Contents

Preface .............................................................................................................................................................. xvii
Audience ................................................................................................................................................................ xvi
Downloading Oracle Communications Documentation .................................................................................. xviii
Document Revision History ........................................................................................................................... xviii
Documentation Accessibility .......................................................................................................................... xviii

1 Introduction

Planning and Designing ........................................................................................................................................ 1-1
Customizing OSM ................................................................................................................................................ 1-1
External Interfaces ................................................................................................................................................ 1-1
OSM Web Services ......................................................................................................................................... 1-1
OSM Automation ............................................................................................................................................. 1-1
OSM Security Callback ............................................................................................................................... 1-2
The OSM XML API ....................................................................................................................................... 1-2
User Interfaces ................................................................................................................................................ 1-2
Behaviors ......................................................................................................................................................... 1-2
Custom Menu Items and Actions ................................................................................................................. 1-2
Localizing OSM ............................................................................................................................................ 1-3
Logging with Apache Log4j ........................................................................................................................ 1-3
Tools for Customizing OSM ........................................................................................................................ 1-3
Design Studio ................................................................................................................................................ 1-3
Apache Ant .................................................................................................................................................... 1-3
The XML Import/Export Application ............................................................................................................ 1-3
About XPath and XQuery .............................................................................................................................. 1-3

2 Planning and Designing the OSM Implementation

Planning the Solution .......................................................................................................................................... 2-1
Integration Design ............................................................................................................................................. 2-2
Configuration Design .................................................................................................................................... 2-2
Cartridge Packaging Design ......................................................................................................................... 2-4
Status Update Design ..................................................................................................................................... 2-4
Status Update Strategies ............................................................................................................................... 2-4
Completing an Automation Task That Handles Concurrent Status Updates ......................................... 2-5
Organizing Design Studio and Naming Conventions .................................................................................. 2-6
### 3 Using Design Studio

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>About Implementing OSM</td>
<td>3-1</td>
</tr>
<tr>
<td>Installing Design Studio</td>
<td>3-2</td>
</tr>
<tr>
<td>Installing the OSM Plug-ins</td>
<td>3-2</td>
</tr>
<tr>
<td>Configuring Design Studio</td>
<td>3-2</td>
</tr>
<tr>
<td>About Design Studio Perspectives and Views</td>
<td>3-3</td>
</tr>
<tr>
<td>Creating and Deploying Custom Files</td>
<td>3-3</td>
</tr>
<tr>
<td>About Cartridges</td>
<td>3-3</td>
</tr>
<tr>
<td>Configuration Guidelines: Steps to Create an Orchestration Cartridge</td>
<td>3-4</td>
</tr>
<tr>
<td>Creating a Cartridge</td>
<td>3-5</td>
</tr>
<tr>
<td>Studio Projects View of a Cartridge</td>
<td>3-6</td>
</tr>
<tr>
<td>Package Explorer View of a Cartridge</td>
<td>3-6</td>
</tr>
<tr>
<td>Adding Content to a Cartridge</td>
<td>3-7</td>
</tr>
<tr>
<td>Handling Multiple Cartridge Versions</td>
<td>3-8</td>
</tr>
<tr>
<td>Packaging and Building a Cartridge</td>
<td>3-9</td>
</tr>
<tr>
<td>Modifying the Build</td>
<td>3-11</td>
</tr>
<tr>
<td>Cleaning and Rebuilding Cartridges Prior to Deployment</td>
<td>3-11</td>
</tr>
<tr>
<td>Optimizing Cartridge Deployment</td>
<td>3-12</td>
</tr>
<tr>
<td>Deploying Multiple Cartridges</td>
<td>3-12</td>
</tr>
<tr>
<td>Deploying Cartridges with Dependencies</td>
<td>3-12</td>
</tr>
<tr>
<td>Building and Deploying Composite Cartridges</td>
<td>3-13</td>
</tr>
<tr>
<td>Setting Cartridge Dependencies</td>
<td>3-13</td>
</tr>
<tr>
<td>Post-Deployment Effect on Numeric Data</td>
<td>3-13</td>
</tr>
<tr>
<td>Post-Deployment Changes to Cartridge</td>
<td>3-13</td>
</tr>
<tr>
<td>Metadata Errors</td>
<td>3-14</td>
</tr>
<tr>
<td>Automating the Cartridge Build and Deployment</td>
<td>3-15</td>
</tr>
<tr>
<td>Maintaining Cartridges Between Releases and Patches</td>
<td>3-15</td>
</tr>
<tr>
<td>Modeling Order Template Structures for Orchestration Plan Generation</td>
<td>3-15</td>
</tr>
<tr>
<td>ControlData/OrderItem</td>
<td>3-16</td>
</tr>
<tr>
<td>ControlData/TransformedOrderItem</td>
<td>3-16</td>
</tr>
<tr>
<td>ControlData/Functions/OrderComponentName</td>
<td>3-16</td>
</tr>
<tr>
<td>Modeling Order Template Structures for Fulfillment States</td>
<td>3-17</td>
</tr>
<tr>
<td>External Fulfillment States</td>
<td>3-17</td>
</tr>
<tr>
<td>Order Fulfillment State</td>
<td>3-17</td>
</tr>
<tr>
<td>Order Item Fulfillment State</td>
<td>3-17</td>
</tr>
<tr>
<td>Point of No Return</td>
<td>3-18</td>
</tr>
<tr>
<td>Fulfillment State and Point of No Return Initial Values</td>
<td>3-18</td>
</tr>
<tr>
<td>Sample XQuery for Changing Default Data Locations</td>
<td>3-18</td>
</tr>
<tr>
<td>Orchestration Order Template Definition</td>
<td>3-20</td>
</tr>
<tr>
<td>Modeling Orders With Data Fields Above 1000 Characters</td>
<td>3-21</td>
</tr>
<tr>
<td>Using XML Types for Data Fields Above 1000 Characters</td>
<td>3-21</td>
</tr>
<tr>
<td>Using Order Remarks for Data Fields Above 1000 Characters</td>
<td>3-22</td>
</tr>
<tr>
<td>Using Attachments for Data Fields Above 1000 Characters</td>
<td>3-22</td>
</tr>
<tr>
<td>Creating and Deploying Custom Files</td>
<td>3-22</td>
</tr>
<tr>
<td>About XML Catalogs</td>
<td>3-23</td>
</tr>
<tr>
<td>Using XML Catalogs in OSM</td>
<td>3-23</td>
</tr>
<tr>
<td>Resource Packaging Considerations for Using XML Catalogs</td>
<td>3-25</td>
</tr>
</tbody>
</table>
4 Using OSM Web Services

About Web Services

About OSM Web Services

Request Validations

Accessing the WSDL Files

Using the SOAP Standard Message Format

Message Header

Message Body

Testing

Order States and Transitions

Web Services Sample

About OSM Web Service Operations

Parameters

Fault Types

Request and Response Examples

Web Service Operations Used for Order Management

CreateOrderBySpecification

CreateOrder

FindOrder

GetOrder

UpdateOrder

SuspendOrder

ResumeOrder

CancelOrder

AbortOrder

FailOrder

ResolveFailure

Web Service Operations Used for Problem Order Diagnosis

GetOrderProcessHistory

GetOrderCompensations

GetCompensationPlan

Navigating WSDL and XSD Files

WSDL File

XSD File

Request and Response Examples

CreateOrderBySpecification Examples

GetOrder Examples

UpdateOrder Examples

SuspendOrder Examples
ResumeOrder Examples.................................................................................................................. 4-48
CancelOrder Examples.................................................................................................................. 4-49
GetOrderProcessHistory Examples............................................................................................... 4-50
GetOrderCompensations Examples............................................................................................... 4-60
GetCompensationPlan Examples................................................................................................. 4-61

5 Using Automation

About Automations and the Automation Framework....................................................................... 5-1
About Internal and External Events that Trigger Automations....................................................... 5-3
About Automations in the Order and Task Contexts....................................................................... 5-4
About Accessing the XML API in Automations.............................................................................. 5-6
About Correlation and Message Property Selectors......................................................................... 5-7
  Correlating Requests from OSM to Responses from External Systems......................................... 5-7
  Intercommunication Between Orders in the Same Domain............................................................. 5-7
About Message Property Selectors.................................................................................................. 5-8
About Automation Plug-in Communication Options........................................................................ 5-8
  No External Communication: Data Processing Only.................................................................... 5-9
  Fire-and-Forget Communication: Message Sent to External Systems........................................... 5-9
  Synchronous Communication: Single Request and Response......................................................... 5-9
  Synchronous Communication: Multiple Requests and Responses.............................................. 5-10
  Asynchronous Communication: Single or Multiple Requests and Responses.............................. 5-12
Storing Response Message as XML Type Parameters....................................................................... 5-14
About Custom Automation Plug-ins............................................................................................... 5-14
  Defining the Custom Automation Plug-in...................................................................................... 5-15
  About the XML Template................................................................................................................ 5-15
About Creating Custom Automation Plug-ins.................................................................................. 5-16
  inputXML Argument........................................................................................................................ 5-16
  AutomationContext Argument and Casting the Context Argument.............................................. 5-17
  outboundMessage Argument........................................................................................................... 5-17
Accessing JDBC from Within an Automation Plug-in ...................................................................... 5-17
Compiling the Custom Automation Plug-in....................................................................................... 5-18
About Predefined Automation Plug-ins............................................................................................ 5-19
XSLT Sender.................................................................................................................................... 5-19
  Defining the Automation.................................................................................................................. 5-19
  Writing the XSLT............................................................................................................................... 5-21
  Steps to Follow When Using XSLT Sender...................................................................................... 5-22
XSLT Automator............................................................................................................................... 5-22
  Defining the Automation.................................................................................................................. 5-22
  Writing the XSLT............................................................................................................................... 5-23
  Steps to Follow When Using XSLT Automator.............................................................................. 5-24
XQuery Sender................................................................................................................................ 5-24
  Defining the Automation.................................................................................................................. 5-24
  Writing the XQuery............................................................................................................................ 5-25
  Steps to Follow When Using XQuery Sender.................................................................................. 5-26
XQuery Automator............................................................................................................................. 5-26
  Defining the Automation.................................................................................................................. 5-26
  Writing the XQuery............................................................................................................................ 5-27
9 Using Data Providers to Retrieve Data

About Data Providers and Adapters................................................................. 9-1
Data Provider Interface Tab ........................................................................ 9-2
Accessing Data through Data Providers .................................................... 9-3
Augmenting or Overriding Data ................................................................. 9-3
Order ............................................................................................................. 9-4
  Parameters................................................................................................. 9-4
Property File ............................................................................................... 9-6
  Parameters................................................................................................. 9-6
SOAP ............................................................................................................. 9-6
  Parameters................................................................................................. 9-6
  Example of soap.body Parameter .......................................................... 9-7
  Example of soap.envelope Parameter ..................................................... 9-8
XML Attachment ....................................................................................... 9-9
  Parameters................................................................................................. 9-10
XML File .................................................................................................... 9-10
  Parameters................................................................................................. 9-10
XML Validation ......................................................................................... 9-10
  Parameters................................................................................................. 9-10
JDBC ............................................................................................................ 9-10
  Parameters................................................................................................. 9-10
Web Service ............................................................................................... 9-11
  Parameters................................................................................................. 9-11
  Adding a New Web Service Data Provider ............................................ 9-12
  Sample soap.request XQuery ................................................................. 9-12
  Accessing Data......................................................................................... 9-12
Custom Data Providers ............................................................................ 9-13
  Handling Parameters................................................................................ 9-14

10 Localizing OSM

About Localization.................................................................................... 10-1
Localizing OSM ....................................................................................... 10-1
  Localizing the XML Import/Export Application .................................... 10-5
Additional Considerations for Localizing OSM ...................................... 10-6
  Support for Different Locales ................................................................. 10-6
  Character Set Encoding .......................................................................... 10-6
  Localization of Settings ......................................................................... 10-6
About NLS Database Configuration ....................................................... 10-6
Oracle Database Character Set .............................................................. 10-6
NLS Environment .................................................................................. 10-7
NLS_LANG Parameter ............................................................................ 10-7
ORA_NLS33 Environment Variable ....................................................... 10-8
About OSM Database Error Messages ................................................. 10-8
About Application Server Strings ....................................................... 10-11
About OSM Process Definition Data .................................................. 10-12
  om_application_function ............................................................... 10-13
  om_attribute_type ........................................................................... 10-13
  om_order_data_dictionary ............................................................ 10-14
  om_order_remarks_type ............................................................... 10-14
  om_order_type_category ............................................................... 10-14
  om_process ..................................................................................... 10-14
  om_process_status ......................................................................... 10-14
  om_responsibility ......................................................................... 10-15
  om_rule_def .................................................................................... 10-15
  om_state ........................................................................................ 10-15
  om_state_category ......................................................................... 10-16
  om_status_category ....................................................................... 10-16
  om_task .......................................................................................... 10-16
  om_view_order_node_label ............................................................ 10-17
About Generic Preferences ..................................................................... 10-17
  om_generic_mnemonic .................................................................. 10-17
Localizing the Task Web Client .......................................................... 10-19
  Task Web Client Localization Resource Bundles .......................... 10-20
  Localizing Date, Time and Currency Formats .............................. 10-20
  Localizing Text and Error Messages .......................................... 10-22
  Localizing Page Titles .................................................................. 10-22
  Localizing Image References ....................................................... 10-22
  Inserting New Images .................................................................. 10-22
  Editing the First Day of the Week ................................................. 10-22
  Editing the Task Web Client Gantt Chart ..................................... 10-22
  Editing the Boolean Data Element Values .................................. 10-22
  Editing the Number of Records Displayed in the Worklist .......... 10-22
  Editing and Replacing Task Web Client Icons ............................ 10-23
Localizing the Order Management Web Client ................................ 10-23
  Changing the Order Management Web Client Logo Image and Text 10-25

11 Using XPath Functions

About XPath Functions ......................................................................... 11-1
Node Set Functions ............................................................................... 11-1
  number last() ................................................................................ 11-2
  number position() ......................................................................... 11-2
  number count(node-set) ............................................................... 11-2
  node-set id(object) ....................................................................... 11-2
  string local-name(node-set?) ...................................................... 11-2
  string namespace-uri(node-set?) ................................................ 11-2
string name(node-set?) ................................................................. 11-2
node-set evaluate(string) .......................................................... 11-2
node-set match(node-set, string) ............................................. 11-3
node-set instance(string) .......................................................... 11-3

String Functions ................................................................. 11-3
string string(object?) .............................................................. 11-3
string concat(string, string, string*) .................................... 11-3
string starts-with(string, string) .......................................... 11-3
string contains(string, string) ................................................ 11-4
string substring-before(string, string) ............................... 11-4
string substring-after(string, string) ................................. 11-4
string substring(string, number, number?) ........................ 11-4
number string-length(string?) ................................................ 11-4
string normalize-space(string?) ............................................. 11-4
string translate(string, string, string) ............................... 11-4
string lower-case(string?) ..................................................... 11-4
string upper-case(string?) ..................................................... 11-5
string ends-with(string, string) ............................................. 11-5

Boolean Functions .......................................................... 11-5
Boolean boolean(object) ....................................................... 11-5
Boolean not(boolean) ............................................................. 11-5
Boolean true() ................................................................. 11-5
Boolean false() ................................................................. 11-5
Boolean boolean-from-string(string) ................................. 11-5
object if(boolean, object, object) ............................... 11-5

Number Functions .......................................................... 11-5
number number(object?) ...................................................... 11-6
number sum(node-set) .......................................................... 11-6
number floor(number) .......................................................... 11-6
number ceiling(number) ........................................................ 11-6
number round(number) ........................................................ 11-6
number avg(node-set) ........................................................... 11-6
number min(node-set) .......................................................... 11-6
number max(node-set) .......................................................... 11-6
number count-not-empty(node-set) ................................. 11-7

XPath 1.0 Reference .......................................................... 11-7
Location Paths [XPath §2] .................................................... 11-7
Location Paths [XPath §2.1] .................................................... 11-7
Axis Specifiers [XPath §2.2] ................................................... 11-7
Node Tests [XPath §2] ............................................................ 11-7
Abbreviated Syntax for Location Paths ............................ 11-7
Predicate [XPath §2.4] .......................................................... 11-7
Variable Reference [XPath §3.7] ......................................... 11-8
XPath ................................................................................. 11-8
XPath Operators .............................................................. 11-8
Node-sets [XPath §3.3] ....................................................... 11-8
Booleans [XPath §3.4] ......................................................... 11-8
12 Process Example

About the Process Example ........................................................................................................ 12-1
Process Flow Diagram ............................................................................................................... 12-2
About the Tasks Defined in the Process Example .................................................................. 12-4

Initiate Order .......................................................................................................................... 12-4
Activities ................................................................................................................................. 12-4
Input Data .............................................................................................................................. 12-4
Output Data ............................................................................................................................ 12-4
Completion .............................................................................................................................. 12-7

ILEC CO CLLI Identified? (Rule Task) .................................................................................. 12-7
Task Activities ....................................................................................................................... 12-7
Input Data .............................................................................................................................. 12-7
Output Data ............................................................................................................................ 12-7
Completion .............................................................................................................................. 12-7

Identify ILEC CO ................................................................................................................... 12-7
Task Activities ....................................................................................................................... 12-8
Input Data .............................................................................................................................. 12-8
Output Data ............................................................................................................................ 12-8
Completion .............................................................................................................................. 12-8

Assign Port .............................................................................................................................. 12-8
Task Activities ....................................................................................................................... 12-8
Input Data .............................................................................................................................. 12-9
Output Data ............................................................................................................................ 12-9
Completion .............................................................................................................................. 12-9

Notify Customer of Delay ..................................................................................................... 12-9
Task Activities ....................................................................................................................... 12-9
Input Data .............................................................................................................................. 12-10
Output Data ........................................................................................................................... 12-10
Completion ............................................................................................................................. 12-10

Cancel Order .......................................................................................................................... 12-10
Task Activities ....................................................................................................................... 12-10
Input Data .............................................................................................................................. 12-10
Output Data ............................................................................................................................ 12-10
Completion.................................................................................................................. 12-11
Wait for Engineering ........................................................................................................... 12-11
  Task Activities .............................................................................................................. 12-11
  Input Data ..................................................................................................................... 12-11
  Output Data .................................................................................................................. 12-11
  Completion ................................................................................................................... 12-11
Send LSR to ILEC ............................................................................................................. 12-11
  Task Activities .............................................................................................................. 12-11
  Input Data ..................................................................................................................... 12-12
  Output Data .................................................................................................................. 12-12
  Completion ................................................................................................................... 12-12
Receive LSR Confirmation ................................................................................................. 12-12
  Task Activities .............................................................................................................. 12-12
  Input Data ..................................................................................................................... 12-12
  Output Data .................................................................................................................. 12-13
  Completion ................................................................................................................... 12-13
Handle LSR Problem .......................................................................................................... 12-13
  Task Activities .............................................................................................................. 12-13
  Input Data ..................................................................................................................... 12-13
  Output Data .................................................................................................................. 12-14
  Completion ................................................................................................................... 12-14
LNP Required? (Rule Task) .............................................................................................. 12-14
  Task Activities .............................................................................................................. 12-14
  Input Data ..................................................................................................................... 12-14
  Output Data .................................................................................................................. 12-14
  Completion ................................................................................................................... 12-14
Send LNP Porting Request ................................................................................................. 12-14
  Task Activities .............................................................................................................. 12-14
  Input Data ..................................................................................................................... 12-14
  Output Data .................................................................................................................. 12-14
  Completion ................................................................................................................... 12-15
Receive LNP Confirmation ................................................................................................. 12-15
  Task Activities .............................................................................................................. 12-15
  Input Data ..................................................................................................................... 12-15
  Output Data .................................................................................................................. 12-15
  Completion ................................................................................................................... 12-15
Handle LNP Problem .......................................................................................................... 12-15
  Task Activities .............................................................................................................. 12-15
  Input Data ..................................................................................................................... 12-16
  Output Data .................................................................................................................. 12-16
  Completion ................................................................................................................... 12-16
Activate Port and Services on CLEC Switch ...................................................................... 12-16
  Task Activities .............................................................................................................. 12-16
  Input Data ..................................................................................................................... 12-16
  Output Data .................................................................................................................. 12-16
  Completion ................................................................................................................... 12-17
Handle Activation Problem ............................................................................................... 12-17
A AutomationMap.xml File

AutomationMap.xml Examples for Automated Tasks ............................................. A-1
  XSLTsender Internal Event Receiver ................................................................. A-1
    Notes Common to All Examples ................................................................. A-2
    Notes on Example ..................................................................................... A-2
  XSLTsender External Event Receiver ............................................................ A-2
    Notes on Example ..................................................................................... A-3
  XSLTAutomator Internal Event Receiver ......................................................... A-3
    Notes on Example ..................................................................................... A-4
  XSLTAutomator External Event Receiver ....................................................... A-4
    Notes on Example ..................................................................................... A-5
  Custom Automation Internal Event Receiver ................................................ A-6

A-1

A-2

A-3

A-4

A-5

A-6

A-7
B Automation: Start to Finish

Assumptions ........................................................................................................... B-1
Getting Started ....................................................................................................... B-1
Defining an Automated Task ..................................................................................... B-3
Writing the Custom Automation Plug-in ................................................................. B-3
Defining the Custom Automation Plug-in ................................................................. B-3
Defining the Automation .......................................................................................... B-4
Defining the Process ................................................................................................ B-4
Building the Cartridge .............................................................................................. B-4
Packaging and Deploying the Cartridge ................................................................. B-4
Triggering the Automation in OSM ....................................................................... B-5

C Behaviors Quick Reference

OSM Behavior Type Overview ............................................................................... C-1
Common Behavior Elements ................................................................................... C-3
  Annotation Element .............................................................................................. C-3
  Description Element ............................................................................................ C-3
  Instance Element ................................................................................................. C-3
  Adapter Element [externalInstanceType] ........................................................ C-3
  Parameter Element [externalInstanceType] ....................................................... C-3
  Cache Element ..................................................................................................... C-3
  Expression Element ............................................................................................ C-3
Declaring Behaviors in OSM XML Model ................................................................. C-3
  Data Dictionary Level .......................................................................................... C-3
  Master Order Template Level ............................................................................. C-4
  View Level ......................................................................................................... C-4
Data Provider Overview ......................................................................................... C-4
Programmatic Behavior Implementation Overview ............................................... C-5

D Automation and Compensation Examples

Predefined Automation Plug-ins ........................................................................... D-1
  Message Example ............................................................................................... D-1
  Automation Plug-in XQuery Examples ............................................................. D-4
Internal XQuery Sender ......................................................................................... D-4
Preface

This document provides information about the following customizable areas of Oracle Communications Order and Service Management (OSM):

- Web Services
- Automation
- Security Callback
- Behaviors
- Custom menu items and action items
- Localization of OSM

This document also provides a process example of a CLEC provisioning a POTS customer using unbundled local loop to illustrate various customizations.

Audience

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

- System interfaces
- Extensible Markup Language (XML)
- Java development
- Java Messaging Service (JMS)
- Web services

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

- Oracle Communications Design Studio configuration
- Orders
- Order states
- Tasks
- Task states
- Notifications
- Behaviors
- Web services
Downloading Oracle Communications Documentation

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

http://docs.oracle.com

Document Revision History

The following table lists the revision history for this guide.

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E41523-04</td>
<td>December 2015</td>
<td>This version includes the following new section: &quot;Storing Response Message as XML Type Parameters.&quot;</td>
</tr>
</tbody>
</table>

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.
This chapter provides an introduction to customizing Oracle Communications Order and Service Management (OSM) interfaces.

Planning and Designing
Before customizing OSM, it is important to understand what needs to be done and to design the solution properly.

This topic is further explored in "Planning and Designing the OSM Implementation".

Customizing OSM
There are two areas of OSM that you can customize:

- External interfaces, which interact with other systems and which you customize to meet specific business requirements. This includes OSM Web Services, OSM automation and OSM Security callback.
- User interfaces, which you customized per installation or per individual user. This includes using behaviors to manipulate data, adding custom menu actions of the Task Web client, and localizing user interfaces.

External Interfaces
The two primary external interfaces for performing automated fulfillment are OSM Web Services and OSM automation. Additional external interfaces include OSM Security Callback and the OSM XML API.

OSM Web Services
OSM Web Services provide the primary interface for \textit{in-bound} order operations such as creating or canceling an order. Web Services are typically initiated from customer relationship management (CRM) systems and other order sources that need to create and manage orders in OSM.

This topic is further explored in Chapter 4, "Using OSM Web Services."

OSM Automation
OSM automation provides the primary interface for \textit{outbound} operations to interact with external systems to achieve automated order fulfillment. Outbound operations are initiated by OSM through automated tasks and automated notifications.
Automated tasks and automated notifications are not limited to outbound operations: Automated tasks can send outbound messages to external systems and also receive in-bound messages back from the external systems. (Automated notifications only send outbound messages to external systems; they cannot receive in-bound messages.) Additionally, automated tasks and automated notifications can perform internal business logic or update the OSM database.

This topic is further explored in Chapter 5, "Using Automation." Additional information on automation is presented in Appendix A, "AutomationMap.xml File" and Appendix B, "Automation: Start to Finish."

**OSM Security Callback**

OSM Security Callback allows you to generate an audit trail log of users before they gain access to order data that is considered sensitive. OSM provides a callback interface that is designed to intercept order access from defined functions.

This topic is further explored in Chapter 7, "Using OSM Security Callback."

**The OSM XML API**

The OSM XML API provides functions that perform tasks such as creating and updating orders, transitioning tasks from state to state or from task to task, adding attachments and remarks, retrieving task worklists, and so on. These functions are now primarily deprecated in favor of the OSM Task Web Client and the OSM Web Services.

You can, however, still use some OSM XML API functions from automations plug-ins in various scenarios. See OSM XML API Developer’s Guide for more information about when to use the XML API in automations. See Appendix D, "Automation and Compensation Examples" for examples that illustrate various scenarios where the XML API is used in automations.

**User Interfaces**

The following sections briefly describe the ways you can customize the OSM user interfaces (UIs).

**Behaviors**

Behaviors provide the ability to customize data validation and data presentation in both the Task Web client and the Order Management Web client. OSM defines several behavior types, and you can define instances of behavior types on data elements defined in the data dictionary, for an order, or for a task.

For information about behaviors, see OSM Concepts.

**Custom Menu Items and Actions**

The custom menu actions and items feature provides the ability to configure custom menu items and actions that are called from the Context menu of the Task Web client Worklist and Query Result pages.

This topic is further explored in Chapter 8, "Using Custom Menu Items and Actions."
Localizing OSM

Localizing OSM is the process of changing the user interfaces from the original language in which it was written to another language. You can localize the Order Management Web UI and the Task Web UI. This process involves modifying OSM XML files.

This topic is further explored in Chapter 10, "Localizing OSM."

Logging with Apache Log4J

Oracle recommends that you use Log4j to generate and manage the system log messages. See OSM System Administrator’s Guide for more information.

Tools for Customizing OSM

Several tools are available to you when customizing OSM, as described in the following sections.

Design Studio

Oracle Communications Design Studio is an Eclipse-based integrated development environment (IDE). Design Studio is a separate software that comes with your OSM installation, along with Design Studio plug-ins specific to OSM that enable you to configure and customize OSM. Detailed information on using Design Studio to customize OSM is presented in Chapter 3, "Using Design Studio."

Apache Ant

Apache Ant is an open source software application often used for automating application build processes. See OSM Installation Guide for the required version of Ant.

Ant uses XML to define targets which are executable commands that perform a specific task. By default, the XML file is named build.xml.

Installing Design Studio OSM-specific plug-ins provide the build.xml and the automationBuild.xml files, which can be used to automate building automation plug-ins. Ant is also used by the XML Import/Export application, as described in the following section.

See Chapter 3, "Using Design Studio" for information on installing Ant.

The XML Import/Export Application

OSM includes the option to install the XML Import/Export application, a set of customizable Ant commands that help you manage data when dealing with multiple OSM development and test environments.

You can also use the XML Import/Export application to manage data when dealing with multiple OSM production environments. This topic is further explored in OSM System Administrator’s Guide.

About XPath and XQuery

To model OSM orders, you must have a working knowledge of the XPath and XQuery languages.
You typically use XPath statements to specify the location of data in OSM entities. You use XQuery statements to find and filter data needed for OSM functionality. You can use XQuery in situations where a more expressive language or transformation abilities are needed.

An XPath tutorial is available at the W3Schools web site:

http://www.w3schools.com/Xpath/default.asp

An XQuery tutorial is also available at the W3Schools web site:

http://www.w3schools.com/xquery/default.asp

---

**Note:** In OSM, XQuery statements are limited to a maximum of 4000 characters.
This chapter provides an overview of planning and design considerations for your Oracle Communications Order and Service Management (OSM) implementation.

Planning the Solution

Before you model your order fulfillment process, you must complete a planning phase, in which you analyze your business and system requirements. These requirements may include systems outside OSM, in order to ensure that the solution works as a complete system. The results of this analysis help you determine how to model your order processing.

This phase can include the following:

- **Sales catalog analysis.** The objectives of this analysis are:
  - To understand and categorize the types of products in your catalog; for example, service-related products, billing-only products, installation products, and physical goods
  - To understand how products can be grouped based on similar characteristics into product specifications. For example, you might group broadband access products into a product specification
  - For each product specification, to understand the order items required, and the actions required to fulfill the order items

- **Business process analysis.** The objective of this analysis is to describe the business processes that will be supported by OSM. This would include, for example, how products are added to your catalog, how order fallout is managed, and how you communicate with other companies (such as shipping companies) involved in the order process.

  The analysis focuses on:
  - The organizations involved in the business processes
  - The activities that need to be performed by each organization

- **IT application analysis.** The objectives of this analysis are:
  - To identify and understand the capabilities and constraints of the applications that are part of the order processing solution; for example, activation applications, inventory applications, and billing applications.
Integration Design

To understand the fulfillment functions the applications support, and how those functions are implemented.

To understand how those applications communicate with OSM to exchange information and carry out fulfillment actions.

**Network services analysis.** The objectives of this analysis are:

- To identify the services supported by your network.
- To map those services to the sales catalog. This identifies the link between the front-office sales catalog and the back-end network.

This analysis helps translate the commercial products into the services that need to be provisioned on the network.

Integration Design

Every order management implementation involves integration between multiple systems as a part of the end-to-end fulfillment lifecycle. To plan how to integrate OSM with external systems, do the following:

- Make a list of the systems involved in the process and technologies they support for integration.
- Make a list of actions that can be automated.
- Make a list of integration points between OSM and external systems. For each integration point:
  - Understand the details about the data being passed from one application to another.
  - Identify and understand the function calls and APIs being invoked each way for inter-application communication.
  - Understand the error handling mechanism.
  - Understand any special performance or failover requirements, for example, number of retries for failed operations.

Configuration Design

Table 2–1 lists the considerations you must take into account, and the entities and configurations you use to implement them in Oracle Communications Design Studio.

**Table 2–1 Order Processing Considerations and Related Metadata Elements in OSM**

<table>
<thead>
<tr>
<th>Question</th>
<th>Metadata Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>What are the data elements related to the incoming order?</td>
<td>Data dictionary</td>
</tr>
<tr>
<td>What additional data is generated by the applications that are a part of the fulfillment process; for example, billing data?</td>
<td></td>
</tr>
<tr>
<td>What are the data elements needed by individual tasks?</td>
<td></td>
</tr>
<tr>
<td>How is the order data organized?</td>
<td>Order template and Behaviors</td>
</tr>
<tr>
<td>How will the order be displayed to your order management personnel?</td>
<td></td>
</tr>
<tr>
<td>How will the incoming order format be recognized?</td>
<td>Order recognition rules</td>
</tr>
</tbody>
</table>
Table 2–1  (Cont.) Order Processing Considerations and Related Metadata Elements in

<table>
<thead>
<tr>
<th>Question</th>
<th>Metadata Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>How can I validate specific content on the incoming order data?</td>
<td>Order recognition rule and Validation X-Query</td>
</tr>
<tr>
<td>How will OSM identify and understand the structure of the individual line items on the order?</td>
<td>Order item specifications</td>
</tr>
<tr>
<td>How will OSM determine the fulfillment actions that need to be performed for line items on the order?</td>
<td>Fulfillment patterns</td>
</tr>
<tr>
<td>How will OSM recognize which set of fulfillment actions need to be performed for an order?</td>
<td>Fulfillment modes</td>
</tr>
<tr>
<td>What are the fulfillment actions involved in the processing? Fulfillment actions include actions such as delivering a service, canceling a service, and qualifying a service.</td>
<td>Order component specifications - Fulfillment actions</td>
</tr>
<tr>
<td>Which systems are involved in the fulfillment lifecycle?</td>
<td>Order component specifications - Fulfillment target systems</td>
</tr>
<tr>
<td>What is the processing granularity for each target fulfillment system? For example, is it:</td>
<td>Order component specifications - processing granularity</td>
</tr>
<tr>
<td>• One line item per request.</td>
<td></td>
</tr>
<tr>
<td>• Bundle of line items per request.</td>
<td></td>
</tr>
<tr>
<td>How will the order fulfillment actions be realized? Does the action need to be manual or can it be automated?</td>
<td>Processes and tasks</td>
</tr>
<tr>
<td>What are the steps in decomposing the sales order into a set of fulfillment requests to be executed by the different fulfillment systems?</td>
<td>Orchestration stages</td>
</tr>
<tr>
<td>What is the sequence in which the decomposition should happen?</td>
<td>Orchestration sequences</td>
</tr>
<tr>
<td>What are the teams and groups of people involved in executing the fulfillment actions?</td>
<td>Roles (called Workgroups in the Administrator application)</td>
</tr>
<tr>
<td>What are the policies that control the execution of the order, concerning which items can be modified, and when, and who is authorized to make modifications.</td>
<td>Order lifecycle policies</td>
</tr>
<tr>
<td>Is there a specific order in which the fulfillment actions must be performed?</td>
<td>Dependencies</td>
</tr>
<tr>
<td>Is there a stage in processing beyond which no revision to an in-flight order can be allowed?</td>
<td>Point of no return</td>
</tr>
<tr>
<td>What actions need to be performed when an in-flight order is revised?</td>
<td>Compensation configuration</td>
</tr>
<tr>
<td>How will external systems be notified of status changes or order data changes in OSM?</td>
<td>Notifications</td>
</tr>
<tr>
<td>How will OSM order processing personnel be notified if an order is in jeopardy.</td>
<td>Order Data Changed Notifications. Line item data fields returned from fulfillment, status and milestones on the order data</td>
</tr>
<tr>
<td>What are the key data elements of the order that need to be tracked? Changes to these data elements might need to be communicated to external systems.</td>
<td></td>
</tr>
<tr>
<td>How can I customize the appearance and functionality of the Task Web client?</td>
<td>Behaviors (called View Rules in previous releases)</td>
</tr>
<tr>
<td>How will data be validated?</td>
<td></td>
</tr>
</tbody>
</table>
Cartridge Packaging Design

When you model OSM entities, you can define separate cartridges and combine them in a single solution. This allows you to create individual cartridges for specific purposes, and to create a library of cartridges which can be shared across multiple solutions. This approach can result in lower maintenance, better performance, and easier collaboration within the implementation team.

While each deployment has its own specific considerations, Oracle recommends that you consider the following guidelines:

- Build separate cartridges based on function. For example, build separate cartridges for central order management operations and service order management operations.
- Put configuration elements that are more commonly changed into a separate cartridge for ease of maintenance.
- If a component has distinct notification and status management requirements, use a separate cartridge and pass the data to this "sub-order" from the main order. For example, if the process of shipping a piece of equipment involves interactions with three to four systems and multiple notifications to other systems, consider creating a “Shipping order management cartridge” to handle this requirement.
- For a given cartridge, limit the total number of sub-processes and the number of tasks per sub-process for ease of maintenance. Consider limiting both to ten or fewer, although in some situations more might be required.
- Consider defining a cartridge which contains only a data dictionary with data nodes and structures that are specific to a technology or service or space. Other cartridges can then reference this data.
- Model a separate cartridge for the product specification definitions since they are modified frequently.

Status Update Design

Status values for an order item and for the whole order often need to be sent to the upstream system that submitted the original request. There are a number of ways to achieve this.

Status Update Strategies

- Use an event notification triggered by a change to order data, or when an order reaches an order milestone; for example, completed. The notification runs an automation plug-in that sends a status message to the upstream system. The automation plug-in should have all of the values for status data defined in its view, in order to calculate an aggregated status value.

Be aware that there can be race conditions if multiple status updates are executed in parallel. Since each update is taking a snapshot at a particular moment, it is possible that none of the status updates will have a snapshot that includes all of
the final values. This strategy is better used when there are no multiple concurrent status updates.

- Configure an automation to generate a status message whenever the order changes state. Because order state changes are generally less frequent than data changes, this may be a higher performance option.

- It is possible to configure status update functions as order components and make them first-class members of the orchestration plan. However, it is not desirable to do this in most circumstances, because this can quickly lead to a large increase in the number of tasks in the cartridge. If used, this option will only work if status updates are only sent at a specific point in the orchestration plan, for example as the last function after provisioning and billing.

Completing an Automation Task That Handles Concurrent Status Updates

An automated task can process multiple responses from external systems. For example, an activation task might receive the status for each service on the activation request. The activation task needs this information to determine when the activation has been completed by the external system, at which point the task can transition to the Completed state.

- The external system can include data that indicates that all of the requests have been completed. Typically, this is a message indicating that the response is the last response, and there will be no further messages.

- If the external system cannot report that the last request has been processed, the automation task must ensure that a response has been received for each request sent to the external system.

When OSM must determine the last response, there are special considerations for concurrent status updates. If the automated task needs to track the status of all responses, and multiple responses are processed concurrently, the automation receiver instances executing concurrently do not have visibility to status updates from the other receivers. The receiver may never execute with the task data that contains all status updates and so never encounters a condition where it can complete the task. This situation can be handled by configuring an automated notification plug-in that monitors the status fields and creates a notification whenever the data changes.

*Figure 2–1* shows a sequence diagram for the concurrent status update notification process.
Organizing Design Studio and Naming Conventions

Oracle recommends that you determine a set of naming conventions for the Design Studio entities being created and a directory structure to contain those elements that is appropriate to your implementation. Following is an example set of naming conventions for selected configuration elements within Design Studio. However, each project team should determine what conventions are suitable for a particular project.

**Table 2–2  Suggested Design Studio Naming Conventions**

<table>
<thead>
<tr>
<th>Metadata Element</th>
<th>Naming Convention</th>
<th>Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order recognition rules</td>
<td>Use the convention of OrderTypeORR</td>
<td>SalesOrderORR ProvisioningOrderORR</td>
</tr>
<tr>
<td>Order item specifications</td>
<td>Use the convention of OrderItemTypeItemSpec</td>
<td>CentralOrderItemSpec</td>
</tr>
<tr>
<td>Fulfillment patterns</td>
<td>Use a name that indicates the fulfillment flow being supported. For example: FP_FulfillmentType_SpecType</td>
<td>FP_Service_Broadband</td>
</tr>
</tbody>
</table>

The notification plug-in is triggered every time the status field is updated by the automation receiver. The notification plug-in executes in a separate transaction after each receiver update, and can check the status responses to determine if all responses have been received for each action request. When all responses are received, the notification plug-in can generate a message to trigger an automation receiver. This receiver is correlated to the original sender by means of an ID set by the sender specifically for tracking the status updates. The receiver is then run with the task data that contains all of the status responses and it can complete the task.
<table>
<thead>
<tr>
<th>Metadata Element</th>
<th>Naming Convention</th>
<th>Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fulfillment modes</td>
<td>Use names that clearly identify the type of action to be taken</td>
<td>DELIVER QUALIFY QUOTE</td>
</tr>
<tr>
<td>Order component specifications: Fulfillment actions</td>
<td>Use names that indicate the function of the fulfillment action; for example, Billing, or Shipping.</td>
<td>ResidentialBillingFunction EnterpriseBillingFunction ProvisioningFunction</td>
</tr>
<tr>
<td>Order component specifications: Fulfillment target systems</td>
<td>Use names that indicate the function of the target system; for example, Billing, or Shipping.</td>
<td>BillingSystem CRMSystem ServiceManagementSystem</td>
</tr>
<tr>
<td>Order component specifications: Processing Granularity</td>
<td>Use names that correspond to the order structure defined in the product catalog; for example, Item, Bundle, and Order.</td>
<td>ItemBased BundleBased OrderBased</td>
</tr>
<tr>
<td>Orchestration stages</td>
<td>Use names that describe the stages.</td>
<td>DetermineFulfillmentFunction Stage</td>
</tr>
<tr>
<td>Orchestration sequences</td>
<td>Use the convention CartridgeNameSequence</td>
<td>SalesOrderFulfillmentSequence</td>
</tr>
<tr>
<td>Decomposition rules</td>
<td>Use the naming convention DR_FunctionName_To_SystemNameName for system decomposition rules Use the naming convention DR_DetermineGranularity_Name For_NameName for granularity decomposition rules</td>
<td>DR_BillingFunction_To_ResBRM DR_DetermineGranularity_Name_For_BillingFunction</td>
</tr>
</tbody>
</table>
This chapter provides information on Oracle Communications Design Studio and the plug-ins specific to Oracle Communications Order and Service Management (OSM) that enable you to configure and customize OSM.

