XML API GetOrder.Response. See *OSM Developer's Guide* for an example of a GetOrder response message with indexing.

The following example triggers different order data updates based on the status message returned from an external system. In this case, the external system is another OSM instance running in the SOM role:

```
declare namespace oms="urn:com:metasolv:oms:xmlapi:1";
declare namespace automator =
"java:oracle.communications.ordermanagement.automation.plugin.ScriptReceiverContextInvoca
tion";
declare namespace context = "java:com.mslv.oms.automation.TaskContext";
declare namespace log = "java:org.apache.commons.logging.Log";
declare namespace su="http://StatusUpdate";
declare namespace so="http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/SalesOrder/V2";
declare namespace corecom="http://xmlns.oracle.com/EnterpriseObjects/Core/Common/V2";

declare variable $automator external;
declare variable $context external;
declare variable $log external;

let $response := fn:root()/su:StatusUpdate (: fn:root(.) :)
let $items := fn:root()/su:StatusUpdate/su:OrderItem

let $taskData := fn:root(automator:getOrderAsDOM($automator))/oms:GetOrder.Response
let $component := if (fn:exists($taskData/oms:_root/oms:ControlData/oms:Functions/*/
oms:componentKey)) then $taskData/oms:_root/oms:ControlData/oms:Functions/
*[fn:position()=1] else ()

return (
if($response/su:status/text()='SOM_Completed') then (
    log:info($log,concat('Received SOM Status Update: SOM_Completed; ', $response/
su:status/text())),
    automator:setUpdateOrder($automator,"true"),
    context:completeTaskOnExit($context,"success"),
    (
        <OrderDataUpdate xmlns="http://www.metasolv.com/OMS/OrderDataUpdate/2002/10/25">
            {
            for $item in $items
            for $parent in $item/su:ParentLineId
            for $orderComponentItem in $component/oms:orderItem[oms:orderItemRef/
oms:LineXmlData/so:SalesOrderLine/corecom:Identification/corecom:ApplicationObjectKey/
corecom:ID/text() = $parent/text()]
            return (
                    <Update path="{fn:concat("/ControlData/Functions/Provision/
orderItem[@index='",fn:data($orderComponentItem/@index),"']")}">
                       <ExternalFulfillmentState>{$item/su:Status/text()}</
ExternalFulfillmentState>
                    </Update>
              )

            }
        </OrderDataUpdate>
    )
) else if($response/su:status/text()='SOM_Failed') then (
```

```
    log:info($log,concat('Received SOM Status Update: SOM_Failed; ', $response/su:status/
text())),
    automator:setUpdateOrder($automator,"true"),
    context:completeTaskOnExit($context,"failure"),
    (
        <OrderDataUpdate xmlns="http://www.metasolv.com/OMS/OrderDataUpdate/2002/10/25">
            {
            for $item in $items
            for $parent in $item/su:ParentLineId
            for $orderComponentItem in $component/oms:orderItem[oms:orderItemRef/
oms:LineXmlData/so:SalesOrderLine/corecom:Identification/corecom:ApplicationObjectKey/
corecom:ID/text() = $parent/text()]
            return (
                    <Update path="{fn:concat("/ControlData/Functions/Provision/
orderItem[@index='",fn:data($orderComponentItem/@index),"']")}">
                       <ExternalFulfillmentState>{$item/su:Status/text()}</
ExternalFulfillmentState>
                    </Update>
              )

            }
        </OrderDataUpdate>
    )
) else (
    log:info($log,concat('Received SOM Status Update: SOM_InProgress or SOM_Canceled;
', $response/su:status/text())),
    automator:setUpdateOrder($automator,"true"),
    (
        <OrderDataUpdate xmlns="http://www.metasolv.com/OMS/OrderDataUpdate/2002/10/25">
            {
            for $item in $items
            for $parent in $item/su:ParentLineId
            for $orderComponentItem in $component/oms:orderItem[oms:orderItemRef/
oms:LineXmlData/so:SalesOrderLine/corecom:Identification/corecom:ApplicationObjectKey/
corecom:ID/text() = $parent/text()]
            return (
                    <Update path="{fn:concat("/ControlData/Functions/Provision/
orderItem[@index='",fn:data($orderComponentItem/@index),"']")}">
                       <ExternalFulfillmentState>{$item/su:Status/text()}</
ExternalFulfillmentState>
                    </Update>
              )

            }
        </OrderDataUpdate>
    )
)
)
```

External XQuery Sender

The Automated Task editor external XQuery sender receives task data from an external system, then sends the data (after possibly transforming the data) to another external system or even returns the data back to the original external system. This XQuery combines characteristics of external XQuery automators and internal XQuery senders. See "External XQuery Automator" and "Internal XQuery Sender " for more information.

> **Note:**
>
> You must declare ScriptSenderContextInvocation in any external XQuery sender which inherits the ScriptReceiverContextInvocation class and methods used in internal or external automators.

Internal XQuery Automator

The Automated Task editor internal XQuery automator receives task data from OSM, then processes the data. For example, such an automation might perform computational actions on the data or other similar logic. This XQuery combines characteristics of external XQuery automators and internal XQuery senders. See "External XQuery Automator" and "Internal XQuery Sender " for more information.

> **Note:**
>
> You must declare ScriptReceiverContextInvocation class in an internal XQuery automator.

Automation Plug-in XSLT Examples

The following topics provide XSLT automation plug-in examples for automation tasks:

- Internal XSLT Sender
- External XSLT Automator
- External XSLT Sender
- Internal XSLT Automator

Internal XSLT Sender

The Automated Task editor internal XSLT automator receives task data from OSM and sends data to an external system. You can send a message to an external system using whatever protocol that system requires, such as, Telnet, HTTP, CORBA, SOAP, or web services.

The XSLT has the following characteristics:

- XSLT context: The input document for any automated task automation plug-in is the order data defined in the Automation Task editor Task Data tab. You can access this data by declaring the TaskContext OSM Java class. Always declare this class along with the context java variable. For example:

  ```
  xmlns:context="java:com.mslv.oms.automation.TaskContext"
  ...
  <xsl:param name="context"/>
  ```

- Initial namespace declarations: You must declare ScriptSenderContextInvocation in any internal XSLT automator which extends ScriptReceiverContextInvocation. Always declare this class along with the automator java variable. For example:

  ```
  xmlns:automator="java:oracle.communications.ordermanagement.automation.plugin.ScriptSenderContextInvocation"
  ```

```
...
<xsl:param name="automator"/>
```

Oracle recommends that you use the standard Apache log class. Always declare this class along with the log java variable.

```
xmlns:log="java:org.apache.commons.logging.Log"
...
<xsl:param name="log"/>
```

You must use the TextMessage class for sending JMS based messages. Always declare this class along with the outboundMessage Java variable. You can use JMS text based messages to send OSM Web Service messages to other OSM systems, such as a service order from an OSM COM system to an OSM SOM system.

```
xmlns:outboundMessage="java:javax.jms.TextMessage"
...
<xsl:param name="outboundMessage"/>
```

> **Note:**
>
> If you need to support any other protocol for sending messages, you can implement a custom Java automation plug-in for the protocol or import a helper function implementation that supports the protocol.

- Body: The body for an internal XSLT sender can contain the following elements:
  - Use outboundMessage to set up the standard WebLogic JMS message properties for web services:

    ```
    <xsl:variable name="outboundMessage"
    select="java:setStringProperty($outboundMessage, '_wls_mimehdrContent_Type',
    'text/xml; charset=&quot;utf-8&quot;')"/>
    ```

  - Use outboundMessage to set up the OSM Web Service URI JMS message property:

    ```
    <xsl:variable name="outboundMessage"
    select="java:setStringProperty($outboundMessage, 'URI', '/osm/wsapi')"/>
    ```

  - You can optionally use outboundMessage with the XML API to populate a JMS property value from order data. For example this code sets up an Ora_OSM_COM_OrderId parameter that is populated with the OSM order ID:

    ```
    <xsl:variable name="outboundMessage"
    select="java:setStringProperty($outboundMessage, 'Ora_OSM_COM_OrderId', /
    oms:GetOrder.Response/oms:OrderID)"/>
    ```

  - You can optionally use outboundMessage to set the JMS Correlation ID for the automation task before sending the message. This allows OSM to route a return message with the same corresponding JMS property value to an external XQuery automator on the same automation task as the original sender automation plug-in. For example, the following code sets the JMS correlation ID using the original OSM COM order:

    ```
    <xsl:variable name="void" select="java:setJMSCorrelationID($outboundMessage,
    concat($order/oms:_root/oms:messageXmlData/ebo:ProcessSalesOrderFulfillmentEBM/
    ebo:DataArea/ebo:ProcessSalesOrderFulfillment/corecom:Identification/corecom:ID/
    text(),'-COM'))"/>
    ```

    If this code were applied to "Message Example", the return value would be a concatenation of ScenarioA2 and -COM: `ScenarioA2-COM`.

> **Note:**
>
> Other correlation scenarios are possible. For example, you may send a message from automation task without expecting any response to the same automation task. In this scenario, another automation task further down in the process may be dedicated to receiving the response message, in which case an automation plug-in would be required that would set the correlation ID expected from the return message for that automated task. See the chapter about using automation in *OSM Developer's Guide* for more information about asynchronous communication scenarios.

- Access to the task level order data (the task view) using the XML API `GetOrder.Response` function call. For example, the following code provides access to all order data passed into the task as a variable that is then used in other variables to access different parts of the data:

```
    <xsl:template match="/">
        <xsl:variable name="order" select="oms:GetOrder.Response"/>
        <xsl:variable name="othervariable" select="$order/oms:_root/
oms:orderid"/>
```

- Any XSLT logic your plug-in requires, such as if-then or if-then-else statements that evaluate based on one or more parameters within the response message. For example, there could be a choice of two or more messages that could be sent depending on the order data values, or you might log a message.
- A completeTaskOnExit method statement that completes the plug-in and transitions the task to the next task based on the status selected if the plug-in is intended to end the task. Typically, an automated task would contain an internal XSLT sender plug-in for sending a message and an external XSLT receiver plug-in for receiving a message, but you can also create an automation that only sends an order with another automation that receives the order. This can be useful if the response message takes a long time to return. If you are expecting the system to respond that you sent the message to, you must configure the internal XSLT sender with a reply to queue that listens for a message acknowledgement, whether the response is returned to an external automator on the same automation task or on another automation task.