About Implementing OSM

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

You can approach the creation of process-based orders by modeling the process flow first or by defining data first and analyzing the process flow needed to support the data model. You can also use a hybrid approach, defining data as needed while modeling your process flow.

When you use Design Studio, you create order specifications for the types of orders your business uses. You also create specifications for each of the elements in an order model; for example, order item specifications, and order component specifications.

You also use Design Studio to define how OSM fulfills orders. For example, you design the way orders are recognized, how they are decomposed, the processes and data required to complete them, and how to manage fallout.

You create projects to contain the order specifications that you develop. When a project is complete, you use Design Studio to build a cartridge that is deployed into OSM.

You also define the following entities in Design Studio to further specify particular aspects of order management.

- **Roles.** You define roles to control the tasks that OSM users can perform, such as running reports or managing fallout. See the discussion of roles in OSM System Administrator’s Guide.

- **Behaviors.** Behaviors provide a way to exercise greater control over validation and presentation of order data to Order Management Web client and Task Web client users. For example, you can configure how to validate that the correct number of digits is entered in a postal code. See OSM System Administrator’s Guide for more information.

- **Order life cycles.** An order life-cycle policy defines which states and transactions an order can experience while it is being fulfilled. The order life-cycle policy also determines which roles can perform which transactions. For example, while an order is in the In Progress state, you might want your Customer Service role to be
able to perform the Update Order, Cancel Order, and Suspend Order transactions, while your Fallout role is able to perform the Raise Exceptions transaction.

- **Rules.** Rules are used in many OSM activities to evaluate conditions and determine next process steps.

- **Notifications.** You use notifications to inform order processing personnel about the status of the order, manual tasks that need to be performed, and problems with order fulfillment.

Design Studio is based on Eclipse, an industry-standard, open-source development environment.

See the Design Studio Help for more information.

Installing Design Studio

Design Studio is the primary application for modeling orders, orchestration details, processes, and creating automated tasks. For directions on how to install Design Studio, which includes the installation of Eclipse, see *Design Studio Installation Guide*. For version compatibility information for Design Studio and plug-ins, see *OSM Installation Guide*.

Installing the OSM Plug-ins

For directions on how to install the OSM plug-ins, see *Design Studio Installation Guide*. The section on installing Design Studio features into Eclipse describes how to install all available Oracle Communications plug-ins.

For OSM, you need to install the following plug-ins:

- **Design Studio Platform:** This is the core plug-in required for working with all the Oracle Communications applications that use Design Studio.

- **Design Studio for Order and Service Management:** This Design Studio feature enables you to configure entities used by OSM.

If you plan to integrate OSM with Oracle Communications ASAP or with Oracle Communications IP Service Activator, or if you plan to use the orchestration capabilities of OSM, you need to install the following plug-ins:

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

- **Design Studio for Order and Service Management Orchestration:** This Design Studio feature enables you to configure orchestration-related entities for OSM.

Configuring Design Studio

Before developing with Design Studio, configure your Oracle WebLogic Server, Java, and OSM SDK home directories:

1. From the *Window* menu, select *Preferences*.
2. In the left pane tree view, expand *Oracle Design Studio* and select *Order and Service Management Preferences*.
3. In the *Deploy Properties* section:
a. Set the WebLogic Home field: Click Browse and navigate to your WebLogic Server installation directory; for example, C:\Oracle\Middleware\wlserver_10.3.

b. Set the Java SDK Home field: Click Browse and navigate to your Java installation directory; for example, C:\Program Files\Java\jdk170_51.

c. Set the OSM Home field: Click Browse and navigate to your OSM SDK installation directory; for example, C:\OSM\SDK.

4. In the Delete Orphaned Task References With Activity section, select the Prompt radio button.

5. Click OK.

About Design Studio Perspectives and Views

Perspectives define your Workbench layout and provide different functionality for working with different types of resources. Several perspectives are available within Design Studio. The Java perspective, Design perspective, and Environment perspective are commonly used when customizing OSM.

Within a given perspective, views further define your Workbench layout and provide different presentations of resources. Several views are available within Design Studio, and the available views are dependent upon the perspective.

For example, within the Java perspective, commonly used views include:

- Ant
- Navigator
- Package Explorer

Within the Design perspective, commonly used views include:

- Studio Projects
- Package Explorer
- Solution

Within the Environment perspective, commonly used views include:

- Cartridge Management
- Environment

For instructions on how to open perspectives and views in Design Studio, see the Design Studio Help.

About Cartridges

Cartridges contain OSM customizations, such as order templates, order processes, automated tasks, and custom automation plug-ins. Use Design Studio to develop cartridges that model your business logic.

When customizing OSM, you can create one or more cartridges, depending on how you choose to organize the customizations. For example, you may create a cartridge that contains only order templates, another that contains only processes, another that contains only automations, and so on. Or, you may create one cartridge per business area where all order templates, processes, and automations for one business area are defined in one cartridge, while all order templates, processes, and automations for another business area are defined in a different cartridge.
Everything you create in Design Studio (order templates, processes, automations, and so on) resides in a cartridge. After the initial build of the cartridge project, a corresponding portal archive (PAR) file is created. A PAR file is a standard ZIP file with a .par extension that contains Java classes, Web resources, and other files required to run the application. The name you choose for the cartridge becomes the name of the PAR file. The PAR file is saved to the cartridgeName/cartridgeBin directory, as seen from the Java perspective Package Explorer view. Everything you create in Design Studio that resides in a cartridge is saved to the corresponding PAR file.

There are two types of cartridges in OSM: component cartridges and composite cartridges. Most cartridges (and all cartridges in versions of OSM prior to 7.2) are component cartridges. An OSM composite cartridge is a type of cartridge that references other cartridges, called component cartridges, within a single logical scope. You use component cartridges and composite cartridges together to develop, deploy, and maintain OSM solutions. A composite cartridge is used to assemble an OSM solution from a collection of component cartridges. When you deploy the composite cartridge into the run-time environment, all of the component cartridges that are referenced in the composite cartridge are deployed as a solution in a single action.

Composite cartridge projects may contain any number of component cartridges, but not other composite cartridges. You cannot create entities directly in a composite cartridge project; rather, you create the entities in the component cartridges that are referenced in the composite cartridge.

You can configure how orders are fulfilled by deploying cartridges in different ways. For example:

- You can deploy different cartridges on different instances of OSM. For example, you can deploy a specific set of cartridges on an instance of OSM that is dedicated to central order management.
- You can make changes to a cartridge after the cartridge has been deployed to the OSM server by making changes to the original cartridge in Design Studio and then redeploying the cartridge.
- You can fulfill orders differently by using functionality deployed by different cartridges.
- You can fulfill orders differently based on functionality deployed by different versions of the same cartridge.

**Configuration Guidelines: Steps to Create an Orchestration Cartridge**

The following is an outline of creating a cartridge using Design Studio.

- Create a new OSM project in Design Studio.
- Create new role that will own the execution of the activities.
  - Associate the role to manual and automated tasks.
  - Define a query task for the role, which will be used later for defining which data is available when a user performs a query.
- Create a new ControlData section in the data schema and populate it.
  
  The *OracleComms_OSM_CommonDataDictionary* model project contains predefined base data elements for control data. It is recommended that you use the data schema of this model project to add the *ControlData* data element in the data schema.

If you define ControlData in the data schema manually:
– Set Cardinality: it should be able to accept multiple order items.
– The componentKey element in ControlData is 100 chars long. Others do not have an explicit length.
– Display names should be the same as real names.

■ Populate the main data schema. This will hold all the individual data elements and data structures that contain the order data.

■ Configure recognition rules.
  – Set name space upon creation.
  – Fill in recognition and transformation tabs.

■ Define order item specs.
  – Define data instance to support product spec mapping.

Create or import at least a skeleton order item specification that contains the base ControlData/OrderItem structure. The OracleComms_OSM_CommonDataDictionary model project contains this base structure.

■ Define manual tasks and sub-processes that will be mapped in order component specs.

■ Define order component specifications.

■ Define fulfillment modes (which must be unique in the workspace).

■ Create orchestration stages.

■ Create a three-stage orchestration sequence with the following stages: Function, followed by system, followed by processing granularity.

■ Create an orchestration process.

■ Create OSM fulfillment patterns that realize the conceptual model fulfillment patterns.

■ Create products and fulfillment patterns in the conceptual model.

■ Define decomposition rules.
  – Use two types of decomposition rules: System decomposition rules to map the functions to a system and processing granularity decomposition rules to map a function-system combination to the "level of execution," for example order or order item level.
  – Even in cases where there is only one system of a given fulfillment type, use dummy decomposition rules (a 1:1 mapping) for ease of maintenance if more systems are added later.

■ Create the creation task and add the needed data.

■ Create an order life-cycle policy.

■ Complete the order.

Creating a Cartridge

For instructions on how to create a cartridge in Design Studio, see the Design Studio Help.
When you create and build a cartridge, OSM gives the cartridge and resultant PAR file the same name. The name is based on what was entered in the Project name field when creating the cartridge.

**Studio Projects View of a Cartridge**

Figure 3–1 shows an example of a cartridge, named TestProject, as it appears in the Design perspective Studio Projects view (left side). The corresponding Order and Service Management Project editor is also shown (right side).

![Figure 3–1 Studio Projects View of a Cartridge](image)

Expand the cartridge in the Studio Projects view (on the left in Figure 3–1) to see the contents created with each cartridge. When you initially create a new OSM component cartridge, you must define:

- An order
- A creation task
- A default process
- A role that grants creation permissions
- An order life-cycle policy
- An order template
- Order permissions

A composite cartridge does not require these entities.

**Package Explorer View of a Cartridge**

Switching to the Java perspective Package Explorer view and expanding the cartridge displays the file types of the contents created with each cartridge.

Figure 3–2 shows an example of a cartridge, named myCartridge, as it appears in the Java perspective Package Explorer view. The corresponding Order and Service Management Project editor is also shown.
In the example, *myCartridge* was entered in the **Project name** field when creating the cartridge. As a result, the Java perspective Package Explorer view shows:

- **myCartridge**: The Design Studio Order and Service Management Cartridge project.
- **myCartridge/dataDictionary/myCartridge.xsd**: The schema file used internally by Design Studio.
- **myCartridge/model/myCartridgeOrder.order**: The Order editor.
- **myCartridge/myCartridge.osmCartridge**: The Order and Service Management Project editor shown on the right side of Figure 3–2.

After creating the cartridge, an immediate build of the project creates additional directories and files in the cartridge, as shown in Figure 3–3. The directories include, among others:

- **cartridgeBin**
  
  This directory contains the *myCartridge.par* file, which contains the Design Studio entity files and is deployed to the OSM server.

- **customAutomation**
  
  This directory is created with the cartridge, but the **automationMap.xsd** and **databasePlugin.xsd** files are pulled into the cartridge with the build.
Adding Content to a Cartridge

For instructions on how to add content to a cartridge in Design Studio, see the Design Studio Help.

To see the OSM-specific options that are available to add to a cartridge:

- From the Studio menu, select New, then select Order and Service Management.
For example, some of options may be Automated Task, Custom Automation Plug-in, Manual Task, Order, and Process. When content is added to a cartridge, Design Studio automatically groups the content based on these options.

For example, when an Automated Task is created, it is grouped under **Automated Tasks**. When a Manual Task is created, it is grouped under **Manual Tasks**. These groupings are not physical directories, and they display only in the Studio Projects view of the Design perspective. Figure 3–4 illustrates this concept, showing four Design Studio entity instances created from two different Design Studio entity types.

### Figure 3–4 Cartridge Contents

![Cartridge Contents](image)

### Handling Multiple Cartridge Versions

Some additional configuration is necessary to deploy multiple versions of a cartridge to an OSM environment. All of the following considerations should be taken into account:

- **Default** field in Project editor Properties tab:
  
  Only one version of a cartridge should be set as the default version of the cartridge. For example, if you have versions 1.0.0 and 2.0.0 of an OSM cartridge deployed, only one of them should have the **Default** box in the editor checked.

  See the Design Studio Help for the Project editor **Properties** tab for more information.

- **Composite and component cartridge versions**:
  
  When you update a component of a composite cartridge, you do not always need to update the version of the composite cartridge as well.

  For example, if CompositeCartridge version 1.0 references ComponentCartridgeA version 1.0 and ComponentCartridgeB version 1.0, when you update...
ComponentCartridgeA to version 1.2, CompositeCartridge and ComponentCartridgeB can both remain at version 1.0.

- Cartridge versioning using XML Catalog:

XML Catalog should be used to allow multiple cartridge versions to refer to their own set of resources using the cartridge model variable CARTRIDGE_VERSION.

If a rewriteURI entry in the XML Catalog contains a version-specific portion in the URI such as “1.0.0” in the following:

```xml
<rewriteURI uriStartString="http://company.com/"
rewritePrefix="osmmodel:///MyCartridge-Resources/1.0.0/resources"/>
```

the version-specific portion of the rewriteURI must be updated to point to the correct cartridge version.

See “Using XML Catalogs to Support Cartridge Versioning” for more information.

- Automation – External Event Receiver:

When there are multiple versions of automation external event receivers listening to the same JMS Source, this can pose a problem because the automation external event receivers may consume the wrong version of the message. To configure the correct version of the message to be consumed, a version identifier should be encoded in either the JMS message properties or the XML message body. This will allow the automation to consume the right version of the message through either the Message Property Selector or XML Message Body Selector option in the External Event Receiver tab of the automation property.

See the Design Studio Help for the **External Event Receiver** sub-tab of the properties view in the automated task editor **Automation** tab.

- Order Recognition Rule:

When there are multiple versions of a cartridge with orchestration entities, order recognition rules should be modeled to recognize a specific version of the order instead of the default version. To recognize a specific version of the order, the Target Order Version of the order recognition rule should be set to the version of the cartridge where the specific version of the order resides.

**Note:** When an order recognition rule is used in a composite cartridge and there are multiple versions of the composite cartridge, the Target Order Version of the order recognition rule should be set to the version of the composite cartridge that contains the target order as part of the solution.

For example, we might have version 1.0.0 of the OsmCentralOMExample-Solution composite cartridge with the following dependent cartridges:

- OsmCentralOMExample-Orchestration version 1.2.0
- OsmCentralOMExampleOrder is defined here
- OsmCentralOMExample-ProductClass version 2.0.0
- OsmCentralOMExample-ProductSpec version 2.0.0
- OsmCentralOMExample-Topology version 1.1.0

The target order version of the order recognition rule should be set to 1.0.0, because the order recognition rule is used as part of a composite cartridge and the version of the composite cartridge that contains the order is 1.0.0.
Packaging and Building a Cartridge

Use Design Studio to package a cartridge by specifying entities to include in the cartridge. By default, all entities created within the cartridge are included unless otherwise specified on the Order and Service Management Project editor Packaging tab.

To build cartridges:

- From the Project menu, select Build.

Oracle recommends that you periodically clean the project prior to a build (see "Cleaning and Rebuilding Cartridges Prior to Deployment").

For instructions on how to package and build a cartridge, see the Design Studio Help.

Modifying the Build

If you need to modify the build performed by Eclipse, you can modify the build files that are provided with the creation of each cartridge. Common modifications include adding logic to the build file for the generation of Java code and the creation of JAR files. The build files are:

- CartridgeName/src/build.xml
- CartridgeName/src/automationBuild.xml

As part of the build process, the CartridgeName/lib directory is copied into the automation staging area, resulting in the lib directory being included in the automation.ear file. The CartridgeName/src/build.xml file can be customized to add files to the lib directory before this happens. For example, you may want to get a JAR file from another project as part of the build or do some other custom staging activity. Nothing in the lib directory goes on the classpath automatically. You can do this manually as well.

Deploying a Cartridge

Design Studio allows you to deploy a cartridge to an OSM environment. However, you must first specify the OSM environment by creating a Design Studio environment entity. The Design Studio environment entity defines the connection information for the server hosting the OSM environment to which you plan to deploy your cartridge. For instructions on how to define an environment in Design Studio, or instructions on how to do any of the steps described in this section, see the Design Studio Help.

---

**Note:** You do not need to define the environment every time you deploy a cartridge; the environment is only be defined once per OSM environment.

---

**Note:** If you are deploying to a clustered environment, specify the proxy server when defining the Environment Design Studio entity.

---

To deploy a cartridge:

1. Ensure the OSM environment to which you are deploying is running.
2. Ensure the Environment Design Studio entity is defined for the OSM environment to which you are deploying.
3. From Design Studio, deploy the cartridge to the OSM server, which can be done one of two ways:
   a. From the Studio menu, select **Deploy**.
      A menu is displayed, listing the cartridges within your workspace that are available for deployment.
   b. Select the cartridge to deploy.
      A menu is displayed, listing the configured OSM environments available to which you can deploy a cartridge.
   c. Select the environment to which the cartridge is to be deployed.

OR

a. Open the Environment perspective Cartridge Management view.
   A list of the cartridges within your workspace that are available for deployment is displayed.

b. Select a cartridge to deploy.

c. Click **Deploy**.

---

**Note:** You can also undeploy previously deployed cartridges from the Environment perspective Cartridge Management view.

---

**Cleaning and Rebuilding Cartridges Prior to Deployment**

Cleaning and rebuilding a cartridge is not included as a deployment step because it is not required for a successful deployment. However, Oracle recommends that you periodically clean and rebuild a cartridge prior to deployment because multiple people can work in the same cartridge; cleaning and rebuilding the cartridge picks up these changes, ensuring that the cartridge is in its current state.

**Optimizing Cartridge Deployment**

During the development process, you can save time by redeploying your changes only, rather than redeploying the entire application. For more information about this option, see the Design Studio Platform Help.

**Deploying Multiple Cartridges**

You can simultaneously deploy multiple cartridges when deploying from the Environment perspective Cartridge Management view. When you select multiple cartridges for deployment, the system deploys the cartridges individually based on any existing cartridge dependencies. The system prevents you from deploying cartridges independently of those cartridges upon which they depend. See "Deploying Cartridges with Dependencies".

**Deploying Cartridges with Dependencies**

A cartridge can be dependent upon information defined in another cartridge. When dependencies exist between cartridges, the build of the cartridge with the dependency extracts the dependent information from the built cartridge upon which it is dependent and copies it to the cartridge being built. As a result, the cartridges can be deployed independently from each other.

For example, CartridgeA is created and defines **phoneNumber** as a data element in a data schema. CartridgeB is then created, and **phoneNumber** is added to a CartridgeB
order template. This causes CartridgeB to be dependent upon CartridgeA. CartridgeA is built first. When CartridgeB is built, the **phoneNumber** data element is extracted from CartridgeA and copied to cartridgeB. As a result, cartridgeB can be deployed even if CartridgeA is not deployed.

---

**Caution:** Cartridges should not be circularly dependent upon each other (CartridgeA is dependent upon CartridgeB and CartridgeB is dependent upon CartridgeA). If you define cartridges with a circular dependency, the cartridge build will fail, with an error like, "CartridgeA Cartridge Model Dependency Error – Cyclic dependency exists: CartridgeA <- CartridgeB."

If there is a composite cartridge that refers to cartridgeX or cartridgeY, the composite cartridge build will also fail, as a result of the component cartridge builds failing.

---

**Building and Deploying Composite Cartridges**

When you build and package a composite cartridge, it is packaged as a single PAR file which contains:

- All non-orchestration entities aggregated and packaged into the composite cartridge
- A PAR file for each component cartridge referenced in the composite cartridge

When a composite cartridge is deployed, it includes all of the OSM non-orchestration entities and all component cartridges referenced in the composite cartridge, if they are either changed or not currently deployed.

---

**Setting Cartridge Dependencies**

Projects have dependencies on other projects when entities in one project reference entities in a different project. If you configure a cartridge to reference content in other cartridges without declaring project dependencies, Design Studio creates a warning. For information about how to set cartridge dependencies, see the Design Studio Platform Help.

---

**Post-Deployment Effect on Numeric Data**

When defining a data element in Design Studio, you have the option of defining numeric data as type int, double, float, or decimal. OSM does not directly support these data types. Rather, the OSM Data Dictionary defines the data type numeric. When a cartridge containing the data types int, double, float, or decimal is deployed to the OSM server, the data types are converted to the OSM Data Dictionary type numeric.

---

**Post-Deployment Changes to Cartridge**

You can make changes to a cartridge after the cartridge has been deployed to the OSM server by making changes to the original cartridge in Design Studio and then redeploying the cartridge. Before doing this, you should back up the original cartridge, because exporting a deployed Design Studio cartridge back out of OSM into Design Studio is not supported.
**Metadata Errors**

Metadata errors can cause order processing failures and can occur in any cartridge with orchestration model entities. Metadata is the information used to represent OSM modeled entities such as order templates, order components, order items, tasks, decomposition rules and so on. If there are no metadata errors, the cartridge models deployed are valid.

Metadata errors occur when OSM references an entity that is missing or the modeling for an entity is incorrect (for example, a data type for an entity is incorrectly entered).

OSM detects and logs metadata errors during the following procedures:

- Deploying a cartridge to a server
- Restarting an OSM server
- Refreshing OSM metadata with the OSM Administrator application or with an Ant refresh

These actions reload OSM metadata, and errors are detected while running validation constraints against certain orchestration model entities. Table 3-1 lists the orchestration entities that are currently validated.

<table>
<thead>
<tr>
<th>Entity Type</th>
<th>Schema Constraint Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OrchestrationStageType</td>
<td>Verifies that the value for the element dependsOnStage is a valid stage.</td>
</tr>
<tr>
<td></td>
<td>dependsOnStage is empty if the stage is independent. A stage is valid if it is defined in</td>
</tr>
<tr>
<td></td>
<td>the orchestrationSequence of orchestrationModel.</td>
</tr>
<tr>
<td>OrderComponentSpecRef</td>
<td>Verifies that this reference is pointing to a valid OrderComponentSpec.</td>
</tr>
<tr>
<td></td>
<td>OrderComponentSpec is valid if it is defined in orchestrationModel.</td>
</tr>
<tr>
<td>OrchestrationConditionRef</td>
<td>Verifies that this reference is pointing to a valid orchestration condition.</td>
</tr>
<tr>
<td></td>
<td>An orchestration is valid if it is defined in orderItemSpec of orchestrationModel.</td>
</tr>
<tr>
<td>DurationType</td>
<td>Verifies that a valid duration value is specified.</td>
</tr>
<tr>
<td>ProductSpecRef</td>
<td>Verifies that this reference is pointing to a valid ProductSpec.</td>
</tr>
<tr>
<td></td>
<td>ProductSpec is valid if it is defined in orchestrationModel.</td>
</tr>
<tr>
<td>OrderItemSpecRef</td>
<td>Verifies that this reference is pointing to a valid OrderItemSpec.</td>
</tr>
<tr>
<td></td>
<td>OrderItemSpec is valid if it is defined in orchestrationModel.</td>
</tr>
</tbody>
</table>

After rebuilding or deploying a cartridge, check for metadata errors. Search for the string "Metadata Errors" in the Console view in Design Studio. If you are not using Design Studio to deploy cartridges, look in the Oracle WebLogic Server logs for the same string.

Metadata errors appear together in a numbered list. For example:

```
Metadata Modeling Errors
**************************
1) Metadata error Severity:ERROR Description:Invalid ProductSpec[name=NonService.Offer, namespace:CommunicationsSalesOrderFulfillmentPIP] Cartridge Name:TypicalSalesOrderFulfillment Version:1.0.0 EntityName:TypicalSalesOrderFulfillment Version:1.0.0 EntityName:TypicalSalesOrderFulfillment Version:1.0.0
```

where
- **Severity** can be an ERROR, WARNING or CRITICAL.
- **Description** describes the failure and provides the entity type, name and name space.
- **Cartridge Name** is the name of the Cartridge that is reporting the problem.
- **Version** is the cartridge version.
- **EntityName** and **Entity Type** are the name and type of the entity reporting the metadata error and its name space. In some cases, the modeled entity within the cartridge is invalid. In other cases, the modeled entity is referring to another entity which is missing or invalid.

If you find metadata errors, it most likely means that OSM is calling on an entity that is missing, has the wrong name, or has a value that is incompatible for the entity type.

To fix the problem, clean and rebuild your cartridges, and make sure all related cartridges are deployed. If you still have metadata errors, it may mean that you have errors in your data. In this case you will have to use Design Studio to re-validate your model. See “Cleaning and Rebuilding Cartridges Prior to Deployment”.

### Automating the Cartridge Build and Deployment

Automating the cartridge build and deployment is accomplished by creating a process that builds and deploys a cartridge outside of the Design Studio environment, and then scheduling that process to run using a batch file. You can automate the cartridge build and deployment for development, test, and production environments.

For more information, see the discussion of working with automated builds in *Design Studio System Administrator’s Guide*.

### Maintaining Cartridges Between Releases and Patches

Cartridges can be maintained between releases of OSM. If you have cartridges built in a prior release, you must clean and rebuild the cartridge in the new version of Design Studio before it can be deployed to the new version of OSM.

For example, OSM 7.0 is released with Design Studio 3.0. You build several cartridges in Design Studio 3.0 that you deploy to your OSM 7.0 environment. Later, you upgrade to OSM 7.1, which is released with Design Studio 3.1. The cartridges you created in Design Studio 3.0 can be imported into Design Studio 3.1, cleaned, rebuilt, and deployed to your OSM 7.1 environment.

Cartridges can be maintained between patches of OSM by cleaning and rebuilding the cartridge in the current version of Design Studio, which would not change with a patch (this would only change with a new release of OSM). Not every patch requires this maintenance; it depends on whether the changes in the patch affect cartridge content.

### Modeling Order Template Structures for Orchestration Plan Generation

Orchestration plan generation requires a specific order template structure which you must model at design time.

- ControlData
- OrderItem
- TransformedOrderItem
- Functions
  - OrderComponentName
ControlData/OrderItem

This is a multi-instance node that OSM populates with a set of order items generated off the in-bound message. The children of this structure must exactly match the set of order item properties defined on the Order Item specification editor in Design Studio.

The OracleComms_OSM_CommonDataDictionary model project contains predefined base data elements for control data. It is recommended that you use the data schema of this model project to add the ControlData/OrderItem data element to the order item specification Order Template tab.

See the Design Studio Help for instructions on modeling the ControlData/OrderItem structure.

ControlData/TransformedOrderItem

This is a multi-instance node that OSM populates with a set of order items generated by the order transformation manager.

The OracleComms_OSM_CommonDataDictionary model project contains predefined base data elements for control data. It is recommended that you use the data schema of this model project to add the ControlData/OrderItem data element to the order item specification Order Template tab.

See the Design Studio Help for instructions on modeling the ControlData/TransformedOrderItem structure.

ControlData/Functions/OrderComponentName

This is a multi-instance node that OSM populates with the set of order components generated by executing the decomposition rules through an orchestration sequence. OrderComponentName must be defined for each order component included in a fulfillment pattern’s orchestration plan. This section of the ControlData represents all of the order components in the orchestration plan. If you use the OracleComms_OSM_CommonDataDictionary model project, Design Studio automatically generates data (OrderComponentName) and adds it to the ControlData/Functions structure for each order component that is associated to the fulfillment pattern that is part of the orchestration plan.

Each order component is assigned a unique key, which is stored in componentKey. OSM generates a key by concatenating the names of the order components traversed through the orchestration stages. An example componentKey name would be FulfillBilling.MobileBillingSystem.WholeOrderGranularity. The Order Component specification editor’s Component ID tab can be used to customize an order component’s name. The component ID can be used to implement processing granularity, as an example. In BundleGranularity, a component ID string would be generated for each order item based on the bundle that it belongs to. OSM then groups order items by component ID into order components.

OSM populates the calculatedStartDate (dateTime type) and duration (string type) nodes for each ControlData/Function. With calculatedStartDate and duration per Function, both central order management and service order management solutions can
use these values as the requested delivery date for the order line in a downstream system. Based on the modeling done in the Order Component Specification entity, the date does affect the runtime behavior of the order component. If there is a Duration Value associated with a dependency, it is used in the order component start date calculation since this value is relative value to the orchestration dependency.

OSM populates the multi-instance `orderItem` node with the set of order items that have been decomposed into this order component. The order items are accessed through `orderItemRef`, which is a reference node to `ControlData/OrderItem`. A reference node is used to point to the actual storage location of the order item so that updates to the order item data are reflected in all order components the order item is referenced from.

**Modeling Order Template Structures for Fulfillment States**

Fulfillment state processing requires specific structures and data elements inside the order template. The specific locations of the data can be changed using XML catalog: the default locations are presented here. See "About XML Catalogs" for more information about using XML catalogs in OSM. See "Sample XQuery for Changing Default Data Locations" for more information about changing the default data locations.

**External Fulfillment States**

External fulfillment state information is populated for order components. Write the automation code so that it populates the information in the correct place.

The default location for external fulfillment state information is `ControlData/OrderItem/OrderComponentName/ExternalFulfillmentState`.

**Order Fulfillment State**

OSM populates the order fulfillment state based on the configuration in the order fulfillment state composition rule set. For more information about order fulfillment state composition, see OSM Concepts.

The default location for OSM to populate the order fulfillment state is `ControlData/OrderFulfillmentState`. The Data Dictionary contains a root-level `OrderFulfillmentState` element. For cartridges created in a pre-7.2 version of OSM, drag the root-level `OrderFulfillmentState` element into the `ControlData` node on the order. For new cartridges, the element will get added automatically to the order template as a child of `ControlData`.

**Order Item Fulfillment State**

OSM populates the order item fulfillment state based on the configuration in the order item fulfillment state composition rule set. For more information about order item fulfillment state composition, see OSM Concepts.

The default location for OSM to populate the order item fulfillment state is `ControlData/OrderItem/OrderItemFulfillmentState`. The Data Dictionary contains a root-level `OrderItemFulfillmentState` element. For order items in cartridges created in a pre-7.2 version of OSM, drag the root-level `OrderItemFulfillmentState` element into the `ControlData/OrderItem` node on the order. For new cartridges and order items, the element will get added automatically to the order template as a child of `ControlData/OrderItem`.
Point of No Return

If points of no return have been configured using fulfillment states, OSM populates the point of no return when processing the order item fulfillment state composition rules. For more information about points of no return, see OSM Concepts.

The default location for OSM to populate the point of no return value is ControlData/OrderItem/PointOfNoReturn.

Fulfillment State and Point of No Return Initial Values

You can set initial values for order item fulfillment states and points of no return, so that these values will appear on the order before any processing takes place. See "Sample XQuery for Changing Default Data Locations" for more information about setting these values.

Sample XQuery for Changing Default Data Locations

To change the default locations and set initial values for point of no return and order item fulfillment state, include an XQuery file in the XML catalog. To use the defaults, do not provide a file.

To include your custom XQuery file in the cartridge, include a line similar to the following in the XML catalog file for your cartridge:

```
<rewriteURI uriStartString="cp:oracle/communications/ordermanagement/execution"
    rewritePrefix="osmmodel://[/CartridgeName/CartridgeVersion/resources/Directory]"/>
```

For more information about using XML catalogs, see "About XML Catalogs".

If you choose to configure a custom file, you should include all of the functions, even those for defaults you are not changing. This will clarify the configuration and assist in maintenance activities. The purpose of each function is indicated in comments in the file. For all values that specify order template locations (for example /OrderLifeCycleManagement), begin the value with a forward slash, as shown below.

```xquery
xquery version "1.0";
module namespace fulfillmentstatemodule = "http://xmlns.oracle.com/communications/ordermanagement/fulfillmentstatemodule";

declare namespace saxon="http://saxon.sf.net/";
declare namespace xsl="http://www.w3.org/1999/XSL/Transform"
declare namespace oms = "urn:com:metasolv:oms:xmlapi:1";

(: Returns the composite fulfillment state path for an order. :) declare function fulfillmentstatemodule:getOrderCompositeFulfillmentStatePath ($orderMnemonic as xs:string) as xs:string {
    "/ControlData/OrderFulfillmentState" }

(: Returns the composite fulfillment state path for an order item. :) declare function fulfillmentstatemodule:getOrderItemCompositeFulfillmentStatePath ($orderMnemonic as xs:string) as xs:string {
    "/ControlData/OrderItem/OrderItemFulfillmentState" }

(: Returns the default order item external fulfillment state path. :) declare function fulfillmentstatemodule:getOrderItemExternalFulfillmentStatePath ($orderMnemonic as xs:string) as xs:string {
    "ExternalFulfillmentState" }

(: Returns the default type of the order item external fulfillment state path. Valid values are RELATIVE_PATH and ABSOLUTE_PATH. :)
declare function fulfillmentstatemodule:getOrderItemExternalFulfillmentStatePathType($orderMnemonic as xs:string) as xs:string {
  "RELATIVE_PATH" 
};

(: Returns the point of no return path for an order item. :) 
declare function fulfillmentstatemodule:getOrderItemPoNRPath ($orderMnemonic as xs:string) as xs:string {
  "/ControlData/OrderItem/PointOfNoReturn" 
};

(: Returns the name of the initial fulfillment state. :) 
declare function fulfillmentstatemodule:getOrderInitialFulfillmentStateName ($orderMnemonic as xs:string) as xs:string {
  "" 
};

(: Returns the namespace of the initial fulfillment state. :) 
declare function fulfillmentstatemodule:getOrderInitialFulfillmentStateNamespace ($orderMnemonic as xs:string) as xs:string {
  "" 
};

(: Returns the initial point of no return value of an fulfillment state. :) 
declare function fulfillmentstatemodule:getOrderItemInitialPoNR($orderMnemonic as xs:string) as xs:string {
  "" 
};

declare function fulfillmentstatemodule:getExternalFulfillmentStates($orderData as element()) as element()?
{
  let $orderMnemonic :=
    if (fn:exists($orderData/OrderType))
      then $orderData/OrderType/text()
      else ""
  let $orderItems := $orderData/_root/ControlData/OrderItem
  where (fn:exists($orderItems))
  return
    <oms:ExternalFulfillmentStates>
    {
      for $orderItem in $orderItems
        let $orderItemIndex := $orderItem/@index
        let $components := $orderData/_root/ControlData/Functions/*[orderItem/orderItemRef/@referencedIndex=$orderItemIndex]
        let $externalFulfillmentStatePath :=
          fulfillmentstatemodule:getOrderItemExternalFulfillmentStatePath($orderMnemonic)
        let $externalFulfillmentStatePathExistsCheck :=
          fn:concat($externalFulfillmentStatePath, "[text()!='']")
        let $externalFulfillmentStateExists :=
          fn:exists($components/orderItem[orderItemRef/@referencedIndex=$orderItemIndex]/sax on:evaluate($externalFulfillmentStatePathExistsCheck))
        where (fn:exists($components) and
          $externalFulfillmentStateExists=fn:true())
        return
          <oms:OrderItemExternalFulfillmentState index="$orderItemIndex">
          {
            for $component in $components
              let $componentKey :=
                fn:normalize-space($component/componentKey/text())
              let $componentId := $component/@index
              let $externalFulfillmentStateValuePath :=
                fn:concat($externalFulfillmentStatePath, "[last()]/text()")
          
          } 
        
      
    } 
  
  
}
Orchestration Order Template Definition

Define the orchestration data on the entity that best reflects its structure, rather than defining all of the data on the order specification. Design Studio generates the order level order template by aggregating the order template definitions for the order item specifications and order components with any data defined at the order level.

You should define data at the level where it is needed:

- **Order Item specification**: Define **ControlData/OrderItem** and all of the order item properties on the order item specification.

  The OracleComms_OSM_CommonDataDictionary model project contains predefined base data elements for control data. It is recommended that you use the data schema of this model project to add the ControlData/OrderItem base data element to the order item specification Order Template tab.

- **Order component**: Define **ControlData/Functions/OrderComponentName** and any other data needed by the tasks in the process that execute this component in the appropriate order component template.

If you use the OracleComms_OSM_CommonDataDictionary model project (recommended) and your orchestration entities are preconfigured correctly, Design Studio automatically generates this structure on the order template of the order component and the order template of the order.

Using this method supports:

- **Encapsulation**

- **Re-factoring**: Modify order template data at the entity level to which it is associated because this highlights the connection between an entity and its order template data.

- **Maintenance**: Modifications to order item specification and order component templates help the designer understand the impact of changes, including possible breaks in compatibility.

- **Traceability**: Using this method provides direct traceability from order template data to the modeling entity to which it is attached.
Modeling Orders With Data Fields Above 1000 Characters

Standard OSM Design Studio data elements and structures can support a maximum of 1000 characters. However, in some cases it may be necessary to model data that exceed this limit. Before you model order data fields than can contain more than 1000 characters, you must carefully decide whether these fields are necessary. Unnecessary data within an order can reduce the order processing performance of OSM.

The following sections describe ways to achieve data length for OSM data above 1000 characters.

Using XML Types for Data Fields Above 1000 Characters

In Design Studio, you can model data dictionary structures as XML types from the Order specification, Order Template, Properties sub-tab, Order Data sub-tab. The structure must be empty and contain no children elements or structures for it to be designated as XML type. Structures defined as XML types in the data dictionary can contain XML documents. You can also use XML schema files to validate the XML structures in the XML types.

Oracle recommends this option when the data is not human editable or readable in the OSM user interfaces because the data is represented as XML. For example, the XML data can be captured as follows, where `<largetext>` is the name of the structure designated as XML type:

```
<largetext>
Text to be inserted here
</largetext>
```

When you have defined the XML type structures in the Order specification Order Template, then included them as a part of Manual or Automated Task Data, you can access the XML data using:

- The OSM Task Web client Order Editor screen (see OSM Task Web Client User’s Guide for more information).
- XML API GetOrder and UpdateOrder transactions (see OSM XML API Developer’s Guide for more information).
- Web Services Getorder and UpdateOrder OSM transactions (see "Using OSM Web Services" for more information).
- Order access and updates performed using Automated Task automation plug-ins (see "Using Automation" for more information).

This approach has the following limitations:

- You cannot specify XML type data as significant for amendment processing. Changes to this data does not trigger compensation.
- XML types are not visible in the OSM reporting interface.

To enable XML schema validation:

1. Create schema files for the required XML data type.
2. Use the Java perspective Package Explorer view to copy the schema files into the cartridge project data dictionary folder where the XML data type has been defined.
Using Order Remarks for Data Fields Above 1000 Characters

You can add Remarks that contain text to orders during order processing. Remarks can be retrieved and updated using:

- The OSM Task Web client Remarks and Attachments screens (see OSM Task Web Client User’s Guide for more information).
- XML API GetOrder and UpdateOrder transactions (see OSM XML API Developer’s Guide for more information).
- Web Services GetOrder and UpdateOrder transactions (see "Using OSM Web Services" for more information).
- Order access and updates performed using Automated Task automation plug-ins (see "Using Automation" for more information).

This approach has the following limitations:

- Remarks can store up to 4000 bytes of data. Depending upon the character set configured in your database, the number of characters will vary.
- Remarks associated with orders are only editable for a certain time after you add them. This time limit is defined by the remark_change_timeout_hours parameter contained in the oms-config.xml file. You can edit the value associated to this parameter to change the number of hours that remarks are editable. The default value is 24 hours. See OSM System Administrator’s Guide for more information about editing the oms-config.xml file.

Using Attachments for Data Fields Above 1000 Characters

You can also add file attachments to remarks. File attachments can contain large amounts of data and you can store them in different formats. You can access attachments with:

- The OSM Task Web client using the Remarks and Attachments screens (see OSM Task Web Client User’s Guide for more information).
- XML API GetOrder and UpdateOrder transactions (see OSM XML API Developer’s Guide for more information).
- Web Services GetOrder and UpdateOrder transactions (see "Using OSM Web Services" for more information).

Attachments are governed by the max_attachment_size parameter in the oms-config.xml file. You can edit the value associated to this parameter to change the maximum attachment size. The default value is 10MB. See OSM System Administrator’s Guide for more information about editing the oms-config.xml file.

---

**Note:** When the remark change threshold is exceeded (remark_change_timeout_hours), you can no longer add or delete attachments to the remark.

---

Creating and Deploying Custom Files

Custom files can be created and written in Design Studio. This can be done by creating a cartridge and adding custom Java, JavaScript, XSLT, XSD, or XML files to the cartridge. In the Java perspective Package Explorer view, you can create package structures and custom files as needed.
OSM JAR files are not automatically pulled into the project library list within Design Studio; depending on what the custom files reference, you may need to update the project library list to include external JAR files.

Custom files added to a cartridge are deployed to the OSM server as part of the cartridge.

About XML Catalogs

XML Catalogs are logical structures that act like address books or directories. XML Catalogs contain entries that indicate a placeholder location and then provide the path to the location to be used. 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 can update the URI to adapt to different environments by resolving the location of the URI in your data model with the location it is mapped to in the XML Catalog. For example:

- OSM resolves a URI against a test server in a test environment and resolves that URI against a different server in a development environment.
- OSM resolves the location of files in a developer’s local workspace to the location of equivalent files available to the OSM server at a generic URI. You might use XML Catalogs in this way for XQuery module import statements that at design time need to refer to files in your local workspace but at run time need to refer to files within the resources directory of a deployed cartridge.
- Your OSM model might reference a resource located on the Internet. If your server deployment runs behind a firewall with no Internet access, you can load the resource behind the firewall and use an XML Catalog to redirect the URI of the Internet location to the location of the resource behind the firewall.

See the OASIS Web site at:
http://www.oasis-open.org/committees/entity/spec-2001-08-06.html
for more information on XML Catalogs and valid XML Catalog entries.

See "Using XML Catalogs in OSM" for information on how you can use XML Catalogs in your OSM development.

Using XML Catalogs in OSM

In Design Studio, you model behaviors such as business rules and other model components, which OSM uses at run time to satisfy your business requirements for order processing. The model components used at run time to manage and fulfill orders are referred to as OSM resources and are often contained in resource files. Examples of resource files include XQuery files, XSLT files, custom JAR files, third-party JAR files, and XML files such as a product specification mapping file. There can be a large quantity of resources and some of those resources must reference each other. Resources in OSM can be referenced through URI locators in your data model.

A resource must reside on some physical location on a system. Each system has its own unique directory structure. If you use static values or constants to indicate the location of a resource when defining the URI locator for that resource in your data model, the resource will not be accessible if you deploy your cartridge to other systems where the resource is in a different directory. Thus, using static values to indicate the location of a resource limits the portability of your cartridge solution to other systems or run-time environments. XML Catalogs solve this problem by redirecting the URI defined in your data model to the URI where the resource actually
resides in whichever run-time environment you deploy your cartridge. XML Catalogs provide a redirection from a URI to another URI. By redirecting the resource URI locators, XML Catalogs serve to insulate your cartridge solution from environment configuration.

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.

OSM processes XML Catalogs in the order you specify them, as follows:

- Specified in your OSM cartridge projects
  XML Catalogs specified in your OSM cartridge projects are packaged as part of the cartridges and deployed to the OSM server. The XML Catalog manages only the resource files in the resources folder of your cartridge project. When you deploy a cartridge with XML Catalog support enabled, the contents of the resources folder are loaded into a virtual file system. Those resources are available through URI redirection to any other deployed cartridges. XML Catalogs can be defined in any cartridge, and those defined in one cartridge can reference resources in other cartridges. All of the XML Catalogs deployed on the OSM server are stored in memory and rebuilt each time the metadata refreshes. If there are conflicting XML Catalog entries, the latest entry loaded overwrites the earlier entry. See “Defining rewriteURI Entries in XML Catalogs” for information on how to avoid conflicting entries.

- Specified on the OSM server
  XML Catalogs specified on the OSM server are defined in the `oms-config.xml` file and are loaded ahead of the XML Catalogs specified in OSM cartridge projects. XML Catalogs defined on the server are global in scope, applying to all cartridges. XML Catalogs specified on the OSM server override the URI mapping of XML Catalogs in cartridge projects. URIs mapped in `oms-config.xml` are resolved for each specific environment. For example, a cartridge developer can specify an XML Catalog in `oms-config.xml` to point certain URIs defined in the data model to her own local Design Studio workspace, allowing her to change the contents of the resources locally and test the changes without having to redeploy the entire cartridge. Because OSM uses XML Catalogs that are specified on the OSM server to resolve URIs to be environment specific, XML Catalogs specified in OSM cartridge projects should not reference URI locations that are environment specific (such as drive letters).

Following are some examples of data OSM looks up from resource files at run time that you could use the XML catalog to redefine:

- Automation logic: You can configure XQuery and XSLT automators with the XML Catalog to specify the XQuery/XSLT file that drives the automation logic.

- Data from a data provider: A data instance provider can use the XML Catalog to specify a resource for providing the data loaded by the provider.

- Order item properties (for orchestration orders): Order item properties can be configured to be loaded through a URI locator. You can configure the XML Catalog to redirect the URI to specify the XQuery file that implements determining the property value.

- Decomposition rules (for orchestration orders): Decomposition rules can be configured to be loaded through a URI locator. You can configure the XML Catalog
to redirect the URI to specify the XQuery file that implements determining the decomposition condition.

See "Specifying XML Catalogs for OSM" for instructions on how to specify XML Catalogs.

You can use the XML Catalog as a tool to perform cartridge versioning, to shorten development cycles, to allow for cartridge extensibility, and to insulate test and production environments from development-specific environments. See "Examples of Using XML Catalogs" for examples of these uses of the XML Catalog.

You can specify a common resources cartridge project that contains all of the shared resources across multiple cartridge projects. Defining the XML Catalog in this common resources cartridge consolidates the XML Catalog entries in one file which makes it easy to identify and eliminate conflicting catalog entries. See "Resource Packaging Considerations for Using XML Catalogs" for information on how you can package your resources when using XML Catalogs.

You can use any valid XML Catalog entry in your XML Catalog, but the rewriteURI entry is the most useful for OSM. See “Defining rewriteURI Entries in XML Catalogs” for information on defining rewriteURI entries for OSM.

**Resource Packaging Considerations for Using XML Catalogs**

You can specify a common resources cartridge project that contains all of the shared resources across multiple cartridge projects. Defining the XML Catalog in this common resources cartridge consolidates the XML Catalog entries in one file which makes it easy to identify and eliminate conflicting catalog entries. When you specify a common resources cartridge project in this way, other projects with model entities that reference the shared resources do not need to have an XML Catalog defined.

When you define resource properties in Design Studio, you can indicate to retrieve the resource by expression, file, or URI. XML Catalogs apply only to the URI option.

Consider the following when making your decision on which option to choose:

- Select **Expression** when the XQuery expression is short (only a few lines in length) and is not shared by other resources.
- Select **File**, also referred to as **Bundle In**, when the XQuery configuration is longer (more than a few lines in length) and is not shared by other resources. Use this method for resources that are not expected to change. You will not be able to access the resource except in the physical location specified. In addition, a resource referenced through **File** or **Bundle In** must exist in the same project as the entity referencing it.
- Select **URI** when the XQuery configuration is shared by multiple configurations and is located in a remote URI location to be accessed through the specified URI. If the XQuery configuration requires frequent changes, even though it is only used in one cartridge, you may want to use the URI option and also package the XQuery in a separate cartridge. That way, you can modify and redeploy the resource without having to compile and redeploy the possibly larger cartridge that uses it.

*Figure 3–5* shows the **Expression**, **File**, and **URI** options in the **XQuery** tab of a Design Studio editor:
Oracle recommends you package resources in the following ways:

- Package resources to be used by a single cartridge in the cartridge itself. Select File or Bundle in when you define the resource properties in Design Studio.

- Package resources to be used by multiple cartridges into a shared or common resources cartridge and do the following:
  - Configure the resources to be retrieved by a URI. Select URI when you define the resource properties in the XQuery and XSLT tabs of the Design Studio editor.
  - Configure OSM to access the resources inside of the deployed common resources cartridge through a URI locator.

Another reason to package resources in a common resource cartridge is when you need to change those resources frequently and they are used by a large cartridge that has automation and model entities that take a long time to build, package, and deploy. By packaging resources that change frequently in a common resources cartridge, you avoid having to rebuild the larger cartridge each time you change the resources.

**Defining rewriteURI Entries in XML Catalogs**

This section describes how to define a rewriteURI entry in the XML Catalog for OSM. See "Using XML Catalogs in OSM" for general information about XML Catalogs and how they work with OSM.

You can use any valid XML Catalog entry in your XML Catalog, but the rewriteURI entry is the most commonly used entry for OSM. OSM uses the rewriteURI entry to replace the starting string of a URI (such as a URL) with an alternative string. For example, OSM could replace \texttt{http://somewhere.org/something} at run time with \texttt{http://myhost/something}.

During data modeling, you can define a URI locator (such as a URL) to access a resource as part of the OSM data model by using the XQuery and XSLT tabs of various Design Studio editors. For example, in the Order Recognition Rule editor you specify a URI to denote that the XQuery configuration for the recognition rule is hosted in a remote URI location such as \texttt{http://osm_server/AIARecognitionRule.xqy}. You can use the XML Catalog for any of the URIs you specify in the Design Studio editors. OSM uses the rewriteURI entry of the XML Catalog to update URIs you defined in your data model to adapt to different environments.
OSM replaces the starting string of a URI/URL with an alternative string as specified by the rewriteURI entry in the XML Catalog. For example, for this rewriteURI entry:

```
<rewriteURI uriStartString="http://example.org/somewhere"
 rewritePrefix="http://192.0.2.0/foo"/>
```

when OSM processes a URI that starts with `http://example.org/somewhere`, it replaces that starting string with `http://192.0.2.0/foo`. A URI you define in Design Studio as `http://example.org/somewhere/myfolder/myfile.txt` resolves as `http://192.0.2.0/foo/myfolder/myfile.txt` at run time.

---

**Note:** The uriStartString and the rewritePrefix attributes can be any valid URI: they do not have to be an IP address or host name.

---

uriStartString is set to the start of the resource URI you defined in Design Studio and rewritePrefix is set to the string OSM replaces uriStartString with after you deploy the cartridge.

To reference resources packaged inside of an OSM cartridge, you can use the OSM model scheme (“osmmodel”) rather than the traditional URI schemes (HTTP, FTP, and so on) to define the URI. For example, for this rewriteURI entry:

```
<rewriteURI uriStartString="http://example.org/somewhere"
 rewritePrefix="osmmodel:///MyCartridge/1.0.0/resources"/>
```

when OSM processes a URI that starts with `http://example.org/somewhere`, it replaces that starting string with `osmmodel:///MyCartridge/1.0.0/resources`. A URI you defined in Design Studio as `http://example.org/somewhere/myfolder/myfile.txt` is resolved as `osmmodel:///MyCartridge/1.0.0/resources/myfolder/myfile.txt`.

This allows you to leverage the contents of the resources directory in each OSM cartridge at run time.

The format of an OSM model schema URI is:

```
osmmodel:///CartridgeName/CartridgeVersion/resources
```

where:

- `osmmodel` indicates a location inside of a deployed OSM cartridge
- `CartridgeName` is the name of your cartridge
- `CartridgeVersion` is the version of the cartridge (specified in the Project editor)

The default cartridge version uses the value `default`.

---

**Note:** See "Using XML Catalogs to Support Cartridge Versioning" for more information on cartridge versioning.

---

To enable cartridges to refer to resources contained in other cartridges in a non-version specific way, you refer to the default cartridge version. To refer to the default cartridge version, use the OSM model schema URI:

```
osmmodel:///cartridge_name/default/resources
```

See "Using XML Catalogs to Support Cartridge Versioning" for information on how the XML Catalog supports cartridge versioning.
When defining XML Catalog entries, do not define mappings that can be satisfied by more than one entry. The following example shows two rewriteURI entries that can be used by OSM at run time to resolve the same URI locator in two different ways:

```xml
<rewriteURI
uriStartString="http://oracle.communications.ordermanagement.sample.centralom.resources/com"
rewritePrefix="osmmodel:///CommonResourcesCartridge/1.0.0/resources/com"/>
<rewriteURI
uriStartString="http://oracle.communications.ordermanagement.sample.centralom.resources"
rewritePrefix="osmmodel:///CommonResourcesCartridge/1.0.0/resources/comMapping"/>
```

Using the preceding rewriteURI entries, OSM can resolve the URI locator http://oracle.communications.ordermanagement.sample.centralom.resources/com/foo.xml as

```
osmmodel:///CommonResourcesCartridge/1.0.0/resources/com/foo.xml or
osmmodel:///CommonResourcesCartridge/1.0.0/resources/comMapping/com/foo.xml
```

### Specifying XML Catalogs for OSM
You specify XML Catalogs for an OSM cartridge project in the `cartridgeProject\xmlCatalogs\core` directory (where `cartridgeProject` is the root of the project directory). In this directory, you create your XML Catalog file (you can use any filename such as `core.xml` or `catalog.xml`) and define your catalog entries within it. Design Studio automatically generates a template XML Catalog file `cartridgeProject\xmlCatalogs\core\xmlCatalogCoreTemplate.xml`.

You specify XML Catalogs on the OSM server in the OSM configuration entry `oracle.communications.ordermanagement.util.net.CatalogUriResolver.DefaultXmlCatalogsUris`. By specifying XML Catalog files on the OSM server, you can operationally modify how OSM resolves URIs without changing the contents of a cartridge. See "Using XML Catalogs in OSM" for information on how OSM resolves URIs based on the XML Catalogs you specify on the OSM server.

To specify XML Catalogs on the OSM server:

1. Add or modify a configuration entry for `oracle.communications.ordermanagement.util.net.CatalogUriResolver.DefaultXmlCatalogsUris` in the `oms-config.xml` file.

   See the chapter on configuring OSM with `oms-config.xml` in OSM System Administrator’s Guide for detailed instructions on accessing and modifying the `oms-config.xml` file.

2. Enter the XML Catalog entries you require.
   - Multiple XML Catalogs can be specified separated by a semicolon (`;`).
   - Use any standard XML Catalog entry. The rewriteURI entry is the most commonly used for OSM. See "Defining rewriteURI Entries in XML Catalogs" for information on defining rewriteURI entries.
3. Save the file.

**Enabling and Disabling XML Catalog Support**

XML Catalog support is enabled by default for all cartridges and is required to be enabled.

If your target run-time software version is OSM 7.0.2 or earlier, you can disable XML Catalog support for a cartridge (or re-enable it) by using the cartridge model variable XML_CATALOG_SUPPORT. For information on disabling or re-enabling XML Catalog support for a cartridge, see the Design Studio Help.

**Examples of Using XML Catalogs**

This section provides the following examples of how you can use the XML Catalog:

- Using XML Catalogs to Support Cartridge Versioning
- Using XML Catalogs to Load Resources from a Development File System
- Using XML Catalogs to Insulate Run-Time Environments from Development

**Using XML Catalogs to Support Cartridge Versioning**

Cartridge versioning requires multiple versions of cartridges to reference their own versioned set of resources. For example, if you have version 1.0 and version 2.0 of an OSM cartridge deployed, you might have version-specific XQuery or JAR files that need to be used depending on which cartridge you are using. XML Catalogs ensure that the cartridges reference the correct resources.

To use XML Catalogs to support cartridge versioning:

1. In Design Studio, on the **Model Variables** tab of the Project editor, set the CARTRIDGE_VERSION model variable to the version number of the cartridge.
   
   For more information about model variables, see the Design Studio Help.

2. In the parts of your model where you need OSM to substitute the version number, use \%(CARTRIDGE_VERSION)\%. For example:

   http://company.com/\%(CARTRIDGE_VERSION)\%/xquery/myFile.xqy

3. In the XML Catalog, define the rewriteURI entries as follows:

   - If the cartridge is a component of a composite cartridge, use the CARTRIDGE_VERSION model variable. For example:

   ```xml
   <rewriteURI uriStartString="http://company.com/\%(CARTRIDGE_VERSION)"
   rewritePrefix="osmmodel:///MyCartridge-Resources/\%(CARTRIDGE_VERSION)/resources"/>
   ```

   **Important:** The XML Catalog entries you specify are applied system wide. Ensure that resources are uniquely identifiable to a single catalog entry so that the correct resource can be located.

   **Note:** This configuration defines the XML Catalog entries inline in the oms-config.xml configuration file.
For more information about component and composite cartridges, see "About Cartridges".

- If the cartridge is not a component of a composite cartridge, use the specific cartridge version number. Do not use a model variable. For example:

  <rewriteURI uriStartString="http://company.com/1.0" rewritePrefix="osmmodel:///MyCartridge-Resources/1.0/resources"/>

  When you deploy the cartridge, OSM replaces all instances of `%(CARTRIDGE_VERSION)` with the value that you set on the Project editor Model Variables tab.

4. When you create a new version of a cartridge, update the CARTRIDGE_VERSION model variable with the new version number.

5. In the XML Catalog, update the rewriteURI entries as follows:

- If the cartridge is a component of a composite cartridge, no further updates are required. Because you used the model variable in the rewriteURI entries, OSM automatically replaces the model variable with the new version number when you deploy the cartridge.

- If the cartridge is not a component of a composite cartridge, update the cartridge version number in each rewriteURI. For example:

  <rewriteURI uriStartString="http://company.com/1.5" rewritePrefix="osmmodel:///MyCartridge-Resources/1.5/resources"/>

Using XML Catalogs to Load Resources from a Development File System

To shorten development cycle times that involve numerous coding, building, deployment, and test cycles, you can use the XML Catalog to load resources from a development file system. By using the XML Catalog in this way, you can test changes to resources located within the cartridge without needing to rebuild and repack the cartridge. Rebuilding and repackaging can be slow and CPU intensive because Design Studio needs to rebuild the deployment EAR file before any changes can be tested. By redirecting the URIs to a local resource, you can change XQuery, XSLT, XML, or Java code and immediately test the changes without having to rebuild, repack, and redeploy (Java code would still need to be rebuilt but not repackaged and redeployed).

For example, use the XML Catalog to instruct OSM to load resources:

- From the development file system during development

- From the cartridge PAR file after testing

Locate resources on the file system instead of from within the cartridge PAR file so that configuration changes made to a resource are picked up by the run-time environment without having to rebuild and redeploy the cartridge. After testing is complete, the URI is redirected to load resources from the cartridge PAR file.

To redirect the URI so that OSM loads resources from the development file system:

1. In the Package Explorer view in Design Studio, navigate to the cartridgeProject/xmlCatalogs/core/ directory.

2. Create or edit the catalog.xml file. You can create the file by renaming a copy of xmlCatalogCoreTemplate.xml.

3. Create the XML Catalog entry:

   <rewriteURI uriStartString="http://example.org/somewhere*" rewritePrefix="file:///localhost/dev/env1/mycartridge/resources"/>
OSM loads all resources that start with http://example.org/somewhere from the file system located on localhost at /dev/env1/mycartridge/resources.

To redirect the URI so that OSM loads resources from the cartridge PAR file after testing is complete, change the preceding configuration to:

```xml
<rewriteURI uriStartString="http://example.org/somewhere"
    rewritePrefix="osmmodel://MyCartridge/1.0.0/resources"/>
```

OSM loads all resources that start with http://example.org/somewhere from the cartridge PAR file.

The XML Catalog supports resource extensibility in a cartridge solution because URIs can be easily rewritten to change the location from which resources are loaded. The XML Catalog allows you to redirect the cartridge solution to use customized resources different from the ones that were originally provided by the cartridge solution.

**Using XML Catalogs to Insulate Run-Time Environments from Development**

To insulate test and production environments from development-specific environments, you can use the XML Catalog. When you develop your code, you can set your XML Catalog to point to local resources on your file system on your laptop. Assume you have an automated test environment that runs daily tests on certain cartridges that use resources on the testing box. In production, you would use the XML Catalog to point resources to your production machines. Note that in this example the resources are not bundled inside of the cartridges.
This chapter describes Oracle Communications Order and Service Management (OSM) Web Services, which provides the primary interface for in-bound order operations such as creating or canceling an order.

About Web Services

Web services support interoperable machine-to-machine interaction over a network. Web services are Web APIs that can be accessed over a network, such as the Internet, and run on a remote system hosting the requested services, as is the case with OSM. Web service interfaces are described by the Web service definition language (WSDL). WSDL is an XML-based language that is used in combination with simple object access protocol (SOAP) and XML Schema to provide Web services over the Internet. A client program connecting to a Web service can read the WSDL to determine what operations are available on the server. Any special data types used are embedded in the WSDL file in the form of XML Schema. The client can then use SOAP to actually call one of the operations defined in the WSDL.

About OSM Web Services

The OSM Web Services provide the primary interface for in-bound order operations such as creating, updating, or canceling an order. Web Services are typically initiated from Customer Relationship Management (CRM) systems and other order sources that need to create and manage orders in OSM. OSM Web Services use the SOAP standard.

The OSM Web Service operations are defined in WSDL files. The operations are listed below, and grouped by WSDL file.

OrderManagement.wsdl

- CreateOrder
- CreateOrderBySpecification
- FindOrder
- GetOrder
- UpdateOrder
- SuspendOrder
- ResumeOrder
- CancelOrder
- AbortOrder
About OSM Web Services

- FailOrder
- ResolveFailure

OrderManagementDiag.wsdl
- GetOrderProcessHistory
- GetOrderCompensations
- GetCompensationPlan

These services can be accessed using HTTP, HTTPS, or JMS as the transport protocol. JMS is a reliable, asynchronous messaging transport with guaranteed delivery while HTTP is synchronous and less reliable.

Request Validations

All Web Service requests are validated by the server based on the rules defined in the schema files. If a validation error is encountered, the server returns a fault message detailing the validation error so it can be resolved.

Accessing the WSDL Files

OSM Web Services are part of the OSM installation. The OSM WSDL files and supporting schema files (XSD files) are located in the OSM_home/SDK/WebService/wsdl directory.

Alternatively, you can access the OSM WSDL by entering the following in your Web browser after you have installed, configured, and deployed the OSM server:

http://server:port/OrderManagement/wsapi for Web service operations used for order management.

and


where server is the specific server on which the application is deployed and port is the port on which the application listens. Users who access the WSDL this way must be configured in the WebLogic console with usernames and passwords and must belong to the group OMS_ws_api.

The syntax of each OSM Web service operation is specified using the XML schema, which is associated with the WSDL for the Web service, and is the same for HTTP, HTTPS, and JMS port types. The JNDI name for the JMS request queue is available in the WSDL file.

Using the SOAP Standard Message Format

OSM Web Services use the SOAP standard message format, which includes a header and a body.

Message Header

OSM Web Services require that security related information be provided in the message header. The user name and password for the Web Service authorized user must be included in each request using the elements <wsse:UserName> and <wsse:PasswordText>, as shown in Example 4–1.
Example 4–1 Message Header

```xml
<soapenv:Header>
  <wsse:Security soapenv:mustUnderstand="1" xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
    <wsse:UsernameToken wsu:Id="UsernameToken-4799946" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
      <wsse:Username>administrator</wsse:Username>
      <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#PasswordText">administrator</wsse:Password>
    </wsse:UsernameToken>
  </wsse:Security>
</soapenv:Header>
```

Message Body

The message body contains the data payload. The data payload varies depending on the specific request, as shown in Example 4–2.

Example 4–2 Message Body

```xml
<soapenv:Body>
  <createOrderBySpecification>
    <specification>.
      .
    </specification>
  </createOrderBySpecification>
</soapenv:Body>
```

Response messages include a data payload containing the result of the method call.

Testing

Test OSM Web Services with software such as SoapUI or HermesJMS. Information on such open source test software is available on the internet.

Note: With OSM 7.2, the context-root for OSM applications changed to /OrderManagement. OSM redirects requests specifying the old URIs to the current ones. However, soapUI 2.5.1 does not correctly handle redirects. soapUI3.x or above correctly handles redirects.

Note: If you are using soapUI for testing in a clustered WebLogic environment, enable preemptive authentication in soapUI by selecting Preferences, then HTTP Settings, then Authenticate Preemptively.

Without this, soapUI sends requests without authentication. The request is rejected and then resent with authentication. Because of OSM's load balancing approach in a clustered WebLogic environment, the second request is sent to a different managed server, distorting load balancing. For example, if a cluster has only two managed servers and you employ round-robin load balancing, all authenticated requests will be sent to the same managed server.
Regardless of the software used to test OSM Web Services, you must ensure the clocks are synchronized between the test client and the server hosting the Web services. The synchronization can be done manually, or by using Network Time Protocol (NTP). The following errors are encountered if the clocks are not synchronized:

- Failing to submit order to server_name server from my local system.
- Security token failed to validate. weblogic.xml.crypto.wss.
  SecurityTokenValidateResult@11f081b[status false][msg UNT Error:Message
  Created time past the current time even accounting for set clock skew.

---

**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.

---

**Order States and Transitions**

Several of the OSM Web Service operations initiate a transition from one order state to another. For example, CancelOrder initiates a transition from either an in progress or suspended order state to the cancelling order state. Any transition that occurs within a Web Service operation is described in the Expected Outcome section for that particular operation as described in “About OSM Web Service Operations”. To learn more about order states and their transitions, see OSM Concepts.

**Web Services Sample**

Your OSM installation provides a Web Service sample that demonstrates how OSM web services are called. The sample is available in the OSM_home/SDK/Samples/Web Services directory. The sample includes both HTTP and JMS clients, and demonstrates the use of the web service operations:

- CreateOrderBySpecification
- GetOrder
- UpdateOrder

The GetOrder and UpdateOrder operations depend on the order ID that is provided in the CreateOrderBySpecification response. Before you can run the sample, you must configure it to reflect your environment. See the ReadMe.txt file for detailed instructions on configuring, building, and running the sample.

**About OSM Web Service Operations**

The remaining sections of this chapter describes each Web Service operation, and includes the following information per operation:

- **Preconditions:** Describes any conditions that must exist prior to calling the request.
- **Expected Outcome:** Describes the expected outcome that occurs as a result of the request.
Parameters

The parameters defined by each Web Service are not provided in this documentation because the information is available in the XSD files provided with your OSM installation. For information on determining the input and output parameters for any given Web Service, see “Navigating WSDL and XSD Files”.

Fault Types

The possible fault types that each Web Service may throw is not provided in this documentation because the information is available in the WSDL files provided with your OSM installation. For information on determining the fault types that any given Web Service may throw, see “Navigating WSDL and XSD Files”.

Request and Response Examples

Request and response examples for each Web Service is not provided in this documentation. However, several request and response examples are provided, which you can parlay to other Web Services. See "Request and Response Examples", which also provides information on how to generate XML examples for any given Web Service operation.
Web Service Operations Used for Order Management

This section describes Web service operations used for order management. This includes creating, retrieving, updating and cancelling an order. Order management operations are defined in the `OrderManagementWS.wsdl` file.

Each operation lists preconditions that must exist for a successful invocation of the Web Service operation. However, the following preconditions are common to all operations, so they are listed here rather than repeated for each operation:

- Web Service calls are authenticated by the server based on the user ID and password provided in the request header. Only requests that pass authentication are processed by the server.
- API users must belong to the WebLogic group, OMS_ws_api.
CreateOrderBySpecification

This operation creates a service order.

Preconditions

- The order specification referenced on the request is defined in the metadata and has been deployed to the target OSM server.
- The content of the order detail that is provided on the request must conform to the order specification referenced on the request.
- The user performing the transaction is a member of at least one workgroup that has been granted permission on the creation task for the referenced order specification.

Expected Outcome

The order is created and processing begins. If the newly created order is matched against an existing order (based on the key defined on the order's specification), then this new order is an amendment to an existing order, and information regarding the amended order and status of the amendment is returned.

If the newly created order is not an amendment, the order is transitioned to the open.running.in_progress state by specifying StartOrder=true.

Alternate Outcome with Start Order Set to False

The order is created but processing does not begin. The order is in the open.not_running.not_started state. The order can be further updated and started through the UpdateOrder operation.

Attachments

You can add attachments through the createOrderBySpecification operation. Attachments are added by populating the Remark element, which provides a place to define a text remark as well as an attachment. The attachment is added by populating the Attachment element, which is a child element of Remark. Within the Attachment element, you can define a sequence of file names and their corresponding file types. For additional information, see the OrderManagementWS.xsd file, which defines these elements.

Reference Nodes

Reference nodes are pointers to values contained in different data nodes, and they enable you to create information once and reuse it in multiple locations in your data model. You set up reference nodes at order creation time.

To set up reference nodes in an order, when creating the order, you must explicitly give the referred-to field an index, and then refer to it with [#] in the reference. For an example that demonstrates how to set up reference nodes at order creation time as part of coding the automation plug-ins that call the CreateOrderBySpecification Web service operation, see "Request Example - Setting Up Reference Nodes".
CreateOrder

This operation creates a new order.

Preconditions

- The content of the order detail that is provided on the request must conform to a defined recognition rule.
- The user performing the transaction is a member of at least one workgroup that has been granted permission on the creation task for the order.

Expected Outcome

The order is created and processing begins. The order is transitioned to the open.running.in_progress state by specifying StartOrder=true.
**FindOrder**

This operation finds a set of orders that match all the conditions defined in the select clause. The `SelectBy` element specifies which orders will be returned.

---

**Note:** If you choose to specify the name of the cartridge in the `SelectBy` element of the request and you do not specify the cartridge version, only orders from the default version will be returned. If you wish to retrieve orders from all of the versions of the specified cartridge, include “*” as the cartridge version.

---

Results can contain a combination of flexible headers and task data. The calling user must belong to a role with permissions to view the order. If the user does not have the permission, no data is returned.

Flexible Headers are user-defined columns which are displayed while viewing order details. Flexible headers are set by OSM administrators. Generally the path of a flexible header is `/<WebService>/<ElementGroup>/<FlexibleHeader>`. Note that `/<WebService>` is preceded by a single slash (/). A double slash (//) or no slash will yield different results. See *OSM XML API Developer’s Guide* for details on how to query and retrieve orders that include available flexible headers using the XML API.

**Preconditions**

- The order being retrieved must exist. If the order does not exist, FindOrder returns an empty set.

**Expected Outcome**

Order data that meets specified selection criteria is returned in the specified sequence and is viewed through the specified filter.
GetOrder

This operation retrieves an order. A summary of the order is returned, along with the detailed order data based on a specified order view template. See also "GetOrder Examples".

Parameters

OrderId
The identification of the order to be retrieved.

View
The name of the view (query task) used to determine the order data that is returned. You must associate the task data you want to return to a role in the Oracle Communications Design Studio Order editor Permissions Query Task sub tab and set a query task with the data to be returned as the Default query task.

The following parameters are optional:

RemarkFilter
Controls how remarks and attachments are returned.

RetrieveRemarks
Set to true if remarks and associated attachments should be returned.

AttachmentFilter
If RetrieveRemarks is set to true, zero or more filters (FileNameMatch, MinSize, MaxSize, Format) may control how attachments are returned. Attachment filters are processed in the order they are provided. If no filters are provided, then no attachments are returned.

OrderDataFilter
Parent element for the Condition child element that specifies which order data to return in the OrderUpdate response message specified in the View. OSM returns only the multi-instance data specified in the condition. This filtering functionality improves OSM performance, especially when requesting multi-instance data in large orders from the OSM server.

- **Condition**: An XPath 1.0 expression against the order data defined by the ResponseView. OSM only returns order data selected by the expression but not the sibling nodes. For example, in a situation where a customer has multiple addresses instances (where address is a multi-instance element), the following expression ensures that OSM returns only the address element that contains a child street element with the specified street address. The response includes all child nodes of the address element (city, postal code, and street). The sibling instances of the address element and their child elements (city, street, and postal code) are not returned.

    <ord:OrderDataFilter>
      <ord:Condition>/subscriber_info/address/[street='190 Drive']</ord:Condition>
    </ord:OrderDataFilter>

    All other parent or sibling elements would be returned. For example any sibling elements of subscriber_info, or sibling elements of address (except for the multiple instances of the sibling address element).
Preconditions

- The specified order exists.
- The user performing the transaction is a member of one or more workgroups that has been assigned the specified view for the order definition in question.

Expected Outcome

The order summary and detail are returned. If the order contains any remarks or attachments, they are returned based on the filters set on the request.
**UpdateOrder**

This operation allows order data to be updated, and allows orders that have been created but not started (in the `open.not_running.not_started` state) to be started. See also "UpdateOrder Examples".

The updateOrder operation defines different ways to update the order:

- **UpdatedOrder**: Provides the ability to update the order by supplying a complete order. The existing order is updated (elements added, changed, or deleted) to match the supplied order.

- **UpdatedNodes**: Provides the ability to update the order by supplying only the nodes to be updated (elements added or changed). Deletes are not performed using UpdatedNodes. The nodes are supplied in the format of the existing order.

- **DataChange**: Provides the ability to update the order by supplying a series of add, update, and delete elements that are used to manipulate the order.

---

**Note**: If you update an order either to add a node (which includes providing a value to a node that did not previously have one) or to delete a node (which includes setting the value of a node to null), the OSM order transformation manager will not propagate the change in either the forward or reverse direction. For more information about data propagation, see the discussion of mapping rules in the Design Studio Modeling OSM Orchestration Help.

---

You can specify and filter which data to return in response to the UpdateOrder requests:

- **ResponseView**: An optional parameter that defines the name of the view that specifies what parameters are returned in UpdateOrder responses. If the UpdateOrder request results in a fulfillment state update, the response auto-filters nodes to only include the affected OrderItem and OrderComponent instances.

- **OrderDataFilter**: Parent element for the **Condition** child element that specifies which order data to return in the OrderUpdate response message specified in the **ResponseView**.

- **Condition**: An XPath 1.0 expression against the order data defined by the ResponseView. OSM only returns order data selected by the expression but not the sibling nodes. For example, in a situation where a customer has multiple addresses instances (where address is a multi-instance element), the following expression ensures that OSM returns only the address element that contains a child street element with the specified street address. The response includes all child nodes of the address element (city, postal code, and street). The sibling instances of the address element and their child elements (city, street, and postal code) are not returned.

  ```xml
  <ord:OrderDataFilter>
    <ord:Condition>/subscriber_info/address/[street='190 Drive']</ord:Condition>
  </ord:OrderDataFilter>
  
  All other parent or sibling elements would be returned. For example any sibling elements of subscriber_info, or sibling elements of address (except for the multiple instances of the sibling address element).```
In addition, you can directly specify order fulfillment using the **ExternalFulfillmentStates** element rather than do so with Add or Update statement on an UpdateOrder. This optional approach improves order processing efficiency, especially in large orders. The **ExternalFulfillmentStates** element has the following child elements:

- **OrderItemOrderComponentFulfillmentState**: The parent element to the children elements that specify the new external fulfillment state of an order component and order item.
  - **ExternalFulfillmentState**: The new external fulfillment state.
  - **OrderComponentIndex**: The order component index. Every order component element must specify a unique index attribute. In most cases, the automation running the XML API OrderUpdate already knows which order component the update is for.
  - **OrderItemIndex**: The order item index. Every order item element must specify a unique index attribute. In most cases, the automation running the XML API OrderUpdate already knows which order component the update is for.

For samples of updateOrder, see OSM_home/SDK/Samples/WebService. You must use the OSM installer to install the SDK sample files.

**Preconditions**

- The user performing the transaction is a member of at least one workgroup (role) that has been granted permission on the creation task (view) for the order specification associated with the order.
- The order is in the open.not_running.not_started state.

**Note:** These preconditions apply if the order is in the not_started state. You can update the order data when the order is running, if the order life-cycle policy permissions allow it for the task you want to update.

You must associate the task data you want to update to a role in the Design Studio Order editor Permissions Query Task sub tab and set a query task with the required data as the Default query task. You can associate only one role per task in the Order editor. The user specified in the UpdateOrder header must be a member of this role.

**Expected Outcome**

The order’s data is updated successfully but remains in the open.not_running.not_started state. The order can be further updated or started by additional calls to the UpdateOrder operation.

**Attachments**

You can add attachments through the updateOrder operation. Attachments are added by populating the **Remark** element, which provides a place to define a text remark as well as an attachment. The attachment is added by populating the **Attachment** element, which is a child element of **Remark**. Within the **Attachment** element, you can define a sequence of file names and their corresponding file types. For additional information, see the OrderManagementWS.xsd file, which defines these elements.
SuspendOrder

This operation suspends an order thereby preventing work items associated with the order from being processed. A suspended order must be resumed before its associated work items can once again be processed.

Preconditions

- The current state of the specified order is open.running.in_progress or open.not_running.not_started.
- The target state of the order is not set.
- The order life-cycle policy associated with the order's specification has the Suspend Order transaction enabled from the open.running.in_progress state or from the not_started state.
- The user performing the transaction is a member of one or more of the workgroups associated with the Suspend Order transaction referenced in the precondition.

Expected Outcome

The order is successfully transitioned to the open.not_running.suspended state. Users are restricted from processing work items associated with the suspended order.

Alternate Outcome with Grace Period

The order enters into a grace period that allows all work items that are currently accepted to be processed. During the grace period, the current order state remains open.running.in_progress and the target state is set to open.not_running.suspended. The order will complete the transition to the open.not_running.suspended state when all accepted work items for the order are completed or the grace period expires, whichever comes first. New work items cannot be accepted during the grace period.

The grace period may be configured on the order state policy and/or specified on this call.
ResumeOrder

This operation resumes an order that is currently suspended or cancelled so that work items associated with the order are allowed to be processed.

Preconditions

- The current state of the specified order is either open.not_running.suspended or open.not_running_cancelled.
- The target state of the order is not set.
- The order life-cycle policy associated with the order's specification has the Resume Order transaction enabled from the open.not_running.suspended state or open.not_running.cancelled state.
- The user performing the transaction is a member of one or more of the workgroups associated with the Resume Order transaction referenced in precondition.

Expected Outcome

The order is successfully transitioned to the open.running.in_progress or open.not_running.not_started state. Authorized users may process work items associated with the specified order.
CancelOrder

This operation cancels an order. All outstanding work items associated with the order are deleted, and all complete work items associated with the order are compensated (undone).

Preconditions

- The current state of the specified order is open.running.in_progress or open.not_running.suspended.
- The target state of the order is not set.
- The order life-cycle policy associated with the order's specification has the Cancel Order transaction enabled from the current order state (open.running.in_progress state or open.not_running.suspended).
- The user performing the transaction is a member of one or more of the workgroups associated with the Cancel Order transaction referenced in precondition.

Expected Outcome

The order is successfully transitioned to the open.running.compensating.cancelling state. Incomplete work items associated with the order are deleted. Completed work items associated with the specified order are compensated. Once compensation completes, the order is transitioned to open.not_running.cancelled.

Alternate Outcome with Grace Period

The order enters into a grace period that allows all work items that are currently accepted to be processed. During the grace period, the current order state remains at its current value (open.running.in_progress or open.not_running.suspended) and the target order state is set to open.running.compensating.cancelling. The order will complete the transition to the open.running.compensating.cancelling state when all accepted work items for the order are completed or the grace period expires, whichever comes first. New work items cannot be accepted during the grace period. The grace period may be configured on the order life-cycle policy and/or specified on this call.
AbortOrder

This operation aborts an order, and aborts all work items associated with the order. You can grant permissions for this operation by editing the Abort Order transaction in the order life-cycle policy associated with the order's specification in Design Studio.

Preconditions

- The user performing the operation must be a member of one or more of the workgroups associated with the Abort Order transaction.

Expected Outcome

The order is successfully transitioned to the closed.aborted state. Users are restricted from processing the aborted order.
FailOrder

This operation fails an order. A failure must be resolved before the order can proceed any further. You can grant permissions for this operation by editing the Fail Order transaction in the order life-cycle policy associated with the order’s specification in Design Studio.

Preconditions

- The user performing the operation must be a member of one or more of the workgroups associated with the Fail Order transaction.

Expected Outcome

The order is successfully transitioned to the `open.not_running.failed` state. Users are restricted from processing work items associated with the failed order.

Alternate Outcome With Grace Period

The order enters into a grace period that allows all work items that are currently accepted to be processed. During the grace period, the current order state remains `open.running.in_progress` and the target state is set to `open.not_running.failed`. The order will complete the transition to the `open.not_running.failed` state when all accepted work items for the order are completed or the grace period expires, whichever comes first. New work items cannot be accepted during the grace period. The grace period may be configured on the order state policy or specified on this call.
ResolveFailure

This operation transits an order that is currently failed back to its state prior to entering the current failed state. You can grant permissions for this operation by editing the Manage Order Fallout transaction in the order life-cycle policy associated with the order's specification in Design Studio.

Preconditions

- The current state of the specified order is open.not_running.failed.
- The user performing the operation must be a member of one or more of the workgroups associated with the Manage Order Fallout transaction.

Expected Outcome

The order is successfully transitioned to its previous state.
Web Service Operations Used for Problem Order Diagnosis

This section describes Web service operations used for diagnosing problem orders. This includes getting order process history, compensation history and compensation details. Order diagnosis operations are defined in the OrderManagementDiag.wsdl file.
GetOrderProcessHistory

This operation returns process history perspective of an order. The different kinds of process history perspectives are:

- **Original**: An order that has never been compensated and has only one (the original) process history perspective. For an order that has been compensated, the original process history perspective includes all tasks created before the first compensation for the order has started.

- **Latest**: Includes all tasks that have never been compensated.

- **Identified by compensationID**: A new process history perspective is created for the compensation of each order that has been started. A task must satisfy the following conditions to be included in the process history perspective that is identified by compensation:
  - To be created before any later compensation has started (if any).
  - Not to be compensated in any prior compensation.

When a task is "redo" compensated, the "redo" compensator replaces the task in all subsequent process history perspectives. When a task is "undone," it is not included into any subsequent process history perspectives. Tasks that are compensated in the compensation context that the process history is requested for are included in the response and their compensation details are provided.

Use the GetOrderCompensations operation to retrieve information about order compensations, including their IDs. See "GetOrderCompensations".

**Preconditions**

- The specified order exists.

**Expected Outcome**

The process history perspective for the order is returned.
GetOrderCompensations

This operation retrieves the history of all compensations for an order. For each compensation, the data returned includes the type of compensation, submission date, start date (optional), and completion date (optional).

Preconditions

- The specified order must exist.

- The specified order must be in the `open.running.compensating.amending` or `open.running.compensating.cancelling` state.

Expected Outcome

The order compensation plan information is returned as a set of compensation tasks, along with the compensation dependencies between them.
GetCompensationPlan

This operation retrieves compensation plan details for an order. For each compensation plan, the data returned includes the type of compensation, active compensation task information, pending compensation task information, and the state transition history for compensation tasks.

Preconditions

- The specified order must exist.
- The specified order must be open.running.compensating.amending or open.running.compensating.cancelling.