The following example provides the code for an XSLT that sends a message from an OSM system in the COM role to an OSM system in the SOM role using the OSM Web Service interface and assumes JMS communication over T3S.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" xmlns="http://www.metasolv.com/OMS/OrderDataUpdate"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:java="http://xml.apache.org/xslt/java"
    xmlns:xalan="http://xml.apache.org/xslt"
    xmlns:oms="urn:com:metasolv:oms:xmlapi:1"
xmlns:automator="java:oracle.communications.ordermanagement.automation.plugin.ScriptSende
rContextInvocation"
    xmlns:context="java:com.mslv.oms.automation.TaskContext"
    xmlns:log="java:org.apache.commons.logging.Log"
    xmlns:outboundMessage="java:javax.jms.TextMessage"
    xmlns:to="http://TechnicalOrder"
    xmlns:provord="http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/
ProvisioningOrder/V1"
    xmlns:corecom="http://xmlns.oracle.com/EnterpriseObjects/Core/Common/V2"
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:ebo="http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/SalesOrder/V2"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
    exclude-result-prefixes="xsl java xalan oms com ser soapenv xsi"
    xmlns:fn="http://www.w3.org/2005/02/xpath-functions">

    <!-- * -->
    <xsl:param name="automator"/>
    <xsl:param name="log"/>
    <xsl:param name="context"/>
    <xsl:param name="outboundMessage"/>

    <!-- * -->

    <xsl:output method="xml" indent="yes" omit-xml-declaration="no" xalan:indent-
amount="5"/>
    <xsl:template match="/">
        <xsl:variable name="order" select="oms:GetOrder.Response"/>
        <xsl:variable name="technicalActions" select="$order/oms:_root/
oms:TechnicalActions"/>
        <xsl:variable name="ebm" select="$order/oms:_root/oms:messageXmlData"/>
        <xsl:variable name="bi" select="$order/oms:_root/
oms:CaptureInteractionResponse"/>
        <xsl:variable name="outboundMessage"
select="java:setStringProperty($outboundMessage, '_wls_mimehdrContent_Type', 'text/xml;
charset=&amp;quot;utf-8&amp;quot;')"/>
        <xsl:variable name="outboundMessage"
select="java:setStringProperty($outboundMessage, 'URI', '/osm/wsapi')"/>
        <xsl:variable name="outboundMessage"
select="java:setStringProperty($outboundMessage, 'Ora_OSM_COM_OrderId', /
oms:GetOrder.Response/oms:OrderID)"/>
        <xsl:variable name="void" select="java:setJMSCorrelationID($outboundMessage,
concat($order/oms:_root/oms:messageXmlData/ebo:ProcessSalesOrderFulfillmentEBM/
ebo:DataArea/ebo:ProcessSalesOrderFulfillment/corecom:Identification/corecom:ID/text(),'-
COM'))"/>
        <xsl:variable name="log" select=java:info($log,concat('Sending Service Order for
COM order: ', $order/oms:OrderID))"/>
        <xsl:call-template name="sendSomOrder"/>
    </xsl:template>
    <!-- ================================
         Create the SOAP message for the sendSomOrder call
         ================================= -->
    <xsl:template name="sendSomOrder">

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ord="http://xmlns.oracle.com/communications/ordermanagement">
        <soapenv:Header>
        <wsse:Security xmlns:wsse = "http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-wssecurity-secext-1.0.xsd" soapenv:mustUnderstand="1">
        <wsse:UsernameToken xmlns:wsu = "http://docs.oasis-open.org/wss/2004/01/
oasis-200401-wss-wssecurity-utility-1.0.xsd" wsu:Id="UsernameToken-4799946">
        <wsse:Username>demo</wsse:Username>
        <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
username-token-profile-1.0#PasswordText">passw0rd</wsse:Password>
        </wsse:UsernameToken>
        </wsse:Security>
        </soapenv:Header>
   <soapenv:Body>
      <ord:CreateOrder>
                <ebo:ProcessProvisioningOrderEBM xmlns:ebo="http://xmlns.oracle.com/
EnterpriseObjects/Core/EBO/ProvisioningOrder/V1">
<ebo:DataArea>
                            <corecom:Process xmlns="http://xmlns.oracle.com/
EnterpriseObjects/Core/EBO/ProvisioningOrder/V1" xmlns:corecom="http://xmlns.oracle.com/
EnterpriseObjects/Core/Common/V2" xmlns:aia="http://www.oracle.com/XSL/Transform/java/
```

```
oracle.apps.aia.core.xpath.AIAFunctions" xmlns:xref="http://www.oracle.com/XSL/Transform/
java/oracle.tip.xref.xpath.XRefXPathFunctions" xmlns:oms="urn:com:metasolv:oms:xmlapi:1"
xmlns:provord="http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/ProvisioningOrder/V1"/>
                              <provord:ProcessProvisioningOrder xmlns="http://
xmlns.oracle.com/EnterpriseObjects/Core/EBO/ProvisioningOrder/V1" xmlns:aia="http://
www.oracle.com/XSL/Transform/java/oracle.apps.aia.core.xpath.AIAFunctions"
xmlns:xref="http://www.oracle.com/XSL/Transform/java/
oracle.tip.xref.xpath.XRefXPathFunctions" xmlns:oms="urn:com:metasolv:oms:xmlapi:1"
xmlns:provord="http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/ProvisioningOrder/V1">
                              <corecom:SalesOrderReference>
                                   <corecom:SalesOrderIdentification>
                                       {$order/oms:_root/oms:ServiceOrder/cord:Order/
cord:CustomerDetails/cord:OrderNumber/corecom:Identification/*}
                                   </corecom:SalesOrderIdentification>
                              </corecom:SalesOrderReference>
                              <provord:RequestedDeliveryDateTime>2010-07-16T08:24:38Z </
provord:RequestedDeliveryDateTime>
                              <provord:TypeCode>SALES ORDER</provord:TypeCode>
                              <provord:FulfillmentPriorityCode>5</
provord:FulfillmentPriorityCode>
                              <provord:FulfillmentSuccessCode>DEFAULT </
provord:FulfillmentSuccessCode>
                              <provord:FulfillmentModeCode>DELIVER</
provord:FulfillmentModeCode>
                              <provord:ProcessingNumber/>
                              <provord:ProcessingTypeCode/>
                              <corecom:Status xmlns:corecom="http://xmlns.oracle.com/
EnterpriseObjects/Core/Common/V2">
                                   <corecom:Code>IN PROGRESS</corecom:Code>
                              </corecom:Status>
                              <corecom:BusinessUnitReference xmlns:corecom="http://
xmlns.oracle.com/EnterpriseObjects/Core/Common/V2">
                                   <corecom:BusinessUnitIdentification>
                                       <corecom:ID schemeID="ORGANIZATION_ID"
schemeAgencyID="SEBL_01">0-R9NH</corecom:ID>
                                   </corecom:BusinessUnitIdentification>
                              </corecom:BusinessUnitReference>
                              {$order/oms:_root/oms:ServiceOrder/cord:Order/
cord:CustomerDetails/cord:CustomerParty/corecom:CustomerPartyReference}
                              <corecom:ParentProvisioningOrderReference
xmlns:corecom="http://xmlns.oracle.com/EnterpriseObjects/Core/Common/V2">
                                   <corecom:ProvisioningOrderIdentification>
                                       <corecom:BusinessComponentID
schemeID="SALESORDER_ID" schemeAgencyID="COMMON"/>
                                   </corecom:ProvisioningOrderIdentification>
                              </corecom:ParentProvisioningOrderReference>
                              {
                                   for $x in $order/oms:_root/oms:ServiceOrder/cord:Order/
cord:ServiceOrderLine
                                   return
                                   <provord:ProvisioningOrderLine>
                                   <corecom:Identification xmlns:corecom="http://
xmlns.oracle.com/EnterpriseObjects/Core/Common/V2">
                                        <corecom:BusinessComponentID>{concat($x/@id,'')} </
corecom:BusinessComponentID>
                                        <corecom:ID schemeID="SALESORDER_LINEID"
schemeAgencyID="SEBL_01">{concat($x/@id,'')}</corecom:ID>
                                        <corecom:ApplicationObjectKey>
                                             <corecom:ID schemeID="SALESORDER_LINEID"
schemeAgencyID="SEBL_01">{concat($x/@id,'')}</corecom:ID>
                                        </corecom:ApplicationObjectKey>
                                   </corecom:Identification>
```

```
                                    <provord:OrderQuantity>1</provord:OrderQuantity>
                                    <provord:ServiceActionCode>{$x/cord:Action/text()} </
provord:ServiceActionCode>
                                    <provord:ServicePointCode/>
                                    <corecom:Status xmlns:corecom="http://xmlns.oracle.com/
EnterpriseObjects/Core/Common/V2">
                                        <corecom:Code>IN PROGRESS</corecom:Code>
                                    </corecom:Status>
                                    <corecom:ServiceAddress xmlns:corecom="http://
xmlns.oracle.com/EnterpriseObjects/Core/Common/V2">
                                         <corecom:Identification>
                                             <corecom:BusinessComponentID
schemeAgencyID="COMMON"
schemeID="CUSTOMERPARTY_ADDRESSID">2d323733323231313531313836313331</
corecom:BusinessComponentID>
                                             <corecom:ApplicationObjectKey>
                                                  <corecom:ID schemeAgencyID="SEBL_01"
schemeID="CUSTOMERPARTY_ADDRESSID">88-2KKNH</corecom:ID>
                                             </corecom:ApplicationObjectKey>
                                         </corecom:Identification>
                                         <corecom:LineOne>{$x/cord:Address/cord:LineOne/
text()} </corecom:LineOne>
                                         <corecom:CityName>{$x/cord:Address/cord:CityName/
text()} </corecom:CityName>
                                         <corecom:StateName>{$x/cord:Address/cord:StateName/
text()} </corecom:StateName>
                                         <corecom:ProvinceName>{$x/cord:Address/
cord:ProvinceName/ text()}</corecom:ProvinceName>
                                         <corecom:CountryCode>{$x/cord:Address/
cord:CountryCode /text()}</corecom:CountryCode>
                                         <corecom:PostalCode>{$x/cord:Address/
cord:PostalCode /text()}</corecom:PostalCode>
                                    </corecom:ServiceAddress>
                                    <corecom:ItemReference xmlns:corecom="http://
xmlns.oracle.com/EnterpriseObjects/Core/Common/V2">
                                    <corecom:ItemIdentification>
                                         <corecom:BusinessComponentID
schemeAgencyID="COMMON" schemeID="ITEM_ITEMID"/>
                                         <corecom:ApplicationObjectKey>
                                             <corecom:ID schemeID="ITEM_ITEMID"
schemeAgencyID="SEBL_01">{concat($x/cord:InstanceID/text(),'')}</corecom:ID>
                                         </corecom:ApplicationObjectKey>
                                         <corecom:AlternateObjectKey>
                                             <corecom:ContextID/>
                                         </corecom:AlternateObjectKey>
                                         <corecom:SupplierItemID/>
                                    </corecom:ItemIdentification>
                                    <corecom:Name>{concat($x/@name,'')}</corecom:Name>
                                        <corecom:ClassificationCode
listID="PermittedTypeCode"></corecom:ClassificationCode>
                                         <corecom:ClassificationCode
listID="BillingProductTypeCode"/>
                                         <corecom:ClassificationCode
listID="FulfillmentItemCode">{concat($x/@name,'')}</corecom:ClassificationCode>
                                         <corecom:ServiceIndicator>false</
corecom:ServiceIndicator>
                                         <corecom:TypeCode>SERVICE</corecom:TypeCode>
                                         <corecom:Description/>
                                         <corecom:SpecificationGroup>
                                             <corecom:Name>ExtensibleAttributes</
corecom:Name>
                                             {
```