Expected Outcome

The order compensation plan information is returned.
Navigating WSDL and XSD Files

This section describes how to navigate the WSDL and XSD files to determine the input parameters, responses, and fault types that a given Web Service operation defines. The information is presented through an example that is applicable to all operations.
WSDL File

Example 4–3 is an excerpt from the OrderManagementWS.wsdl file that shows how a typical Web Service operation is defined.

**Example 4–3  WSDL Operation Definition**

```
<wsdl:operation name="CreateOrderBySpecification">
    <wsdl:input message="prov:CreateOrderBySpecificationRequest"/>
    <wsdl:output message="prov:CreateOrderBySpecificationResponse"/>
    <wsdl:input message="prov:CreateOrderBySpecificationRequest"/>
    <wsdl:output message="prov:CreateOrderBySpecificationResponse"/>
    <wsdl:fault name="InvalidOrderSpecificationFault" message="prov:CreateOrder_faultMsg"/>
    <wsdl:input message="prov:CreateOrderBySpecificationRequest"/>
    <wsdl:output message="prov:CreateOrderBySpecificationResponse"/>
    <wsdl:fault name="TransactionNotAllowedFault" message="prov:CreateOrder_faultMsg1"/>
    <wsdl:input message="prov:CreateOrderBySpecificationRequest"/>
    <wsdl:output message="prov:CreateOrderBySpecificationResponse"/>
    <wsdl:fault name="InvalidOrderDataFault" message="prov:CreateOrder_faultMsg3"/>
</wsdl:operation>
```

The WSDL file defines each operation in the same manner, which provides the following information:

- **Operation name**: The name of the Web Service operation.
- **Input message**: The request structure that is defined in the corresponding XSD file.
- **Output message**: The response structure that is defined in the corresponding XSD file.
- **Fault names**: The exception structures that are defined in the corresponding XSD file.

The WSDL file tells you what request goes with what response, and what exceptions the request may throw. Each Web Service operation defines a request and a response, which are the input and output parameters. The request and response structures are defined in the corresponding XSD file. For example, the CreateOrderBySpecification operation is defined in the OrderManagementWS.wsdl file, and the corresponding XSD file is OrderManagementWS.xsd.
XSD File

This section describes how to navigate the XSD files. The request and response structures defined in the XSD are used by the Web Service operations as input and output parameters. This section provides graphics of the XSD in various states of expansion. You can view the XSD using any XML application, such as XMLSpy.

XMLSpy offers several ways to view XML files. XSD files containing large structures can be very difficult to read. The examples provided in this section show how to view XSD files using the Schema/WSDL Design view, which allows you to view the top level structures and then expand and collapse them as needed. Viewing the XML structure in this manner automatically pulls in any referenced structures, removing the need to scroll around to locate them.

**Note:** If you are using an application other than XMLSpy to view XML files, your views of the XSD may differ from the examples used in this section.

Determining Input Parameters (Request)

Figure 4–1 shows a portion of the OrderManagementWS.xsd file in the Schema/WSDL Design view, as it appears when first opened. This is the top level of the view, which lists all simpleType, complexType, and elements that are defined in the file.

**Figure 4–1 Schema/WSDL Design View**

```
| complexType     | GetOrderRequestType |
| complexType     | GetOrderResponseType |
| element         | CreateOrderBySpecification |
| element         | CreateOrderBySpecificationResponse |
| element         | GetOrder |
| element         | GetOrderResponse |
```

From the top level, clicking the grey box located to the left of any element or complexType expands the structure. Continuing with the example, Figure 4–2 shows the result of clicking the grey box located to the left of CreateOrderBySpecification.

**Figure 4–2 Expanded Structure**

From this level, you can see that CreateOrderBySpecification defines CreateOrderBySpecificationRequestType, but you cannot see what CreateOrderBySpecificationRequestType actually defines. Clicking the "+" located within the CreateOrderBySpecificationRequestType structure box expands the structure. Figure 4–3 and Figure 4–4 show the result of this action.
Figure 4–3  Further Expanded Structure

```
<osm:CreateOrderBySpecificationRequestType>
  <Specification>
    <Reference>
      The order reference. The order reference is typically used to store an upstream order ID for tracking purposes, however it can be used for any required purpose.
    </Reference>
    <Priority>
      The priority of the order. If the priority value is outside the valid priority range specified as part of the order definition, then the priority value will be automatically rounded to the nearest valid value.
    </Priority>
    <AutoAddMandatoryData>
      This element controls whether or not the system will automatically add mandatory data that is missing from a request. Missing mandatory data will be added using default values defined in metadata.
    </AutoAddMandatoryData>
    <StartOrder>
      This element controls whether or not the order will be automatically started or not.
    </StartOrder>
  </Specification>
</osm:CreateOrderBySpecificationRequestType>
```
From this level, you can see that CreateOrderBySpecificationRequestType defines:

- Specification
- Reference
- Priority
- AutoAddMandatoryData
- StartOrder
- Data
- Remark

However, you cannot see what the Specification, Data, or Remark structures define. As with the previous level, you can expand any of these structures by clicking the “+” located to the right of the structure name. Clicking the “+” located within the Data and Remark structure box expands the structures. Figure 4–5 shows the result of this action.
Expanding the Specification, Data, and Remark structures shows additional defined structures and fields. In this example, note that the structure defined under the Data structure (OrderDataType any) is a structure that is defined in Design Studio. For example, you may define five different order templates, so the structure under the Data structure varies depending on the order type. The order-specific data in the request is validated by the server through the creation task view.

**Note:** To collapse any of the structures at any level, click "+" located near the structure name. You can also collapse all structures and return to the top level by clicking the collapse button, located in the upper left corner as shown in Figure 4–3, "Further Expanded Structure". The collapse button is only visible in the upper left corner, so you must scroll all the way up and all the way to the left to see it.

**Determining Output Parameters (Response)**

You can expand the response structure defined for an operation. Figure 4–6 shows the top level of the OrderManagementWS.xsd file in Schema/WSDL Design view.
Continuing with our example, expand CreateOrderBySpecificationResponse to
determine the expected response.

**Figure 4–6 Schema/WSDL Design View**

```
<complexType name="GetOrderRequestType">
  <element name="CreateOrderBySpecification"/>
  <element name="CreateOrderBySpecificationResponse"/>
  <element name="GetOrder"/>
  <element name="GetOrderResponse"/>
</complexType>
```

Figure 4–7 shows the expected response defined by
CreateOrderBySpecificationResponse, which can be expanded even further.

**Figure 4–7 Expanded Structure**

Determining Fault Types

You can expand the fault names defined for the operation. Continuing with the
CreateOrderBySpecification example, InvalidOrderSpecificationFault,
TransactionNotAllowedFault, and InvalidOrderDataFault are all defined as top level
structures in the *OrderManagementWS.xsd* file.
Request and Response Examples

This section provides sample XML requests and sample XML responses. Sample XML for any Web Service operation can be generated from the XSD using any XML application such as XMLSpy.

To generate a sample XML file using XMLSpy:

1. Open an XSD file in XMLSpy.
2. From the menu, select DTD/Schema, then select Generate Sample XML File.
   - The Select a Root Element dialogue box opens, which lists all root elements defined in the XSD, such as CreateOrder, CreateOrderResponse, FindOrder, FindOrderResponse, and so on.
3. Select a root element and click OK.
   - The Generate Sample XML File dialogue box appears, which provides a few selection options such as generating non-mandatory elements and attributes, the number of structures to generate for structures that are defined as a sequence, and whether or not to populate the XML with data.
4. Choose the appropriate options and click OK.
   - The generated XML displays within a new file, Untitled.xml.

Generating XML in this manner does not generate the SOAP header and body. However, the SOAP header and body can be manually inserted into the generated XML.
CreateOrderBySpecification Examples

This section provides a request example and a response example for the CreateOrderBySpecification operation.

Request Example

Example 4–4 CreateOrderBySpecificationRequest

```xml
<?xml version = '1.0' encoding = 'UTF-8'?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:ws="http://xmlns.oracle.com/communications/ordermanagement">
 <soapenv:Header>
  <wsse:Security soapenv:mustUnderstand="1"
   xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-sec
text-1.0.xsd">
   <wsse:UsernameToken wsu:Id="UsernameToken-4799946"
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utili
ty-1.0.xsd">
    <wsse:Username>administrator</wsse:Username>
    <wsse:Password
      Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profi
le-1.0#PasswordText">administrator</wsse:Password>
   </wsse:UsernameToken>
  </wsse:Security>
 </soapenv:Header>
 <soapenv:Body>
  <ws:CreateOrderBySpecification>
   <ws:Specification>
    <ws:Cartridge>
     <ws:Name>view_framework_demo</ws:Name>
     <ws:Version>1.0</ws:Version>
    </ws:Cartridge>
    <ws:Type>vf_demo</ws:Type>
    <ws:Source>web</ws:Source>
   </ws:Specification>
   <ws:Reference>test message</ws:Reference>
   <ws:Priority>5</ws:Priority>
   <ws:AutoAddMandatoryData>true</ws:AutoAddMandatoryData>
   <ws:StartOrder>true</ws:StartOrder>
   <ws:Data>
    <_root>
     <account_information>
      <amount_owing>553</amount_owing>
     </account_information>
    </_root>
   </ws:Data>
  </ws:CreateOrderBySpecification>
 </soapenv:Body>
</soapenv:Envelope>
```

4-32 OSM Developer's Guide
Request and Response Examples

Response Example

Example 4–5  CreateOrderBySpecificationResponse

Request Example - Setting Up Reference Nodes

This example demonstrates how to set up reference nodes in the task data of the creation task when you code the CreateOrderBySpecification call in your XQuery or XSLT or Java automation plug-ins.
When creating the order you must explicitly give the referred-to field an index, and then refer to it with {#} in the reference. You assign the index and code it when you write your automation plug-in code (XQuery/XSLT/Java code).

In this example, `<LineItem index="1">` is the index value you defined to this LineItem instance in your automation plug-in code. The index value must be unique within this CreateOrderBySpecification order data; this allows you to refer to this instance later as `<LineItem_refNode>{1}</LineItem_refNode>` to point to a single data node location in the order template at order creation time.

**Example 4–6 CreateOrderBySpecificationRequest - Setting Up Reference Nodes**

```xml
<ord:CreateOrderBySpecificationRequest>
  <ord:Specification>
    <ord:Cartridge>
      <ord:Name>ReferenceDebug</ord:Name>
      <ord:Version>1.0.0</ord:Version>
    </ord:Cartridge>
    <ord:Type>ReferenceDebugOrder</ord:Type>
    <ord:Source>ReferenceDebugOrder</ord:Source>
    <ord:View>ReferenceDebugCreationTask</ord:View>
  </ord:Specification>
  <ord:Reference>created from SoapUI</ord:Reference>
  <ord:Priority>5</ord:Priority>
  <ord:AutoAddMandatoryData>false</ord:AutoAddMandatoryData>
  <ord:StartOrder>false</ord:StartOrder>
  <!--Optional:-->
  <ord:Data>
    <_root>
      <Data>
        <LineItem index="1">
          <ID>1</ID>
        </LineItem>
        <LineItem index="2">
          <ID>2</ID>
        </LineItem>
      </Data>
      <References>
        <LineItem_refNode>{1}</LineItem_refNode>
      </References>
    </_root>
  </ord:Data>
</ord:CreateOrderBySpecificationRequest>
```

**Note:** A creation task is selected for an order in the Order Editor Details tab of Design Studio. In this example, the name of the creation task (for which the CreateOrderBySpecification call requires the order data) is in the parameter `<ord:View>ReferenceDebugCreationTask</ord:View>`. 

When creating the order you must explicitly give the referred-to field an index, and then refer to it with {#} in the reference. You assign the index and code it when you write your automation plug-in code (XQuery/XSLT/Java code).

In this example, `<LineItem index="1">` is the index value you defined to this LineItem instance in your automation plug-in code. The index value must be unique within this CreateOrderBySpecification order data; this allows you to refer to this instance later as `<LineItem_refNode>{1}</LineItem_refNode>` to point to a single data node location in the order template at order creation time.
GetOrder Examples

This section provides a request example and a response example for the GetOrder operation.

Request and Response Example

Example 4–7 shows a GetOrderRequest that specifies that the data defined by the demo_query query task be returned in the GetOrderResponse from the order with order ID 9.

**Example 4–7 GetOrderRequest**

```xml
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/
xmlns:ord="http://xmlns.oracle.com/communications/ordermanagement"
xmlns:ws="http://xmlns.oracle.com/communications/ordermanagement">
  <soapenv:Header>
    <wsse:Security soapenv:mustUnderstand="1"
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-sec
text-1.0.xsd">
      <wsse:UsernameToken wsu:Id="UsernameToken-4799946"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utili
ty-1.0.xsd">
        <wsse:Username>User</wsse:Username>
        <wsse:Password
Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profi
gle-1.0#PasswordText">password</wsse:Password>
      </wsse:UsernameToken>
    </wsse:Security>
  </soapenv:Header>
  <soapenv:Body>
    <ord:GetOrder>
      <ord:OrderId>9</ord:OrderId>
      <ord:View>demo_query</ord:View>
    </ord:GetOrder>
  </soapenv:Body>
</soapenv:Envelope>
```

Example 4–8 shows the GetOrderResponse returned for the GetOrderRequest in Example 4–7.

**Example 4–8 GetOrderResponse**

```xml
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Body>
    <GetOrderResponse xmlns="http://xmlns.oracle.com/communications/ordermanagement">
      <OrderSummary>
        <Id>9</Id>
        <Specification>
          <Cartridge>
            <Name>bb_ocm_demo</Name>
            <Version>1.0.0.0</Version>
          </Cartridge>
          <Type>add_adsl_siebel</Type>
          <Source>add_adsl_siebel</Source>
        </Specification>
        <State>open.running.in_progress</State>
      </OrderSummary>
    </GetOrderResponse>
  </env:Body>
</env:Envelope>
```
Request and Response Example with OrderDataFilter

Example 4–9 shows a GetOrderRequest that specifies that the data defined by the demo_query query task be returned in the GetOrderResponse from the order with order ID 9. The GetOrderRequest also includes an OrderDataFilter that specifies that only the address instance with a corresponding street value of “190 Attwell Drive” should return in the GetOrderResponse.

Example 4–9 GetOrderRequest with OrderDataFilter

```xml
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/
xmlns:ord="http://xmlns.oracle.com/communications/ordermanagement"
xmlns:ws="http://xmlns.oracle.com/communications/ordermanagement">
  <soapenv:Header>
    <wsse:Security soapenv:mustUnderstand="1"
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-sec
text-1.0.xsd">
      <wsse:UsernameToken wsu:Id="UsernameToken-4799946"/>
    </wsse:Security>
  </soapenv:Header>
  <soapenv:Body>
    ...GetOrderRequest content...
  </soapenv:Body>
</soapenv:Envelope>
```
Example 4–10 shows the GetOrderResponse returned for the GetOrderRequest in Example 4–9.

**Example 4–10 GetOrderResponse with OrderDataFilter**

```xml
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Header/>
  <env:Body>
    <GetOrderResponse xmlns="http://xmlns.oracle.com/communications/ordermanagement">
      <OrderSummary>
        <Id>9</Id>
        <Specification>
          <Cartridge>
            <Name>bb_ocm_demo</Name>
            <Version>1.0.0.0.0</Version>
          </Cartridge>
          <Type>add_adsl_siebel</Type>
          <Source>add_adsl_siebel</Source>
        </Specification>
        <State>open.running.in_progress</State>
        <Reference/>
        <CreatedDate>2014-10-30T08:24:15.000-07:00</CreatedDate>
        <ExpectedDuration>P1D</ExpectedDuration>
        <ExpectedOrderCompletionDate>2014-10-31T08:24:26.000-07:00</ExpectedOrderCompletionDate>
        <ProcessStatus>n/a</ProcessStatus>
        <Priority>5</Priority>
      </OrderSummary>
      <Data>
        <osmc:_root index="0" xmlns:osmc="urn:oracle:names:ordermanagement:cartridge:bb_ocm_demo:1.0.0.0.0:view:demo_query">
          <osmc:subscriber_info index="1414682666683">
              <osmc:address index="1414682666684">
                  <osmc:street index="1414682666685">190 Attwell Drive</osmc:street>
              </osmc:address>
          </osmc:subscriber_info>
        </osmc:_root>
      </Data>
    </GetOrderResponse>
  </env:Body>
</env:Envelope>
```
Response Example - Order with Distributed Order Template Elements and Transformed Order Items

Example 4–11 Partial GetOrderResponse containing Distributed Order Template Data

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Body>
    <GetOrderResponse xmlns="http://xmlns.oracle.com/communications/ordermanagement">
      <OrderSummary>
        <Id>20</Id>
        <Specification>
          <Cartridge>
            <Name>OsmCentralOMExample-Solution</Name>
            <Version>4.0.0.0</Version>
          </Cartridge>
          <Type>OsmCentralOMExampleOrder</Type>
          <Source>OsmCentralOMExampleOrder</Source>
        </Specification>
        <State>open.running.in_progress</State>
        <Reference>Order1397235767310</Reference>
        <CreatedDate>2014-04-11T10:03:28.000-07:00</CreatedDate>
        <RequestedDeliveryDate>2014-03-31T07:05:00.000-07:00</RequestedDeliveryDate>
        <ExpectedStartDate>2014-04-11T03:28.495-07:00</ExpectedStartDate>
        <ExpectedDuration>PT0S</ExpectedDuration>
        <ExpectedOrderCompletionDate>2014-04-11T03:28.495-07:00</ExpectedOrderCompletionDate>
        <Priority>5</Priority>
      </OrderSummary>
      <Data>
        <osmc:_root xmlns:osmc="urn:oracle:names:ordermanagement:cartridge:OsmCentralOMExample-Solution:4.0.0.0:view:OsmCentralOMExampleQueryTask">
          <osmc:OrderHeader>
            <osmc:numSalesOrder>Order1397235767310</osmc:numSalesOrder>
            <osmc:typeOrder>Add</osmc:typeOrder>
          </osmc:OrderHeader>
          <osmc:CustomerDetails>
            <osmc:nameLocation>Location1</osmc:nameLocation>
            <osmc:typeAddress>Building</osmc:typeAddress>
          </osmc:CustomerDetails>
          <osmc:AccountDetails>
            <osmc:numAccount>TEL1234</osmc:numAccount>
          </osmc:AccountDetails>
        </osmc:_root>
      </Data>
    </GetOrderResponse>
  </env:Body>
</env:Envelope>
```
<osmc:status index="26">Existing</osmc:status>
<osmc:corporate index="27">PoC</osmc:corporate>
<osmc:clientSince index="31">1986-12-31-08:00</osmc:clientSince>
<osmc:category index="32">Corporate</osmc:category>
</osmc:AccountDetails>
<osmc:ControlData index="1397235812801">
<osmc:Functions index="1397235812888">
<osmc:ProvisioningFunction index="1397235811141">
<osmc:transformedOrderItem index="1397235812898">
<osmc:orderItemRef xsi:type="ct160:TransformedOrderLineType" type="&lt;http://www.oracle.com/otm/cso&gt;TransformedOrderLineType" index="1397235812899" referencedIndex="1397235811129"
xmlns:ct160="http://www.oracle.com/otm/cso"
xlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<ct135:LineId index="1397235812853"
<ct160:dynamicParams xsi:type="ct264:SA_Provision_BroadbandInternetType" type="&lt;OracleComms_Model_BroadbandInternet/4.0.0.0.0&gt;SA_Provision_BroadbandInternetType" index="1397235812848" xmlns:ct264="OracleComms_Model_BroadbandInternet/4.0.0.0.0">
<ct264:DownloadSpeed index="1397235812851">50</ct264:DownloadSpeed>
<ct264:QoS index="1397235812852">Data</ct264:QoS>
<ct264:UploadSpeed index="1397235812850">3.072</ct264:UploadSpeed>
</ct160:dynamicParams>
<ct160:Recognition index="1397235812846" xsi:type="ct264:SA_Provision_BroadbandInternetType" type="&lt;OracleComms_Model_BroadbandInternet/4.0.0.0.0&gt;SA_Provision_BroadbandInternetType" index="1397235812848" xmlns:ct264="OracleComms_Model_BroadbandInternet/4.0.0.0.0">
<ct160:LineName index="1397235812847">
SA_Provision_BroadbandInternetSpec [Add]
</ct160:LineName>
<ct160:FulfillmentPattern index="1397235812855">
</ct160:FulfillmentPattern>
<ct160:Action index="1397235812854">Add</ct160:Action>
</osmc:orderItemRef>
</osmc:transformedOrderItem>
</osmc:Functions>
</osmc:ControlData>
<osmc:Components index="1397235812889">
<osmc:ProvisioningFunction index="1397235811200">
<osmc:orderItem index="1397235812892">
<osmc:orderItemRef xsi:type="ct211:CustomerOrderItemSpecificationType" type="&lt;http://oracle.communications.ordermanagement.unsupported.centralom&gt;CustomerOrderItemSpecificationType" index="1397235812893" referencedIndex="1397235811126"
xlns:ct211="http://oracle.communications.ordermanagement.unsupported.centralom"
xlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<ct211:productSpec index="1397235812811">Broadband Service Feature Class</ct211:productSpec>
<ct211:fulfPatt index="1397235812802">Service.Broadband</ct211:fulfPatt>
<ct211:lineId index="1397235812809">1</ct211:lineId>
<ct211:lineItemName index="1397235812808">Brilliant Broadband [Add]</ct211:lineItemName>
<ct211:requestedDeliveryDate index="1397235812806">2014-03-31T07:05:00-07:00</ct211:requestedDeliveryDate>
</osmc:orderItem>
</osmc:ProvisioningFunction>
</osmc:Components>
<ct211:requestedDeliveryDate>Rio de Janeiro</ct211:region>
<ct211:typeCode>BUNDLE</ct211:typeCode>
<ct211:lineItemPayload index="1397235812805">
<im:lineId>1</im:lineId>
<im:promotionalSalesOrderLineReference>1</im:promotionalSalesOrderLineReference>
<im:serviceId/>
<im:requestedDeliveryDate>2014-03-31T07:05:00</im:requestedDeliveryDate>
<im:serviceInstance>N</im:serviceInstance>
<im:serviceAddress>
<im:nameLocation>Location1</im:nameLocation>
<im:typeAddress>Building</im:typeAddress>
</im:serviceAddress>
<im:itemReference>
<im:name>Brilliant Broadband</im:name>
<im:typeCode>BUNDLE</im:typeCode>
<im:primaryClassificationCode>Broadband Service Feature Class</im:primaryClassificationCode>
</im:itemReference>
</im:salesOrderLine>
</ct211:lineItemPayload>
<ct211:Recognition index="1397235812810">
Broadband Service Feature Class
</ct211:Recognition>
<ct211:Action index="1397235812812">Add</ct211:Action>
<ct211:ServiceInstance index="1397235812807">N</ct211:ServiceInstance>
</osmc:orderItemRef>
</osmc:orderItem>
[...]
<osmc:calculatedStartDate index="1397235812890">2014-03-31T07:05:00-07:00</osmc:calculatedStartDate>
<osmc:duration index="1397235812891">PT0S</osmc:duration>
</osmc:ProvisioningFunction>
</osmc:Functions>
<ct211:productSpec index="1397235812811">Broadband Service Feature Class</ct211:productSpec>
<ct211:fulfPatt index="1397235812802">Service.Broadband</ct211:fulfPatt>
<ct211:lineId index="1397235812809">1</ct211:lineId>
<ct211:lineItemName index="1397235812808">Brilliant Broadband [Add]</ct211:lineItemName>
<ct211:requestedDeliveryDate index="1397235812806">2014-03-31T07:05:00-07:00</ct211:requestedDeliveryDate>

Request and Response Examples

Using OSM Web Services

<ct211:requestedDeliveryDate>
<ct211:region index="1397235812803">Rio de Janeiro</ct211:region>
<ct211:typeCode index="1397235812813">BUNDLE</ct211:typeCode>
<ct211:lineItemPayload index="1397235812805">
<im:salesOrderLine
xmlns:im="http://xmlns.oracle.com/InputMessage">
<im:lineId>1</im:lineId>
<im:promotionalSalesOrderLineReference>1</im:promotionalSalesOrderLineReference>
<im:serviceId/>
<im:requestedDeliveryDate>2014-03-31T07:05:00</im:requestedDeliveryDate>
<im:serviceInstance>N</im:serviceInstance>
<im:serviceAddress>
<im:nameLocation>Location1</im:nameLocation>
<im:typeAddress>Building</im:typeAddress>
</im:serviceAddress>
<im:itemReference>
<im:name>Brilliant Broadband</im:name>
<im:typeCode>BUNDLE</im:typeCode>
<im:primaryClassificationCode>
Broadband Service Feature Class
</im:primaryClassificationCode>
<im:specificationGroup/>
</im:itemReference>
</im:salesOrderLine>
</ct211:lineItemPayload>
<ct211:Recognition index="1397235812810">
Broadband Service Feature Class
</ct211:Recognition>
<ct211:Action index="1397235812812">Add</ct211:Action>
<ct211:ServiceInstance index="1397235812807">N</ct211:ServiceInstance>
</osmc:OrderItem>

<ct160:dynamicParams xsi:type="ct264:SA_Provision_BroadbandInternetType" type="[OracleComms_Model_BroadbandInternet/4.0.0.0.0]SA_Provision_BroadbandInternetType" index="1397235812848" xmlns:ct264="OracleComms_Model_BroadbandInternet/4.0.0.0.0">
<ct264:DownloadSpeed index="1397235812851">50</ct264:DownloadSpeed>
<ct264:UploadSpeed index="1397235812850">3.072</ct264:UploadSpeed>
</ct160:dynamicParams>
<ct160:FulfillmentPattern index="1397235812846">[OracleComms_Model_BroadbandInternet/4.0.0.0.0]SA_Provision_BroadbandInternetSpec</ct160:FulfillmentPattern>
</osmc:TransformedOrderItem>
<ct160:Action index="1397235812854">Add</ct160:Action>
</osmc:TransformedOrderItem>
<osmc:MappingContext index="1397235812856">
<osmc:ProviderFunction index="1397235812857">
<osmc:namespace index="1397235812858">
OracleComms_Model_Base/4.0.0.0
</osmc:namespace>
<osmc:name index="1397235812859">CalculateServiceOrder</osmc:name>
<osmc:TargetMapping index="1397235812860">
<osmc:target index="1397235812861">CSO_2</osmc:target>
<osmc:SourceMapping index="1397235812862">
<osmc:source index="1397235812863">2</osmc:source>
<osmc:InstantiatingMappingRule index="1397235812864">
<osmc:namespace index="1397235812865">
http://www.oracle.com/otm/cso
</osmc:namespace>
<osmc:name index="1397235812866">BroadbandMappingRule_Broadband_PS_SA_Provision_BroadbandInternet_Primary_---g---U---R---Q1zkDkw</osmc:name>
</osmc:InstantiatingMappingRule>
</osmc:SourceMapping>
[...]
</osmc:TargetMapping>
</osmc:ProviderFunction>
</osmc:MappingContext>
</osmc:ControlData>
<osmc:BillingProfile index="4">
<osmc:mediaType index="5">1</osmc:mediaType>
<osmc:typeInvoice index="6">Summary</osmc:typeInvoice>
<osmc:billingCycle index="7">Q11</osmc:billingCycle>
<osmc:exemptionICMS index="8">Yes</osmc:exemptionICMS>
<osmc:empresaFaturamento index="9">Oi Fixed</osmc:empresaFaturamento>
<osmc:paymentMethod index="10">1</osmc:paymentMethod>
</osmc:BillingProfile>
</osmc:_root>
</Data>
</GetOrderResponse>
</env:Body>
</env:Envelope>
UpdateOrder Examples

This section provides request examples and a response example for the UpdateOrder operation.

Request Examples

Example 4–12 UpdateOrderRequest

```xml
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ws="http://xmlns.oracle.com/communications/ordermanagement">
  <soapenv:Header>
    <wsse:Security soapenv:mustUnderstand="1"
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-sec
text-1.0.xsd">
      <wsse:UsernameToken wsu:Id="UsernameToken-4799946"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utili
ty-1.0.xsd">
        <wsse:Username>administrator</wsse:Username>
        <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profi
gle-1.0#PasswordText">administrator</wsse:Password>
      </wsse:UsernameToken>
    </wsse:Security>
  </soapenv:Header>
  <soapenv:Body>
    <ws:UpdateOrder>
      <ws:OrderId>4</ws:OrderId>
      <ws:View>enter_payment_details</ws:View>
      <ws:UpdatedOrder>
        <_root>
          <account_information>
            <amount_owing>222</amount_owing>
          </account_information>
        </_root>
      </ws:UpdatedOrder>
    </soapenv:Body>
  </soapenv:Envelope>
```

Example 4–13 UpdateOrderRequest: Update nodes

```xml
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ws="http://xmlns.oracle.com/communications/ordermanagement">
  <soapenv:Header>
    <wsse:Security soapenv:mustUnderstand="1"
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-sec
text-1.0.xsd">
      <wsse:UsernameToken wsu:Id="UsernameToken-4799946"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utili
ty-1.0.xsd">
        <wsse:Username>administrator</wsse:Username>
        <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profi
gle-1.0#PasswordText">administrator</wsse:Password>
      </wsse:UsernameToken>
    </wsse:Security>
  </soapenv:Header>
  <soapenv:Body>
    <ws:UpdateOrder>
      <ws:OrderId>4</ws:OrderId>
      <ws:View>enter_payment_details</ws:View>
      <ws:UpdatedOrder>
        <_root>
          <account_information>
            <amount_owing>222</amount_owing>
          </account_information>
        </_root>
      </ws:UpdatedOrder>
    </soapenv:Body>
  </soapenv:Envelope>
```
Example 4–14  UpdateOrderRequest: Data change

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:ws="http://xmlns.oracle.com/communications/ordermanagement">
    <soapenv:Header>
        <wsse:Security soapenv:mustUnderstand="1"
            xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
            <wsse:UsernameToken wsu:Id="UsernameToken-4799946"
                xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
                <wsse:Username>administrator</wsse:Username>
                <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#PasswordText">administrator</wsse:Password>
            </wsse:UsernameToken>
        </wsse:Security>
    </soapenv:Header>
    <soapenv:Body>
        <ws:UpdateOrder>
            <ws:OrderId>4</ws:OrderId>
            <ws:DataChange>
                <ws:Update Path="/account_information/amount_owing">444</ws:Update>
            </ws:DataChange>
            <ws:StartOrder>false</ws:StartOrder>
            <ws:View>enter_payment_details</ws:View>
        </ws:UpdateOrder>
    </soapenv:Body>
</soapenv:Envelope>
Example 4-15  UpdateOrderRequest: Data change with Distributed Order Template

<ord:UpdateOrder>
    <ord:OrderId>123</ord:OrderId>
    <ord:View>OsmCentralOMExampleQueryTask</ord:View>
    <ord:DataChange>
        <ord:Update Path="/ControlData/OrderItem[@index='111223333'][@type='{http://oracle.communications.ordermanagement.unsupported.centralom}CustomerOrderItemSpecificationType']/dynamicParams[@index='222333444'][@type='{OracleComms_Model_BroadbandInternet/4.0.0.0.0}Broadband_Bandwidth_PSType']/UploadSpeed">
            10000
        </ord:Update>
    </ord:DataChange>
</ord:UpdateOrder>

Request and Response Example with ResponseView and OrderDataFiltering

Example 4-16 shows the standard response message returned from an UpdateOrderRequest.

Example 4-16  UpdateOrderResponse

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:ws="http://xmlns.oracle.com/communications/ordermanagement">
    <ws:UpdateOrderResponse>
        <ws:OrderId>2180</ws:OrderId>
        <ws:State>open.running.in_progress</ws:State>
    </ws:UpdateOrderResponse>
</soapenv:Envelope>

Example 4-17 shows an UpdateOrderRequest that adds a new customer address instance to the order that includes a ResponseView query task that defines the data to be returned in the UpdateOrderResponse message. The UpdateOrderRequest also includes an OrderDataFilter that specifies that only the new address instance with street value of "112 Update Drive" must be returned.

Example 4-17  UpdateOrderRequest with ResponseView and OrderDataFilter

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:ws="http://xmlns.oracle.com/communications/ordermanagement">
    <ws:UpdateOrderResponse>
        <ws:OrderId>2180</ws:OrderId>
        <ws:State>open.running.in_progress</ws:State>
    </ws:UpdateOrderResponse>
</soapenv:Envelope>
Example 4–18 shows the UpdateOrderResponse to the UpdateOrderRequest in Example 4–17.

**Example 4–18  UpdateOrderResponse with ResponseView and OrderDataFilter Applied**

```xml
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Header/>
  <env:Body>
    <UpdateOrderResponse xmlns="http://xmlns.oracle.com/communications/ordermanagement">
      <OrderId>10</OrderId>
      <State>open.running.in_progress</State>
      <Data>
        <osmc:_root index="0" xmlns:osmc="urn:oracle:names:ordermanagement:cartridge:bb_ocm_demo:1.0.0.0.0:view:demo_query">
          <osmc:subscriber_info index="1414771747926"/>
          <osmc:address index="1414771747949">
            <osmc:city index="1414771747950">TO</osmc:city>
            <osmc:postal_code index="1414771747951">A1A1A1</osmc:postal_code>
            <osmc:street index="1414771747952">112 Update Drive</osmc:street>
          </osmc:address>
          <osmc:primary_phone_number index="1414771747932">222122222</osmc:primary_phone_number>
          <osmc:name index="1414771747931">John Doe</osmc:name>
          <osmc:subscriber_info/>
          <osmc:adsl_service_details index="1414771747933"/>
          <osmc:bandwidth index="1414771747934">3</osmc:bandwidth>
        </osmc:_root>
        <Data>
          </Data>
    </UpdateOrderResponse>
  </env:Body>
</env:Envelope>
```
SuspendOrder Examples

This section provides a request example and a response example for the SuspendOrder operation.

Request Example

**Example 4–19  SuspendOrderRequest**

```xml
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ws="http://xmlns.oracle.com/communications/ordermanagement">
  <soapenv:Header>
    <wsse:Security soapenv:mustUnderstand="1"
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-sec
text-1.0.xsd">
      <wsse:UsernameToken wsu:Id="UsernameToken-4799946"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utili
ty-1.0.xsd">
        <wsse:Username>administrator</wsse:Username>
        <wsse:Password
Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profi
le-1.0#PasswordText">administrator</wsse:Password>
      </wsse:UsernameToken>
    </wsse:Security>
  </soapenv:Header>
  <soapenv:Body>
    <ws:SuspendOrder>
      <ws:OrderId>145</ws:OrderId>
      <!--Optional:-->
      <ws:Reason>test</ws:Reason>
      <!--You have a CHOICE of the next 2 items at this level-->
      <!--Optional:-->
      <ws:GracePeriodExpiryDate>?<ws:GracePeriodExpiryDate>
      <!--Optional:-->
      <ws:GracePeriodExpiry>?<ws:GracePeriodExpiry>
      <!--Optional:-->
      <ws:EventInterval>?<ws:EventInterval>
    </ws:SuspendOrder>
  </soapenv:Body>
</soapenv:Envelope>
```

Response Example

**Example 4–20  SuspendOrderResponse**

```xml
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ws="http://xmlns.oracle.com/communications/ordermanagement">
  <soapenv:Body>
    <ws:SuspendOrderResponse
xmlns:ws="http://xmlns.oracle.com/communications/ordermanagement">
      <ws:OrderId>145</ws:OrderId>
    </ws:SuspendOrderResponse>
  </soapenv:Body>
</soapenv:Envelope>
```
ResumeOrder Examples

This section provides a request example and a response example for the ResumeOrder operation.

Request Example

*Example 4–21 ResumeOrderRequest*

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ws="http://xmlns.oracle.com/communications/ordermanagement">
  <soapenv:Header>
    <wsse:Security soapenv:mustUnderstand="1"
    xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-sec
    ext-1.0.xsd">
      <wsse:UsernameToken wsu:Id="UsernameToken-4799946"
      xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-util
    ity-1.0.xsd">
        <wsse:Username>administrator</wsse:Username>
        <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profi
        le-1.0#PasswordText">administrator</wsse:Password>
      </wsse:UsernameToken>
  </soapenv:Security>
  <soapenv:Body>
    <ws:ResumeOrder>
      <ws:OrderId>1176</ws:OrderId>
    </ws:ResumeOrder>
  </soapenv:Body>
</soapenv:Envelope>
```

Response Example

*Example 4–22 ResumeOrderResponse*

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <ws:ResumeOrderResponse xmlns:ws="http://xmlns.oracle.com/communications/ordermanagement">
      <ws:OrderId>1176</ws:OrderId>
    </ws:ResumeOrderResponse>
  </soapenv:Body>
</soapenv:Envelope>
```
CancelOrder Examples

This section provides a request example and a response example for the CancelOrder operation.

Request Example

Example 4–23  CancelOrderRequest

```xml
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ws="http://xmlns.oracle.com/communications/ordermanagement">
  <soapenv:Header>
    <wsse:Security soapenv:mustUnderstand="1"
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-sec
      ext-1.0.xsd">
      <wsse:UsernameToken wsu:Id="UsernameToken-4799946"
        xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
        <wsse:Username>administrator</wsse:Username>
        <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profi
        le-1.0#PasswordText">administrator</wsse:Password>
      </wsse:UsernameToken>
    </wsse:Security>
  </soapenv:Header>
  <soapenv:Body>
    <ws:CancelOrder>
      <ws:OrderId>1316</ws:OrderId>
    </ws:CancelOrder>
  </soapenv:Body>
</soapenv:Envelope>
```

Response Example

Example 4–24  CancelOrderResponse

```xml
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header/>
  <soapenv:Body>
    <ws:CancelOrderResponse
      xmlns:ws="http://xmlns.oracle.com/communications/ordermanagement">
      <ws:OrderId>1316</ws:OrderId>
    </ws:CancelOrderResponse>
  </soapenv:Body>
</soapenv:Envelope>
```
GetOrderProcessHistory Examples

This section provides a request example and a response example for the GetOrderProcessHistory operation.

GetOrderProcessHistory Requests

Example 4–25  GetOrderProcessHistory Requests by CompensationID

```xml
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ord="http://xmlns.oracle.com/communications/ordermanagement">
  <soapenv:Header/>
  <soapenv:Body>
    <ord:GetOrderProcessHistory>
      <OrderId>6</OrderId>
      <CompensationId>1</CompensationId>
    </ord:GetOrderProcessHistory>
  </soapenv:Body>
</soapenv:Envelope>
```

Example 4–26  GetOrderProcessHistory Requests by Perspective (Original)

```xml
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ord="http://xmlns.oracle.com/communications/ordermanagement">
  <soapenv:Header/>
  <soapenv:Body>
    <ord:GetOrderProcessHistory>
      <OrderId>6</OrderId>
      <Perspective>original</Perspective>
    </ord:GetOrderProcessHistory>
  </soapenv:Body>
</soapenv:Envelope>
```

Example 4–27  GetOrderProcessHistory Requests by Perspective (Latest)

```xml
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ord="http://xmlns.oracle.com/communications/ordermanagement">
  <soapenv:Header/>
  <soapenv:Body>
    <ord:GetOrderProcessHistory>
      <OrderId>6</OrderId>
      <Perspective>latest</Perspective>
    </ord:GetOrderProcessHistory>
  </soapenv:Body>
</soapenv:Envelope>
```

GetOrderProcessHistory Responses

Example 4–28  GetOrderProcessHistory Response by CompensationID

```xml
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
xmlls:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <env:Header/>
  <env:Body>
    <ord:GetOrderProcessHistoryResponse
xmlns:ord="http://xmlns.oracle.com/communications/ordermanagement">
```
<OrderId>6</OrderId>
</Cartridge>

<Compensation xsi:type="ord:AmendmentCompensationInfoType">
  <CompensationId>1</CompensationId>
  <CompensationType>amend</CompensationType>
  <Submitted>2015-06-02T08:24:29.975-07:00</Submitted>
  <Started>2015-06-02T08:24:31.000-07:00</Started>
  <AmendmentOrderId>7</AmendmentOrderId>
</Compensation>

<ProcessHistory>
  <Item xsi:type="ord:WorkItemType"
    Id="2"
    TaskName="OsmCentralOMExampleOrder">
    <TaskType>creation</TaskType>
    <StartDate>2015-06-02T08:23:43.000-07:00</StartDate>
    <EndDate>2015-06-02T08:23:43.000-07:00</EndDate>
    <User>omsadmin</User>
    <ActualDuration>PT0.000S</ActualDuration>
  </Item>

  <Item xsi:type="ord:ContainerItemType"
    Id="102"
    TaskName="SyncCustomerFunction_CustomerSystemSI">
    <TaskType>subprocess</TaskType>
    <StartDate>2015-06-02T08:23:43.000-07:00</StartDate>
    <EndDate>2015-06-02T08:23:49.000-07:00</EndDate>
    <User>omsadmin</User>
    <ActualDuration>PT6.000S</ActualDuration>
    <Scope>
      <Item xsi:type="ord:WorkItemType"
        Id="1205"
        TaskName="configureCustomerSystemTask">
        <TaskType>automated</TaskType>
        <StartDate>2015-06-02T08:23:46.000-07:00</StartDate>
        <EndDate>2015-06-02T08:23:49.000-07:00</EndDate>
        <User>omsadmin</User>
        <ActualDuration>PT3.000S</ActualDuration>
      </Item>
      <Scope/>
    </Scope>
  </Item>

  <Item xsi:type="ord:ContainerItemType"
    Id="202"
    TaskName="MarketingFunction_MarketingSI">
    <TaskType>subprocess</TaskType>
    <StartDate>2015-06-02T08:23:43.000-07:00</StartDate>
    <EndDate>2015-06-02T08:23:46.000-07:00</EndDate>
    <User>omsadmin</User>
    <ActualDuration>PT3.000S</ActualDuration>
    <Scope>
      <Item xsi:type="ord:WorkItemType"
        Id="1105"
        TaskName="configureMarketingTask">
        <TaskType>automated</TaskType>
        <StartDate>2015-06-02T08:23:46.000-07:00</StartDate>
        <EndDate>2015-06-02T08:23:48.000-07:00</EndDate>
        <CompensateeRole>
<Item xsi:type="ord:WorkItemType">
  <Id>1409</Id>
  <TaskName>MarketingBaseTask</TaskName>
  <TaskType>manual</TaskType>
  <StartDate>2015-06-02T08:23:48.000-07:00</StartDate>
  <EndDate>2015-07-23T13:40:03.849-07:00</EndDate>
  <User>omsadmin</User>
  <ActualDuration>PT2.000S</ActualDuration>
</Item>

<Item xsi:type="ord:WorkItemType">
  <Id>904</Id>
  <TaskName>configureBillingTask</TaskName>
  <TaskType>automated</TaskType>
  <StartDate>2015-06-02T08:23:45.000-07:00</StartDate>
  <EndDate>2015-06-02T08:23:46.000-07:00</EndDate>
  <CompensateeRole>
    <ExecutionMode>redo</ExecutionMode>
    <CompensatorId>2117</CompensatorId>
    <CompensatorState>accepted</CompensatorState>
  </CompensateeRole>
  <User>omsadmin</User>
  <ActualDuration>PT1.000S</ActualDuration>
</Item>

<Item xsi:type="ord:WorkItemType">
  <Id>402</Id>
  <TaskName>CollectionsFunction_CollectionsSI</TaskName>
  <TaskType>subprocess</TaskType>
  <StartDate>2015-06-02T08:23:43.000-07:00</StartDate>
  <EndDate>2015-06-02T08:24:04.000-07:00</EndDate>
  <User>omsadmin</User>
  <ActualDuration>PT21.000S</ActualDuration>
</Item>
<Id>1305</Id>
<TaskName>configureCollectionsTask</TaskName>
<TaskType>automated</TaskType>
<StartDate>2015-06-02T08:23:46.000-07:00</StartDate>
<EndDate>2015-06-02T08:24:04.000-07:00</EndDate>
<User>omsadmin</User>
<ActualDuration>PT18.000S</ActualDuration>
</Item>
</Scope>
</Item>
<Item xsi:type="ord:ContainerItemType">
<Id>502</Id>
<TaskName>ProvisioningFunction_ProvisioningSI</TaskName>
<TaskType>subprocess</TaskType>
<StartDate>2015-06-02T08:23:43.000-07:00</StartDate>
<EndDate>2015-06-02T08:23:45.000-07:00</EndDate>
<User>omsadmin</User>
<ActualDuration>PT2.000S</ActualDuration>
<Scope>
<Item xsi:type="ord:WorkItemType">
<Id>602</Id>
<TaskName>routeToProvisioningTask</TaskName>
<TaskType>automated</TaskType>
<StartDate>2015-06-02T08:23:43.000-07:00</StartDate>
<EndDate>2015-06-02T08:23:43.000-07:00</EndDate>
<CompensateeRole>
<ExecutionMode>redo</ExecutionMode>
<CompensatorId>1912</CompensatorId>
<CompensatorState>completed</CompensatorState>
</CompensateeRole>
<User>omsadmin</User>
<ActualDuration>PT0.000S</ActualDuration>
</Item>
<Item xsi:type="ord:WorkItemType">
<Id>704</Id>
<TaskName>activationOrderAdslRegion2Task</TaskName>
<TaskType>automated</TaskType>
<StartDate>2015-06-02T08:23:43.000-07:00</StartDate>
<EndDate>2015-06-02T08:23:45.000-07:00</EndDate>
<CompensateeRole>
<ExecutionMode>redo</ExecutionMode>
<CompensatorId>2014</CompensatorId>
<CompensatorState>completed</CompensatorState>
</CompensateeRole>
<User>omsadmin</User>
<ActualDuration>PT2.000S</ActualDuration>
</Item>
</Scope>
</Item>
<Item xsi:type="ord:ContainerItemType">
<Id>2</Id>
<TaskName>ProvisioningFunction_ProvisioningSI</TaskName>
<TaskType>subprocess</TaskType>
<StartDate>2015-06-02T08:23:43.000-07:00</StartDate>
<EndDate>2015-06-02T08:23:45.000-07:00</EndDate>
<User>omsadmin</User>
<ActualDuration>PT2.000S</ActualDuration>
</Item>
</Scope>
</Item>
</Scope>
</Request and Response Examples
Using OSM Web Services 4-53
<Link>
  <Source>2</Source>
  <Target>202</Target>
</Link>
<Link>
  <Source>2</Source>
  <Target>302</Target>
</Link>
<Link>
  <Source>2</Source>
  <Target>402</Target>
</Link>
<Link>
  <Source>2</Source>
  <Target>502</Target>
</Link></ProcessHistory></env:Body></env:Envelope>

Example 4–29  GetOrderProcessHistory Response by Perspective (Original)

<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <env:Header/>
  <env:Body>
    <ord:GetOrderProcessHistoryResponse
      xmlns:ord="http://xmlns.oracle.com/communications/ordermanagement">
      <OrderId>6</OrderId>
      <Cartridge>
        <ord:Name>OsmCentralOMExample-Solution</ord:Name>
        <ord:Version>4.0.0.0.0</ord:Version>
      </Cartridge>
      <ProcessHistory>
        <Item xsi:type="ord:WorkItemType">
          <Id>2</Id>
          <TaskName>OsmCentralOMExampleOrder_OsmCentralOMExampleOrder</TaskName>
          <TaskType>creation</TaskType>
          <StartDate>2015-06-02T08:23:43.000-07:00</StartDate>
          <EndDate>2015-06-02T08:23:43.000-07:00</EndDate>
          <User>omsadmin</User>
          <ActualDuration>PT0.000S</ActualDuration>
        </Item>
        <Item xsi:type="ord:ContainerItemType">
          <Id>102</Id>
          <TaskName>SyncCustomerFunction_CustomerSystemSI</TaskName>
          <TaskType>subprocess</TaskType>
          <StartDate>2015-06-02T08:23:43.000-07:00</StartDate>
          <EndDate>2015-06-02T08:23:49.000-07:00</EndDate>
          <User>omsadmin</User>
          <ActualDuration>PT6.000S</ActualDuration>
        </Item>
        <Item xsi:type="ord:ContainerItemType">
          <Id>202</Id>
          <TaskName>MarketingFunction_MarketingSI</TaskName>
        </Item>
      </ProcessHistory>
    </ord:GetOrderProcessHistoryResponse>
  </env:Body>
</env:Envelope>
<TaskType>subprocess</TaskType>
<StartDate>2015-06-02T08:23:43.000-07:00</StartDate>
<EndDate>2015-06-02T08:23:46.000-07:00</EndDate>
<User>omsadmin</User>
<ActualDuration>PT3.000S</ActualDuration>
</Item>
<Item xsi:type="ord:ContainerItemType">
  <Id>302</Id>
  <TaskName>BillingFunction_BillingSI</TaskName>
  <TaskType>subprocess</TaskType>
  <StartDate>2015-06-02T08:23:43.000-07:00</StartDate>
  <EndDate>2015-06-02T08:23:46.000-07:00</EndDate>
  <User>omsadmin</User>
  <ActualDuration>PT3.000S</ActualDuration>
</Item>
<Item xsi:type="ord:ContainerItemType">
  <Id>402</Id>
  <TaskName>CollectionsFunction_CollectionsSI</TaskName>
  <TaskType>subprocess</TaskType>
  <StartDate>2015-06-02T08:23:43.000-07:00</StartDate>
  <EndDate>2015-06-02T08:24:04.000-07:00</EndDate>
  <User>omsadmin</User>
  <ActualDuration>PT21.000S</ActualDuration>
</Item>
<Item xsi:type="ord:ContainerItemType">
  <Id>502</Id>
  <TaskName>ProvisioningFunction_ProvisioningSI</TaskName>
  <TaskType>subprocess</TaskType>
  <StartDate>2015-06-02T08:23:43.000-07:00</StartDate>
  <EndDate>2015-06-02T08:23:45.000-07:00</EndDate>
  <User>omsadmin</User>
  <ActualDuration>PT2.000S</ActualDuration>
</Item>
</Links>
</ProcessHistory>
</ord:GetOrderProcessHistoryResponse>
</env:Body>
</env:Envelope>
Example 4–30  GetOrderProcessHistory Response by Perspective (Latest)

<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <env:Header/>
  <env:Body>
    <ord:GetOrderProcessHistoryResponse
      xmlns:ord="http://xmlns.oracle.com/communications/ordermanagement">
      <OrderId>6</OrderId>
      <Cartridge>
        <ord:Name>OsmCentralOMExample-Solution</ord:Name>
        <ord:Version>4.0.0.0.0</ord:Version>
      </Cartridge>
      <ProcessHistory>
        <Item xsi:type="ord:WorkItemType">
          <Id>2</Id>
          <TaskName>OsmCentralOMExampleOrder_OsmCentralOMExampleOrder</TaskName>
          <TaskType>creation</TaskType>
          <StartDate>2015-06-02T08:23:43.000-07:00</StartDate>
          <EndDate>2015-06-02T08:23:43.000-07:00</EndDate>
          <User>omsadmin</User>
          <ActualDuration>PT0.000S</ActualDuration>
        </Item>
        <Item xsi:type="ord:ContainerItemType">
          <Id>102</Id>
          <TaskName>SyncCustomerFunction_CustomerSystemSI</TaskName>
          <TaskType>subprocess</TaskType>
          <StartDate>2015-06-02T08:23:43.000-07:00</StartDate>
          <EndDate>2015-06-02T08:23:49.000-07:00</EndDate>
          <User>omsadmin</User>
          <ActualDuration>PT6.000S</ActualDuration>
          <Scope>
            <Item xsi:type="ord:WorkItemType">
              <Id>1205</Id>
              <TaskName>configureCustomerSystemTask</TaskName>
              <TaskType>automated</TaskType>
              <StartDate>2015-06-02T08:23:46.000-07:00</StartDate>
              <EndDate>2015-06-02T08:23:49.000-07:00</EndDate>
              <User>omsadmin</User>
              <ActualDuration>PT3.000S</ActualDuration>
            </Item>
            <Link/>
          </Scope>
        </Item>
        <Item xsi:type="ord:ContainerItemType">
          <Id>202</Id>
          <TaskName>MarketingFunction_MarketingSI</TaskName>
          <TaskType>subprocess</TaskType>
          <StartDate>2015-06-02T08:23:43.000-07:00</StartDate>
          <EndDate>2015-06-02T08:23:46.000-07:00</EndDate>
          <User>omsadmin</User>
          <ActualDuration>PT3.000S</ActualDuration>
          <Scope>
            <Item xsi:type="ord:WorkItemType">
              <Id>1105</Id>
              <TaskName>configureMarketingTask</TaskName>
              <TaskType>automated</TaskType>
              <StartDate>2015-06-02T08:23:46.000-07:00</StartDate>
              <EndDate>2015-06-02T08:23:48.000-07:00</EndDate>
              <CompensateeRole>
            </Item>
          </Scope>
        </Item>
      </ProcessHistory>
    </ord:GetOrderProcessHistoryResponse>
  </env:Body>
</env:Envelope>
<ExecutionMode>undo</ExecutionMode>
<CompensatorId>1810</CompensatorId>
<CompensatorState>completed</CompensatorState>
</CompensateeRole>
<User>omsadmin</User>
<ActualDuration>PT2.000S</ActualDuration>
</Item>
<Item xsi:type="ord:WorkItemType">
<Id>1409</Id>
<TaskName>MarketingBaseTask</TaskName>
<TaskType>manual</TaskType>
<StartDate>2015-06-02T08:23:48.000-07:00</StartDate>
<EndDate>2015-07-24T08:19:07.207-07:00</EndDate>
<User>omsadmin</User>
<ActualDuration>PT2.000S</ActualDuration>
</Item>
</Scope>
</Item>
<Item xsi:type="ord:ContainerItemType">
<Id>302</Id>
<TaskName>BillingFunction_BillingSI</TaskName>
<TaskType>subprocess</TaskType>
<StartDate>2015-06-02T08:23:43.000-07:00</StartDate>
<EndDate>2015-06-02T08:23:46.000-07:00</EndDate>
<User>omsadmin</User>
<ActualDuration>PT3.000S</ActualDuration>
</Scope>
</Item>
<Item xsi:type="ord:ContainerItemType">
<Id>402</Id>
<TaskName>CollectionsFunction_CollectionsSI</TaskName>
<TaskType>subprocess</TaskType>
<StartDate>2015-06-02T08:23:43.000-07:00</StartDate>
<EndDate>2015-06-02T08:23:46.000-07:00</EndDate>
<User>omsadmin</User>
<ActualDuration>PT21.000S</ActualDuration>
</Scope>
</Item>
<Id>1305</Id>
<TaskName>configureCollectionsTask</TaskName>
<TaskType>automated</TaskType>
<StartDate>2015-06-02T08:23:46.000-07:00</StartDate>
<EndDate>2015-06-02T08:24.000-07:00</EndDate>
<User>omsadmin</User>
<ActualDuration>PT18.000S</ActualDuration>
</Item>
</Scope>
</Item>
<Item xsi:type="ord:ContainerItemType">
<Id>502</Id>
<TaskName>ProvisioningFunction_ProvisioningSI</TaskName>
<TaskType>subprocess</TaskType>
<StartDate>2015-06-02T08:23:43.000-07:00</StartDate>
<EndDate>2015-06-02T08:23:45.000-07:00</EndDate>
<User>omsadmin</User>
<ActualDuration>PT2.000S</ActualDuration>
</Scope>
</Item>
<Item xsi:type="ord:WorkItemType">
<Id>602</Id>
<TaskName>routeToProvisioningTask</TaskName>
<TaskType>automated</TaskType>
<StartDate>2015-06-02T08:23:43.000-07:00</StartDate>
<EndDate>2015-06-02T08:23:43.000-07:00</EndDate>
<CompensateeRole>
<ExecutionMode>redo</ExecutionMode>
<CompensatorId>1912</CompensatorId>
<CompensatorState>completed</CompensatorState>
</CompensateeRole>
<User>omsadmin</User>
<ActualDuration>PT0.000S</ActualDuration>
</Item>
</Scope>
</Item>
<Item xsi:type="ord:WorkItemType">
<Id>704</Id>
<TaskName>activationOrderAdslRegion2Task</TaskName>
<TaskType>automated</TaskType>
<StartDate>2015-06-02T08:23:43.000-07:00</StartDate>
<EndDate>2015-06-02T08:23:45.000-07:00</EndDate>
<CompensateeRole>
<ExecutionMode>redo</ExecutionMode>
<CompensatorId>2014</CompensatorId>
<CompensatorState>completed</CompensatorState>
</CompensateeRole>
<User>omsadmin</User>
<ActualDuration>PT2.000S</ActualDuration>
</Item>
</Scope>
</Item>
<Links>
<Link>
<Source>602</Source>
<Target>704</Target>
</Link>
</Links>
</Item>
</Scope>
</Item>
<Links>
<Link>
<Source>2</Source>
<Target>102</Target>
</Link>
</Links>
</Item>


</Links>
</ProcessHistory>
</ord:GetOrderProcessHistoryResponse>
</env:Body>
</env:Envelope>
GetOrderCompensations Examples

This section provides a request example and a response example for the GetOrderCompensations operation.

Example 4–31 GetOrderCompensations
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ord="http://xmlns.oracle.com/communications/ordermanagement">
<soapenv:Header/>
<soapenv:Body>
<ord:GetOrderCompensations>
<OrderId>6</OrderId>
</ord:GetOrderCompensations>
</soapenv:Body>
</soapenv:Envelope>

Example 4–32 GetOrderCompensationsResponse
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<env:Header/>
<env:Body>
<ord:GetOrderCompensationsResponse
xmlns:ord="http://xmlns.oracle.com/communications/ordermanagement">
<OrderId>6</OrderId>
<Compensation xsi:type="ord:AmendmentCompensationInfoType">
<CompensationId>1</CompensationId>
<CompensationType>amend</CompensationType>
<Submitted>2015-06-02T08:24:29.975-07:00</Submitted>
<Started>2015-06-02T08:24:31.000-07:00</Started>
<AmendmentOrderId>7</AmendmentOrderId>
</Compensation>
</ord:GetOrderCompensationsResponse>
</env:Body>
</env:Envelope>
GetCompensationPlan Examples

This section provides a request example and a response example for the GetOrderCompensations operation.

**Example 4–33  GetOrderCompensationPlan**

```xml
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/
xmlns:ord="http://xmlns.oracle.com/communications/ordermanagement">
  <soapenv:Header/>
  <soapenv:Body>
    <ord:GetCompensationPlan>
      <OrderId>6</OrderId>
    </ord:GetCompensationPlan>
  </soapenv:Body>
</soapenv:Envelope>
```

**Example 4–34  GetOrderCompensationPlanResponse**

```xml
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ord="http://xmlns.oracle.com/communications/ordermanagement">
  <env:Header/>
  <env:Body>
    <ord:GetCompensationPlanResponse
xmlns:ord="http://xmlns.oracle.com/communications/ordermanagement">
      <OrderId>6</OrderId>
      <CompensationId>1</CompensationId>
      <CompensationType>amend</CompensationType>
      <ActiveItem>
        <TaskName>SyncCustomerFunction_CustomerSystemSI</TaskName>
        <ExecutionMode>redo</ExecutionMode>
        <CompensationWorkItem>self</CompensationWorkItem>
        <FlowItemId>102</FlowItemId>
      </ActiveItem>
      <ActiveItem>
        <TaskName>CollectionsFunction_CollectionsSI</TaskName>
        <ExecutionMode>redo</ExecutionMode>
        <CompensationWorkItem>self</CompensationWorkItem>
        <FlowItemId>402</FlowItemId>
      </ActiveItem>
      <ActiveItem>
        <TaskName>configureBillingTask</TaskName>
        <ExecutionMode>redo</ExecutionMode>
        <CompensationWorkItem>self</CompensationWorkItem>
        <FlowItemId>1005</FlowItemId>
      </ActiveItem>
      <PendingItem>
        <TaskName>cdiTask</TaskName>
        <ExecutionMode>redo</ExecutionMode>
        <CompensationWorkItem>self</CompensationWorkItem>
        <FlowItemId>1508</FlowItemId>
      </PendingItem>
    </ord:GetCompensationPlanResponse>
  </env:Body>
</env:Envelope>
```
<PendingItem>
  <TaskName>Configure Collections Task</TaskName>
  <ExecutionMode>redo</ExecutionMode>
  <CompenationWorkItem>self</CompenationWorkItem>
  <FlowItemId>1608</FlowItemId>
  <WaitsFor>
    <TaskName>CollectionsFunction_CollectionsSI</TaskName>
    <ExecutionMode>redo</ExecutionMode>
    <PositionedInFlow>before</PositionedInFlow>
    <CompenationWorkItem>self</CompenationWorkItem>
    <FlowItemId>1609</FlowItemId>
  </WaitsFor>
</PendingItem>
</env:GetCompensationPlanResponse>
</env:Body>
</env:Envelope>
This chapter describes the Oracle Communications Order and Service Management (OSM) automation, which enables you to configure and automatically run automated tasks and notifications.

About Automations and the Automation Framework

The OSM automation framework provides the primary interface for outbound and inbound operations that interact with external systems for automated order fulfillment. The automation framework also provides internal data processing for automated tasks within a process workflow. You can create notifications for individual tasks or at the order level that trigger automations. See OSM Concepts for information about automated tasks and notifications.

To run automated tasks, notifications at the task level, or notification at the order level, you write automation plug-ins. The automation framework runs the automation plug-ins within the context of these tasks and notifications which defines what order data is available to the automation.

An automation plug-in can be a:
- Custom automation plug-in, which is an automation plug-in that you write, consisting of custom business logic in the form of Java code.
- Predefined automation plug-in, which is an automation plug-in that is provided with the OSM installation that you can augment with your business logic requirements.

OSM provides the following predefined automation plug-ins:
- XSLT Plug-in. A plug-in that uses XSLT to generate outbound messages and process in-bound messages.
- XQuery Plug-in. A plug-in that uses XQuery to generate outbound messages and process in-bound messages.
- JDBC Plug-in. A plug-in that uses JDBC to retrieve or update data in the database.
- Email Plug-in. A plug-in available for notifications that send email messages to external systems.

The automation framework simplifies the process of sending messages to external systems. The automation framework does the following:
- Uses the JMS communication protocol.
- Establishes and maintains the various JMS connections.
- Constructs the JMS messages, setting the required message properties.
Correlates requests from OSM with responses from external systems.

 Guarantees delivery of the message and handles any errors or exceptions. It retries messages until the message delivers.

 Handles poison messages. For example, if the message is undeliverable for some reason.

When OSM sends a message to an external system using an automation plug-in, the following processing flow generally occurs:

1. OSM runs an automated task that triggers an automation called a sender plug-in.

2. The automation framework adds properties to the outbound message to correlate external system responses to requests. For example, for a predefined XQuery or XSLT sender plug-in:
   a. The sender plug-in sets a property on the outbound JMS message as the correlation property.
   b. The automation framework saves the message properties set for each message with the event information.
   c. The automation framework sets the replyTo JMS property on the JMS request based on properties configured for the sender plug-in.

3. The automation framework sends the JMS message to the JMS queue and destination type that the external system must subscribe to in order to consume based on properties configured for the sender plug-in.

**Note:** Custom automations are not restricted to JMS but can use any communication protocol, such as HTTP or FTP. See “About Custom Automation Plug-ins” for more information.

When OSM receives a message in response to the request, the following process flow generally occurs:

1. After processing the request, the external system copies the properties from the incoming request to the outgoing response.

2. The external system sends the response message to the reply to queue based on the replyTo JMS property in the request.

3. The automation framework routes the response from the queue to the plug-in. The plug-in that receives the response is called an automator.

4. The automation framework uses the message properties of the response, plus the correlation information, to reload a Context for the response message, which is in this scenario the task that sent the original request.

5. The automator performs business logic, such as updating order data and completing the task.

You can create custom or predefined plug-ins using Oracle Communications Design Studio Help.

**Figure 5–1** shows the flow of an automated task with a notification that call their corresponding automation plug-in. Design Studio provides the ability to map a specific automated task (Task A) to a specific automation plug-in (Automation Plug-in A), or a specific automated notification (Notification B) to a specific automation plug-in (Automation Plug-in B). This is called automation mapping. The mappings are saved to a cartridge, which is then deployed to the OSM server. OSM processes the
automated tasks which trigger the mapped automation plug-ins when specific events occur. See "About Creating Automations in Design Studio" and "About Internal and External Events that Trigger Automations" for more information.

Figure 5–1 Automation Flow

About Internal and External Events that Trigger Automations

When you create an automation plug-in for a task, task notification, or order notification in Design Studio, you bring up the Add Automation dialog box to create a plug-in for the task or notification, give it a name, and select the Automation Type (for example, one of the predefined automations or a custom automation). There are two basic types of automation plug-ins: Sender and Automator. Use the Automator type if you want the plug-in to receive data and perform work on the data. Use the Sender type if you want the plug-in to receive data, perform work, then send the data to external systems.

You must also define where you expect the plug-in to receive its data when you set the plug-in Event Type, which specifies whether the new plug-in receives data events internally from OSM or from external systems. The choices are as follows:

- **Internal Event Receiver** (default choice): Internal receiver indicates that the source of event for plug-ins is internal to OSM. OSM makes order data available to these type of plug-ins in their respective contexts (see "About Automations in the Order and Task Contexts" for more information). For internal event receivers, the following happens:
  - An event occurs within OSM.
  - OSM creates a message and sends it to the internal message queue that is part of the OSM installation. The OSM order priority is mapped to the JMS priority to prioritize internal events.
  - The automation framework subscribes to the internal message queue as part of the OSM installation.
  - The message is picked up by the automation framework and processed.

- **External Event Receiver**: The data made available to the automation plug-in comes from a message sent from an external system. For external event receivers, the following happens:
— An event occurs within an external system, such as an OSM automation plug-in sends a message that arrives at the external system.

— The external system creates a response message and sends it to an external message queue. You must explicitly create the external message queues.

— The automation framework subscribes to the external message queue through the information you define on the **External Event Receiver** tab of the automation definition.

— The message is picked up by the automation framework and processed.

Automated notifications are always defined as internal event receivers because, as the name implies, notifications are used to notify OSM users or other areas of the OSM system of some event occurring within OSM. That is why notifications do not receive messages from external systems; the information with which to notify always originates within OSM.

The new plug-in appears in the **Automation** list. Once you add a plug-in to the your automated task, you define the plug-in properties. See the Design Studio Help for further information.

Automation automator or sender plug-ins that are external event receivers (process responses from external systems) listen for responses (JMS messages) from external systems on an external message queue (JMS queue). These are responses to previously sent messages that are correlated back to a task based on correlation ID. In some cases you must specify filter criteria, defined in Design Studio as a message property selector, which OSM uses to filter messages on the JMS queue. A task only receives messages from queues that match the message property. If a message is selected, then message correlation occurs as normal and the automated task receives the message. The external system must echo back the filter criteria information by extracting and reinserting it into its response. See "About Correlation and Message Property Selectors" for more information.

**About Automations in the Order and Task Contexts**

You can configure automations in various contexts, such as automated tasks, notifications configured for automated tasks, notifications configured for manual tasks, notifications configured in process flows, and notifications configured at the order level. The data available to these automations depends on which of these contexts the automation is triggered. The two main contexts from which depend all the other contexts are the order context and the task context.

The data available to an automation plug-in in the task context is restricted to the data defined in the automated task’s **Task Data** tab. The data available to an automation plug-in in the order context is restricted to the data defined in the order specification, **Permission** tab, **Query Task** subtab. This subtab links to a manual task designated as a query task that defines the data available to order level notifications but is not part of any process flow.

When you create custom automations, you can access these contexts from the OSM Java API **com.mslv.automation.oms.AutomationContext** class which is the parent class of **OrderContext** which is in turn the parent of the **TaskContext**. These are either parent or sibling classes for all the other contexts. You never need to import the AutomationContext because it is inherited by all the other contexts. You can also declare these contexts in predefined automation plug-ins.

Each context class provides methods (or inherits them from parent classes) that you can use in automation plug-ins to perform various functions such as:
- Updating order data
- Transitioning the task to a new state
- Suspending the task
- Completing the task
- Getting order task data for use in business logic
- Transition the order into a failed state

Figure 5–2 shows the class hierarchy stemming from the AutomationContext.

**Figure 5–2 Context Object Class Hierarchy**

Some of the methods that the task context inherits from the order context behave differently when run from the task context. For example, the update order method run from the task context can generate historical and contemporary order perspectives that can be used in order amendment analysis, while the update order method run from the order context does not. See "About Compensation for Automations" for more information.

Table 5–1 shows the Design Studio entity where you can configure automations, the types of events that trigger the automations, and the context that gets passed into the plug-in.
### Table 5–1 Context Objects Passed To Plug-in

<table>
<thead>
<tr>
<th>Automation Plug-in Trigger</th>
<th>Design Studio Definition Location</th>
<th>OSM Event</th>
<th>OSM Event Type</th>
<th>Context Object Passed To Plug-in</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automated task</td>
<td>Task editor, Automation tab</td>
<td>Task state transitions to Received</td>
<td>Task Event</td>
<td>TaskContext</td>
</tr>
<tr>
<td>Order milestone-based event notification</td>
<td>Order editor, Events tab</td>
<td>Order reaches specified milestone</td>
<td>Order Notification Event</td>
<td>OrderNotificationContext</td>
</tr>
<tr>
<td>Task state-based event notification</td>
<td>Task editor, Events tab</td>
<td>Task reaches specified state</td>
<td>Task Notification Event</td>
<td>TaskNotificationContext</td>
</tr>
<tr>
<td>Task state-based event notification</td>
<td>Process editor, Events tab on Properties view of a task in the process</td>
<td>Task reaches specified state, then data condition specified by rule evaluates to true.</td>
<td>Task Notification Event</td>
<td>TaskNotificationContext</td>
</tr>
<tr>
<td>Task status-based event notification</td>
<td>Process editor, Events tab on Properties view of a status in the process</td>
<td>Task reaches specified status, then data condition specified by rule evaluates to true.</td>
<td>Task Notification Event</td>
<td>TaskNotificationContext</td>
</tr>
<tr>
<td>Order data changed event notification</td>
<td>Order editor, Notifications tab</td>
<td>Specified order data changes.</td>
<td>Order Notification Event</td>
<td>OrderDataChangeNotificationContext</td>
</tr>
<tr>
<td>Order jeopardy notification</td>
<td>Order editor, Jeopardy tab</td>
<td>At polling, data condition defined by rule evaluates to true.</td>
<td>System Notification Event</td>
<td>OrderNotificationContext</td>
</tr>
<tr>
<td>Task jeopardy notification</td>
<td>Task editor, Jeopardy tab</td>
<td>At polling, data condition defined by rule evaluates to true.</td>
<td>System Notification Event</td>
<td>If the task level jeopardy condition <strong>Multiple events per Task instance</strong> is set, then TaskNotificationContext is passed. Otherwise OrderNotificationCenterContext is passed.</td>
</tr>
</tbody>
</table>

All context objects are located in the `OSM_home/SDK/automation/automationdeploy_bin/automation_plugins.jar` file. All context objects are defined in the same package: `com.mslv.automation.oms`.

### About Accessing the XML API in Automations

You can use the XMP API from within automations. To access the XML APIs from within a custom automation plug-in, API users must belong to a WebLogic group that provides privilege to access the APIs. For accessing the XML APIs from within a custom automation plug-in, that WebLogic group is OSM_automation. So, to access the APIs from within a custom automation plug-in, the API user must belong to the WebLogic group OSM_automation.
See the Design Studio Help for further information regarding the **Run As** field, which defines the user of the automation.

**About Correlation and Message Property Selectors**

Correlation is a property that associates an incoming external system message with an outbound OSM message previously sent to initiate communication with the external system. In some situations you may need additional message filtering using message property selectors.

**Correlating Requests from OSM to Responses from External Systems**

You can set the JMS ID Correlation parameter in messages sent from OSM to external systems to correlate response messages from the external system with the original request. If you expect the correlated response to return to the task that originally sent the message, then you do not need to programmatically set the correlation ID for the task because this is done for the task when the original sender sent the message. If you expect the correlated response to return to a different task (a receiver task) than the one that sent the message, then you must programmatically set the correlation ID for the outgoing JMS message in the sending task, and configure the receiver task to use the matching correlation ID. For more information about this second scenario, see "Asynchronous Communication: Single or Multiple Requests and Responses". In both scenarios, OSM compares the JMSCorrelationID with the correlation ID set for the task and associates the two messages if the respective values match.

---

**Note:** No correlation configuration is required at the external system that sends the response message.

---

Correlation is of two types: Message Property and XML Body correlation.

In Message Property correlation, you specify a message header as the correlation ID in the outbound OSM message. For example:

```
outboundMessage:setJMSCorrelationID($outboundMessage, $corrID)
```

You can also specify additional message header properties in the outbound message. For example:

```
outboundMessage:setStringProperty($outboundMessage, $HEADER1, $corrID)
```

By default, Message Property correlation uses JMSCorrelationID as the correlation ID. The XML Body correlation uses an XPath expression to retrieve the correlation ID from the body of the XML message.

See "Internal XQuery Sender" and "Internal XSLT Sender" for examples of predefined XQuery and XSLT sender that set correlation ID for the outgoing messages. See "Internal Custom Java Sender" for an example of a custom Java sender that sets the correlation ID for the outgoing message.

**Intercommunication Between Orders in the Same Domain**

There is a special consideration when managing intercommunication between orders, and by extension cartridges that are deployed in the same domain. This situation can occur whenever there are two or more cartridges deployed in the same OSM server that need to communicate with each other.

The automation sender in the child cartridge needs to use the correlation ID specified by the parent order’s task. By default, OSM uses the JMSCorrelationID property in the
message header as the correlation ID. However, if both parent and child task senders
use the same JMSCorrelationID property as the correlation ID, there is a potential
situation where duplicate entries will exist in the OSM database with the same
correlation ID, resulting in an error when the parent receiver tries to look up an
automation context.

The design guideline to handle this is as follows:

- For the parent automation sender, set the JMSCorrelationID header either
  programmatically, or allow the system to auto-generate this value.
- For the child automation sender, set the JMSCorrelationID header to a different
  correlation ID than what the parent task sent, for example by using a different
  algorithm than the one used in the automator for the parent, or allowing the
  system to auto-generate a value. Define a separate custom field in the JMS header
  to contain the correlation ID expected by the parent task.
- For the parent automation receiver, use the message property correlation
  configuration to retrieve the correlation ID from the custom defined JMS header
  field. This will prevent multiple entries with the same correlation ID in the
  database and will allow the parent task to correlate the automation context
  properly.

**About Message Property Selectors**

An automation task may have one or more external event receivers listening on the
JMS queue.

If the automation task has *only one* external event receiver, and you are using the
Optimized build-and-deploy mode to build and deploy cartridges, you do not need to
specify a message property selector. The automation tasks can use the JMS queue
without the need for filter criteria.

You must specify a unique message property selector for the event receiver if any of
the following situations apply:

- If the automation task has *more than one* external event receiver listening on the
  same JMS queue. For example, if you defined multiple automation plug-in
  external event receivers for the same automation task.
- If applications other than OSM share the same queue that an external event
  receiver is listening on.
- If you use the Legacy build-and-deploy mode to build and deploy cartridges.
- If you use the Both (Allow server preference to decide) build-and-deploy mode to
  build and deploy cartridges and configure the Internal dispatch mode for the OSM
  server.

For information on how OSM processes plug-ins according to the build-and-deploy
mode you set, see "**About Dispatch Modes for Automation Plug-ins**". For information
on message property selector filter criteria, see the Design Studio Help.

**About Automation Plug-in Communication Options**

Automated tasks and the automation plug-ins they trigger can handle asynchronous
or synchronous communication. Automated notifications and the automation plug-ins
they trigger can handle asynchronous communication only because an automated
notification can not be defined as external event receiver, so it can not process a
response.
No External Communication: Data Processing Only
You can define an automation as an internal event receiver that extends AbstractAutomator. In this scenario, the input data is coming from OSM and not being sent anywhere, so there is no communication with an external system. The automation plug-in may perform some internal calculation, or just complete the task. Use this scenario for order or task level notifications because notifications do not require responses. You can also use this scenario with automated task plug-ins.

Figure 5–1, "Automation Flow" illustrates this scenario. In the figure, Automation Plug-ins A and B are internal event receivers/automators.

Fire-and-Forget Communication: Message Sent to External Systems
You can define an automation as an internal event receiver that extends AbstractSendAutomator. In this scenario, the input data is coming from OSM and being sent to an external system. The automation plug-in sends an asynchronous "fire-and-forget" message. That is, it completes the task and sends a message to an external system, but does not expect a response back from the external system.

Figure 5–3 illustrates this scenario, which builds on Figure 5–1, "Automation Flow". In the figure, Automation Plug-in A is an internal event receiver/sender.

Synchronous Communication: Single Request and Response
You can define an automated task that defines two automation plug-ins:

- You can define the first automation as an internal event receiver that extends AbstractSendAutomator. In this scenario, the input data is coming from OSM and being sent to an external system. The automation plug-in sends a synchronous message which expects a response back from the external system.

- You can define the second automation as an external event receiver that extends AbstractAutomator. In this scenario, the input data is coming from the external system.
system (it is the response from the message sent by the first automation) and not being sent anywhere. The automation plug-in processes the response and completes the task.

Figure 5–4 illustrates this scenario, which builds upon Figure 5–3. In the figure, Automation Plug-in A-1 is an internal event receiver/sender, and Automation Plug-in A-2 is an external event receiver/automator.

**Figure 5–4  Automation Flow: Simple Synchronous**

---

**Synchronous Communication: Multiple Requests and Responses**

You can define an automated task that defines multiple automation plug-ins:

- You can define the first automation as an internal event receiver that extends AbstractSendAutomator. In this scenario, the input data is coming from OSM and being sent to an external system. The automation plug-in sends a synchronous message which expects a response back from the external system.

- You can define the second automation as an external event receiver that extends AbstractSendAutomator. In this scenario, the input data is coming from the external system (it is the response from the message sent by the first automation) and being sent back to the external system. The automation plug-in processes the response and replies back by sending another message.

- You can define the third automation as an external event receiver that extends AbstractAutomator. In this scenario, the input data is coming from the external system (it is the response from the second message sent by the second automation) and not being sent anywhere. The automation plug-in processes the response and completes the task.
Figure 5–5 illustrates this scenario, which builds upon Figure 5–4. In the figure, Automation Plug-in A-1 is an internal event receiver/sender, Automation Plug-in A-2 is an external event receiver/sender, and Automation Plug-in A-3 is an external event receiver/automator.

There can be multiple exchanges in such a scenario; this is just an example. After some number of messages back and forth, the final automation must be an external event receiver that extends AbstractAutomator, to complete the task. This example shows communication with two different external systems; however, steps 8-13 could continue communications with External System X, rather than with External System Y.

**Figure 5–5  Automation Flow: Complex Synchronous**
Asynchronous Communication: Single or Multiple Requests and Responses

In the synchronous communication scenario one task sends a single message and expects a response in return (see "Synchronous Communication: Single Request and Response"). While the task is waiting for the response to return, the order data associated to that task is not available for amendment processing, effectively blocking any revision order changes or cancelation request involving that task. This scenario is normally not a problem when the response returns quickly but for more asynchronous communication where the message can take a longer time to return, the scenario described in this section is more appropriate so as to avoid unnecessarily long delays in order amendments or cancelation requests.

You can define an automated task that defines a single automation as an internal event receiver that extends AbstractSendAutomator. In this scenario, the input data is coming from OSM and being sent to an external system. The automation plug-in sets a correlation ID and sends a message. In this case, however, OSM expects a response back from the external system but to a different task.

In this scenario, you must programmatically set the correlation ID for the outgoing message in the sending task. You cannot use the OSM auto-generated correlation ID functionality. For more information, see "Correlating Requests from OSM to Responses from External Systems".

You can define the second automated task with two automation plug-ins:

- The first plug-in is an internal event receiver that extends AbstractAutomator. In this scenario, the input data is coming from the previous task that sent the initial message and correlation ID to the external system. The automation plug-in configures the correlation ID to correspond to the correlation ID configured on the previous task so that the message is routed to the right location. In addition, this automator uses the taskContext suspendTask method to transition the task to a new customer defined task state (for example, a state called waitingforresponse) and also has the ability to suspend the task. When a task is in the suspended state, it can be amended.

- The second plug-in is an external event receiver that extends AbstractAutomator. In this scenario, the input data is coming from the response to the message sent by the previous task. When the response arrives, the event automatically transitions the task to a new state (for example, a state called waitForProvisioningCompleted) that moves the task out of the suspended state and completes that task.

Figure 5–6 illustrates this scenario, which is a variant of Figure 5–4. In the figure, Automation Plug-in A-1 is an internal event receiver/sender. Automation plug-in B-1 sets the correlation ID and suspends the task, and Automation Plug-in B-2 is an external event receiver/automator.
You can also apply this asynchronous communication to the synchronous communication scenario where one task sends and receives multiple messages (see "Synchronous Communication: Multiple Requests and Responses"). In Figure 5–5, replace plug-in A-3 with a new task that includes two automation plug-ins that set the expect correlation ID, suspend the task so that the task data can be amended or canceled while it is waiting for the response, and then completes the task when the response returns.
Storing Response Message as XML Type Parameters

When you receive response message from external fulfillment systems, you may want to store response message data on the OSM order. To do this, you can use a parameter that you designate as XML Type in the Design Studio Order editor Order Template tab.

However, you must strip the envelope, header, and body from the response message before storing data in this way. Having XML type data that includes the envelope, header, or body prevents OSM from sending any subsequent Web Service request messages because Web Service message envelopes, headers, or body cannot be nested.

For example, you could receive response data and assign it to a variable, such as $wsResponseDataXmlData. This variable contains the entire response including the Web Service envelope, header, and body. You could use the following code to strip the envelope, header, and body:

**Example 5–1 Stripping the Envelope, Header, and Body**

```plaintext
let $wsResponseContentXmlData := $wsResponseDataXmlData/env:Envelope/env:Body/*
```

The new $wsResponseContentXmlData variable now contains only the content of the body.

About Custom Automation Plug-ins

All custom automation plug-in Java source files must reside in the cartridgeName/src directory. You can create subdirectories within the src directory as needed. When you compile the source file, the resultant Java class file is placed in the cartridgeName/out directory. Any subdirectories you created within the src directory are reflected in the out directory.

All custom automation plug-ins must extend one of the following automation classes, located in the OSM_home/SDK/automation/automationdeploy_bin/automation_plugins.jar file:

- AbstractAutomator
- AbstractSendAutomator

The custom automation plug-in can directly or indirectly extend AbstractAutomator or AbstractSendAutomator: If needed, there can be one or more layers of inheritance between AbstractAutomator or AbstractSendAutomator, and the automation plug-in.

These classes are hierarchically related. AbstractAutomator is the parent of AbstractSendAutomator as shown in Figure 5–7. Both classes reside in the com.mslv.automation.plugin package.
The AbstractAutomator can receive information, either from OSM or from an external system. The AbstractSendAutomator inherits this ability, so it can also receive information from OSM or from an external system; however, it can also send information. If the purpose of the custom automation plug-in you are writing is to send a message, it should extend the AbstractSendAutomator class; otherwise, it should extend the AbstractAutomator class.

Defining the Custom Automation Plug-in

For every custom automation plug-in you write, you must define a corresponding 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 Design Studio entity. For example, if you create MyCustomPlugin.java and compile it, the result is MyCustomPlugin.class. You then create a new Custom Automation Plug-in entity, and populate the fields defined on the editor.

There is a difference between the terms custom automation plug-in and Custom Automation Plug-in: The former is a custom Java class, the latter is a Design Studio entity.

About the XML Template

The Custom Automation Plug-in editor also defines the XML Template field.

You must provide XML that defines the implementation for your custom automation plug-in. This is done through the <implement> element, as shown in Example 5–2. The <implement> element is defined in the cartridgeName/customAutomation/automationMap.xsd file, which is available with the creation of an OSM cartridge, as described in "Package Explorer View of a Cartridge".

Example 5–2 XML Template

```xml
<implement xsi:type="hw:customImplementation"
xmni:hw="http://www.example.org/Hello/World"
xsi:schemaLocation="http://www.example.org/Hello/World helloWorld.xsd">
  <hw:completionStatus>success</hw:completionStatus>
</implement>
```

You must also provide the corresponding schema file that defines the rules for the XML that you entered in the XML Template field. The schema file name in this example is helloWorld.xsd, shown on the third line of Example 5–2. The content of helloWorld.xsd is shown in Example 5–3.
Example 5–3 Schema for XML Template

```xml
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.example.org/hello/world"
    xmlns:tns="http://www.example.org/hello/world"
    elementFormDefault="qualified"
    <import schemaLocation="automationMap.xsd"
</import>
<complexType name="customImplementation">
    <complexContent>
        <extension base="Q1:Implementation">
            <sequence>
                <element name="completionStatus" type="string"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>
</schema>
```

The schema files you create must reside in the `cartridgeName/customAutomation` directory and the `cartridgeName/resources/automation` directory.

---

**Note:** The generated `automationMap.xml` file includes the `<implement>` element for predefined automation plug-ins, but not for custom automation plug-ins. For additional examples of the `implement` element, see Appendix A, "AutomationMap.xml File".

When looking at the examples, note that the sub-elements defined for the `implement` element differ for senders versus automators.

---

About Creating Custom Automation Plug-ins

AbstractAutomator and AbstractSendAutomator each define abstract methods which require child classes to define those methods. The custom automation plug-in must define a specific method, depending on which Java class the custom automation plug-in extends:

- A custom automation plug-in that extends AbstractAutomator must define the method:
  ```java
  public void run(String inputXML, AutomationContext automationContext)
  ```

- A custom automation plug-in that extends AbstractSendAutomator must define the method:
  ```java
  public void makeRequest(String inputXML, AutomationConext automationContext, TextMessage outboundMessage)
  ```

By defining one of these methods in a custom automation plug-in, when an automated task or automated notification is triggered, OSM can process the automation mapping and call the method, knowing it is defined for the class name provided in the automation mapping.

The following sections describe the arguments used in the run and `makeRequest` methods. See "Custom Java Automation Plug-ins" for sample custom automation senders and receivers that illustrate how you can use these arguments.
About Custom Automation Plug-ins

**inputXML Argument**
The `inputXML` argument is a `java.lang.String` object. The custom automation plug-in does not need to include an import statement for this object because it is included in the hierarchy from which the custom automation is extending.

The `inputXML` argument is the input data in XML document form that can be parsed to access the individual pieces of data. If the automation is defined as an internal event receiver, the XML document defines OSM order data. If the automation is defined as an external event receiver, the XML document defines data from an external source. In either case, you need to know the expected XML definition in order to write the code to parse the data.

Data is not stored at the element for a given XML tag; it is stored at its child, so the approach for retrieving order data is not obvious. A command to retrieve order data looks like this:

```java
Element clli_a = root.getElementsByTagName("clli_a").item(0);
String text = clli_a.getFirstChild().getNodeValue();
```

**AutomationContext Argument and Casting the Context Argument**
Within the custom plug-in, you must determine which context object to expect as an argument, and then cast the `AutomationContext` object to the appropriate child context object (for example, `TaskContext` or `OrderNotificationContext`).

For example, in code below, the expected context object is `TaskContext` and `automationContext` is the name of the `AutomationContext` object argument.