```
                                                for $y in $x/cord:Attribute
                                                return
                                                <corecom:Specification>
                                                    <corecom:ServiceActionCode> </
corecom:ServiceActionCode>
                                                    <corecom:Name>{concat($y/@name,'')} </
corecom:Name>
                                                    <corecom:DataTypeCode>Text</
corecom:DataTypeCode>
                                                    <corecom:Value>{$y/cord:Value/
cord:value/text()} </corecom:Value>
                                                </corecom:Specification>
                                            }
                                        </corecom:SpecificationGroup>
                                        <corecom:PrimaryClassificationCode>{concat($x/
@name,'')} </corecom:PrimaryClassificationCode>
                                        <corecom:ServiceInstanceIndicator>true </
corecom:ServiceInstanceIndicator>
                                    </corecom:ItemReference>
                                    <provord:ProvisioningOrderLineSpecificationGroup>
                                        <corecom:SpecificationGroup>
                                             <corecom:Name>ExtensibleAttributes</
corecom:Name>
                                             <corecom:Specification>
                                                  <corecom:Name>ParentSalesOrderLine</
corecom:Name>
                                                  <corecom:Value>{$x/cord:primaryMapping/
text()} </corecom:Value>
                                             </corecom:Specification>
                                             {
                                                  for $z in $x/cord:secondaryMapping
                                                  return
                                                  <corecom:Specification>
                                                      <corecom:Name>ParentSalesOrderLine</
corecom:Name>
                                                      <corecom:Value>{$z/text()}</corecom:Value>
                                                      </corecom:Specification>
                                             }
                                        </corecom:SpecificationGroup>
                                    </provord:ProvisioningOrderLineSpecificationGroup>
                                    </provord:ProvisioningOrderLine>
                                }
                            </provord:ProcessProvisioningOrder>
                        </ebo:DataArea>
                    </ebo:ProcessProvisioningOrderEBM>
      </ord:CreateOrder>
   </soapenv:Body>
</soapenv:Envelope>
</xsl:template>
    <!-- * -->
    <xsl:template match="* | @* | text()">
        <!-- do nothing -->
        <xsl:apply-templates/>
    </xsl:template>
</xsl:stylesheet>
```

External XSLT Automator

The Automated Task editor external XSLT automator receives task data from an external system and optionally updates OSM order data. The XSLT has the following characteristics:

- XSLT context in prolog: The input document for any automated task automation plug-in is the order data defined in the Automation Task editor Task Data tab. You can access this data by declaring the TaskContext OSM Java class. Always declare this class along with the context java binding. For example:

```
xmlns:context="java:com.mslv.oms.automation.TaskContext"
...
<xsl:param name="context"/>
```

- Prolog: You must declare ScriptReceiverContextInvocation in any external XQuery automator. Typically, you can use the getOrderAsDOM method to receive external messages and the setUpdateOrder method to update the order data. Always declare this class along with the automator java binding. For example:

```
xmlns:automator="java:oracle.communications.ordermanagement.automation.plugin.ScriptR
eceiverContextInvocation"
...
<xsl:param name="automator"/>
```

  Oracle recommends that you use the standard Apache log class. Always declare this class along with the $log java binding.

```
xmlns:log="java:org.apache.commons.logging.Log"
...
<xsl:param name="log"/>
```

  Another necessary declaration includes the xmlapi namespace, that you can use with the ScriptReceiverContextInvocation getOrderAsDom method to retrieve the order data for the task as a variable. This task data variable can be used in an OrderDataUpdate to update the order data with the data values received in the response message, if an update to the order data is required. For example:

```
xmlns:oms="urn:com:metasolv:oms:xmlapi:1"
<xsl:variable name="taskData" select="fn:root(java:getOrderAsDOM($automator))/
oms:GetOrder.Response"/>
```

- Body: The body for an external XSLT automator can contain the following elements:
  - Any XSLT logic your plug-in requires, such as if-then or if-then-else statements that evaluate based on one or more parameters within the response message, or you might log a message.
  - A setUpdateOrder method statement that indicates whether there is an order data update. This method should be identical to what you selected in the Design Studio automation plug-in Properties View XSLT Tab Update Order check box.
  - A completeTaskOnExit method statement that completes the plug-in and transitions the task to the next task based on the status selected, if the plug-in is intended to end the task. Since there can be multiple plug-ins within a task, you would only need this method in the last plug-in listed. For example, the Failed status might transition to a fallout task, and the Succeed status may transition to the next task in the process.
  - An OrderDataUpdate statement that updates the order data based on the information returned in the response. See "Using OrderDataUpdate Elements to Pass Order Modification Data" for more information about structuring order update code.
  - Indexing: Order data in OSM often includes multiple data instances. For example, an orchestration order must include the **ControlData/OrderItem** and **ControlData/Functions** multi-instance nodes. Multi-instance nodes in solution cartridges are possible for any data element where the maximum cardinality of the node is greater than 1. When updating a multi-instance data node using automations use the node index to reference the specific node instance you want to update. The node index is

available in the XML API GetOrder.Response. See *OSM Developer's Guide* for an example of a GetOrder response message with indexing.

The following example triggers different order data updates based on the status message returned from an external system. In this case, the external system is another OSM instance running in the SOM role:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" xmlns="http://www.metasolv.com/OMS/OrderDataUpdate"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:java="http://xml.apache.org/xslt/java"
    xmlns:xalan="http://xml.apache.org/xslt"
    xmlns:oms="urn:com:metasolv:oms:xmlapi:1"

xmlns:automator="java:oracle.communications.ordermanagement.automation.plugin.ScriptRecei
verContextInvocation"
    xmlns:context="java:com.mslv.oms.automation.TaskContext"
    xmlns:log="java:org.apache.commons.logging.Log"
    xmlns:su="http://StatusUpdate"
    xmlns:so="http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/SalesOrder/V2"
    xmlns:corecom="http://xmlns.oracle.com/EnterpriseObjects/Core/Common/V2"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    exclude-result-prefixes="xsl java xalan oms soapenv xsi">

    <!-- * -->
    <xsl:param name="automator"/>
    <xsl:param name="log"/>
    <xsl:param name="context"/>

    <!-- * -->
    <xsl:output method="xml" indent="yes" omit-xml-declaration="no" xalan:indent-
amount="5"/>

    <xsl:template match="/">
     <xsl:variable name="taskData" select="fn:root(java:getOrderAsDOM($automator))/
oms:GetOrder.Response"/>
     <xsl:variable name="response" select="fn:root()/su:StatusUpdate (: fn:root(.) :)"/>
     <xsl:variable name="items" select="fn:root()/su:StatusUpdate/su:OrderItem"/>
    <xsl:variable name="component" select="if (fn:exists($taskData/oms:_root/
oms:ControlData/oms:Functions/*/oms:componentKey)) then $taskData/oms:_root/
oms:ControlData/oms:Functions/*[fn:position()=1] else ()"/>
        <xsl:apply-templates/>
</xsl:template>

<!-- Match the status SOM_Complete -->
    <xsl:template match="$response[su:status/text()='SOM_Completed']">
     <xsl:variable name="log" select="java:info($log,concat('Received SOM Status Update:
SOM_Completed; ', $response/su:status/text()))"/>
     <xsl:variable name="automator" select="java:setUpdateOrder($automator, true())"/>
     <xsl:variable name="context" select="java:completeTaskOnExit($context, success())"/>
     <OrderDataUpdate xmlns="http://www.metasolv.com/OMS/OrderDataUpdate/2002/10/25">
        <xsl:for-each select="su:ParentLineId">
            <xsl:variable name="parent" select="."/>
            <xsl:for-each select="$component/oms:orderItem[oms:orderItemRef/
oms:LineXmlData/so:SalesOrderLine/corecom:Identification/corecom:ApplicationObjectKey/
corecom:ID/text() = $parent/text()]">
                <xsl:variable name="index" select="@index"/>
                    <Update path="{fn:concat("/ControlData/Functions/Provision/
orderItem[@index='",fn:data($orderComponentItem/@index),"']")}>
                     <ExternalFulfillmentState>{$item/su:Status/text()}</
ExternalFulfillmentState>
                </Update>
```

```
            </xsl:for-each>
         </xsl:for-each>
      </OrderDataUpdate>
     </xsl:template>

<!-- Match the status SOM_Failed -->
     <xsl:template match="$response[su:status/text()='SOM_Failed']">
      <xsl:variable name="log" select="java:info($log,concat('Received SOM Status Update:
SOM_Failed; ', $response/su:status/text()))"/>
      <xsl:variable name="automator" select="java:setUpdateOrder($automator, true())"/>
      <xsl:variable name="context" select="java:completeTaskOnExit($context, success())"/>
      <OrderDataUpdate xmlns="http://www.metasolv.com/OMS/OrderDataUpdate/2002/10/25">
         <xsl:for-each select="su:ParentLineId">
             <xsl:variable name="parent" select="."/>
             <xsl:for-each select="$component/oms:orderItem[oms:orderItemRef/
oms:LineXmlData/so:SalesOrderLine/corecom:Identification/corecom:ApplicationObjectKey/
corecom:ID/text() = $parent/text()]">
                 <xsl:variable name="index" select="@index"/>
                     <Update path="{fn:concat("/ControlData/Functions/Provision/
orderItem[@index='",fn:data($orderComponentItem/@index),"']")}>
                      <ExternalFulfillmentState>{$item/su:Status/text()}</
ExternalFulfillmentState>
                     </Update>
             </xsl:for-each>
         </xsl:for-each>
      /OrderDataUpdate>
     </xsl:template>

     <xsl:template match="$response[su:status/text()='']">
      <xsl:variable name="log" select="java:info($log,concat('Received SOM Status Update:
SOM_InProgress or SOM_Canceled; ', $response/su:status/text()))"/>
      <xsl:variable name="automator" select="java:setUpdateOrder($automator, false())"/>
      <xsl:variable name="context" select="java:completeTaskOnExit($context, success())"/>
      <OrderDataUpdate xmlns="http://www.metasolv.com/OMS/OrderDataUpdate/2002/10/25">
         <xsl:for-each select="su:ParentLineId">
             <xsl:variable name="parent" select="."/>
             <xsl:for-each select="$component/oms:orderItem[oms:orderItemRef/
oms:LineXmlData/so:SalesOrderLine/corecom:Identification/corecom:ApplicationObjectKey/
corecom:ID/text() = $parent/text()]">
                 <xsl:variable name="index" select="@index"/>
                     <Update path="{fn:concat("/ControlData/Functions/Provision/
orderItem[@index='",fn:data($orderComponentItem/@index),"']")}>
                      <ExternalFulfillmentState>{$item/su:Status/text()}</
ExternalFulfillmentState>
                     </Update>
             </xsl:for-each>
         </xsl:for-each>
      </OrderDataUpdate>
     </xsl:template>

<!-- * -->
     <xsl:template match="* | @* | text()">
         <!-- do nothing -->
         <xsl:apply-templates/>
     </xsl:template>
</xsl:stylesheet>
```

External XSLT Sender

The Automated Task editor external XSLT sender receives task data from an external system, then sends the data (after possibly transforming the data) to another external system or even

returns the data back to the original external system. This XSLT combines characteristics of external XSLT automators and internal XSLT senders. See "External XSLT Automator" and "Internal XSLT Sender " for more information.