```java
if (automationContext instanceof TaskContext) {
    TaskContext taskContext = (TaskContext)automationContext;
} else { //log an error }
```

After the `AutomationContext` object is cast to the appropriate context object, all methods on the context object are available to the custom plug-in. See “About Automations in the Order and Task Contexts” for more information.

**outboundMessage Argument**
The `outboundMessage` argument is a `javax.jms.TextMessage` object. The custom automation plug-in does not need to include an import statement for this object because it is included in the hierarchy from which the custom automation is extending.

The `outboundMessage` argument is defined only for the `makeRequest` method; it is not defined for the `run` method. The `makeRequest` method is defined for classes that extend `AbstractSendAutomator`, which automatically send a message to an external system. You can write custom code that populates `outboundMessage`, which is sent to the external message queue defined by the automation definition. You do not have to write custom code to connect to the external system or send the message; OSM automation handles the connection and the message upon completion of the `makeRequest` method.

**Accessing JDBC from Within an Automation Plug-in**
Because custom automation plug-ins run inside a J2EE container, JDBC services are readily available.

To use JDBC from a plug-in, you must create a data source through the WebLogic console. The data source contains all the connection information for your proprietary database, such as host names, user names, passwords, number of connections, and so on.
For information on setting up data sources in WebLogic, see the overview of WebLogic Server applications development in the Oracle WebLogic documentation.

The following code illustrates how to connect to a proprietary database from OSM and perform a "SELECT *".

```java
javax.naming.InitialContext initialContext = new InitialContext();
javax.sql.DataSource datasource = (javax.sql.DataSource) initialContext.lookup("java:comp/env/jdbc/DataSource");

javax.sql.Connection connection = datasource.getConnection();
javax.sql.Statement statement = connection.createStatement();

javax.sql.ResultSet resultSet = statement.executeQuery("SELECT * FROM my_custom_table");
```

Line two, the lookup, uses the JNDI name of the data source as a parameter.

### Compiling the Custom Automation Plug-in

You must include the following JAR files in your project library list for the custom automation plug-in to compile:

- `WLS_home/wlsserver_10.3/server/lib/weblogic.jar`
- `OSM_home/SDK/automation/automationdeploy_bin/automation_plugins.jar`

**Warning:** The version of the `automation_plugins.jar` that you reference to compile the custom automation plug-in must be the same version that resides in the cartridge `osmlib` directory. To verify this, check the date and size of each file. If they are different, use the version that came with the OSM installation. To do so, copy the `automation_plugins.jar` file from the `OSM_home/SDK/automation/automationdeploy_bin` directory to the `osmlib` directory of your cartridge. After the file is copied to the cartridge, clean and rebuild the cartridge.

Depending on the content of the custom automation plug-in, you may also need to include additional JAR files.

To include a JAR file in the project library list:

1. From the Design Studio menu bar, select **Project**, then select **Properties**. The **Properties for CartridgeName** window opens.
2. In the left navigation pane, click **Java Build Path**.
3. Click the **Libraries** tab.
4. Click **Add External JARs**. The **Jar Selection** window opens.
5. Navigate to the location of the JAR file and double-click the JAR file to add it to the library list.
About Predefined Automation Plug-ins

The OSM installation provides several predefined automation plug-ins, as described in the following sections. The sections are presented in the order that the predefined automation plug-ins display within Design Studio, on the **Add Automation** window **Automation Type** list field.

All of the predefined automation plug-ins are part of the automation class hierarchy; they extend, either directly or indirectly, the AbstractAutomator class that you use to create custom automations, as shown in **Figure 5–8**.

**Figure 5–8 Predefined Automation Plug-ins Class Hierarchy**

---

**Note:** The XSLT and XQuery Automator predefined automation plug-in Java class are XSLTReceiver and XQueryReceiver. The presentation in Design Studio was changed to remove confusion. The names *receiver* and *sender* imply that one receives and one sends, which is not true: Both receive. The sender just has the added ability to send a message.

**XSLT Sender**

The XSLT Sender predefined automation plug-in provides a way to transform data and send it to an external system using JMS, with you supplying the extensible stylesheet language transformation (XSLT).

**Defining the Automation**

When defining the automation on the **Add Automation** window, select **XSLT Sender** from the **Automation Type** list field.

For an automation defined as an internal event receiver, the XSLT must transform the OSM input data to **SystemY** data, where **SystemY** is the external system that the automation is sending the transformed data to.

For an automation defined as an external event receiver, the XSLT must transform **SystemX** data to **SystemY** data, where **SystemX** is the external system that the
automation is receiving input data from, and *SystemY* is the external system that the automation is sending the transformed data to.

See "Internal XSLT Sender" and "External XSLT Sender" for sample code.

**XSLT Tab**

Selecting *XSLT Sender* from the *Automation Type* list field results in *XSLT* tab being present on the *Properties* view for the automation. The *XSLT* tab is where you specify your XSLT file so the predefined automation plug-in can access it. You can specify your XSLT file in one of three ways by choosing the appropriate radio button:

- When you choose **Bundle in**, you can select your XSLT file from a list that displays all XSLT files defined in the cartridge *resources* directory, which populates the *XSLT* field for you.
- When you choose **Absolute path**, you must enter the path and name of your XSLT file in the *XSLT* field.
- When you choose **URL**, you must enter the unified resource locator (URL) to your XSLT file in the *XSLT* field.

---

**Note:** Oracle recommends that you choose **Bundle in** for production mode because it pulls the XSLT files into the 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 choose **Absolute Path** or **URL** for production mode, you can deploy the EAR file to any server but are responsible for ensuring the XSLT files reside in the 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 XSLT.

The XSLTSender class can cache the associated XSLT file, incurring minimal overhead on each invocation. When the automation is defined to cache the XSLT, the implementation detects at runtime whether the XSLT source has changed by checking the URL modification time and the XSLT is automatically reloaded if required. You can configure caching through the *Maximum Number in Cache* and *Cache Timeout* fields.

You can set exceptions for the XSLT processing by setting the *Exception* field. For automations defined on a task, the *Exception* list field provides the values of *success* and *failure*, which are task statuses. If you define additional task statuses, they also appear in the list. (The *Exception* field is not applicable for automations defined on an order.)

Oracle uses Saxon as the transformer factory to process XSLT. You can specify use of a different transformer factory by specifying a value for the *Transformer Factory* field.

---

**Note:** Oracle recommends that you use the default Saxon transformer factory.

**Routing Tab**

The *Routing* tab consists of two sub-tabs: *To* and *Reply To*. Both sub-tabs define the same set of fields. The *To* sub-tab defines where the outbound message is being routed to, and the *Reply To* sub-tab defines where the in-bound message (replying to the
outbound message) is being routed to. You must set the ReplyTo queue on the sender even if you are processing the return message on a different automation plug-in.

Writing the XSLT

When the XSLT transformer is called, it is passed references to the following named parameters that may be used from within the XSLT:

- **Automator**: The class instance (for example, the instance of XSLTSender that is calling the XSLT).
- **Log**: The automator’s instance of org.apache.commons.logging.Log.
- **Context**: The context object input parameter to the makeRequest method.
- **OutboundMessage**: The outbound JMS TextMessage.

XSLTSender does not automatically complete the associated task after successful processing. If the task needs to be completed, the XSLT must include a call to TaskContext.completeTaskOnExit(java.lang.String s)

as shown in Example 5–4:

**Example 5–4  XSLT Java Call**

```xml
<xsl:stylesheet version="1.0"
     xmlns="http://java.sun.com/products/oss/xml/ServiceActivation"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
     xmlns:java="http://xml.apache.org/xslt/java"
     xmlns:xalan="http://xml.apache.org/xslt"
     exclude-result-prefixes="xsl java xalan sa mslv-sa">
    <!-- * -->
    <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="void1" select="java:info($log,'completing task with status success')"/>
        <xsl:variable name="void" select="java:completeTaskOnExit($context,'success')"/>
    </xsl:template>
    <!-- * -->
    <xsl:template match="* | @* | text()">
        <!-- do nothing -->
        <xsl:apply-templates/>
    </xsl:template>
</xsl:stylesheet>
```

As the XSLT author, you must ensure that the context parameter provided to the automation plug-in, and so to your XSLT, is an instance of TaskContext or TaskNotificationContext. This implementation attempts to complete the associated task, if applicable, on processing failure, using the exception status defined in the AutomationMap.xml file.
Steps to Follow When Using XSLT Sender
The following high level-steps describe how to set up the XSLT Sender predefined automation plug-in:

1. Determine the from and to data that your XSLT is to translate.
2. Write the XSLT.
3. Define automated task or automated notification that will trigger the automation plug-in.
4. Define the automation for automated task or automated notification:
   a. Select XSLT Sender from the Automation Type list field.
   b. For an automated task, define the automation as internal or external event receiver.
   c. Populate all applicable automation Properties tabs, including the tabs specific to this type of automation: the XSLT tab and the Routing tab.
5. Build the cartridge.
6. Deploy the cartridge to the OSM server.
7. From within OSM, perform the event that triggers the automation.
8. XSLTSender uses your XSLT to transform the data and send it to the external system specified by the automation definition.

XSLT Automator
The XSLT Automator predefined automation plug-in provides a way to transform data or update OSM with the transformed data, with you supplying the extensible stylesheet language transformation (XSLT).

Defining the Automation
When defining the automation on the Add Automation window, select XSLT Automator from the Automation Type list field.

For an automation defined as an internal event receiver, the scenario is not very plausible because your corresponding XSLT would not need to transform OSM data to OSM data. However, you can write XSLT that executes Java rather than transforms data, so it is possible to define an XSLT Automator as an internal event receiver, but you can accomplish the same thing by writing a custom automation plug-in. The decision on which to use is based on the complexity of the Java code: If it is fairly short and simple, it may be quicker to use the predefined automation plug-in and just write the XSLT, as opposed to writing the custom automation plug-in.

For an automation defined as an external event receiver, your corresponding XSLT must transform SystemX data to OSM data, where SystemX is the external system that the automation is receiving input data from. You can also specify to update OSM with the transformed data.

See "External XSLT Automator" and "Internal XSLT Automator" for sample code.

XSLT Tab
Selecting XSLT Automator from the Automation Type list field results in XSLT tab being present on the Properties view for the automation. The XSLT tab is where you specify your XSLT so the predefined automation plug-in can access it. You can specify your XSLT in one of three ways by choosing the appropriate radio button:
About Predefined Automation Plug-ins

- When you choose **Bundle in**, you can select your XSLT file from a list that displays all XSLT files defined in the cartridge **resources** directory, which populates the XSLT field for you.

- When you choose **Absolute path**, you must enter the path and name of your XSLT file in the XSLT field.

- When you choose **URL**, you must enter the unified resource locator (URL) that locates your XSLT file in the XSLT field.

---

**Note:** Oracle recommends that you choose **Bundle in** for production mode and **Absolute Path** or **URL** for testing mode.

---

The XSLTReceiver class can cache the associated XSLT file, incurring minimal overhead on each invocation. When the automation is defined to cache the XSLT, the implementation detects at runtime whether the XSLT source has changed by checking the URL modification time; the XSLT is automatically reloaded if required. You can configure caching through the **Maximum Number in Cache** and **Cache Timeout** fields.

You can set exceptions for the XSLT processing by setting the **Exception** field. For automations defined on a task, the **Exception** list field provides the values of **success** and **failure**, which are task statuses. If you define additional task statuses, they also appear in the list. (The **Exception** field is not applicable for automations defined on an order.)

Oracle uses Saxon as the transformer factory to process XSLTs. You can specify to use a different transformer factory by specifying a value for the **Transformer Factory** field.

---

**Note:** Oracle recommends that you use the default Saxon transformer factory.

---

When **XSLT Automator** is selected from the **Automation Type** list, the **XSLT** tab also includes the **Update Order** check box, which is not present when **XSLT Sender** is selected from the **Automation Type** list. If the check box is selected, XSLTReceiver updates OSM with the transformed order data. If the check box is deselected, XSLTReceiver just transforms the data; it does not update OSM with the transformed data.

**Writing the XSLT**

When the XSLT transformer is called, it is passed references to the following named parameters that may be used from within the XSLT:

- **Automator**: The class instance (for example, the instance of XSLTReceiver that is calling the XSLT).

- **Log**: The automator’s instance of org.apache.commons.logging.Log.

- **Context**: The context object input parameter to the makeRequest method.

XSLTReceiver does not automatically complete the associated task after successful processing. If the task needs to be completed, the XSLT must include a call to TaskContext.completeTaskOnExit(java.lang.String s)

as shown in Example 5–4, "XSLT Java Call".

---

**Note:** Oracle recommends that you choose **Bundle in** for production mode and **Absolute Path** or **URL** for testing mode.
As the XSLT author, you must ensure that the context parameter provided to the automation plug-in, and so to your XSLT, is an instance of TaskContext or TaskNotificationContext. This implementation attempts to complete the associated task, if applicable, on processing failure, using the exception status defined in the AutomationMap.xml file.

**Steps to Follow When Using XSLT Automator**

The following high-steps describe how to set up the XSLT Automator predefined automation plug-in:

1. Determine the from and to data that your XSLT is to translate.
2. Write the XSLT.
3. Define automated task or automated notification that will trigger the automation plug-in.
4. Define the automation for automated task or automated notification:
   a. Select *XSLT Automator* from the *Automation Type* list field.
   b. For an automated task, define the automation as internal or external event receiver.
   c. Populate all applicable automation *Properties* tabs, including the tab specific to this type of automation; that is, the *XSLT* tab.
5. Build the cartridge.
6. Deploy the cartridge to the OSM server.
7. From within OSM, perform the event that triggers the automation.
8. XSLTAutomator uses your XSLT to transform the data or updates OSM with the transformed data.

**XQuery Sender**

The XQuery Sender predefined automation plug-in provides a way to extract and manipulate XML data and send it to an external system using JMS, with you supplying the XML query (XQuery).

**Defining the Automation**

When defining the automation on the *Add Automation* window, select *XQuery Sender* from the *Automation Type* list field.

For an automation defined as an internal event receiver, your corresponding XQuery can manipulate OSM data and send it to *SystemY*, where *SystemY* is the external system that the automation is sending the manipulated data to.

For an automation defined as an external event receiver, your corresponding XQuery can manipulate *SystemX* data and send it to *SystemY*, where *SystemX* is the external system that the automation is receiving input data from, and *SystemY* is the external system that the automation is sending the manipulated data to.

See "Internal XQuery Sender" and "External XQuery Sender" for sample code.

**XQuery Tab**

Selecting *XQuery Sender* from the *Automation Type* list field results in *XQuery* tab being present on the *Properties* view for the automation. The *XQuery* tab is where you specify your XQuery file so the predefined automation plug-in can access it. You can
specify your XQuery file in one of three ways by choosing the appropriate radio button:

- When you choose **Bundle in**, you can select your XQuery file from a list that displays all XQuery files defined in the cartridge `resources` directory, which populates the XQuery field for you.
- When you choose **Absolute path**, you must enter the path and name of your XQuery file in the XQuery field.
- When you choose **URL**, you must enter the unified resource locator (URL) to your XQuery file in the XQuery field.

**Note:** Oracle recommends that you choose **Bundle in** for production mode and **Absolute Path** or **URL** for testing mode.

The XQuerySender class can cache the associated XQuery file, incurring minimal overhead on each invocation. When the automation is defined to cache the XQuery, the implementation detects at runtime whether the XQuery source has changed by checking the URL modification time; the XQuery is automatically reloaded if required. You can configure caching through the **Maximum Number in Cache** and **Cache Timeout** fields.

You can set exceptions for the XSLT processing by setting the **Exception** field. For automations defined on a task, the **Exception** list field provides the values of **success** and **failure**, which are task statuses. If you define additional task statuses, they also appear in the list. (The **Exception** field is not applicable for automations defined on an order.)

**Routing Tab**

The **Routing** tab consists of two sub-tabs: **To** and **Reply To**. Both sub-tabs define the same set of fields. The **To** sub-tab defines where the outbound message is being routed to, and the **ReplyTo** sub-tab defines where the in-bound message (replying to the outbound message) is being routed to. You must set the ReplyTo queue on the sender even if you are processing the return message on a different automation plug-in.

**Writing the XQuery**

When the XQuery processor is called, it is passed references to the following named parameters that may be used from within the XQuery:

- **Automator**: The class instance (for example, the instance of XQuerySender that is calling the XSLT).
- **Log**: The automator’s instance of org.apache.commons.logging.Log.
- **Context**: The context object input parameter to the makeRequest method.
- **OutboundMessage**: The outbound JMS TextMessage.

XQuerySender does not automatically complete the associated task after successful processing. If the task needs to be completed, the XQuery must include a call to `TaskContext.completeTaskOnExit(java.lang.String s)` as shown in Example 5–4, "XSLT Java Call".

As the XQuery author, you must ensure that the context parameter provided to the automation plug-in, and so to your XQuery, is an instance of TaskContext or TaskNotificationContext. This implementation attempts to complete the associated
About Predefined Automation Plug-ins

Steps to Follow When Using XQuery Sender

The following high-steps describe how to set up the XQuery Sender predefined automation plug-in:

1. Determine the from and to data that your XQuery is to manipulate.
2. Write the XQuery.
3. Define automated task or automated notification that will trigger the automation plug-in.
4. Define the automation for automated task or automated notification:
   a. Select XQuery Sender from the Automation Type list field.
   b. For an automated task, define the automation as internal or external event receiver.
   c. Populate all applicable automation Properties tabs, including the tabs specific to this type of automation: the XQuery tab and the Routing tab.
5. Build the cartridge.
6. Deploy the cartridge to the OSM server.
7. From within OSM, trigger the automation.
8. XQuerySender uses your XQuery to manipulate the data and send it to the external system specified by the automation definition.

XQuery Automator

The XQuery Automator predefined automation plug-in provides a way to manipulate data or update OSM with the manipulated data, with you supplying the XML Query (XQuery).

Defining the Automation

When defining the automation on the Add Automation window, select XQuery Automator from the Automation Type list field.

For an automation defined as an internal event receiver, your corresponding XQuery can manipulate the OSM input data or specify to update OSM with the manipulated data.

For an automation defined as an external event receiver, your corresponding XQuery can manipulate the SystemX input data, where SystemX is the external system that the automation is receiving input data from. You can also specify to update OSM with the manipulated data.

See "External XQuery Automator" and "Internal XQuery Automator" for sample code.

XQuery Tab

Selecting XQuery Automator from the Automation Type list field results in XQuery tab being present on the Properties view for the automation. The XQuery tab is where you specify your XQuery so the predefined automation plug-in can access it. You can specify your XQuery in one of three ways by choosing the appropriate radio button:
When you choose **Bundle in**, you can select your XQuery file from a list that displays all XQuery files defined in the cartridge **resources** directory, which populates the **XQuery** field for you.

- When you choose **Absolute path**, you must enter the path and name of your XQuery file in the **XQuery** field.

- When you choose **URL**, you must enter the unified resource locator (URL) that locates your XQuery file in the **XQuery** field.

The XQueryReceiver class can cache the associated XQuery file, incurring minimal overhead on each invocation. When the automation is defined to cache the XQuery, the implementation detects at runtime whether the XQuery source has changed by checking the URL modification time; the XQuery is automatically reloaded if required. You can configure caching through the **Maximum Number in Cache** and **Cache Timeout** fields.

You can set exceptions for the XSLT processing by setting the **Exception** field. For automations defined on a task, the **Exception** list field provides the values of **success** and **failure**, which are task statuses. If you define additional task statuses, they also appear in the list. (The **Exception** field is not applicable for automations defined on an order.)

When **XQuery Automator** is selected from the **Automation Type** list, the **XQuery** tab also includes the **Update Order** check box, which is not present when **XQuery Sender** is selected from the **Automation Type** list. If the check box is selected, XQueryReceiver updates OSM with the manipulated data; if the check box is deselected, XQueryReceiver just manipulates the data; it does not update OSM with the manipulated data.

### Writing the XQuery

When the XQuery transformer is called, it is passed references to the following named parameters that may be used from within the XQuery:

- **Automator**: The class instance (for example, the instance of XQueryReceiver that is calling the XSLT).

- **Log**: The automator’s instance of org.apache.commons.logging.Log.

- **Context**: The context object input parameter to the makeRequest method.

XQueryReceiver does not automatically complete the associated task after successful processing. If the task needs to be completed, the XQuery must include a call to `TaskContext.completeTaskOnExit(java.lang.String s)` as shown in Example 5–4, "XSLT Java Call".

As the XQuery author, you must ensure that the context parameter provided to the automation plug-in, and so to your XQuery, is an instance of TaskContext or TaskNotificationContext. This implementation attempts to complete the associated task, if applicable, on processing failure, using the exception status defined in the AutomationMap.xml file.

### Steps to Follow When Using XQuery Automator

The following high-steps describe how to set up the XQuery Automator predefined automation plug-in:

1. Determine the from and to data that your XQuery is to manipulate.

2. Write the XQuery.
3. Define automated task or automated notification that will trigger the automation plug-in.

4. Define the automation for automated task or automated notification:
   a. Select *XQuery Automator* from the *Automation Type* list field.
   b. For an automated task, define the automation as internal or external event receiver.
   c. Populate all applicable automation *Properties* tabs, including the tab specific to this type of automation; that is, the *XQuery* tab.

5. Build the cartridge.

6. Deploy the cartridge to the OSM server.

7. From within OSM, trigger the automation.

8. XQueryReceiver uses your XQuery to manipulate the data or update OSM with the manipulated data.

**DatabasePlugin**

The DatabasePlugin class is a predefined automation plug-in that provides a way to interact with external databases, with you supplying the SQL and stored procedures to query and update a database. The automation plug-in can also be configured to update OSM with data returned from an external database.

DatabasePlugin is slightly different from the previously described predefined automation plug-ins, in that the input is not accessed through a file. Rather, the input is accessed through the XML Template field on the Custom Automation Plug-in editor. Because this predefined automation plug-in requires the use of the XML Template field, it must be defined as a Custom Automation Plug-in. As a result, DatabasePlugin does not appear in the Automation Type list field on the Add Automation window like the other predefined automation plug-ins do.

---

**Note:** The OSM installation provides samples of the DatabasePlugin predefined automation plug-in, located in the SDK/Samples/DatabasePlugin directory.

---

**Defining the Custom Automation Plug-in**

To define the Custom Automation Plug-in for the DatabasePlugin predefined automation plug-in, set the Class field by selecting *DatabasePlugin*. The DatabasePlugin class is located in the \OSM_home\SDK\automation\automationdeploy_bin\automation_plugins.jar file, which comes with your OSM installation.

**XML Template**

The XML Template field consists of one or more statements defined under the root <implementation> element. A statement may update the database, or update OSM order data, or both. All statements share the following characteristics:

- May contain SQL or stored procedure calls.
- May have one or more parameters.
- May return one or more result sets, either as a result of a database query or through a stored procedure OUT parameter.
May contain one or more bind paths.

May be configured to handle database exceptions in an implementation appropriate manner.

May run as a single transaction or in a group.

SQL statements are specified by the `<sql>` element and stored procedure statements are specified by the `<call>` element. The format of the `call` element is expected to be of the form `{ ? = call <procedure-name> [...]}` or `{call <procedure-name> [...]}`. Parameters are declared with the `?` character.

Example 5–5 and Example 5–6 show the SQL statement and the stored procedure call.

**Example 5–5 SQL Statement**

```xml
<implementation xmlns="http://www.oracle.com/Provisioning/database/DatabasePlugin/2006/02/28" ...>
...
<query>
  <sql>SELECT 'dummy' FROM dual</sql>
...
</query>
</implementation>
```

**Example 5–6 Stored Procedure Call**

```xml
<implementation xmlns="http://www.oracle.com/Provisioning/database/DatabasePlugin/2006/02/28" ...>
...
<update>
  <call>{call a_stored_procedure(?)}</call>
...
</update>
</implementation>
```

**Transaction Element**

The `<transaction>` element allows statements to be grouped. All statements contained in a `<transaction>` element will be run as part of a single database transaction. If a statement is defined outside of the `<transaction>` element, it is auto-committed immediately after the statement completes. The available configuration parameters are:

- **dataSource**: Mandatory. Specifies the JNDI name of the SQL data source used to create the database connection for the transaction. This data source must be defined in your WebLogic domain before the plug-in is called.

  **Note**: Do not configure the data source to support global transactions. The plug-in instance is called under an enclosing transaction, making this option illegal.

- **isolationLevel**: Optional. Specifies the transaction isolation level. Valid values are `READ_COMMITTED`, `READ_UNCOMMITTED`, `REPEATABLE_READ`, and `SERIALIZABLE`. `READ_UNCOMMITTED` and `REPEATABLE_READ` are not supported by Oracle.

- **scrollType**: Optional. Specifies the type of result sets to be created as part of the transaction. Valid values are `FORWARD_ONLY`, `SCROLL_SENSITIVE`, and `SCROLL_`
INSENSITIVE. The SCROLL values apply only when more than one ResultSet definition is defined for the same result set.

- **update**: A statement that updates the database, but does not return results.
- **query**: A statement that queries the database for information. The returned results are used to update the order data.

### Example 5–7 Transaction Definition

```xml
<implementation xmlns="http://www.oracle.com/Provisioning/database/DatabasePlugin/2006/02/28" ...>
  <transaction isolationLevel="READ_COMMITTED" scrollType="SCROLL_INSENSITIVE">
    <dataSource>test/dbplugin/datasource</dataSource>
    <query>
      <sql>SELECT 'dummy' FROM dual</sql>
      <resultSet>
        <column number="1" path="/path/to/p6/field"/>
      </resultSet>
    </query>
  </transaction>
</implementation>
```

### Bind Path

The `<bind path>` element provides a way to correlate outbound parameter values and in-bound result set column values. Instances of this result column will be bound to instances of the specified parameter at the specified path; after which their paths may diverge. This attribute is only relevant when a parameter’s path includes a multi-instance group element.

Consider the sample OSM order data shown in Example 5–8 and the corresponding plug-in configuration in Example 5–9.

### Example 5–8 OSM Order Data

```xml
<employees>
  <employee>
    <name>William</name>
    <job/>
  </employee>
  <employee>
    <name>Mary</name>
    <job/>
  </employee>
</employees>
```

### Example 5–9 Plug-in Definition Using a Bind Path

```xml
<implementation xmlns="http://www.oracle.com/Provisioning/database/DatabasePlugin/2006/02/28" ...>
  <query>
    <sql>SELECT job FROM employee WHERE name = ?</sql>
    <bindPath id="emp" path="/employee[2]"/>
    <parameter xsi:type="ProvisioningParameterType" bindPathRef="emp" path="name" type="text="/>
    <resultSet appliesTo="1" appliesToRow="all">
      <column number="1" bindPathRef="emp" path="job" updateOnMatch="true"/>
    </resultSet>
  </query>
</implementation>
```
The emp bind path selects the second employee (with name of Mary). This bind path is used as the basis for the parameter selection and the corresponding result set column value, ensuring the job field that gets updated is the job corresponding with the employee named Mary.

**Parameter**

The `<parameter>` element defines how values are bound to the SQL parameter declarations. Parameters must be defined in the order of the corresponding declarations.

**OSMParameterType**

Specifies a parameter, the value of which will be bound to a `<sql>` or `<call>` statement. Parameters are processed in the order they are declared. The available parameter configuration attributes are:

- `bindPathRef`
- `path`
- `type`
- `mode`

`bindPathRef` and/or `path` provide the value that will be set on the SQL parameter; `type` provides the data type of the value; `mode` specifies whether the parameter is a stored procedure IN, OUT, or INOUT parameter. Each attribute is described in more detail in the sections that follow.

**bindPathRef**: This is the ID value of a bind path defined elsewhere on the statement. Either `bindPathRef`, `path`, or both may be specified. The value bound to the SQL parameter depends on the result of the evaluation of the bind path's XPath expression, as described in the table.

<table>
<thead>
<tr>
<th>XPath Result</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>null</td>
<td>If path is not specified, the SQL parameter is set to null. If path is specified, the SQL parameter is set based on the path evaluation as described below.</td>
</tr>
</tbody>
</table>
| Node-set     | If path is not specified, the SQL parameter is set according to the following algorithm:

  The first node encountered in the node-set is selected.
  If the node is an XML element, the text contained directly under the element is selected as a String (if none, the SQL parameter is set to null).
  If the node is an XML attribute, the value of the attribute is selected as a String.
  Otherwise, the node itself (as a Java Object) is selected.

  The parameter value is set using the selected data based on the parameter's type (see Table 5–4, "OSM Data Type to SQL Data Type Mapping"). |
| Object       | The parameter value is set using the selected data based on the parameter's type (see Table 5–4, "OSM Data Type to SQL Data Type Mapping"). |
**Path**: The XPath selector in the `path` attribute is evaluated against the plug-in’s input data in order to determine the SQL parameter’s value. The context node against which the path expression is evaluated depends on the format of the input data and whether or not `bindPathRef` evaluated to a `node-set` of XML Elements. If the `bindPathRef` evaluated to a `node-set` of `Elements`, the first encountered `Element` is used as the context node for the `path` expression. If the input is an OSM `GetOrder.Response` document, the context node is the `_root` element of the document. Otherwise, the context node is the document root element. The value bound to the SQL parameter depends on the result of the evaluation of the `path`’s XPath expression, as described in Table 5–3.

<table>
<thead>
<tr>
<th>XPath Result</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>null</td>
<td>The SQL parameter is set to null.</td>
</tr>
<tr>
<td>Node-set</td>
<td>The SQL parameter is set according to the following algorithm: The first node encountered in the node-set is selected. If the node is an XML <code>Element</code>, the text contained directly under the <code>Element</code> is selected as a <code>String</code> (if none, the SQL parameter is set to null). If the node is an XML <code>Attribute</code>, the value of the <code>Attribute</code> is selected as a <code>String</code>. Otherwise, the node itself (as a <code>java Object</code>) is selected. The parameter value is set using the selected data based on the parameter’s type (see Table 5–4).</td>
</tr>
<tr>
<td>Object</td>
<td>The parameter value is set using the selected data based on the parameter’s type (see Table 5–4).</td>
</tr>
</tbody>
</table>

**Type**: Specifies the data type of the parameter, which are OSM specific. Valid values are: `boolean`, `currency`, `date`, `dateTime`, `numeric`, `phone`, and `text`. Table 5–4 shows the SQL data type that will be used to set the SQL parameter based on the specified type and the Java class of the parameter value.

<table>
<thead>
<tr>
<th>Type Attribute Value</th>
<th>SQL Data Type¹</th>
<th>Parameter Evaluation²</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td><code>Boolean</code></td>
<td>Evaluated according to <code>java.lang.Boolean.parseBoolean()</code> using the String value of the parameter. OSM values Yes and No are also supported.</td>
</tr>
<tr>
<td>currency</td>
<td><code>double</code></td>
<td>Evaluated according to <code>java.lang.Double.parseDouble()</code> using the String value of the parameter.</td>
</tr>
<tr>
<td>numeric</td>
<td><code>double</code></td>
<td>Evaluated according to <code>java.lang.Double.parseDouble()</code> using the String value of the parameter.</td>
</tr>
<tr>
<td>date</td>
<td><code>date</code></td>
<td>The String value of the parameter is expected to match the format <code>yyyy-MM-dd</code>.</td>
</tr>
<tr>
<td>dateTime</td>
<td><code>timestamp</code></td>
<td>The String value of the parameter is expected to match the format <code>yyyy-MM-dd'T'HH:mm:ss z</code>.</td>
</tr>
<tr>
<td>phone</td>
<td><code>string</code></td>
<td>Evaluated according to <code>java.lang.String.valueOf()</code>.</td>
</tr>
</tbody>
</table>
mode: Specifies the mode of the parameter. Valid values are IN, OUT, and INOUT. Applicable only if the statement is a prepared statement, that is, defined with <call>.

Exception

The exception statement specifies the behavior that the plug-in should exhibit when a particular Java exception is caught during processing. Exceptions can be ignored or they can complete the associated task with a particular exit status.

If the exception is an instance of java.sql.SQLException, behavior may be further constrained to exceptions that have a particular error code or SQL state value. Exception handlers are evaluated in document order; that is, the first exception handler that matches the thrown exception will be used. If no exception handler exists for a thrown exception, it will be wrapped in a com.mslv.oms.automation.plugin.JDBCPluginException and re-thrown.

Creating the JDBC Data Source

The Database Plug-in must be associated with a JDBC data source that:

- Uses a non-XA database drive
- Does not support global transactions (Supports Global Transactions is check box that is available when defining the WebLogic data source configuration).

When creating the JDBC data source:

- Create a JDBC Data Source that refers to the schema under which you are running the scripts.
- The provided Database Plug-in sample assumes that the JNDI name of this Data Source is demo/dbplugin/datasource. However, the Data Source can have any JNDI name, but the configuration XML files in the config directory needs to be updated appropriately.
- For Database Type, select Oracle.
- For Database Driver, select Oracle's Driver @ (Thin) Versions: 9.0.1, 9.2.0, 10.

Deselect the Supports Global Transactions check box. (This check box defaults to being selected, so you must deselect it.)

Exception

If you create a JDBC data source that uses an XA database drive or that supports global transactions, the DatabasePlugin implementation throws the exception shown in Example 5–10.

Example 5–10 Exception

An automation exception has occurred at AutomationDispatcherImpl.runAutomator:/automation/plugin/internal/task/database_plugin_demo/1.0/get_employee_names/do.
The reason is:
com.mslv.oms.automation.AutomationException:
com.mslv.oms.automation.AutomationException:
com.mslv.oms.util.jdbc.exception.UncategorizedSQLException:
Unable to commit transaction.
com.mslv.oms.automation.AutomationException:
com.mslv.oms.automation.AutomationException:
com.mslv.oms.util.jdbc.exception.UncategorizedSQLException:
Unable to commit transaction.
at com.mslv.oms.automation.plugin.AutomationEventHandlerImpl.a(Unknown Source)
at com.mslv.oms.automation.plugin.AutomationEventHandlerImpl.processMessage
(Unknown Source)
at com.mslv.oms.automation.AutomationDispatcher.onMessage(Unknown Source)
at weblogic.ejb.container.internal.MDListener.execute(MDListener.java:429)
at weblogic.ejb.container.internal.MDListener.transactionalOnMessage
(MDListener.java:335)
at weblogic.ejb.container.internal.MDListener.onMessage(MDListener.java:291)
at weblogic.jms.client.JMSSession.onMessage(JMSSession.java:4060)
at weblogic.jms.client.JMSSession.execute(JMSSession.java:3953)
at weblogic.jms.client.JMSSession$UseForRunnable.run(JMSSession.java:4467)
at weblogic.work.ExecuteRequestAdapter.execute(ExecuteRequestAdapter.java:21)
at weblogic.kernel.ExecuteThread.execute(ExecuteThread.java:145)
at weblogic.kernel.ExecuteThread.run(ExecuteThread.java:117)

About Compensation for Automations

The following sections describe how automations can be configured for compensation.

About Execution Modes for Automations

Internal event receiver sender automations triggered from tasks can be run in different execution modes in compensation scenarios. When the task is in a particular execution mode, only those sender automations configured with the corresponding execution mode can run. For example, a task may have three automations, one of which is a sender configured to send messages to external systems in both do and redo mode, with a corresponding automator that receives the responses from those messages. A third sender plug-in could be required for cancelation scenarios or if the task were no longer required in the process flow. This sender would send a cancelation request to the external system that would cancel any of the do or redo operations that had previously occurred on the external system. The response would be returned to the automator plug-in that contains code that can handle any do, redo, or undo request and transition the task as appropriate.

About Automations that Update Order Data and Compensation Analysis

When a revision order triggers compensation analysis for an order, an automation that updates order data may potentially be data included in compensations.

Any update order data changes triggered from automations with TaskContext or TaskNotificationContext objects, regardless of whether the task can be run in different execution modes, participates in task level compensation analysis. Figure 5–9 illustrates how an update order run during the base order processing of Task A is included in the historical perspective of revision 1 Task A.
Any update order data changes triggered from automations with OrderNotificationContext or OrderDataChangeNotificationContext objects do not participate in task level compensation analysis and OSM does not include them in the contemporary or historical perspectives. Nevertheless, OSM includes these data updates into the current perspective and OSM adds the changes to the closest task instances that are created or completed when the data changes occur.

OSM guarantees the accuracy of these data perspectives for the data update changes done in the task context according to the definitions of these data perspectives. Because the update order data changes in the order context are not associated with a specific task, OSM cannot deterministically guarantee that the compensation perspectives (COP or HOP) will reflect the data changes in at the order context consistently and deterministically.

**About Using GetOrder Responses to View Compensation Perspectives**

During the fulfillment process, an order may fail (also known as fallout) for reasons such as insufficient data or incorrect data. You may have to revise the order data to fix the fallout. If there are multiple revisions on the order, you may need access to previous versions of it so you can provide the information required to roll back the order to the corresponding successful state rather than rolling it back to the previous successful state.

Using GetOrder’s TaskExecutionHistory and OrderChangeID elements, you can obtain the order data for all the revisions that happened on an order and use the relevant data in the fulfillment process according to your needs. The GetOrder.Response and GetOrder.Request XML APIs also include these elements and are included with OSM Automation plug-ins.

For example, consider an order which has been revised three times. You can obtain order data of all the three revisions and use the required data for the fulfillment.

*See OSM XML API Developer’s Guide* for more information about these elements.
Use the `GetOrder` function to retrieve the `TaskExecutionHistory` element which returns an `OrderChangeID` associated with each historical perspective.

The following sample code snippet provides the syntax for the `GetOrder` function:

```xml
let $taskData := fn:root(automator:getOrderAsDOM($automator))/oms:GetOrder.Response
let $orderChangeID := $taskData/oms:TaskExecutionHistory/oms:Task[1]/oms:OrderChangeID/text()
let $prevTaskData := fn:root(automator:getOrderAsDOM($automator, $orderChangeID))/oms:GetOrder.Response
```

In the example above, the `OrderChangeID` specifies the revision to look for and roll back. An `OrderChangeID` with a value 0 indicates that it is the original base order with no revisions.

**About Large Orders and Automation Plug-ins**

The following sections provide information about developing and managing automation for large orders.

**Limiting Automation Concurrency in Large Orders**

OSM is designed to provide high levels of order processing concurrency. In most OSM solutions, this high level of concurrency (when coupled with proper system tuning such as database connections, WebLogic Server threads, and so on) is effective at maximizing OSM scalability and performance. However, in some cases, especially when orders are very large or the associated automation plug-in transactions are complex and lengthy, you may need to limit the number of automation plug-in instances that can run at one time. You can restrict the number of automation plug-ins that run concurrently using order automation concurrency control (OACC) policy files (`automationConcurrencyModel.xml`).

To create an OACC policy file:

1. In Design Studio, create a file called `automationConcurrencyModel.xml` and add the file to the resource folder of the cartridge you want the OACC policy to apply to.

2. Add the following snippet to the file after replacing the placeholders:

   ```xml
   <automationConcurrencyModel
   xmlns="http://xmlns.oracle.com/communications/ordermanagement/model">
   <automationConcurrencyPolicy name="name">
     <targetPlugins>
       <cartridgeNamespace>namespace</cartridgeNamespace>
       <cartridgeVersion>version</cartridgeVersion>
       <pluginSelector>pluginSelector</pluginSelector>
     </targetPlugins>
     <scope>scope</scope>
     <concurrencyLevel>concurrencyLevel</concurrencyLevel>
   </automationConcurrencyPolicy>
   </automationConcurrencyModel>
   ```

   where:

   - `name`: A policy name. Within the `automationConcurrencyModel.xml` file you can specify one or more automation concurrency policies. Each policy can be specified within the `automationConcurrencyPolicy` element.
You can use the optional child elements within the `targetPlugins` element to specify plug-ins contained in the `automationMap.xml` files in deployed cartridges or found on the OSM system class path. OSM must match all specified criteria before applying a policy. If no criteria are specified, OSM applies the policy to all deployed plug-ins.

- **namespace**: The value for this field must be a valid cartridge namespace where the automation plug-ins are located.

- **version**: The value for this field cartridge version.

- **pluginSelector**: The value for this field is an XPath 1.0 selector. The context is the `automationMap.xml` file, which defines every automation plug-in associated with a specific cartridge.

For example, the following selector matches all automation plug-ins from a cartridge with namespace foo and version 1.2 that are also external receivers with a receive/jmsSource element).

```
.[cartridgeNamespace="foo"] [cartridgeVersion="1.2.3.4.5"] [count(receive/jmsSource)>0]).
```


- **scope**: A value that specifies the scope of the policy. The values are:

  - **ORDER_ID**: The policy applies to every order on each OSM managed server. This scope is appropriate if you want to limit the degree of automated transactions that can run in parallel within a given order, but do not want to restrict how many separate orders could be running concurrently.

  - **CARTRIDGE_AND_VERSION**: The policy applies to a specific cartridge and version. The policy limits the maximum number of concurrent automated transactions that can occur across all orders from the same cartridge namespace and version. This scope is appropriate if you want to limit how many orders can have transactions concurrently processing that were created from within the same version of the same cartridge.

  - **CARTRIDGE**: The policy applies to a specific cartridge regardless of version. This scope is appropriate if you want to limit how many orders can have transactions concurrently processing that were created from a cartridge with the same namespace regardless of version.

  - **SERVER**: The policy applies to an entire server. The policy limits the maximum number of concurrent automated transactions that can occur across all orders in any one server regardless of cartridge namespace or version. This scope is appropriate if you want to limit how many orders can be processing on any one managed server regardless of the cartridge namespace and version that they are created from.

If plug-ins from two cartridge versions were targeted then there would be two group instances (cartridge X version 1, cartridge X version 2).

- **concurrencyLevel**: A numerical value specifying the maximum concurrency for each managed server that is allowed within the defined scope. A value of 1 or higher limits concurrency to the specified level within the scope. A value of 0 or less means unlimited concurrency (effectively disabling the policy).

3. Save and close the file.
4. Build the cartridge.
5. Deploy the cartridge.

**Note:** You can validate that the OACC policy was applied by verifying the WebLogic server `domain_home/servers/servername/logs` files (where `domain_home` is the directory that contains the configuration for the domain into which OSM is installed, and `servername` is the server whose logs you are checking). Details about deployed OACC policies are listed in the automation plug-in deployment summary.

For example, the following policy limits each order to run one automation plug-in at a time:

```
<?xml version="1.0"?>
<automationConcurrencyModel
xmlns="http://xmlns.oracle.com/communications/ordermanagement/model">
<automationConcurrencyPolicy name="name">
  <targetPlugins>
    <pluginSelector>starts-with(./ejbName,'UpdateOACC')</pluginSelector>
  </targetPlugins>
  <scope>ORDER_ID</scope>
  <concurrencyLevel>1</concurrencyLevel>
</automationConcurrencyPolicy>
<automationConcurrencyPolicy name="policymultithread">
  <targetPlugins>
    <pluginSelector>starts-with(./ejbName,'UpdateMultiThread')</pluginSelector>
  </targetPlugins>
  <scope>ORDER_ID</scope>
  <concurrencyLevel>3</concurrencyLevel>
</automationConcurrencyPolicy>
</automationConcurrencyModel>
```

### Using GetOrder and UpdateOrder API Functions in Large Orders

When you design automation plug-ins or interact with OSM from external applications, you can implement XML API or Web Service GetOrder operations with the OrderDataFilter element that explicitly specifies which parts of the order to return data from. This can enhance performance in cases where orders are very large and complex with hundreds of order items and where returning the complete order in a response would be costly in terms of CPU and memory usage. For example, in many cases, an automation plug-in already has advanced knowledge of an order item line ID which you can use with the OrderDataFilter to specify the exact line ID you want to return data for.


When you use automation plug-ins or external clients, you can create XML API or Web Service UpdateOrder requests with a ResponseView that specifies the order data to be returned in an UpdateOrder response. This ResponseView behaves in the same way as a GetOrder request. You can use the OrderDataFilter with the ResponseView to further specify the returning data. If the response includes a fulfillment state update, then OSM automatically filters the response so that only order items and order components

---

**Note:** You can validate that the OACC policy was applied by verifying the WebLogic server `domain_home/servers/servername/logs` files (where `domain_home` is the directory that contains the configuration for the domain into which OSM is installed, and `servername` is the server whose logs you are checking). Details about deployed OACC policies are listed in the automation plug-in deployment summary.
impacted by the fulfillment state update are included. This auto-filtering of fulfillment state updates in responses avoids expensive XQuery processing within OSM to determine impacted order item and order component fulfillment states. The ResponseView does this by automatically applying an OrderDataFilter from within the OSM Server which can more efficiently perform this filtering action and also avoids having to serialize and parse large amounts of XML not needed by the requesting client or automation plug-in logic.

In addition, you can use the ExternalFulfillmentStates nodes within an XML API or Web Service UpdateOrder to directly update order item fulfillment states. This optional approach improves order processing efficiency because you no longer need complicated XQuery logic to determine the impact of the external fulfillment state change on an order component and order item.


About Creating Automations in Design Studio

The following sections describe Design Studio tasks involved in creating automations.

About Dispatch Modes for Automation Plug-ins

When you build and deploy your OSM cartridges in Design Studio, you can configure a build-and-deploy mode for automation plug-ins included in the cartridge. Design Studio uses the build-and-deploy mode to build and deploy the automation components OSM requires to process the automation plug-ins in the mode you prefer. A dispatch mode setting on the OSM server works in conjunction with the build-and-deploy mode configured in Design Studio in certain cases. The build-and-deploy mode configured in Design Studio controls building and deploying the automation components required for each mode while the dispatch mode setting on the server controls specifying the preferred mode OSM uses at run-time.

The Optimized dispatch mode runs all automation plug-ins inside the oms.ear file. Running all automation plug-ins in oms.ear improves the performance of processing of automated tasks and improves the performance of build and deployment of cartridges with automated tasks; it is the default mode for OSM 7.0.3.

The Legacy dispatch mode runs each automation plug-in included in a cartridge in its own EAR file and is the dispatch mode used for OSM 7.0.2 and earlier releases; this method of building and deploying automation plug-ins is now referred to as the Legacy mode but it is simply the manner in which automation plug-ins were deployed and executed prior to OSM 7.0.3.

Oracle strongly recommends moving automation plug-ins to Optimized mode. Legacy mode is deprecated. To use Optimized mode, you must upgrade your cartridges from OSM 7.0.2 and earlier.
**Tip:** The build-and-deploy mode you set for automation plug-ins in Design Studio indicates whether you want the plug-ins to be able to run within the `oms.ear` file or not. The dispatch mode on the OSM server indicates the ability for OSM to invoke an automation plug-in that is running within the bounds of the `oms.ear` file. If you want the server dispatch mode setting to take precedence, set the Design Studio build-and-deploy mode to Both.

Figure 5-10 illustrates that when automation plug-ins are built and deployed using Legacy mode, internal event receiver type plug-ins are contained within a J2EE application (.ear file) that is distinct from the OSM application (`oms.ear` file).

**Figure 5-10 Legacy Mode Dispatch of Automation Plug-ins**

![Legacy Mode Dispatch Diagram](image)

- JMS listener in figure is `<taskname>.<automation_pluginname>_Automation_MDB`

Figure 5-11 illustrates that when automation plug-ins are built and deployed using Optimized mode, internal event receiver type plug-ins run within the OSM application. Therefore, internal dispatch automation plug-ins no longer require their own J2EE application. The figure also illustrates that with Optimized mode, the business logic of external event receiver type plug-ins is also run within the OSM application and only the automation framework of external event receiver type plug-ins requires its own J2EE application to listen on the external message queue.
External event receiver type automation plug-ins always require their own J2EE application in order to listen on a JMS destination. When you use Optimized mode, all of the business logic for external event receiver type plug-in J2EE applications is executed within the OSM application and they need to be rebuilt only when the JNDI name of the JMS destination changes.

External event receiver type automation plug-ins are made up of both:

- The automation framework (OSM infrastructure) which receives and prepares the incoming message so that it can be executed according to your business logic. The automation framework subscribes to the external message queue (JMS destination) and requires its own J2EE application in order to listen on the external message queue.

- The business logic itself which determines how the incoming message will be processed (for example, XQuery, XSLT, and custom Java class).

When you use Legacy mode, the J2EE application of an external event receiver type automation plug-in contains both the automation framework and the business logic for handling the incoming message.

When you use Optimized mode, the J2EE application of an external event receiver type automation plug-in contains only the minimum amount of automation framework infrastructure that allows it to listen on the external message queue and forward the message to the core OSM application logic. This means the business logic of the automation plug-in is now executed within the OSM application which enables it to more efficiently interact with the core OSM application logic. The automation framework acts primarily to forward the message to OSM. By containing the business logic within the OSM application, the only time you need to rebuild an external event receiver type automation plug-in is when you decide to use a different external message queue (when the JNDI name of the JMS destination changes).

Oracle recommends moving automation plug-ins to use the Optimized dispatch mode. For information on upgrading your cartridges to use the Optimized mode, see the upgrade impacts section of the OSM Installation Guide.
You can set the automation plug-in build-and-deploy mode for all cartridges in the same workspace or for individual cartridges.

By setting the Both (Allow server preference to decide) build-and-deploy mode, Design Studio builds and deploys the automation components required to process the automation plug-in in both Optimized and Legacy modes. In this case, OSM uses the dispatch mode defined on the OSM server to process the automation plug-in. For information on how to set the dispatch mode on the OSM server, see "Setting Dispatch Modes for Automation Plug-ins".

Table 5–5 summarizes the dispatch mode OSM uses at runtime based on how the automation plug-in build-and-deploy mode and the OSM server dispatch mode are set:

<table>
<thead>
<tr>
<th>Automation Plug-in Build-and-Deploy Mode</th>
<th>OSM Server Dispatch Mode</th>
<th>Effective Mode Used at Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimized</td>
<td>Legacy</td>
<td>Optimized</td>
</tr>
<tr>
<td>Optimized</td>
<td>Internal</td>
<td>Optimized</td>
</tr>
<tr>
<td>Legacy</td>
<td>Legacy</td>
<td>Legacy</td>
</tr>
<tr>
<td>Legacy</td>
<td>Internal</td>
<td>Legacy</td>
</tr>
<tr>
<td>Both</td>
<td>Legacy</td>
<td>Legacy</td>
</tr>
<tr>
<td>Both</td>
<td>Internal</td>
<td>Optimized</td>
</tr>
</tbody>
</table>

For information on setting automation plug-in build-and-deploy modes, see "Setting Dispatch Modes for Automation Plug-ins" and the Design Studio Help.

**Setting Dispatch Modes for Automation Plug-ins**


You set a server dispatch mode on the OSM server in the OSM configuration entry com.mslv.oms.automation.AutomationDispatcher.DefaultAutomationPluginDispatch. By setting a server dispatch mode on the OSM server, you indicate your preferred mode for OSM to invoke automation plug-ins if the build-and-deploy mode is set to Both (Allow server preference to decide). Based on the combined configuration of the build-and-deploy mode and the server dispatch mode, OSM determines the run-time behavior of automation plug-ins. See Table 5–5 for a truth table that shows the dispatch mode OSM uses at run time based on how the automation plug-in build-and-deploy mode and the OSM server dispatch mode are set.

To set the OSM server dispatch mode for processing automation plug-ins:

1. Add or modify the following configuration entry in the oms-config.xml file:

See the chapter on configuring OSM with oms-config.xml in OSM System Administrator’s Guide for detailed instructions on accessing and modifying the oms-config.xml file.

2. Modify the configuration entry as follows:
   - To run automation plug-ins in a common EAR file (to use the Optimized dispatch mode), enter INTERNAL (default).
   - To run each automation plug-in its own EAR file (to use the Legacy dispatch mode), enter LEGACY.

3. Save the file.

For information on how OSM processes automation plug-ins based on the build-and-deploy modes you set for the cartridge and the dispatch mode you set for the OSM server, see "About Dispatch Modes for Automation Plug-ins".

About Automation Maps

After you have defined the automated task or automated notification, and defined the automation for it, a successful build of the project automatically generates the automationMap.xml file:

- This file is governed by the rules defined in the cartridgeName/customAutomation/automationMap.xsd file, which is only visible when in the Java perspective. The customAutomation directory and XSD file are present with the creation of an OSM cartridge.

- This file is placed in the cartridgeName/cartridgeBuild/automation directory, which is only visible when in the Java perspective.

About Editing the Automation Map

If you are deploying a cartridge outside Design Studio, for example using OSM’s cartridge management tools, the first time you upgrade a cartridge from a pre-OSM 7.0.3 version to a version of OSM that is 7.0.3 or later, you need to update the automationMap.xml manually. You need to add two elements to each <taskAutomator> element:

<cartridgeNamespace>Namespace</cartridgeNamespace>
<cartridgeVersion>Version</cartridgeVersion>

These elements are required because of changes to the automationMap.xsd.

If you are upgrading a pre-OSM 7.0.3 cartridge created in Design Studio, to a version that is 7.0.3 or later, no manual change is required.

Appendix A, "AutomationMap.xml File" provides numerous examples of generated XML for automations defined for automated tasks and automated notifications. The information is not included in this chapter because Oracle recommends that when defining the automation, you take the defaults and allow the project build to generate the automationMap.xml file. The information in the appendix is provided for in-depth understanding of the file should you need to modify it for some rare, obscure business reason.

About Mnemonic Values for Design Studio Entities in Automation Maps

For automations defined as internal event receivers, the automationMap.xml generates the <mnemonic> element. This value of this element varies as described in Table 5–6.
The String value of the mnemonic element cannot exceed a length of fifty characters. If the length is greater than fifty, the following build error is encountered:


### Table 5–6 Mnemonic Values

<table>
<thead>
<tr>
<th>Automated Task or Automated Notification</th>
<th><code>&lt;mnemonic&gt;</code> value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automated task</td>
<td><code>taskName</code></td>
</tr>
<tr>
<td>Order milestone-based event notification</td>
<td>The <code>&lt;mnemonic&gt;</code> element is not generated for order milestone-based event notifications.</td>
</tr>
<tr>
<td>Task state-based event notification (task Events tab)</td>
<td><code>taskName</code></td>
</tr>
<tr>
<td>Task state-based event notification (process Events tab)</td>
<td><code>processName_eventName</code></td>
</tr>
<tr>
<td>Task status-based event notification</td>
<td><code>processName_eventName</code></td>
</tr>
<tr>
<td>Order data changed event notification</td>
<td><code>orderName_eventNotificationName</code></td>
</tr>
<tr>
<td>Order jeopardy notification</td>
<td><code>orderName_jeopardyName</code></td>
</tr>
<tr>
<td>Task jeopardy notification</td>
<td><code>taskName_jeopardyName</code></td>
</tr>
</tbody>
</table>

### About Managing Automations

The following sections describe automation management topics.

### Building and Deploying Automation Plug-ins

Building and deploying an automation plug-in is a matter of building and deploying the cartridge that defines the automation plug-in. See "Packaging and Building a Cartridge" and "Deploying a Cartridge".

### Automating the Build and Deploy

You can also automate and build the deploy of an automation plug-in by automating the build and deploy of the cartridge that defines the automation plug-in. See "Automating the Cartridge Build and Deployment".

### Upgrading Automation Plug-ins

If you are upgrading from a previous release of OSM, and the previous release included automation plug-ins (custom or predefined), the same steps that are required to define a new automation plug-in are required to define the existing automation plug-in in the new release, with the exception of writing of the actual custom Java code.

For example, if the previous release included the automation plug-in `genericPlugin`, to upgrade `genericPlugin` in the new release you need to:

- Define the trigger in Design Studio
Troubleshooting Automations

The following sections describe various automation troubleshooting scenarios.

getProductBuildVersion() method

If you encounter a problem when attempting to run an automation, you must verify that you are not using multiple versions of the automation_plugins.jar file. You do this by checking that the date and size of the file are the same in the following locations:

- When you create a new cartridge in Design Studio, the automation_plugins.jar file is placed in the osmlib directory of the cartridge. Verify the date and size of the file by viewing your Eclipse workspace in Windows Explorer, and navigating to the osmlib directory of the cartridge you created within your workspace.

- When you install OSM, the automation_plugins.jar file is placed in the OSM_home/SDK/automation/automationdeploy_bin directory. This is the version of the automation_plugins.jar file that your project library list references to compile the cartridge project containing the automation. (See "Compiling the Custom Automation Plug-in" for more information.) Verify the date and size of the file by viewing your installation directory, and navigating to the OSM_home/SDK/automation/automationdeploy_bin directory.

If the two versions of the file are not the same, use the version from the OSM installation:

1. Copy the automation_plugins.jar file from the OSM_home/SDK/automation/automationdeploy_bin directory to the osmlib directory of your cartridge within your Eclipse workspace.
2. Clean and rebuild the cartridge.
3. Redeploy the cartridge.
4. Run the automation.

Note: When the versions of the automation_plugins.jar file are not the same, you may also encounter a marshalling error when deploying the cartridge, prior to attempting to run the automation. The marshalling error, which states that it cannot find the getProductBuildVersion() method, displays on the WebLogic console; it does not display in Design Studio when deploying the cartridge. If you encounter this error, the resolution is the same. Follow the steps described above.
Using Order Metrics Manager

This chapter describes the Oracle Communications Order and Service Management (OSM) Order Metrics Manager feature, which allows you to extract metric data from your OSM system and view that data using a variety of tools.

About Order Metrics Manager ADML Files

OSM provides an Order Metrics Manager API that collects metric data about your system. A set of XML files, called ADML files, contain the metric rules that allows Order Metrics Manager to aggregate the metric data.

ADML files are automatically loaded to the correct directory when you run the installer. If the ADML files are not loaded correctly, you can load them manually. For more information, see the topic about manually loading metric rules files in OSM Installation Guide.

You can access ADML files in order to see the metrics that Order Metrics Manager is collecting. For OSM and other Oracle products and product suites, ADML files are located in the following directory:

`$MW_home/oracle_common/modules/oracle.em_11.1.1/server_config/adml`

where `$MW_home` is the location where the Oracle Middleware product is installed.

---

**Note:** ADML files are the technical implementation of the API, therefore you cannot customize these files. You can, however, create your own custom ADML files. For information about creating ADML files, see Oracle Fusion Middleware documentation.

---

Viewing Metrics

The Oracle Application Management Pack plug-in that is available with Oracle Enterprise Manager provides a graphical view of metrics data. For more information about viewing metrics data using the Application Management Pack interface, see OSM System Administrator’s Guide and Oracle Application Management Pack for Oracle E-Business Suite User’s Guide.

There are a number of other tools that you can use to view and retrieve or dump the metric data that Order Metrics Manager extracts from your OSM system. You can use the following:

- Oracle Dynamic Monitoring Service (DMS) Spy servlet
- WebLogic Scripting Tool (WLST)
Java Management Extensions (JMX)
Oracle Fusion Middleware Console
Oracle Enterprise Manager

The DMS interface is provided with your OSM installation and displays metric data in sets of tables. A set of DMS tables displays OSM metric data, and a set of aggregated tables displays OSM metrics that are defined based on existing metrics.

The DMS interface can also display sets of tables for WebLogic and JMX metrics, and for non-J2EE metrics, which are about remote processes from the Oracle HTTP server. For more information about viewing metrics data using the DMS interface, see OSM System Administrator’s Guide.

For more information about DMS and using the other tools that are listed in this section to view metric data, see Oracle Fusion Middleware Performance and Tuning Guide.
This chapter describes the Oracle Communications Order and Service Management (OSM) Security Callback feature, which allows you to generate an audit trail log of users before they gain access to order data that is deemed to be sensitive.

**About Security Callback**

OSM provides a callback interface that is designed to intercept order access from the following functions:

- GetOrder
- Web Service GetOrder
- Order Automation Context getOrder()
- XML API GetOrder.Request, GetNextOrderAtTask.Request, GetOrderAtTask.Request
- Opening the order from the Task Web client Worklist and Query pages
- Worklist
- XML API WorkList.Request
- Query
- XML API Query.Request
- OrderDataHistory
- Task Web client Order Data History page (clicking on view or node URLs in Order Editor)
- XML API GetOrderDataHistory.Request

The callback is called before sensitive order data is about to be retrieved or displayed to a user. The normal security authorization for the call being made remains in place and runs before this callback interface.

**About the Security Callback Interface**

The security callback interface (contained in the com.mslv.osm.security Java package) is implemented by a registered custom class which calls the defined method (single order or result set) and passes information about the order which has been exposed to the user. In the single order or result set method, the custom class can be passed either a single order or a result, depending on which interface it is invoked. For example, if
you select multiple orders in a worklist, the security callback would be passed a result set of orders.

For more information about the security callback interface, install the OSM SDK and extract the `OSM_home/SDK/osm7.x.y.z-javadocs.zip` OSM Java docs (where `OSM_home` is the directory in which the OSM software is installed and `x.y.z` are the software release, patch, and build numbers). See OSM Installation Guide for more information about installing the OSM SDK.

package com.mslv.oms.security;

import java.util.Collection;

/**
 * The interface provides the callback to user defined custom code in which
 * the external call accesses the order.
 *
 */
public interface OrderViewAccessProvider extends Callback {

    /**
     * Called before the details of an order are retrieved for a user. This occurs
     * when an order is displayed in the order editor, or retrieved via APIs e.g.
     * GetOrder, GetWorklist, GetQuery, GetOrderDataHistory, GetOrderAtTask,
     * GetNextOrderTask (xmlapi only).
     * @param userId The user that accessing the order.
     * @param orderId OSM order ID.
     * @param cartridgeName The cartridge name that order belongs to.
     * @param cartridgeVersion The cartridge version that order belongs to.
     * @param orderType The order type.
     * @param orderSource The order source.
     * @param view The view mnemonic.
     * @throws OrderViewAccessNotAllowedException This embeds any custom code
     * application exceptions. The exception to differentiate unexpected exception
     * that may be occurring in custom code.
     * Any exceptions other than OrderViewAccessException suppressed and logged by
     * OSM core.
     * @see OrderViewAccessNotAllowedException
     */
    public void checkOrderAccess(String userId, String orderId, String cartridgeName,
                                  String cartridgeVersion, String orderType,
                                  String orderSource, String view)
            throws OrderViewAccessNotAllowedException;

    /**
     * Invoked before a summary of an order is displayed for a user. This occurs
     * before an order is returned on a worklist or query. Note that multiple order
     * summaries may be passed through the supplied array. This allows the core to
     * optimize invocations of this method to pass multiple orders at the same time.
     * @param userSummaryInfo
     *   The collection of Workgroups of the user who is accessing the order.
     * @param orderSummaryInfo
     *   The collection of order summaries accessed by the function.
     * @return order IDs
     *   The collection of order IDs that need be filterd from the order list. If
     *   returns null or empty collection, Order and Service Management returns the whole
     *   list.
     * @throws OrderViewAccessNotAllowedException
     *   This may embed any custom code application exceptions. Order and Service
     *   Management core would deny the access to all orders if the exception is thrown.
     *   Other exceptions suppressed and logged by Order and Service Management core.
     * @see OrderViewAccessNotAllowedException
     */
    public void checkOrderSummaryAccess(Collection<Workgroup> userSummaryInfo,
                                         Collection<OrderSummaryInfo> orderSummaryInfo)
            throws OrderViewAccessNotAllowedException;
}
Security Callback Sample

Exceptions

OSM blocks order access if it catches an `OrderViewAccessNotAllowedException` from the callback call, regardless of the method called. Other types of exceptions are simply logged and users are not blocked from order access or retrieval.

Security Callback Sample

You can find the following sample in the installation’s `SDK/Samples/SecurityCallback` directory.

```java
import java.util.Collection;
import java.util.Map;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import com.mslv.oms.security.OrderSummaryInfo;
import com.mslv.oms.security.OrderViewAccessNotAllowedException;
import com.mslv.oms.security.OrderViewAccessProvider;
import com.mslv.oms.security.UserSummaryInfo;

/**
 * The sample provides an example of security callback.
 * *
 */

public class MyViewAccessCallback implements OrderViewAccessProvider {
    private static final Log LOG = LogFactory.getLog(MyViewAccessCallback.class);

    /**
     * Invoked before the details of an order are retrieved for a user. This occurs when an order is displayed in the order editor, or retrieved via APIs (e.g. GetOrder, GetWorklist, GetQuery, GetOrderDataHistory, GetOrderAtTask and GetNextOrderAtTask).
     * *
     * @param userId
     *            The user that accessing the order.
     * @param orderId
     *            Order and Service Management order ID.
     * @param cartridgeName
     *            The cartridge name that order belongs to.
     * @param cartridgeVersion
     *            The cartridge version that order belongs to.
     * @param orderType
     *            The order type.
     * @param orderSource
     *            The order source.
     * @param view
     *            The view mnemonic.
     * @throws OrderViewAccessNotAllowedException
     */
```
public void checkOrderAccess(final String userId, final String orderId, final String cartridgeName, final String cartridgeVersion, final String orderType, final String orderSource, final String view) throws OrderViewAccessNotAllowedException {
    if (LOG.isInfoEnabled()) {
        LOG.info("MyViewAccessCallback called on checkOrderViewDetail:" +
                "by:" + userId + " orderID:" + orderId + " cartridgeName:" +
                cartridgeName + " cartridgeVersion:" + cartridgeVersion + " orderType:" + orderType + " orderSource:" + orderSource + " view:" + view);}
}
/**
 * Invoked before a summary of an order is displayed for a user. This occurs before an order is returned on a worklist or query. Note that multiple order summaries may be passed through the supplied array. This allows the core to optimize invocations of this method to pass multiple orders at the same time.
 * @param userSummaryInfo    The collection of Workgroups of the user who is accessing the order.
 * @param orderSummaryInfo   The collection of order summaries accessed by the function.
 * @return order IDs
 * The collection of order IDs that need be filtered from the order list. If returns null or empty collection, Order and Service Management returns the whole list.
 * @throws OrderViewAccessNotAllowedException
 * This may embed any custom code application exceptions. Order and Service Management core would deny the access to all orders if the exception is thrown. Other exceptions suppressed and logged by Order and Service Management core.
 * @see OrderViewAccessNotAllowedException
 */
public Collection<String> checkOrderSummaryAccess(final UserSummaryInfo userSummaryInfo, final Collection<OrderSummaryInfo> orderSummaryInfo) throws OrderViewAccessNotAllowedException {
    if (LOG.isInfoEnabled()) {
        LOG.info("MyViewAccessCallback called on checkOrderViewSummary");
    }
    for (final OrderSummaryInfo order : orderSummaryInfo) {
        if (LOG.isInfoEnabled()) {
            LOG.info("MyViewAccessCallback called on checkOrderViewSummary" +
                    " by:" + userSummaryInfo.getUserId() + " orderID:" +
                    order.getOrderId() + " orderHistID:" +
                    order.getHistId() + " cartridgeName:" +
                    order.getCartridgeName() + " cartridgeVersion:" +
                    order.getCartridgeVersion() + " orderType:" + order.getOrderType() + " orderSource:" +
                    order.getOrderSource() + " view:" + view);} 
    } 
    return new ArrayList<String>(); // Example implementation
}
Security Callback Sample

```java
order.getTaskInfo().getOrderHistId() + " cartridgeName:" + order.getCartridgeName() + " cartridgeVersion:" +
order.getCartridgeVersion() + " orderType:" + order.getOrderType() + " orderSource:" +
order.getOrderSource() + " orderState:" + order.getOrderState() + " targetOrderState:" + order.getTargetOrderState() + " reference:" + order.getReference() + " priority:" + order.getPriority() + " processStatus:" +
order.getProcessStatus() + " orderCreationDate:" + order.getOrderCreationDate() + " orderCompletedDate:" +
order.getOrderCompletedDate() + " expectedOrderCompletionDate:" + order.getExpectedOrderCompletionDate() + " expectedGracePeriodCompletionDate:" + order.getExpectedGracePeriodCompletionDate() + " taskMnemonic:" + order.getTaskInfo().getTaskMnemonic() + " taskState:" + order.getTaskInfo().getTaskState() + " taskStartDate:" + order.getTaskInfo().getTaskStartDate() + " executionMode:" +
order.getTaskInfo().getExecutionMode() + " expectedTaskCompletionDate:" + order.getTaskInfo().getExpectedTaskCompletionDate());
```

```java
final Map<String, String> flexHeadersWithValues = order.getFlexibleHeaders();
if (LOG.isInfoEnabled()) {
    LOG.info("FlexibleHeaders in the form {(MnemonicPath=Value)} " + flexHeadersWithValues);
}
```

```java
return null;
}
}
```

### Configuring Security Callbacks

Complete the following steps to configure your callback implementation.

1. **Implement the interface** `OrderViewAccessProvider`.

   OSM provides the `osmcommon.jar` file, which includes the callback interface and exception `OrderViewAccessException`. The JAR file can be obtained by unpacking the `oms.ear` file.

2. **Register the callback through the `oms-config.xml` file.**

   ```xml
   <oms-parameter>
   <oms-parameter-name>com.mslv.oms.security.OrderViewAccessProvider</oms-parameter-name>
   <oms-parameter-value>callbackexamples.MyViewAccessCallback</oms-parameter-value>
   </oms-parameter>
   
   See the chapter on configuring OSM with `oms-config.xml` in OSM System Administrator's Guide for detailed instructions on accessing and modifying the `oms-config.xml` file.

3. **Compile and package the callback implementation in the `customization.jar` file.**
4. Modify the security.jar manifest to include any required JAR for the custom code to run.

5. Repack oms.ear with customization.jar and any custom code dependent libraries using the scripts provided.
This chapter describes the Oracle Communications Order and Service Management (OSM) Custom Menu and Action feature, which allows you to configure custom menu items and actions that are called from the context menu of the Task Web client Worklist and Query Result pages.

About Custom Menu Items and Actions

A custom menu action calls customer-specific business logic, for example, enabling a print job of tasks in the Worklist. The custom business logic can easily interact with the OSM server through the XML API.

You define custom menu items and actions using a model in an XML file. Actions are defined globally across all cartridges, and may be called for any task or group of tasks. The action is available to all users. Actions that call the XML API are done within the Web client session, so access privileges to the API are based on the Web client user’s workgroup privileges.

Additionally, API users must belong to a WebLogic group that provides privilege to access the APIs. For custom menu and action items, that WebLogic group is OMS_xml_api. So, to access the APIs through custom menu items and actions, the API user must belong to the WebLogic group OMS_xml_api.

About the File Name and Location

The metadata definition for custom menu action is supported through a standalone configuration file that is loaded and run at runtime. The OSM Administrator application (or an Ant task in the Cartridge Development Kit) can be used to trigger a reload of the configuration file.

The name and location of the custom menu action file is defined in the oms-config.xml file.

A working model, which includes a sample configuration file, Javascript file, and ReadMe, is available in the OSM_home/SDK/Samples/CustomMenuAndAction directory.

About the Model Definition

The definition of the model must follow the XML schema menuAction.xsd located in the OSM_home/SDK/XMLImportExport directory. The action and menuItem elements are described below.
**Action Definition**

Table 8–1 lists the action elements.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the action referenced by the menu item.</td>
</tr>
<tr>
<td>xsi:type</td>
<td>There are three types of actions:</td>
</tr>
<tr>
<td></td>
<td>javascript ActionType - Defines a Javascript function as part of the &lt;implementation&gt; element. The function may be embedded directly in the element, in which case it should not be wrapped in a function name () {} construction, or it may be located in an external file which can be called from the &lt;implementation&gt; element.</td>
</tr>
<tr>
<td></td>
<td>orderContext ActionType - Similar to javascript ActionType, can make use of Javascripts in the same way. In addition, has an object named orderContext which is accessible from within the Javascript. Refer to this object as part of the function. If the function is defined in an external file and the &lt;implementation&gt; contains a call to that function, pass orderContext as a parameter to the function.</td>
</tr>
<tr>
<td></td>
<td>uri ActionType - Forwards you to the supplied URI, which opens in a new window in the browser. The URI is supplied as part of the &lt;implementation&gt; element.</td>
</tr>
<tr>
<td>description</td>
<td>The description of the action that appears on the context menu when no menu item description is supplied.</td>
</tr>
<tr>
<td>hint</td>
<td>The tool tip associated with the action.</td>
</tr>
<tr>
<td>icon</td>
<td>The icon associated with the action. Icons must be packed as part of the oms.ear file (oms.ear/oms.war/images).</td>
</tr>
<tr>
<td>implementation</td>
<td>The implementation of the action, e.g. Javascript function, orderContext, URI. May also contain a href, which is a URI pointing to a Javascript file.</td>
</tr>
<tr>
<td>uri</td>
<td>The path to a local directory, Web page address, or any point of content.</td>
</tr>
</tbody>
</table>

**OrderContext and Orders**

An orderContext ActionType action is supplied with an object named orderContext. This object contains an array of orders which, in turn, contains information about the orders for which the action was called. Table 8–2 shows the method calls that can be made on the orderContext and order objects.

<table>
<thead>
<tr>
<th>Object</th>
<th>Methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>orderContext</td>
<td>getOrders()</td>
<td>Call this method to get the selected orders.</td>
</tr>
<tr>
<td>order</td>
<td>getOrderId()</td>
<td>Call this method to get the order ID for the order.</td>
</tr>
<tr>
<td>order</td>
<td>getOrderHistId()</td>
<td>Call this method to get the order history ID for the order.</td>
</tr>
<tr>
<td>order</td>
<td>getOrderTypeId()</td>
<td>Call this method to get the order type ID for the order.</td>
</tr>
<tr>
<td>order</td>
<td>getOrderSourceId()</td>
<td>Call this method to get the order source ID for the order.</td>
</tr>
<tr>
<td>order</td>
<td>getState()</td>
<td>Call this method to get the state of the order.</td>
</tr>
</tbody>
</table>
Calling the XML API

The function, `callXmlApi()`, makes it easier for action implementations to call the XML API. The function takes the XML API request document as an argument and returns the response XML document.

Sample Action Implementations

This section provides some samples of the different types of actions that you can configure in your custom action and menu XML file.

```xml
<action name="get_worklist_through_xml_api" xsi:type="javascriptActionType">
  <description>Get worklist through XML API</description>
  <hint>An XML API call</hint>
  <icon>/oms/images/delete_node.gif</icon>
  <implementation>
    var callString=prompt("Please enter the XML API statement", "<Worklist.Request xmlns='urn:com:oracle:oms:xmlapi:1'> </Worklist.Request>" );
    var returnDoc=callXmlApi(callString);
    alert("Returned document: "+ returnDoc.xml);
  </implementation>
</action>

<action name="test_order_context" xsi:type="orderContextActionType">
  <description>Test Order Context</description>
  <implementation>
    var orders=orderContext.getOrders();
    var callString="<GetOrder.Request xmlns='urn:com:oracle:oms:xmlapi:1'>"
    callString = callString + "<OrderID>" + orders[0].getOrderId() + "</OrderID>
    callString = callString + "<Accept>false</Accept>
    callString = callString + "<OrderHistID>" + orders[0].getOrderHistId() + "</OrderHistID>
    callString = callString + "</GetOrder.Request>");
    returnDoc = callXmlApi(callString);
    alert('result: ' + returnDoc.xml);
  </implementation>
</action>

<action name="test_js_file" xsi:type="orderContextActionType">
  <description>Test Order Context</description>
  <implementation href="file:///$bea_home/user_projects/domains/provisioning/foo.js">
    test_js_file(orderContext);
  </implementation>
</action>

<action name="go_to_about" xsi:type="uriActionType">
  <description>Show OSM About</description>
  <icon>/oms/images/mslv_logo1.jpg</icon>
  <uri>/oms/about</uri>
</action>
```

Menu Item Definition

### Table 8–3  Menu Item Elements

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the menu item (internal reference only).</td>
</tr>
<tr>
<td>description</td>
<td>The description of the menu item that appears on the context menu.</td>
</tr>
<tr>
<td>enabled</td>
<td>Can be set to true() or false() when used in conjunction with an XPath to other fields. Not supported in this version.</td>
</tr>
</tbody>
</table>
Sample Menu Item Definition

```xml
<menuItem name="get_worklist">
  <description>Get worklist through XML API</description>
  <enabled>true()</enabled>
  <visible>true()</visible>
  <displayStyle>ICON TEXT</displayStyle>
  <action>
    <name>get_worklist_through_xml_api</name>
  </action>
</menuItem>
```

Setting Up the Environment

Once you have defined the elements in your configuration file, you must set up the environment before running the file. There are three methods for doing this:

- File system path method: This is the simplest configuration method for a single environment. It does not require any cartridges for its implementation. However, it does require you to unpack, repack, and redeploy the oms.ear file for each environment with a different file location and every time the file location changes.

- XML Catalog (Static Relative Location) method: This method uses the XML Catalog function in Oracle Communications Design Studio. It allows you to deploy the resources with a cartridge, and configure a static location for the source files based on their location in the Design Studio files hierarchy. This means that the files can be deployed from Studio to environments in different locations, and having different file structures, without needing any further manual intervention.

- XML Catalog (rewriteURI) method: This method uses the XML Catalog function in Design Studio. It provides a mechanism for you to define the location of the files dynamically, either to an absolute file location or to a location relative to the current Design Studio environment. You can then change the location for the files without having to edit the oms-config.xml file. This could be especially useful while you are developing or unit testing the configuration, as you could define a local directory for the files and change them without having to redeploy the cartridge after each change.