> **Note:**
>
> You must declare ScriptSenderContextInvocation in any external XSLT sender which inherits the ScriptReceiverContextInvocation class and methods used in internal or external automators.

Internal XSLT Automator

The Automated Task editor internal XSLT automator receives task data from OSM, then processes the data. For example, such an automation might perform computational actions on the data or other similar logic. This XSLT combines characteristics of external XSLT automators and internal XSLT senders. See "External XSLT Automator" and "Internal XSLT Sender " for more information.

> **Note:**
>
> You must declare ScriptReceiverContextInvocation class in an internal XSLT automator.

Automation Plug-in Examples for Events, Jeopardies, and Notifications

The following topics provide XQuery automation plug-in examples for:

- Event Automators
- Jeopardy Automators
- Order Notification Automation Plug-ins

Event Automators

An event automation plug-in can be triggered when an order or a task transitions into a defined milestone. The automation can be any internal XQuery, XSLT, or custom automation since the milestone event, by definition, can only be triggered by milestones happening within an order or a task. For more information about the characteristics for these automations, see "Automation Plug-in XQuery Examples", "Automation Plug-in XSLT Examples", and "Custom Java Automation Plug-ins."

> **Note:**
>
> For an event automation plug-in you must declare the OrderNotificationContext instead of TaskContext. For example:
>
> ```
> declare namespace context =
> "java:com.mslv.oms.automation.OrderNotificationContext";
> ```

The following example is an internal sender automation plug-in that uses methods available to the OrderNotificationContext class to get milestone data from the order and sends an notification message to an external system. Because this sender does not expect a response message (a fire-and-forget message), you must use the OrderNotificationContext class ackNotificationOnExit method to clear the JMS correlation ID for the notification. Also, events do not transition tasks, so you must not specify completeTaskOnExit in a notification.

```
declare namespace saxon="http://saxon.sf.net/";
declare namespace xsl="http://www.w3.org/1999/XSL/Transform";
declare namespace log = "java:org.apache.commons.logging.Log";
declare namespace outboundMessage = "java:javax.jms.TextMessage";
declare namespace oms="urn:com:metasolv:oms:xmlapi:1";
declare namespace osm="http://xmlns.oracle.com/communications/ordermanagement/model";
declare namespace context = "java:com.mslv.oms.automation.OrderNotificationContext";

declare variable $context external;
declare variable $log external;
declare variable $outboundMessage external;

let $taskData := fn:root(.)/oms:GetOrder.Response
let $correlationId := $taskData/oms:_root/oms:Id/text()
let $controlDataArea := if (fn:exists($taskData/oms:_root/oms:ControlData))
                        then $taskData/oms:_root/oms:ControlData
                        else ()

return
(
log:info($log, fn:concat('COMCartridge: Invoking orderCompletionNotification for
order[',$taskData/oms:OrderID/text(),'] with correlation [', $correlationId,']')),
context:ackNotificationOnExit($context),
outboundMessage:setStringProperty($outboundMessage, "COMCorrelationID", $correlationId),
outboundMessage:setStringProperty($outboundMessage, "SUB_FOLDER_NAME", $taskData/
oms:_root/oms:OrderNumber/text()),
outboundMessage:setStringProperty($outboundMessage, "COMMilestone",
"COMOrderCompleteEvent"),
<orderNotification xmlns="http://xmlns.oracle.com/communications/sce/dictionary/
CommonResourcesCartridge/Notifications"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <OSMOrderId>{$taskData/oms:OrderID/text()}</OSMOrderId>
    <Id>{$correlationId}</Id>
    <OrderNumber>{$taskData/oms:_root/oms:OrderNumber/text()}</OrderNumber>
    {
    for $serviceInstance in $controlDataArea/oms:OrderItem
    return
        <Instance>
            <InstanceID>{$serviceInstance/oms:instanceID/text()}</InstanceID>
            <OrderLineId>{$serviceInstance/oms:orderLineId/text()}</OrderLineId>
            <Status>{$serviceInstance/oms:status/text()}</Status>
        </Instance>
    }
</orderNotification>
)
```

Jeopardy Automators

An order jeopardy automation plug-in can be triggered when a particular condition is met, such as when a task exceeds the expected duration configured for the task or when the process that the task is a part of exceeds its excepted process duration. The automation can be any internal XQuery, XSLT, or custom automation since the jeopardy, by definition, can only be triggered by events happening within the task or the process. For more information about the

characteristics for these automations, see "Automation Plug-in XQuery Examples", "Automation Plug-in XSLT Examples", and "Custom Java Automation Plug-ins."

> **Note:**
>
> For an order level jeopardy automation plug-in you must declare the OrderNotificationContext instead of TaskContext. For example:
>
> ```
> declare namespace context =
> "java:com.mslv.oms.automation.OrderNotificationContext";
> ```
>
> For a task level jeopardy automation plug-in, if the task level jeopardy condition **Multiple events per Task instance** is set indicating that the task is a multi-instance task and the event should be triggered for each instance, then you must declare `TaskNotificationContext` so that the task data is passed to each instance of the event. If the task is not a multi-instance task, then `OrderNotificationContext` should be declared.

The following example is an internal automator plug-in that uses methods available to the OrderNotificationContext class to get notification details from the task in combination with the XML API Notification.Request that logs the jeopardy notification details. Other jeopardy examples could also send an email or trigger a pager.

```
declare namespace oms="urn:com:metasolv:oms:xmlapi:1";
declare namespace automator =
"java:oracle.communications.ordermanagement.automation.plugin.ScriptReceiverContextInvoca
tion";
declare namespace context = "java:com.mslv.oms.automation.OrderNotificationContext";
declare namespace log = "java:org.apache.commons.logging.Log";

declare option saxon:output "method=xml";
declare option saxon:output "saxon:indent-spaces=2";

declare variable $automator external;
declare variable $context external;
declare variable $log external;

declare variable $exitStatus := "success";

let $thisOrderId := context:getOrderId($context)
(: let $taskMnemonic := context:getTaskMnemonic($context) :)
let $notificationName := context:getNotificationName($context)
let $notificationType := context:getNotificationType($context)
let $orderId := fn:root(.)/oms:GetOrder.Response/oms:_root/oms:orderId
let $xmlRequest := '<Notifications.Request xmlns="urn:com:metasolv:oms:xmlapi:1" />'
let $notifications := context:processXMLRequest($context, $xmlRequest)
return (
  log:info($log, fn:concat("XQuery jeopardy: order[", $thisOrderId,
    "], notificationContext [", context:getClass($context),
    "], notificationName[", $notificationName,
    "], notificationType[", $notificationType,
    "], notifications[", $notifications,
    "] entered order ID [", $orderId/text(), "]")),
  <placeholder/>
)
```

Order Notification Automation Plug-ins

An order notification automation plug-in can be triggered when specified data changes in the order. For example, you can monitor order status changes using the orchestration data element **ControlData/OrderFulfillmentState** or individual order item status changes using **ControlData/OrderItem/OrderItemFulfillmentState** so OSM triggers an internal XQuery sender automation plug-in that sends these status changes to another system, such as from a SOM OSM system to a COM OSM system, or from a COM OSM system to a CRM.

The automation can be any internal XQuery, XSLT, or custom automation since the notification, by definition, can only be triggered by a change in the internal order data. For more information about the characteristics for these automations, see "Automation Plug-in XQuery Examples", "Automation Plug-in XSLT Examples", and "Custom Java Automation Plug-ins."

> **Note:**
>
> For an order notification automation plug-in you must declare the OrderDataChangeNotificationContext instead of TaskContext. For example:
>
> ```
> declare namespace context =
> "java:com.mslv.oms.automation.OrderDataChangeNotificationContext";
> ```

The following example is an internal XQuery sender that sends any order and order item fulfillment state changes to another OSM system. It also provides stubs for transforming the fulfillment states to external system message formats.

```
declare namespace osm="urn:com:metasolv:oms:xmlapi:1";
declare namespace log = "java:org.apache.commons.logging.Log";
declare namespace to="http://TechnicalOrder";
declare namespace automator =
"java:oracle.communications.ordermanagement.automation.plugin.ScriptSenderContextInvocati
on";
declare namespace su="http://StatusUpdate";
declare namespace context =
"java:com.mslv.oms.automation.OrderDataChangeNotificationContext";
declare namespace outboundMessage = "java:javax.jms.TextMessage";

declare variable $log external;
declare variable $outboundMessage external;

(:
   This function is for indication purposes only.
   OSM Fulfillment State can be mapped according the expectation of Upstream
:)
declare function local:getUpstreamFulfillmentState($fulfillmentState as xs:string) as
xs:string {
    (: fn:concat('Order_Upstream_' , $fulfillmentState) :)
    fn:concat('' , $fulfillmentState)
};

(:
   This function is for indication purposes only.
   OSM Fulfillment State can be mapped according the expectation of Upstream
:)
declare function local:getUpstreamOrderItemFulfillmentState($fulfillmentState as
xs:string) as xs:string {
```

```
    (: fn:concat('OrderItem_Upstream_' , $fulfillmentState) :)
    fn:concat('' , $fulfillmentState)
};

let $order := ..//osm:GetOrder.Response
let $orderFulfillmentState := $order/osm:_root/osm:ControlData/osm:OrderFulfillmentState
let $mappedUpstreamFulfillmentState := if(exists($orderFulfillmentState)) then
local:getUpstreamFulfillmentState($orderFulfillmentState/text()) else ()

return
(
log:info($log,'Sending Upstream Fulfillment State'),
outboundMessage:setStringProperty($outboundMessage, "SOMTOMCorrelationHeader",
concat($order/osm:_root/osm:messageXmlData/to:TechnicalOrder/to:SOMOrderId/text(),'-
SOM')),
if (fn:count($order/osm:_root/osm:ControlData/osm:OrderItem)=0) then (
<StatusUpdate xmlns="http://StatusUpdate">
<numSalesOrder>{$order/osm:Reference/text()}</numSalesOrder>
<numOrder>{$order/osm:OrderID/text()}</numOrder>
<typeOrder>{$order//osm:OrderHeader/osm:typeOrder/text()}</typeOrder>
<errorCode>0</errorCode>
<status>cancelled</status>
</StatusUpdate>
) else (
<StatusUpdate xmlns="http://StatusUpdate">
<numSalesOrder>{$order/osm:Reference/text()}</numSalesOrder>
<numOrder>{$order/osm:OrderID/text()}</numOrder>
<typeOrder>{$order//osm:OrderHeader/osm:typeOrder/text()}</typeOrder>
<errorCode>0</errorCode>
<status>{$mappedUpstreamFulfillmentState}</status>
{
    for $orderItem in $order/osm:_root/osm:ControlData/osm:OrderItem
    where exists($orderItem/osm:OrderItemFulfillmentState)
    return
        <OrderItem>
            <LineName>{$orderItem/osm:LineName/text()}</LineName>
            <LineId>{$orderItem/osm:LineId/text()}</LineId>
            <ParentLineId>{$orderItem/osm:ParentLineId/text()}</ParentLineId>
            <SpecificationName>{$orderItem/osm:TypeCode/text()}</SpecificationName>
            <Status>{local:getUpstreamOrderItemFulfillmentState($orderItem/
osm:OrderItemFulfillmentState/text())}</Status>
        </OrderItem>
    }
</StatusUpdate>
)
)
```

Custom Java Automation Plug-ins

This topic provides common usage examples for custom Java automation plug-ins.

- Internal Custom Java Automator
- Internal Custom Java Sender
- External Custom Java Automator that Changes the OSM Task Status
- External Custom Java Automator that Updates Order Data
- Using OrderDataUpdate Elements to Pass Order Modification Data
- Examples of Sending Messages to External Systems

- Examples of Handling Responses from External Systems

Internal Custom Java Automator

A basic internal custom Java automator has the following characteristics:

- The name of the custom automation package. For example:

```
package com.mslv.oms.sample.atm_frame;
```

- Import statements required for this custom automation plug-in. For example:

```
import com.mslv.oms.automation.plugin.*;
import com.mslv.oms.automation.*;
import java.rmi.*;
```

- An arbitrary class name that extends AbstractAutomator. For the automation framework to call an internal custom Java automator, the plug-in must extend the AbstractAutomator class. This class resides in the com.mslv.automation.plugin package. For example:

```
public class MyPlugin extends AbstractAutomator {
```

- The required run method, as dictated by the parent class, AbstractAutomator:

```
protected void run(String inputXML, AutomationContext context)
                throws com.mslv.oms.automation.AutomationException {
```

- Cast the AutomationContext object to the TaskContext object. This example assumes that the custom automation plug-in is triggered by an automated task, so the code is expecting the context input an argument to be an instance of the TaskContext object.

```
TaskContext taskContext = (TaskContext)context;
```

> **Note:**
>
> You can use the TaskContext object to do many things, such as complete the task, suspend it, and so on. For more information about this class, see the OSM Javadocs.

- Call a method on the TaskContext object to retrieve the task name.

```
String taskName = taskContext.getTaskMnemonic();
```

- Add any require business logic.

```
this.performAutomation(taskname);
```

The following example shows the minimal amount of code required for a custom automation plug-in to run. This example assumes that it is triggered by an automated task.

```
package com.mslv.oms.sample.atm_frame;

  import com.mslv.oms.automation.plugin.*;
  import com.mslv.oms.automation.*;
  import java.rmi.*;

  public class MyPlugin extends AbstractAutomator {
   protected void run(String inputXML, AutomationContext context)
                  throws com.mslv.oms.automation.AutomationException {
      try {
        TaskContext taskContext = (TaskContext)context;
        String taskName = taskContext.getTaskMnemonic();
        this.performAutomation(taskname);
```

```
    catch(RemoteException ex) {
      throw new AutomationException(ex); }
    catch(AutomationException x) {
      throw x; }
  }
}
```

Internal Custom Java Sender

A basic internal custom Java sender has the following characteristics:

- The name of the custom automation package. For example:

```
package com.mslv.oms.sample.atm_frame;
```

- Import statements required for this custom automation plug-in. For example:

```
import com.mslv.oms.automation.plugin.*;
import com.mslv.oms.automation.*;
import java.rmi.*;
```

- An arbitrary class name that extends AbstractSendAutomator. For the automation framework to call an internal custom Java sender, the plug-in must extend the AbstractSendAutomator class. This class resides in the com.mslv.automation.plugin package. For example:

```
public class MyPlugin extends AbstractSendAutomator {
```

- The required run method, as dictated by the parent class, AbstractSendAutomator

```
protected void run(String inputXML, AutomationContext context)
                throws com.mslv.oms.automation.AutomationException {
```

- Cast the AutomationContext object to the TaskContext object. This example assumes that the custom automation plug-in is triggered by an automated task, so the code is expecting the context input an argument to be an instance of the TaskContext object.

```
TaskContext taskContext = (TaskContext)context;
```

> **Note:**
>
> You can use the TaskContext object to do many things, such as complete the task, suspend it, and so on. For more information about this class, see the OSM Javadocs.

- Call a method on the TaskContext object to retrieve the task name.

```
String taskName = taskContext.getTaskMnemonic();
```

- Sets the text for the outbound message, which is sent to the external message queue defined by the automation definition. The custom code does not establish a connection to an external system or send the message; the automation framework handles the connection and sends the message upon completion of the makeRequest method.

```
outboundMessage.setText("Received task event for task = " + taskName);}
```

> **Note:**
>
> OSM provides outboundMessage in the OSM automation framework as a JMS message with text content. If you require other message formats or protocols, do not use outboundMessage. You must implement an internal custom java automator or helper class with the required code.

The following example shows the minimal amount of code required for a custom automation plug-in that sends data to run. This example assumes that it is triggered by an automated task.

```
package com.mslv.oms.sample.atm_frame;

import com.mslv.oms.automation.plugin.*;
import com.mslv.oms.automation.*;
import javax.jms.TextMessage;
import java.rmi.*;

public class MyPlugin extends AbstractSendAutomator {
  protected void makeRequest(String inputXML, AutomationContext context,
                TextMessage outboundMessage)
                throws com.mslv.oms.automation.AutomationException {
    try {
      TaskContext taskContext = (TaskContext)context;
      String taskName = taskContext.getTaskMnemonic();

      // optional - You can use this code if you want to define your own correlation
ID rather than an autogenerated correlation ID.
      Correlator correlator = getCorrelator(context);
      correlator.add(createCustomCorrelationId(taskContext));

      outboundMessage.setText("Received task event for task = " + taskName);}
    catch(javax.jms.JMSException ex) {
      throw new AutomationException(ex); }
    catch(RemoteException x) {
      throw new AutomationException(x); }
  }

  private String createCustomCorrelationId(TaskContext taskContext) {
   // Create a custom correlation ID using task name and unique order history ID
   // Actual correlation calculation depends on solution logic
      String corrId = taskContext.getTaskMnemonic()
                    + "-"
                    + String.valueOf(taskContext.getOrderHistoryId());
      return corrId;
  }

}
```

External Custom Java Automator that Changes the OSM Task Status

A basic external custom Java automator that changes the OSM task status has the following characteristics:

- The name of the custom automation package. For example:

  ```
  package com.mslv.oms.sample.atm_frame;
  ```

- Import statements required for this custom automation plug-in. For example:

```
import com.mslv.oms.automation.plugin.*;
import com.mslv.oms.automation.*;
import java.rmi.*;
```

- An arbitrary class name that extends AbstractAutomator. For the automation framework to call an external custom Java sender, the plug-in must extend the AbstractAutomator class. This class resides in the com.mslv.automation.plugin package. The name reflects that this example is an external event receiver, receiving information from ASAP. For example:

```
public class AsapResponseHandler extends AbstractAutomator {
```

- The required run method, as dictated by the parent class, AbstractAutomator.

```
public void run(String inputXML, AutomationContext task)
            throws AutomationException {
```

- Cast the AutomationContext object to the TaskContext object. This example assumes that the custom automation plug-in is triggered by an automated task, so the code is expecting the context input an argument to be an instance of the TaskContext object.

```
TaskContext taskContext = (TaskContext)context;
```

> **Note:**
>
> You can use the TaskContext object to do many things, such as complete the task, suspend it, and so on. For more information about this class, see the OSM Javadocs.

- Call a method on the TaskContext object to retrieve the task name.

```
String taskName = taskContext.getTaskMnemonic();
```

- Logs the information regarding the response that the plug-in is handling. AtmFrameCatalogLogger is available to this example plug-in based on the package in which the plug-in resides. You must replace this with your own solution logic.

```
AtmFrameCatalogLogger.logTaskEventResponse
  (taskName,tctx.getOrderId(),tctx.getOrderHistoryId(),inputXML);
```

> **Note:**
>
> The automation framework keeps track of the order ID and the order history ID of the task that triggered the automation. There are two ways you can get the Order History ID:
>
> – By parsing the inputXML
>
> – By calling the TaskContext.getOrderHistoryId method as shown in this example.
>
> In most cases, these return the same order history ID. However, if you use automation to handle task events, the order history ID obtained from:
>
> – Parsing the inputXML returns the order history ID as it was when the task was generated
>
> – Calling the TaskContext.getOrderHistoryID method returns the order history ID as it is now (current)

- Update the task status by calling a method on the TaskContext object.

```
          tctx.completeTaskOnExit("activation_successful"); }
```

The following example shows an external custom automator that updates the OSM task status. This example assumes that the automation definition is an external event receiver that is receiving a message from ASAP, and that it is triggered by an automated task.

```
package com.mslv.oms.sample.atm_frame;

import com.mslv.oms.automation.plugin.*;
import com.mslv.oms.automation.*;
import java.rmi.*;

public class AsapResponseHandler extends AbstractAutomator {
  public void run(String inputXML, AutomationContext task)
              throws AutomationException {
    try {
      TaskContext tctx = (TaskContext)task;
      String taskName = tctx.getTaskMnemonic();
      AtmFrameCatalogLogger.logTaskEventResponse
        (taskName,tctx.getOrderId(),tctx.getOrderHistoryId(),inputXML);
      tctx.completeTaskOnExit("activation_successful"); }
    catch(RemoteException ex) {
      throw new AutomationException(ex); }
    catch(AutomationException x) {
      throw x; }
  }
}
```

External Custom Java Automator that Updates Order Data

If an automated task sends data to an external system and the external system sends a response back, you may need to update OSM with the data received from the external system.

The following example shows how to update data in OSM. The code is an example of updating OSM with data received from Oracle Communications Unified Inventory Management (UIM) when calling the server extension `FRDemo.AssignFacilities`.

```
package com.mslv.oms.sample.atm_frame;

import com.mslv.oms.automation.plugin.*;
import com.mslv.oms.automation.*;
import java.rmi.*;
import java.util.*;
import java.io.*;
import java.net.*;
import org.xml.sax.*;
import org.w3c.dom.*;
import javax.xml.parsers.*;

public class UIMResponseHandler extends AbstractAutomator {

  public void run( String inputXML, AutomationContext task)
              throws AutomationException {
    try {
      TaskContext tctx = (TaskContext)task;
      String taskName = tctx.getTaskMnemonic();
      AtmFrameCatalogLogger.logTaskEventResponse
        (taskName,tctx.getOrderId(),tctx.getOrderHistoryId(),inputXML);

      // Using the data returned from UIM, update the OSM order data
      String updateXml = generateOMSUpdateString(inputXML);
      tctx.updateOrderData(updateXml);
```

```
    // Complete the OSM task with the correct status
    tctx.completeTaskOnExit( "success" ); }

  catch(OrderUpdateException ex) {
    throw new AutomationException( ex ); }
  catch(RemoteException ex) {
    throw new AutomationException( ex ); }
 catch(AutomationException x ) {
   throw x; }
}

static private String generateOMSUpdateString(String inputXML) {
  StringBuffer osmUpdate = new StringBuffer("");
  try {
    osmUpdate = new StringBuffer
     ("<OrderDataUpdate xmlns=\"http://www.w3.org/2001/XMLSchema\""+
      " xmlns:xs=\"http://www.w3.org/2001/XMLSchema\"" +
      " xmlns:odu=\"http://www.oracle.com/OMS/OrderDataUpdate\"" +
      " targetNameSpace=\"http://www.oracle.com/OMS/OrderDataUpdate\">");

    // Use updates from UIM to update OSM
    osmUpdate.append("<AddMandatory>true</AddMandatory>");
    DocumentBuilderFactory docBuilderFactory =
      DocumentBuilderFactory.newInstance();
    DocumentBuilder parser = docBuilderFactory.newDocumentBuilder();
    Document doc = parser.parse(new StringBufferInputStream(inputXML));
    Element root = doc.getDocumentElement();
    root.normalize();
    NodeList a_site_list = root.getElementsByTagName("a_site information");
    NodeList a_site_data = a_site_list.item(0).getChildNodes();

    for(int i=0;i<a_site_data.getLength();i++) {
      Element e = (Element)a_site_data.item(i);
      osmUpdate.append("<Add path=\"/a_site_information/");
      osmUpdate.append(e.getTagName());
      osmUpdate.append("\">");
      osmUpdate.append(e.getFirstChild().getNodeValue());
      osmUpdate.append("</Add>");
    }

    NodeList z_site_list = root.getElementsByTagName("z_site_information");
    NodeList z_site_data = z_site_list.item(0).getChildNodes();

    for(int i=0;i<a_site_data.getLength();i++) {
      Element e = (Element)a_site_data.item(i);
      osmUpdate.append("<Add path=\"/z_site_information/");
      osmUpdate.append(e.getTagName());
      osmUpdate.append("\">");
      osmUpdate.append(e.getFirstChild().getNodeValue());
      osmUpdate.append("</Add>");
    }

    osmUpdate.append("</OrderDataUpdate>");

    System.out.println(omsUpdate.toString()); }

catch(Exception e) {
  System.out.println(e.getMessage()); }

return omsUpdate.toString();
```

```
    }
  }
```

The following code snippets from this example show:

- How to display where OSM data is updated, using XML input to describe which data nodes to update.

```
tctx.updateOrderData(updateXml);
```

- How to build the OrderDataUpdate XML string to update the data in OSM using data garnered by parsing the UIM XML. See "Using OrderDataUpdate Elements to Pass Order Modification Data" for more information. This differs for every order template and every external system. This code represents the translation step where you convert the data from the format of an external system to the format that OSM expects.

```
static private String generateOMSUpdateString(String inputXML) {
  StringBuffer osmUpdate = new StringBuffer("");
  try {
    osmUpdate = new StringBuffer
     ("<OrderDataUpdate xmlns=\"http://www.w3.org/2001/XMLSchema\""+
      " xmlns:xs=\"http://www.w3.org/2001/XMLSchema\"" +
      " xmlns:odu=\"http://www.oracle.com/OMS/OrderDataUpdate\"" +
      " targetNameSpace=\"http://www.oracle.com/OMS/OrderDataUpdate\">");

    // Use updates from UIM to update OSM
    osmUpdate.append("<AddMandatory>true</AddMandatory>");
    DocumentBuilderFactory docBuilderFactory =
      DocumentBuilderFactory.newInstance();
    DocumentBuilder parser = docBuilderFactory.newDocumentBuilder();
    Document doc = parser.parse(new StringBufferInputStream(inputXML));
    Element root = doc.getDocumentElement();
    root.normalize();
    NodeList a_site_list = root.getElementsByTagName("a_site information");
    NodeList a_site_data = a_site_list.item(0).getChildNodes();

    for(int i=0;i<a_site_data.getLength();i++) {
      Element e = (Element)a_site_data.item(i);
      osmUpdate.append("<Add path=\"/a_site_information/");
      osmUpdate.append(e.getTagName());
      osmUpdate.append("\">");
      osmUpdate.append(e.getFirstChild().getNodeValue());
      osmUpdate.append("</Add>");
    }

    NodeList z_site_list = root.getElementsByTagName("z_site_information");
    NodeList z_site_data = z_site_list.item(0).getChildNodes();

    for(int i=0;i<a_site_data.getLength();i++) {
      Element e = (Element)a_site_data.item(i);
      osmUpdate.append("<Add path=\"/z_site_information/");
      osmUpdate.append(e.getTagName());
      osmUpdate.append("\">");
      osmUpdate.append(e.getFirstChild().getNodeValue());
      osmUpdate.append("</Add>");
    }

    osmUpdate.append("</OrderDataUpdate>");

    System.out.println(omsUpdate.toString()); }

catch(Exception e) {
  System.out.println(e.getMessage()); }
```

```
return omsUpdate.toString();
}
```

The structure of the XML document to update OSM data is as follows:

```
<OrderDataUpdate xmlns=\"http://www.w3.org/2001/XMLSchema\"
xmlns:xs=\"http://www.w3.org/2001/XMLSchema\"
xmlns:odu=\"http://www.oracle.com/OMS/OrderDataUpdate\"
targetNameSpace=\"http://www.oracle.com/OMS/OrderDataUpdate\">
<AddMandatory>true</AddMandatory>
<Add path=\"/service_details/new_number\">98765</Add>
<Update path=\"/customer_details/service_address/street\">55 Updated St</Update>
<Delete path=\"/service_details/current_account_number\"></Delete>
</OrderDataUpdate>
```

This example illustrates adding a data node (Add path), updating a data node (Update path), and deleting a data node (Delete path).

- How to specify a mandatory parameter. If set to true, the following rules apply:

```
osmUpdate.append("<AddMandatory>true</AddMandatory>");
```

  - If you delete a mandatory node, AddMandatory replaces the node and populates it with the default value.
  - If the update is missing a mandatory node, AddMandatory adds the missing node and populates it with the default value.

> **Note:**
>
> If you add a mandatory field, but do not include a value, AddMandatory will not add a default value and the request will generate an error-error code 200.

Using OrderDataUpdate Elements to Pass Order Modification Data

You use OrderDataUpdate XML elements to pass data add, modify and delete data nodes in an order.

OrderDataUpdate elements can be passed as a parameter to updateOrderData(). XSL translations whose results are passed to setUpdateOrder() must be in OrderDataUpdate format. See the OSM Javadocs for details on both methods. You can also pass OrderDataUpdate format elements to the DataChange Web Service (see the SDK schema OrderManagementWS.xsd) and UpdateOrder.request XML API call (see the SDK schema oms-xmlapi.xsd).

For update and delete operations on multi-instance nodes, you must specify the order node index as it exists in the input XML. Specify the order node index as "[@index='index_value']" where index_value is the order node index.

The following example shows how to specify the addition of an order node with OrderDataUpdate. The path attribute identifies the parent node under which to add the element:

```
<OrderDataUpdate>
  <Add path="/">
    <ProvisioningOrderResponse>
      <OrderInformation>
        <OrderNumber>1238723</OrderNumber>
```

```
      </OrderInformation>
    </ProvisioningOrderResponse>
  </Add>
</OrderDataUpdate>
```

The following example shows a combined update and delete operation on a multi-instance node using OrderDataUpdate. In Delete attributes, the path attribute identifies the data to delete. In Update attributes, the path attribute identifies the data to update. Indexes are required on Update and Delete attributes when modifying multi-instance nodes. Note how the order node index values are specified in the Update and Delete attributes.

```
<OrderDataUpdate>
  <Delete path="/client_info/address[@index='80132']/city" />
  <Update path="/client_info/address[@index='76579']/city">Newark</Update>
  <Update path="/customer_details/service_address/street">55 Updated St</Update>"
  <Delete path="/service_details/current_account_number"></Delete>
</OrderDataUpdate>
```

See "External Custom Java Automator that Updates Order Data" for an example in which OrderDataUpdate XML data is created dynamically within Java code and passed to UpdateOrderData().

The schema for OrderDataUpdate is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://
www.metasolv.com/OMS/OrderDataUpdate" xmlns:odu="http://www.metasolv.com/
OMS/OrderDataUpdate" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">

  <element name="OrderDataUpdate">
  <complexType>
    <choice maxOccurs="unbounded">
      <element ref="odu:Add"/>
      <element ref="odu:Delete"/>
      <element ref="odu:Update"/>
    </choice>
  </complexType>
  </element>

  <element name="Add">
    <annotation>
      <documentation>It contains a node to be added. The path attribute identifies the
parent node under which to add the element.</documentation>
    </annotation>
    <complexType>
      <sequence>
        <any/>
      </sequence>
      <attribute name="path" type="string" use="required"/>
    </complexType>
  </element>

  <element name="Delete">
    <annotation>
      <documentation>It contains a node to be deleted. The path attribute identifies the
node to delete.</documentation>
    </annotation>
    <complexType>
      <attribute name="path" type="string" use="required"/>
    </complexType>
  </element>
```

```
  <element name="Update">
    <annotation>
      <documentation>It contains a node to update. The path attribute identifies the
node to update.</documentation>
    </annotation>
    <complexType>
      <simpleContent>
        <extension base="string">
          <attribute name="path" type="string" use="required"/>
        </extension>
      </simpleContent>
    </complexType>
  </element>
</schema>
```

Examples of Sending Messages to External Systems

Automation simplifies the process of sending messages to external systems. The automation framework does the following:

- Assumes the protocol is JMS. The products (Siebel, OSM, UIM, ASAP, IP Service Activator) all have JMS APIs.
- Takes care of establishing and maintaining the various JMS connections.
- Constructs the JMS messages, setting the required message properties.
- Guarantees delivery of the message and handles any errors or exceptions. It retries until the message is delivered.
- Automatic message correlation.
- Poison message handling.

An OSM event that is sent to an external system follows this process flow:

1. OSM runs an automation that triggers an automation plug-in.
2. Internally, the automation framework maps the plug-in, using the **automationMap.xml** configuration, onto custom business logic and calls the makeRequest method on the custom automator class.
3. The makeRequest method performs some business logic and sets the content of the outbound message.
4. The automation framework adds properties to the outbound message to aid in correlating external system responses to requests.
5. The automation framework uses information from the **automationMap.xml** to send the JMS message to the JMS queue representing the external system.

The following example shows a custom automation plug-in that sends data to an external system.

```
package com.mslv.oms.sample.atm_frame;

import com.mslv.oms.automation.plugin.*;
import com.mslv.oms.automation.*;
import javax.jms.TextMessage;
import java.rmi.*;

public class ObjectelPlugin extends AbstractSendAutomator {

protected void makeRequest(String inputXML, AutomationContext context, TextMessage
```

```
outboundMessage) throws com.mslv.oms.automation.AutomationException {

 try {
 TaskContext taskContext = (TaskContext)context;
 String taskName = taskContext.getTaskMnemonic();
 AtmFrameCatalogLogger.logTaskEvent(taskName, taskContext.getOrderId(),
taskContext.getOrderHistoryId(), inputXML);

 //
 // Set the outgoing message
 //
 String xmlRequest = "<Message
type=\"ni\"><iLibPlus:findFunctionalPortOnLocation.Request xmlns:iLibPlus=\"http://
www.oracle.com/objectel\"><location><DS><AG2ObjectID>189438</
AG2ObjectID><AG2ParentID>189428</AG2ParentID><CLLIX>XML.CO.1</CLLIX><SiteName>XML.CO.1</
SiteName></DS></location><feType>PP</feType><portType>$FEP</
portType><selectionMethod>LOAD_BALANCE</
selectionMethod><portSelectionAttribName><string>AG2ObjectID</
string><string>AG2ParentID</string><string>AG2PortLabel</string></
portSelectionAttribName><portSelectionAttribValue><string>189508</string><string>189478</
string><string>F-31-OC-48</string></portSelectionAttribValue><portUpdateAttribName/
><portUpdateAttribValue/></iLibPlus:findFunctionalPortOnLocation.Request></Message>";
 outboundMessage.setText( xmlRequest );

 } catch( javax.jms.JMSException x ) {
 throw new AutomationException( x );
 } catch(RemoteException ex){
 throw new AutomationException( ex );
 }
 }
 }
```

The following code snippets from this example show:

- how to generate an output XML string. In this example it is hard coded. In a business case you would use business logic to transform OSM data into what the external system expects

  ```
   String xmlRequest = "<Message
  type=\"ni\"><iLibPlus:findFunctionalPortOnLocation.Request xmlns:iLibPlus=\"http://
  www.oracle.com/objectel\"><location><DS><AG2ObjectID>189438</
  AG2ObjectID><AG2ParentID>189428</AG2ParentID><CLLIX>XML.CO.1</
  CLLIX><SiteName>XML.CO.1</SiteName></DS></location><feType>PP</
  feType><portType>$FEP</portType><selectionMethod>LOAD_BALANCE</
  selectionMethod><portSelectionAttribName><string>AG2ObjectID</
  string><string>AG2ParentID</string><string>AG2PortLabel</string></
  portSelectionAttribName><portSelectionAttribValue><string>189508</
  string><string>189478</string><string>F-31-OC-48</string></
  portSelectionAttribValue><portUpdateAttribName/><portUpdateAttribValue/></
  iLibPlus:findFunctionalPortOnLocation.Request></Message>";
  ```

- how to set the output data:

  ```
   outboundMessage.setText( xmlRequest );
  ```

- how this code does not establish a connection to an external system or send a message. After the data is set in the code, the message is automatically sent upon exit of the makeRequest method.

Examples of Handling Responses from External Systems

In Message Property Correlation, the following steps describe how responses from external systems are handled.

1. The plug-in populates the message content.
2. The plug-in sets a property on the outbound JMS message, with name of the value set for correlationproperty in the **automationMap.xml** file, and a value decided by the business logic. For example, you could use this to correlate on a reference number.
3. If the value of the correlationproperty in the **automationMap.xml** file is set to the value JMSCorrelationID, the plug-in is not required to set the property on the outbound message (as described in Step 2). The automation framework does this automatically.
4. The automation framework saves the message properties set for each message with the event information.
5. The automation framework sets the replyTo property on the JMS message.
6. The external system copies the properties on the request message to the response message.
7. The external system sends the message to the reply queue specified in the **automationMap.xml** file.
8. The automation framework uses the configuration in the **automationMap.xml** file to map messages from external systems to plug-ins. The plug-ins are automators written by system integrators. Configuration of an automator for receiving messages from an external system are defined within Design Studio and saved to the **automationMap.xml** file.
9. The automation framework uses the message properties of the response, plus the correlation information saved in step four above, to reload a Context for the response message.
10. The run method of the external system automator is called and is passed the Context created in step 9.
11. The automator performs business logic, such as completing the task.

The following example shows a custom automation plug-in that handles and processes response messages from an external system.

```
package com.mslv.oms.sample.atm_frame;

import com.mslv.oms.automation.plugin.*;
import com.mslv.oms.automation.*;
import java.rmi.*;

public class UIMResponseHandler extends AbstractAutomator {

public void run( String inputXML, AutomationContext task)
throws AutomationException {

try {
TaskContext tctx = (TaskContext)task;

tctx.completeTaskOnExit( "success" );

} catch(RemoteException ex){
throw new AutomationException( ex );
} catch(AutomationException x ) {
throw x;
```

```
 }
 }
}
```

This automation plug-in does not need to send JMS messages to any system, so it extends AbstractAutomator and is intended to process Task automation responses, so it casts the Context to a TaskContext then completes the task.

The following example shows what the external system is expected to do for the message property correlation to work.

```
 public void sendMessage(Message originalMessage) {
 try {
 //
 // Set up the JMS connections
 //
 QueueConnectionFactory connectionFactory =
(QueueConnectionFactory)jndiCtx.lookup(connectionFactoryName);
 QueueConnection queueConnection = connectionFactory.createQueueConnection();
 QueueSession queueSession = queueConnection.createQueueSession(false,
Session.AUTO_ACKNOWLEDGE);
 Queue replyQueue = (Queue)originalMessage.getJMSReplyTo();
 QueueSender queueSender = queueSession.createSender(replyQueue);

 //
 // Create the message
 //
 TextMessage textMessage =
queueSession.createTextMessage(((TextMessage)originalMessage).getText());
 textMessage.setStringProperty("MESSAGE_NAME","ActivationResponse");
 textMessage.setJMSCorrelationID(originalMessage.getJMSCorrelationID());

 //
 // Send the message
 //
 queueSender.send(textMessage, javax.jms.DeliveryMode.PERSISTENT,
javax.jms.Message.DEFAULT_PRIORITY, 1800000);

 } catch(javax.jms.JMSException ex){
 ex.printStackTrace();
 } catch(javax.naming.NamingException ex){
 ex.printStackTrace();
 }
 }
```

The following code snippets from this example show:

- how the external system chooses which JMS destination to send the reply to.

  ```
  Queue replyQueue = (Queue)originalMessage.getJMSReplyTo();
  QueueSender queueSender = queueSession.createSender(replyQueue);
  ```

- the external system setting a property that identifies the nature of the JMS message. This implies that the automation was defined with a message property selector select statement that matches these parameters.

  ```
  textMessage.setStringProperty("MESSAGE_NAME","ActivationResponse");
  ```

- the external system echoing the correlation information onto the reply message. This implies that the automation was defined to correlate based on JMSCorrelationID.

  ```
  textMessage.setJMSCorrelationID(originalMessage.getJMSCorrelationID());
  ```

Compensation XQuery Expressions

The following topics provide information about automation and manual task compensation XQuery expressions.

- Task Re-Evaluation and Rollback XQuery Expressions
- In Progress Compensation Include XQuery Expressions
- In Progress Compensation Complete XQuery Expressions
- In Progress Compensation Grace Period XQuery Expressions

For general OSM XQuery information, see "General XQuery Information".

Task Re-Evaluation and Rollback XQuery Expressions

You can dynamically assign compensation strategies to tasks by creating XQuery expressions in the Design Studio **Task Editor Compensation** tab for re-evaluation compensation strategies or compensation strategies for when a task is no longer required.

> **Note:**
>
> If the XQuery expression is invalid OSM logs the error but does not rollback the transaction. Instead, OSM uses the static compensation strategy as the default.

This section refers to the Design Studio OSM **Automated Task** or **Manual Task** editor, **Compensation** tab Compensation Expression XQuery field for re-evaluation compensation strategies:

- Context: The context for this XQuery is the current order data. You can get the current order data using the XML API GetOrder.Response function.
- Prolog: You can declare the XML API namespace to use the GetOrder.Response function in the XQuery body to extract the order information. You must declare the java:oracle:communications.ordermanagement.compensation. ReevaluationContext OSM Java package that provides methods that access the contemporary and historical order perspectives and compares the two. You can use the results of this comparison to determine what compensation strategy is required for a task based on revision order data.

  For example:

  ```
  declare namespace osm = "urn:com:metasolv:oms:xmlapi:1";
  declare namespace context =
  "java:oracle.communications.ordermanagement.compensation.ReevaluationContext";
  declare namespace log =  "java:org.apache.commons.logging.Log";

  declare variable $log external;
  declare variable $context external;
  ```

  For more information about the classes in the OSM packages, install the OSM SDK and extract the OSM Javadocs from the *OSM_home*/**SDK/osm7.***w.x.y.z*-**javadocs.zip** file (where *OSM_home* is the directory in which the OSM software is installed and *w.x.y.z* represents the specific version numbers for OSM). See *OSM Installation Guide* for more information about installing the OSM SDK.
- Body: The body must return a valid compensation option.

For example, the following XQuery expression creates variables for the ReevaluationContext methods. The expression then checks that a specific value exists in the $value variable and that the value in the $significantValue variable both exists and is significant. If the value exists and is significant, then the expression sets the compensation strategy for the task to **Undo then Do** (undoThenDo in the ReevaluationContext Java class). If not, then the expression sets the compensation strategy to **Redo** (redo in the ReevaluationContext Java class).

```
let $inputDoc := self::node()
let $hopDoc := context:getHistoricalOrderDataAsDOM($context)
let $ropDoc := context:getCurrentOrderDataAsDOM($context)
let $diffDoc := context:getDataChangesAsDOM($context)
let $value := $inputDoc/GetOrder.Response/_root/service[name='BB']//orderItemRef/
specificationGroup//specification[value='100']
let $significantValue := $diffDoc/Changes/Add[@significant='true']/
specification[value='100']
let $currentValue := $ropDoc/ GetOrder.Response/_root/service[name='BB']//
orderItemRef/specificationGroup//specification[value='100']

return if (fn:exists($value) and fn:exists($significantValue))
then
    context:undoThenDo($context)
else
    context:redo($context)
```

This section refers to the Design Studio OSM **Automated Task** or **Manual Task** editor, **Compensation** tab **Compensation Expression** XQuery field for when a task is no longer required. The context, prolog, and body are similar to the XQuery expression for the re-evaluation strategy, except that the XQuery expression implements the java:oracle:communications.ordermanagement.compensation.RollbackContext package.

For example:

```
declare namespace osm = "urn:com:metasolv:oms:xmlapi:1";
declare namespace context =
"java:oracle.communications.ordermanagement.compensation.RollbackContext";
declare namespace log =  "java:org.apache.commons.logging.Log";

declare variable $log external;
declare variable $context external;

let $inputDoc := self::node()
let $hopDoc := context:getHistoricalOrderDataAsDOM($context)
let $ropDoc := context:getCurrentOrderDataAsDOM($context)
let $diffDoc := context:getDataChangesAsDOM($context)

let $value := $inputDoc/GetOrder.Response/_root/service[name='BB']//orderItemRef/
specificationGroup//specification[value='100']
return if (fn:exists($value))
then
    context:undo($context)
else
    context:doNothing($context)
```

In Progress Compensation Include XQuery Expressions

You can determine if an in progress task should be compensated by writing an XQuery expression in the Design Studio **Task Editor Compensation** tab.

> **Note:**
>
> If the XQuery expression is invalid OSM logs the error and includes the in progress task in the compensation plan as it defaults the expression to true.

This section refers to the Design Studio OSM **Automated Task** or **Manual Task** editor, **Compensation** tab, **In Progress Compensation Include Expression** XQuery field for dynamically defining when in progress tasks should be included in compensation. This XQuery expression runs when OSM first analyzes the task for compensation:

- Context: The context for this XQuery is the current task order data. You can get the current task order data using the XML API GetOrder.Response function.
- Prolog: You can declare the XML API namespace to use the GetOrder.Response function in the XQuery body to extract the order information.

  For example:

  ```
  declare namespace osm = "urn:com:metasolv:oms:xmlapi:1";
  declare namespace log =  "java:org.apache.commons.logging.Log";

  declare variable $log external;
  declare variable $context external;
  ```

- Body: Based on task context data, the body must return **true** if the in progress task requires compensation or **false** if it does not.

For example:

```
declare namespace osm = "urn:com:metasolv:oms:xmlapi:1";
declare namespace log =  "java:org.apache.commons.logging.Log";
declare variable $log external;

let $inputDoc := self::node()
let $value := $inputDoc/GetOrder.Response/_root/data

return (
    if (fn:contains($value, "includeInCompensation")) then
        fn:true()
    else
        fn:false()
)
```

In Progress Compensation Complete XQuery Expressions

You can determine when the compensation for an in progress task is complete by writing an XQuery expression in the Design Studio **Task Editor Compensation** tab.

> **Note:**
>
> If the XQuery expression is invalid OSM logs the error and includes the in progress task in the compensation plan as it defaults the expression to true.

This section refers to the Design Studio OSM **Automated Task** or **Manual Task** editor, **Compensation** tab, **In Progress Compensation Complete Expression** XQuery field for

dynamically defining when in progress tasks completes compensation activities. This XQuery expression runs whenever data changes on the compensating task:

- Context: The context for this XQuery is the current task order data. You can get the current task order data using the XML API GetOrder.Response function.
- Prolog: You can declare the XML API namespace to use the GetOrder.Response function in the XQuery body to extract the order information.

  For example:

```
declare namespace osm = "urn:com:metasolv:oms:xmlapi:1";
declare namespace log =  "java:org.apache.commons.logging.Log";

declare variable $log external;
declare variable $context external;
```

- Body: Based on task context data, the body must return **true** if the in progress task has completed all compensation activities or **false** if it has not.

For example:

```
declare namespace osm = "urn:com:metasolv:oms:xmlapi:1";
declare namespace log =  "java:org.apache.commons.logging.Log";
declare variable $log external;
let $inputDoc := self::node()
let $value := $inputDoc/GetOrder.Response/_root/data
return (
    if (fn:contains($value, "compensationDone")) then
                fn:true()
    else
                fn:false()
```

In Progress Compensation Grace Period XQuery Expressions

You can determine whether a grace period should be observed before starting compensation for an in progress task by writing an XQuery expression in the Design Studio **Task Editor Compensation** tab.

> **Note:**
>
> If the XQuery expression is invalid OSM logs the error and includes the in progress task in the compensation plan as it defaults the expression to true.

This section refers to the Design Studio OSM **Automated Task** or **Manual Task** editor, **Compensation** tab, **When an amendment occurs if this task is in progress it will:** tab, **Dynamic Expression** XQuery field for dynamically defining the grace period for an in progress task based on task data. This XQuery expression runs after OSM has determined whether the in progress task needs to be compensated:

- Context: The context for this XQuery is the current task order data. You can get the current task order data using the XML API GetOrder.Response function.
- Prolog: You can declare the XML API namespace to use the GetOrder.Response function in the XQuery body to extract the order information. You can also declare the $gracePeriod variable in the XQuery prolog which contains the grace period specified on the order life-cycle policy.

  For example:

```
declare namespace osm = "urn:com:metasolv:oms:xmlapi:1";
declare namespace log =  "java:org.apache.commons.logging.Log";

declare variable $gracePeriod external;
declare variable $log external;
declare variable $context external;
```

- Body: The XQuery body returns a duration value based on the XQuery you enter:

  ```
  PyYmMdDThHmMsS
  ```

  where

  - **P** begins the expression.
  - *y***Y** specifies the year.
  - *m***M** specifies the month.
  - *d***D** specifies the day.
  - **T** separates the parts of the expression indicating the date from the parts of the expression indicating the time.
  - *h***H** specifies the hour.
  - *m***M** specifies the minutes.
  - *s***S** specifies the seconds.

For example, this XQuery uses order data to define the specific grace period duration for the task. The last statement calls the $gracePeriod variable which represents the grace period duration specified on the order life-cycle policy:

```
declare namespace osm = "urn:com:metasolv:oms:xmlapi:1";
declare namespace log =  "java:org.apache.commons.logging.Log";
declare variable $log external;
declare variable $gracePeriod external;

let $inputDoc := self::node()
let $value := $inputDoc/GetOrder.Response/_root/data

return (
    if (fn:contains($value, '-immediate-')) then
        xs:duration('PT0S')
    else if (fn:contains($value, '-override-')) then
        xs:duration('PT20S')
    else if (fn:contains($value, '-negative-')) then
        xs:duration('-PT10S')
    else if (fn:contains($value, '-invalidNumber-')) then
        fn:number(0)
    else if (fn:contains($value, '-invalidString-')) then
        xs:string('UNKNOWN')
    else
        xs:duration(fn:concat('PT', $gracePeriod, 'S'))
```

Order Jeopardy Automation XQuery Plug-ins

This topic provides information about order jeopardy XQuery expressions. These XQuery expressions apply to order jeopardies configured in the Order Jeopardy editor, not order jeopardies configured in the Order editor.

For general OSM XQuery information, see "General XQuery Information".

You can configure automations for order jeopardies in the Order Jeopardy editor, **Automation** tab. If you choose to use an XQuery automation type, create an XQuery file and reference it in the **Script** subtab **Script** field.

- Context: The context for this XQuery is the Order Jeopardy Notification context.
- Prolog: You should declare the XML namespace for the Order Jeopardy Notification context, and if you are using a date (rather than a duration) you can declare a namespace for the date format as well.

  For example:

  ```
  declare namespace context =
  "java:oracle.communications.ordermanagement.orderjeopardy.automation.OrderJeopardyNot
  ificationContext";
  declare namespace dateFormat = "java:java.text.DateFormat";
  ```

  You should then declare the $context variable to contain the actual context:

  ```
  declare variable $context external;
  ```

  Then if you want to use order data in your XQuery, you can get the order data into a variable. For example:

  ```
  let $orderData := fn:root(automator:getOrderAsDOM($automator))/oms:GetOrder.Response
  ```

  You can then access individual data elements on the order. For example:

  ```
  let $date := $orderData/oms:_root/oms:ojPostponeDate/text()
  ```

- Body: There are several calls you can use in the order jeopardy XQuery file in addition to the normal calls available for notification plug-ins. Following are brief descriptions of the available calls:
  - postponeTimerOnExit(interval): If this call receives a numeric parameter, it postpones the due date for the number of milliseconds contained in the parameter.
  - postponeTimerOnExit(dateTime): If this call receives a date/time parameter, it postpones the due date to the indicated date/time.
  - logAndParkNotificationOnExit(logMessage): This call acknowledges the notification with the passed-in message, but does not reset/deactivate the notification. It will still be available in the Order Management web client.
  - ackNotificationOnExit: This call acknowledges and resets/deactivates the notification.
  - getNotificationAckStatus: This call returns true if the notification has been acknowledged, and false if it has not.

  The following example postpones the jeopardy to a specified date:

  ```
  declare namespace context =
  "java:oracle.communications.ordermanagement.orderjeopardy.automation.OrderJeopardyNot
  ificationContext";
  declare namespace dateFormat = "java:java.text.DateFormat";

  declare variable $context external;

  let $dateFormat := dateFormat:getDateTimeInstance(3, 3)
  let $date := dateFormat:parse($dateFormat, "09/30/15 03:30 PM")
  return
      context:postponeTimerOnExit($context, $date)
  ```