To configure your environment, you must perform the steps in "Setting Up the oms-config.xml File" and only one of the following sections:

- File System Path Environment Configuration Method
- XML Catalog (Static Relative Location) Environment Configuration Method
- XML Catalog (rewriteURI) Environment Configuration Method

### Table 8–3 (Cont.) Menu Item Elements

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>visible</td>
<td>Can be set to true() or false() when used in conjunction with an XPath to other fields. Not supported in this version.</td>
</tr>
<tr>
<td>displayStyle</td>
<td>The display style of the menu item on the context menu, either icon, or text, or both. References the action icon and/or description.</td>
</tr>
<tr>
<td>action</td>
<td>Reference to the action being called.</td>
</tr>
</tbody>
</table>
Setting Up the Environment

Using Custom Menu Items and Actions

Setting Up the oms-config.xml File

All three methods of environment configuration require that you set up the oms-config.xml file. There are two ways of managing this file:

- Using an external oms-config.xml file which is the recommended easy-access method.
- Storing the oms-config.xml file in oms.ear which requires unpacking and repacking oms.ear to modify the file.

For the file system path method, you must edit the oms-config.xml file for each environment where the absolute path to the file is different. For the XML Catalog methods you should only need to perform this procedure once.

See OSM System Administrator’s Guide for more information about editing the oms-config.xml file.

1. Locate the following section of the oms-config.xml file:

   `<oms-parameter>
     <oms-parameter-name>custommenuaction_model_location</oms-parameter-name>
     <oms-parameter-value/>
   </oms-parameter>`

2. Update the `<oms-parameter-value>` tag. The value you use here depends on the environment configuration method you are using.

   - If you are using the file system path method, update the value with the exact path to the configuration file for the current environment. You must perform this procedure for each environment that has a different file path.

     `<oms-parameter>
     <oms-parameter-name>custommenuaction_model_location</oms-parameter-name>
     <oms-parameter-value>/opt/OSM/CustomMenu/custom_menu_action_model.xml
     </oms-parameter>`

   - If you are using the XML Catalog (Static Relative Location) method, you use a relative location based on osmmodel and referring to a directory in the Studio workspace.

     `<oms-parameter>
     <oms-parameter-name>custommenuaction_model_location</oms-parameter-name>
     <oms-parameter-value>
       osmmodel://cartridge_name/cartridge_version/resources/filename.xml
     </oms-parameter>`

   where cartridge_name and cartridge_version represent the name and version of the cartridge where you are planning to include the custom files, and filename.xml is the file with your XML model (for example, custom_menu_action_model.xml).

   - If you are using the XML Catalog (rewriteURI) method, you use a URI that you have determined for this task. It does not have to be a valid URL or any location where the file is located. It will be overwitten with a valid value automatically at runtime.

     `<oms-parameter>`
<oms-parameter-name>custommenuaction_model_location</oms-parameter-name>
<oms-parameter-value>
  http://example.org/somewhere/filename.xml
</oms-parameter-value>
</oms-parameter>

where example.org/somewhere represents a namespace you are using as a convention to refer to this file and filename.xml is the file with your XML model (for example, custom_menu_action_model.xml).

File System Path Environment Configuration Method

You must perform the procedure below for each server environment.

1. Edit your custom menu and action configuration XML file to ensure that it contains the correct location of any external files referenced in it. To find the references, look for the string implementation href in the file. Then change the value to the correct location for the current environment.

2. Save the changes and close the file.

3. Ensure that your custom configuration XML file is located in the directory you specified in step 2 of "Setting Up the oms-config.xml File".

4. Deploy the oms.ear file that contains your oms-config.xml changes to your environment.

XML Catalog (Static Relative Location) Environment Configuration Method

You must perform the procedure below for each Design Studio environment.

1. Edit your custom menu and action configuration XML file to ensure that it contains the correct location of any external files referenced in it. To find the references, look for the string implementation href in the file. Then change the value to the correct location for the current environment.

2. Create or open a cartridge in Design Studio with the name and version that you configured in step 2 of "Setting Up the oms-config.xml File".

3. Ensure that XML_CATALOG_SUPPORT is not set to disable for the cartridge. To check this, open the cartridge definition file, and click on the Cartridge Management Variables tab. By default, XML_CATALOG_SUPPORT is enabled, so if there is no entry in the Cartridge Management Variables table for that parameter, no change is needed. If there is an entry and it is set to disable, remove the entry and save the cartridge definition file.

4. Copy your custom configuration XML file and any files that it references to the location you configured in step 2 of "Setting Up the oms-config.xml File". In the example, you would copy the files to the resources directory for your cartridge.

5. Build and deploy the cartridge.

6. Deploy the oms.ear file that contains your oms-config.xml changes to your environment.

XML Catalog (rewriteURI) Environment Configuration Method

You must perform the procedure below for each Design Studio environment. You must perform steps 4-7 whenever you change the location of the files.
1. Edit your custom menu and action configuration XML file to ensure that it contains the correct location of any external files referenced in it. To find the references, look for the string **implementation href** in the file. Then change the value to the correct location for the current environment.

2. Create or open a cartridge in Design Studio.

3. Ensure that XML_CATALOG_SUPPORT is not set to **disable** for the cartridge. To check this, open the cartridge definition file, and click on the **Cartridge Management Variables** tab. By default, XML_CATALOG_SUPPORT is enabled, so if there is no entry in the Cartridge Management Variables table for that parameter, no change is needed. If there is an entry and it is set to **disable**, remove the entry and save the cartridge definition file.

4. Copy your custom configuration XML file and any files that it references to a location of your choice, either inside or outside of the cartridge directory structure.

5. Create a copy of the **xmlCatalogCoreTemplate.xml** file. It is located in the `xmlCatalogs\core` directory for your cartridge. You can name the copy anything you like as long as it is a different name from the original and it ends with `.xml`.

6. In your new XML file, replace the commented text with a line indicating how you want to translate the URI into a file location. The new line should look something like this:

   `<rewriteURI uriStartString=specified_namespace_string rewritePrefix="file-_location"/>`

   where **specified_namespace_string** refers to the string you specified in step 2 of "Setting Up the oms-config.xml File" and **file_location** refers to the location where you copied your custom configuration files.

   For example, if you have copied the files to a location inside your cartridge directory structure, you would add a line similar to this:

   `<rewriteURI uriStartString=http://example.org/somewhere rewritePrefix="osmmodel:///TestCartridge/1.0.0/resources"/>`

   If you have copied the files to some location outside the Design Studio file structure, you would add a line similar to this:

   `<rewriteURI uriStartString=http://example.org/somewhere rewritePrefix="file:///C:/LocalResourcesFolder/resources"/>`

7. Build and deploy the cartridge.

8. Deploy the **oms.ear** file that contains your **oms-config.xml** changes to your environment.

**Verifying the Changes**

1. If OSM was running when you made the changes to set up the environment, use the OSM Administrator application to refresh the metadata. This loads the latest configuration for the custom menu and actions.

2. Log in to the Task Web client.

3. In the Worklist or Query Results page, select any order and right-click.

   The context menu displays the new menu items, positioned at the bottom of the menu.
This chapter describes how to use data providers to retrieve data when modeling orders in Oracle Communications Order and Service Management (OSM).

About Data Providers and Adapters

An Oracle Communications Design Studio data provider is an instantiation of adapter (which is a Java class) that can retrieve data in an XML format from external systems. Data Providers are used when defining Data Instance behaviors. Design Studio provides several built-in Data Providers 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 “Custom Data Providers” for more information).

In Design Studio, the Data Provider editor Settings tab (Figure 9–1) allows you to set the Data Provider type using Provider Type. Types of Data Providers include:

- Objectel
- Order
- Property File
- SOAP
- XML Attachment
- XML File
- XML Validation
- JDBC
- Web Service
- Custom Data Providers

When you select any of the above choices other than a custom data provider, the Provider Class field becomes disabled and is populated with the OSM implementation of the adapter. When you select Custom, the Provider Class field is enabled because you must supply the class name of the custom adapter that you write. See “Custom Data Providers” for detailed information.
Data Provider Interface Tab

Data providers, both built-in and custom, can take parameters as input, as shown in the Interface tab (Figure 9–2). Parameter names are free-form text, but are dictated by the data provider’s expected input. An asterisk (*) appears next to mandatory parameters, and each parameter’s corresponding value can be specified as either XPath 1.0 or XQuery 1.0. In addition to the functions provided by the XPath 1.0 or XQuery 1.0 standards, OSM provides a custom function, `instance(string)` that allows the output of one data provider to be used as the input of another. The parameters required by each of the built-in data providers is documented in the sections that follow.
Using Data Providers to Retrieve Data

For instructions on how to define these data providers in Studio, including field-level detail, see the Design Studio Help.

### Accessing Data through Data Providers

To use a Data Provider, you include a data element in the order template, define a behavior for it and use an XPath expression to access the Data Provider and extract the data that you wish to display in the data element.

For example, the following XPath illustrates how to call a Web service provider instance named "DataInstance" and return the value of the "my_element" view data element.

```
instance('DataInstance')/Data/_root/my_element
```

For XQuery, you would use `vf:instance0`.

### Augmenting or Overriding Data

In most cases, a data provider references order data from an external source, another behavior, or as static values defined within the data provider. In addition to these options, you can also add explicit parameter values from within an XQuery or XPath that augment or override the parameters defined in the OSM data dictionary.

For example, the following variable can be declared with parameters that have not been defined within the OSM data dictionary from within an XQuery:

```xml
declare variable $dataInstanceParams :=
  <params>
    <oms:url>file://users/bdueck/catalog.xml</oms:url>
    <fooParam>barValue</fooParam>
  </params>;
```
You can call a data instance function using a sequence of parameters declared in the variable above. For example:

``` razor
log:info($log,local-name(vf:instance($order/oms:GetOrder.Response/oms:_
root/oms:data[1],'DataInstance',$dataInstanceParams/*)/*[1]))
```

You can call a data instance function using parameters passed as parameters on the function one by one. For example:

``` razor
log:info($log,local-name(vf:instance($order/oms:GetOrder.Response/oms:_
```

You can call a data instance function using parameters passed as parameters on the function one by one and include two parameters. For example:

``` razor
log:info($log, local-name(vf:instance($order/oms:GetOrder.Response/oms:_
root/oms:data[1],'DataInstance',<oms:url>file://us/catalog.xml</oms:url>,<foo>bar</foo>)/*[1])
```

**Objectel**

This adapter provides a reliable transport call into Objectel. Although JMS is an asynchronous protocol, the Objectel adapter itself is not. While JMS simplifies transaction management, recovery, offline capabilities, and security, these benefits are not relevant when considered within the context of a behavior. 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. It is not recommended to use this adapter in this scenario.

**Parameters**

- **objectel:extensionName**
  
  Description: the name of the Objectel server extension to call.
  
  Mandatory/Optional: Mandatory

- **objectel:jmsFactory**
  
  Description: the name of the JMS factory to be used to access Objectel’s JMS queue.
  
  Mandatory/Optional: Optional
  
  Default value - *com.oracle.objectel.XMLJMS.QueueConnectionFactory*

- **objectel:queue**
  
  Description: The name of the Objectel receive queue.
  
  Mandatory/Optional: Optional
  
  Default value: - *com.oracle.objectel.XMLJMS.QueueConnectionFactory*

- **objectel:allowErrorResponse**
  
  Description: an optional Boolean parameter name that if specified controls what happens if Objectel returns an error response. If this parameter is set to false (default), an error response from Objectel triggers an exception to be thrown which is in turn displayed as a constraint violation. If this parameter is set to true, the error response is returned by the ObjectelAdapter as a valid instance. This allows another Constraint behavior to apply to that same instance and display an error message accordingly. The benefit of using the default (false) is that you do
not have to write an additional behavior to display a default error message. The constraint violation message looks like an exception with a stack trace, but shows the error description returned by Objectel at the top of the message. The benefit of setting this parameter to true is that you have greater control over when the error is shown, at what severity, and what message is displayed.

- false: If this parameter is set to false (the default), an error response from Objectel throws an exception, which is then displayed as a constraint violation. By using false you can avoid writing an additional behavior to display only a default error message. With this method, the constraint violation message looks like an exception with a stack trace, but shows the error description returned by Objectel at the top of the message.

- true: If this parameter is set to true, the error response is returned by the ObjectelAdapter as a valid instance. This allows another Constraint behavior to apply to that same instance and display an error message accordingly. By setting the parameter to true, you have greater control over:
  - When the error should be shown
  - The severity level displayed
  - The exact error message to display.

- All other parameters are passed directly as name/value pairs to the server extension.

**Order**

This adapter lets you use order data from any OSM order as an external instance. This is useful for using related order data from other orders within OSM.

**Parameters**

- **oms:OrderID**
  Description: The order ID of the order to be retrieved.
  Mandatory/Optional: Mandatory

- **oms:View**
  Description: The view to use when retrieving order data.
  Mandatory/Optional: This is required if the OMS:OrderHistID is not supplied.

- **oms:OrderHistID**
  Description: The order history ID to use when retrieving order data.
  Mandatory/Optional: This is required if OMS:View is not supplied.

**Adding a New Order Data Provider**

To add a new Data Provider which uses the Order adapter:

1. In Design Studio, add a new Data Provider. From the Studio menu, select New, then select Order and Service Management, and then select Data Provider.
2. In the Data Provider Wizard, enter a name and folder for the Data Provider and set Provider Type to Order.
   
   The new Order Data Provider is added to the Design Studio project.
3. Edit the Data Provider.

4. In the Data Provider editor, on the Input Parameters section of the Interfaces tab, specify values for either the oms:View or oms:orderHistID parameters.

5. Set the Default Value to either XQuery or XPath and enter your request code in the Default Value edit box.

6. Optionally specify the XML structure of the data in the Results Document edit box.

The definition of GetOrderResponse is located in the order management Web service schema at osm_home/SDK/XMLSchema/GetOrder.xsd.

For more information, see the discussion on Data Providers in the Design Studio Help. Also, see "Accessing Data through Data Providers".

Property File

This adapter retrieves an external Java property file with a given name from the classpath. The format of the XML instance returned by this adapter is specified on the Oracle Technology Network (OTN) documentation Web site at:

http://docs.oracle.com/javase/7/docs/api/java/util/Properties.html

Parameters

- oms:url

Description: Specifies the file name of the Java property file. The file must be on the classpath and must be in the format of a Java property file.

Mandatory/Optional: Mandatory

SOAP

This adapter lets you access Web services from OSM or an external Web service server, using the HTTP protocol. You can call SOAP Web services from OSM or an external Web service server and use the responses within behaviors.

**Note:** If you need to configure a proxy server to access the internet, add the following parameters to the OSM WebLogic server startup script:

```
JAVA_OPTIONS="${JAVA_OPTIONS} -Dhttp.proxyHost=ip_address
-Dhttp.proxyPort=port"
```

where *ip_address* and *port* are the IP address and port of the proxy server.

For Web service calls specific to OSM, use the Web service adapter. See "Web Service".

For general Web services calls, use the Soap adapter.

Parameters

- soap.endpoint

Description: Specifies the URL to which the SOAP request will be sent.
Mandatory/Optional: Mandatory

- **soap.action**
  Description: Contains the URI that identifies the intent of the message.
  Mandatory/Optional: Optional

- **soap.envelope**
  Description: Specifies the root element of a SOAP message.
  Mandatory/Optional: Mandatory, if the `soap.body` parameter is not defined.

- **soap.body**
  Description: Contains the SOAP message intended for the endpoint. If the SOAP body node is not included in the `soap.body` content, it will be added by the SOAPAdapter.
  Mandatory/Optional: Mandatory, if the `soap.envelope` parameter is not defined.

- **soap.header**
  Description: Contains XML data that affects the way the application-specific content is processed by the message provider. If the SOAP header node is not included in the `soap.header` content, it will be added by the SOAPAdapter.
  Mandatory/Optional: Optional

- **oms:credentials.username**
  Description: Specifies an authentication user name.
  Mandatory/Optional: Optional

- **oms:credentials.password**: An optional authentication parameter
  Description: Specifies an authentication password.
  Mandatory/Optional: Optional

- **oms:credentials.scope.host**: An optional authentication parameter
  Description: Specifies an authentication host parameter.
  Mandatory/Optional: Optional

- **soap.allowErrorResponse**
  Description: When set to true, the adapter returns SOAP fault messages to the calling behavior; otherwise, the adapter throws an exception when a SOAP fault response is returned.
  Mandatory/Optional: Mandatory

**Example of soap.body Parameter**

The following is an example of a SOAP body, which would be populated in the `soap.body` parameter.

```xml
<instance name="us-addr" xsi:type="externalInstanceType">
<adapter>com.mslv.oms.view.rule.adapter.SOAPAdapter</adapter>
<parameter name="soap.endpoint">http://ws2.serviceobjects.net/av/AddressValidate.asmx</parameter>
<parameter name="soap.body" xsi:type="xqueryType">
<soap:Body soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
```

Using Data Providers to Retrieve Data  9-7
Example of soap.envelope Parameter

The following is an example of a SOAP envelope, which would be populated in the soap.envelope parameter.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<com:modelEntity xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:adapt="http://xmlns.oracle.com/communications/sce/osm/model/adapter"
 xmlns="http://xmlns.oracle.com/communications/sce/osm/model/adapter"
 xmlns:com="http://www.mslv.com/studio/core/model/common"
 xmlns:prov="http://xmlns.oracle.com/communications/sce/osm/model/provisioning"
 xsi:type="adapt:adapterType" name="Send_Order">
  <com:displayName>Send_Order</com:displayName>
  <com:interface>
    <com:inputParameter xsi:type="adapt:xpathInputParameterType"
 name="soap.endpoint"><adapt:contentType>XPATH</adapt:contentType><adapt:defaultValue>'http://localhost:7001/osm/wsapi'</adapt:defaultValue></com:inputParameter>
    <com:inputParameter xsi:type="adapt:xpathInputParameterType"
 name="soap.envelope"><adapt:contentType>XQUERY</adapt:contentType><adapt:defaultValue>
      <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:osm="http://xmlns.oracle.com/communications/ordermanagement">
        <soapenv:Header>
          <wsse:Security soapenv:mustUnderstand="1"
           xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
          <wsse:UsernameToken wsu:Id="UsernameToken-10570647"> administrator</wsse:UsernameToken>
        </soapenv:Header>
      </soapenv:Envelope>
    </com:inputParameter>
  </com:interface>
</com:modelEntity>
```
This adapter lets you use an attachment from any OSM order as an external instance. It is useful for using related-order-data from other orders within OSM.
Parameters

- **oms:OrderID**
  Description: The order ID of the order to be retrieved.
  Mandatory/Optional: Mandatory

- **oms:FileName**
  Description: The name of the attachment to use when retrieving the order data.
  Mandatory/Optional: Mandatory

### XML File

This adapter lets you use an XML file accessible from any URL as an external instance. It is useful for integrating external XML data located in a file system, FTP site, from HTTP, or in a Java JAR file.

Parameters

- **oms:url**
  Description: The URL of the file to retrieve.
  Mandatory/Optional: Mandatory

### XML Validation

This adapter validates a provided XML instance document according to a user-defined schema. The document may be provided either as a URL or as an element. The schema may also be provided as a URL or as an element. The returned document conforms to the element specified by:

```
http://xmlns.oracle.com/communications/ordermanagement#ValidationResult
```

Parameters

- **document**
  Description: The file name of the XML document to validate.
  Mandatory/Optional: Mandatory

- **schema**
  Description: The file name of the XSD used to perform the XML validation.
  Mandatory/Optional: Mandatory

### JDBC

This adapter lets OSM query any JDBC database, then use the results within a behavior. This adapter is particularly useful for acquiring information stored in an external database.

Parameters

- **oms:dataSource**
Web Service

Using Data Providers to Retrieve Data

Description: The JNDI name of the data source providing the database connection information, for example:

<code>'mslv/oms/oms1/internal/jdbc/DataSource'</code>

The data source must be defined through the WebLogic server console. Mandatory/Optional: Mandatory

- **oms:sql**
  Description: The SQL that is sent to the database. To run a SQL stored procedure, this parameter must comply with the format specified on the Oracle Technology Network (OTN) documentation Web site at:
  
  [http://docs.oracle.com/javase/7/docs/api/java/sql/CallableStatement.html](http://docs.oracle.com/javase/7/docs/api/java/sql/CallableStatement.html)

  Mandatory/Optional: Mandatory

- **in:1 ... in:n**
  Description: 1 to n additional optional input parameters may be supplied that are bound to parameters defined in the oms:sql value.

  Mandatory/Optional: Optional

- **out:1 ... out:n**
  Description: 1 to n additional optional output parameters that are used when calling SQL stored procedures that have output parameters defined. The parameter value specifies the SQL type name of the parameter, and must comply with the format specified on the Oracle Technology Network (OTN) documentation Web site at:
  
  [http://docs.oracle.com/javase/7/docs/api/java/sql/Types.html](http://docs.oracle.com/javase/7/docs/api/java/sql/Types.html)

  Mandatory/Optional: Optional

**Web Service**

This external instance adapter lets you invoke the GetOrder and FindOrder OSM Web service operations. The adapter acts as a wrapper around OSM's Web Services API for these two Web services, allowing these operations to be called from external instances.

For other Web service calls, use the SOAP adapter. See "SOAP."

**Parameters**

- **soap.request**
  Description: Set this parameter to one of the following:
  - The contents of what would normally be in the Body element of the Web service request. For example, ord:GetOrder or ord:FindOrder.
  - A soap:Envelope element, that is, the entire soap request.
  - A soap:Body element, that is, the body element of the soap request.

  Mandatory/Optional: Mandatory

See "GetOrder" and "FindOrder" for more information on these Web service transactions.
Adding a New Web Service Data Provider

To add a new Data Provider which uses the Web Service adapter:

1. In Design Studio, add a new Data Provider. From the Studio menu, select New, then select Order and Service Management, and then select Data Provider.

2. In the Data Provider Wizard, enter a name and folder for the Data Provider and set Provider Type to Web Service.

   The new Web Service Data Provider is added to the Design Studio project.

3. Edit the Data Provider.

4. In the Data Provider editor, on the Input Parameters section of the Interfaces tab, select the soap.request* parameter.

5. Set the Default Value to XQuery and enter the request XQuery code in the Default Value edit box. See "Sample soap.request XQuery" for an example.

   You can optionally specify the request as an XPath instance instead by setting the Default Value to XPath and entering the request XPath code in the Default Value edit box.

6. Optionally specify the XML structure of the data in the Results Document edit box.

   Definitions of FindOrderResponse and GetOrderResponse declarations are located in the order management Web service schema at osm_home\SDK\WebService\wsdl\OrderManagementWS.xsd.

   For more information, see the discussion on Data Providers in the Design Studio Help.

Sample soap.request XQuery

The following is a soap.request XQuery example for a Web services Data Provider. You can also specify the input as a SOAP envelope or a SOAP Body.

```xml
<ord:GetOrder xmlns:ord="http://xmlns.oracle.com/communications/ordermanagement">
  <ord:OrderId>1</ord:OrderId>
  <ord:View>review_details_view</ord:View>
  <ord:AmendmentFilter>
    <ord:LevelOfDetail>AmendmentsSummary</ord:LevelOfDetail>
  </ord:AmendmentFilter>
  <ord:LifecycleEventFilter>
    <ord:RetrieveLifecycleEvents>true</ord:RetrieveLifecycleEvents>
  </ord:LifecycleEventFilter>
</ord:GetOrder>
```

Accessing Data

To use the Data Provider, include a data element in the order template, define a behavior for it, and use an XPath expression to access the Data Provider and extract the data to display in the data element. See "Accessing Data through Data Providers".

Whenever the Web Service adapter is called through a Data Provider, GetOrderRequest is executed and a response returned. If logging is set to debug for the OrderAdapter, a message similar to the one below is displayed on the WebLogic Administration console:

```
<09-Feb-2012 2:57:57,884 IST PM> <DEBUG> <adapter.OsmWebServiceAdapter>
<ExecuteThread: '10' for queue: 'oms_web'> <<GetOrderResponse
xmlns="http://xmlns.oracle.com/communications/ordermanagement">
```
Custom Data Providers

In addition to the built-in data providers described in previous sections, Design Studio supports custom data providers. You can develop a custom data provider class in a project in Design Studio as part of a solution. This provider class must implement the `com.mslv.oms.view.rule.ExternalInstanceAdapter` interface. This interface is documented in the Javadoc distribution found in the OSM SDK.

The implementation class can be made available to the OSM run time system in two ways:

- Package the class into an Java archive (jar file) with an arbitrary name and place the jar file in the resources directory of the Studio project(s) that define Behaviors.
referencing the data provider. The class will be available as soon as the project is deployed

- Add the compiled adapter class to the `customization.jar` file in the `oms.ear` file. The class will be available as soon as the OSM application is redeployed. See *OSM System Administrator’s Guide* for information about unpacking, packing, and redeploying the `oms.ear` file.

The `ExternalInstanceAdapter.retrieveInstance(ViewRuleContext, Map)` method provides a Map of name/value pairs of arguments defined in the data provider’s Design Studio definition and their corresponding values for an invocation of an instance of this class. The `com.mslv.oms.view.rule.adapter.AbstractAdapter` class provides a number of methods to assist in extracting properly type cast parameter values from that Map. `AbstractAdapter` is included in the `automation_plugins.jar` archive found in the `osmLib` directory of a Design Studio OSM project, as well as in the `automation/automationdeploy_bin` subdirectory of an OSM SDK installation.

### Handling Parameters

Custom data providers, like built-in providers, support input parameters. The following examples illustrate how to access those parameters.

**Example 1 (incorrect usage)**

```java
String stringParamValue = (String) parameters.get(MY_STRING_PARAM);
```

The value returned by `parameters.get(...)` may not be a String, resulting in a `ClassCastException` exception.

**Example 2 (incorrect usage)**

```java
String stringParamValue = parameters.get(MY_STRING_PARAM).toString();
```

The `parameters.get()` call may return a null value resulting in a null pointer exception. Also, the value returned may be an XML DOM fragment, requiring a more sophisticated mechanism for value extraction than simply calling `toString()`.

**Example 3 (correct usage)**

```java
String stringParamValue = getStringParam(parameters, MY_STRING_PARAM);
```

The `getStringParam(Map, String)` call automatically performs the appropriate conversion to coerce a parameter value into a String. This method is intended for extracting a required parameter value. If a value for `MY_STRING_PARAM` was not provided, or if the value cannot be coerced into a String, a `BadParameterException` exception is thrown. To retrieve optional parameter values, use `getStringParam(Map, String, String)` instead; see Example 4.

**Example 4 (correct usage)**

```java
String stringParamValue = getStringParam(parameters, MY_STRING_PARAM, "a default value");
```

The `MY_STRING_PARAM` parameter is retrieved as an optional parameter. If a value for `MY_STRING_PARAM` is provided, it is returned, otherwise, “a default value” is returned.

The `AbstractAdapter` class also provides similar methods to extract boolean, numeric, and XML DOM Node parameter values:
The following code snippet illustrates a simple custom data provider class:

```java
/**
 * Copyright © 1998, 2012, Oracle and/or its affiliates. All rights reserved.
 */
package oracle.communications.ordermanagement.example;

import java.util.Map;
import oracle.communications.ordermanagement.util.xml.XMLHelper;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import com.mslv.oms.view.rule.ExternalInstanceAdapter;
import com.mslv.oms.view.rule.ViewRuleContext;
import com.mslv.oms.view.rule.adapter.AbstractAdapter;

/**
 * <p>
 * This class exemplifies a custom Data Provider. In particular, it demonstrates a
 * provider that returns the familiar "Hello World!" example. The data returned by this provider can in turn be used as input to any
 * Behavior type.
 * </p>
 * <p>
 * @author Copyright © 1998, 2012, Oracle and/or its affiliates. All rights
 * reserved.
 * </p>
 * public final class ExampleProvider extends AbstractAdapter implements
 * ExternalInstanceAdapter {

    /**
     * The name of a parameter that specifies the salutation to return from {link
     * #retrieveInstance(ViewRuleContext, Map)}. For example, if
     * a value of {code}Goodbye{code} is specified, the message {code}Goodbye
     * World!{code} will be returned. This example does not require
     * this parameter to exist. If it does not, the message {code}Hello
     * World!{code} will be returned.
     */
```
public static final String SALUTATION_PARAM_NAME = "salutation";

private static final String DEFAULT_SALUTATION = "Hello";

public static final String SALUTATION_PARAM_NAME = "salutation";

private static final String DEFAULT_SALUTATION = "Hello";

/*
**
* This implementation simply returns the root Element of a Document containing the String "Hello World!" in the contents, i.e., the root of the XML:
* <pre>
* &lt;response&gt;
* &lt;message&gt;Hello World!&lt;/message&gt;
* &lt;/response&gt;
* &lt;/pre&gt;
* </pre>
*
* The instance('ExampleProvider') Behavior function resolves to the document root element returned by this method. Therefore, the syntax for locating this provider's message (assuming the Data Provider associated with this class is named
* <code>ExampleProvider</code>) is <code>instance('ExampleProvider')/message</code>.  
* </p>
* 
* @param ctx
* @param params
* @return the root Document Element
*/

@Override
public Element retrieveInstance(final ViewRuleContext ctx, final Map<String, Object> params) throws Exception {

final String salutation = getStringParam(params, SALUTATION_PARAM_NAME, DEFAULT_SALUTATION);

final String response = "<response><message>" + salutation + " World!</message></response>";

/*
* The code invoking this method expects a org.w3c.dom.Document root
* org.w3c.dom.Element. The XMLHelper utility class provides a
* number of DOM manipulation methods, including various String parsers.
*/
final Document responseDoc = XMLHelper.parseText(response, false);
return responseDoc.getDocumentElement();
}
This chapter provides information on localizing the Oracle Communications Order and Service Management (OSM) Web clients. Localization is the process of changing a user interface (UI) from the original language in which it was written to another language. This chapter also provides information for customizing regional settings across all OSM applications. This chapter is intended for service professionals developing a custom installation for their clients.

**About Localization**

Localization is the process of customizing the OSM system for use in a specific market and language. This process includes translating the user interface and documentation, as well as adapting time, date, number formats, and punctuation conventions. It may also include editing or creating new icon graphics.

**Note:** Oracle recommends that you have an experienced localization professional perform the documentation translation and coding localization.

This chapter describes how to:

- Modify the OSM Order Management Web clients and Task Web clients
- Configure regional settings for the OSM Administrator application
- Identify information in the OSM database that requires localization
- Convert the OSM Web clients into a single foreign language.

**Note:** After you run any migration script such as upgrading an installation, you must reapply any customization done to the database.

**Localizing OSM**

Localizing OSM involves the following high-level steps.
1. Uncompress the $OSM_{\text{home}}$/Database/dbinstall/install/omsmodel_I10n.jar file. Set the environment variable PATH to $JDK_{\text{home}}$/bin, then from a command prompt run the command:

```
jar -xvf omsmodel_I10n.jar
```

The uncompressed files are organized into the directories shown in Figure 10–1, "Uncompressed Files".

2. Navigate to the directories listed below, and localize the text strings in the XML files indicated. You must localize this information first so that the database parameters are defined in the target language. For information on how to localize the content of the XML files, see the following sections in this chapter:

- About OSM Database Error Messages
- About Application Server Strings
- About OSM Process Definition Data
- About Generic Preferences

**$\text{OMS/main/model/calendar/_table/_data}$:**
- om_region.xml
- om_schedule.xml

**$\text{OMS/main/model/cartridge/_table/_data}$:**
- om_cartridge.xml
OMS/main/model/notification/_table/_data:
- om_notification_def.xml

OMS/main/model/ordertemplate/_table/_data:
- om_attribute_type.xml
- om_order_data_dictionary.xml
- om_order_hier.xml
- om_orderRemarks_type.xml
- om_order_type_category.xml
- om_view_order_node_label.xml

OMS/main/model/process/_table/_data:
- om_behavior.xml
- om_process.xml
- om_process_point.xml
- om_process_position.xml
- om_process_status.xml
- om_state.xml
- om_state_category.xml
- om_status_category.xml
- om_task.xml
- om_task_state.xml
- om_task_status.xml

OMS/main/model/rule/_table/_data:
- om_rule_def.xml
- om_rule_source.xml
- om_rule_task.xml

OMS/main/model/user/preference/_table/_data:
- om_generic_mnemonic.xml

OMS/main/model/user/privilege/_table/_data:
- om_application_function.xml
- om_responsibility.xml

OMS/main/model/utility/_table/_data:
- om_errors.xml
- om_server_strings.xml

3. Compress **omsmodel_I10n.jar**.

After you have localized the XML files, open a command prompt console, navigate to the directory that contains the JAR file, and run the command:

```bash
jar -cvf omsmodel_I10n.jar .
```
4. Choose option a or option b:

   a. Drop the existing schemas from the database so you can reuse the schema names when you reinstall (Step 5). This is done by running the commands below. Stop the WebLogic server prior to running these commands, and start it again upon completion.

   - OMS user schema:
     ```
     drop user oms_schema_name cascade; commit;
     ```

   - Rule engine schema:
     ```
     drop user rule_engine_schema cascade; commit;
     ```

   - The context:
     ```
     drop context oms_schema_name; commit;
     ```

   b. When reinstalling (Step 5), create a new schema with a different name.

5. Reinstall OSM, this time selecting all of the components you want to install permanently. When the Database Schema Localization Information window appears, select the **Use localized Order and Service Management initial data** check box and specify the path to your localized JAR file, as shown in Figure 10–2, "Installer Window".

![Figure 10–2 Installer Window](image)

The installer localizes the schema using your JAR file.

6. Localize the OSM Web clients as required.

After the database information has been localized and the product has been installed, you can customize the OSM Web clients to suit the needs of the target locale. For more information, see "Localizing the Task Web Client" and "Localizing
Localizing OSM

Localizing the XML Import/Export Application

If you localized OSM, you must also localize the XML Import/Export application. Localizing the XML Import/Export application involves the following high-level steps:

1. Uncompress the `OSM_home/SDK/XMLImportExport/classes/modelHandler.jar` file.
   
   Set the environment variable PATH to `JDK_home/bin`, then from a command prompt run the command:

   ```
   jar -xvf modelHandler.jar
   ```

2. Localize the `sql/default_data.sql` file.
   
   To localize the `default_data.sql` file, you must understand the relationship between the data that was localized in the XML files from the `omsmodel_l10n.jar` file, and the insert scripts defined in the `default_data.sql` file. The following example provides that understanding:

   Suppose `om_state.xml` of `omsmodel_l10n.jar` is localized. The localization corresponds to the INSERT statements of the table OM_STATE defined in `default_data.sql` in the `om_state.xml` file. For example, if the `<state_description>` of `<state_mnemonic>received</state_mnemonic>` is localized as `<state_description>TEMP_received</state_description>`, in `default_data.sql` the INSERT statement that is inserting `received` state in the OM_STATE table must be localized to:

   ```
   [INSERT INTO OM_STATE (STATE_ID, STATE_MNEMONIC, STATE_CATEGORY_ID, STATE_DESCRIPTION,STATE_ICON_ID, CARTRIDGE_ID)
   VALUES (1, 'received', 1, 'TEMP_received', 0, 0); ]
   ```

   Following this example, all fields that are localized in the XML files of `omsmodel_l10n.jar` must also be localized in `default_data.sql`.

3. Compress `modelHandler.jar`.
   
   After you have localized the `sql/default_data.sql` file, open a command prompt, navigate to the directory that contains the JAR file, and run the command:

   ```
   jar -cvf modelHandler.jar .
   ```

Localizing the Encoding Element

To support localization, the `<encoding>` element in the `config.xml` file determines the appropriate language. For information on the `config.xml` file, see OSM System Administrator's Guide.

- Configure the following line in the `config.xml` file with the appropriate ISO character code, shown in bold, below:

  ```
  <encoding>ISO-8859-2</encoding>
  ```

- Be sure that your Microsoft Windows operating system is localized for your language. See Microsoft documentation for instructions.

- Some pattern restrictions for entity names in the XML model schema are provided with the XML Import/Export application. For non-English environments, you must customize the schema for entity name patterns.
Additional Considerations for Localizing OSM

The process of localization also involves the support for different locales, character set encoding, and localization of settings.

Support for Different Locales

Locales are linguistic regions that share spelling conventions. A locale consists of a language code, followed by an optional country code, followed by an optional variant code. For example, "en_US" specifies English in the United States, while "en_GB" specifies English in Great Britain.

Operating systems such as Windows, Linux, Oracle Solaris, and HP-UX support the locale model and provide facilities to properly read and format locale-specific information. Additionally, newer programming languages such as Java provide similar facilities to support localization.

However, HTML does not support localization or have solutions for date or time issues. The localization of HTML requires support from the OSM server and Javascript code sent to the browser.

Character Set Encoding

Characters in an HTML file are stored as numeric values with a range from 0 to 255. When a Web browser displays a character symbol, it uses the numeric value to find the correct symbol to display. The set of symbols displayed for each number is known as a character set. As 256 numbers is insufficient to describe all possible characters that may need to be displayed in all languages, you must specify what character set the browser is to use when it displays an HTML page.

By default, HTML uses the ISO-8859-1 character set, which can display all characters needed for Western European Languages. When you create HTML pages using other character sets (such as ISO-8859-2 for Eastern European Languages), you must use the corresponding encoding.

Localization of Settings

OSM supports localized Windows regional settings: All number, currency, date, and time formats displayed in the OSM Administrator application come from the Windows regional settings.

You can also localize regional settings through the OSM properties file. See "Task Web Client Localization Resource Bundles" for more information on how to do this.

For more information on Windows regional settings, see the Microsoft Windows documentation.

About NLS Database Configuration

Oracle’s National Language Support (NLS) architecture allows you to store, process, and retrieve data in native languages. It ensures that database utilities and error messages, sort order, date, time, monetary, numeric, and calendar conventions automatically adapt to the native language and locale.

Oracle Database Character Set

OSM stores its text data in CHAR and VARCHAR2 columns in an Oracle database. The database character set determines what languages can be represented in the
You must install OSM in an Oracle database with a character set that meets your language requirements. You can specify a character set when creating a database using the \texttt{CHARACTER SET} clause of the \texttt{CREATE DATABASE} statement. A complete list of character sets supported by Oracle is included in the Oracle documentation.

You can use the \texttt{v$nls\_parameters} view to determine the existing database character set. OSM does not use the NCHAR, NVARCHAR2, or NCLOB data types and so has no specific requirements for the alternate character set for the database.

To determine the existing character set:

1. Connect to the Oracle server using SQL*Plus as a user who has access to the \texttt{v$nls\_parameters} view.
2. At the SQL*Plus prompt, enter the following:
   \begin{verbatim}
   SQL> select value from v$nls\_parameters where parameter = 'NLS\_CHARACTERSET';
   \end{verbatim}

   Information similar to the following is returned:
   \begin{verbatim}
   VALUE
   WE8ISO8859P1
   \end{verbatim}

   You can use the \texttt{CHARACTER SET} clause of the \texttt{ALTER DATABASE} statement to change the character set for an existing database.

   \begin{table}
   \centering
   \begin{tabular}{|c|}
   \hline
   \textbf{Note:} You must have SYSDBA system privileges.
   \hline
   \end{tabular}
   \end{table}

   To change the character set:
   
   1. Initiate the database in restricted mode. For example, use the SQL*Plus \texttt{STARTUP RESTRICT} command.
   
   2. Connect to server Manager (svrmgrl in UNIX or svrmgr30 on Windows XP).
   
   3. At the prompt, enter:
      \begin{verbatim}
      ALTER DATABASE db1 CHARACTER SET WE8ISO8859P1;
      \end{verbatim}

      \begin{table}
      \centering
      \begin{tabular}{|c|}
      \hline
      \textbf{Note:} The source character set must be a strict subset of the target character set.
      \hline
      \end{tabular}
      \end{table}

   \textbf{NLS Environment}

   For the OSM Administrator application to support localization, the Oracle NLS parameters must be configured properly in Windows XP.

   On UNIX, the Oracle NLS parameters are configured as environment variables. On Windows XP, the Oracle NLS parameters are configured as registry entries under \texttt{HKEY\_LOCAL\_MACHINE/SOFTWARE/ORACLE}.

   \textbf{NLS\_LANG Parameter}

   Use the NLS\_LANG parameter to set the language, territory, and character set used for the database.
The NLS_LANG parameter has three components: language, territory, and charset, in the form:

\[
\text{NLS\_LANG} = \text{language\_territory.charset}
\]

Each component controls the operation of a subset of the NLS features.

- **Language**: Specifies conventions such as the language used for Oracle messages, as well as day and month names. Each supported language has a unique name, such as French, or German.

- **Territory**: Specifies conventions such as the default calendar, collation, date, monetary, and numeric formats. Each supported territory has a unique name, such as America, France, or Canada.

- **Charset**: Specifies the character set used by the client application. Each supported character set has a unique acronym, such as US7ASCII, WE8ISO8859P1, WE8DEC, WE8EBCDIC500, or JA16EUC.

**ORA_NLS33 Environment Variable**

Ensure the ORA_NLS33 environment is set to `Oracle_home/nls/data`.

You can also set other NLS environment variables. For a complete list of NLS environment variables, see the Oracle documentation.

---

**About OSM Database Error Messages**

You can localize OSM database error messages by editing the error message text in the `om_errors.xml` file.

This file is located in your `OSM_home/Database/dbinstall/install/OMS/main/model/utility/_table/_data` directory.

To edit these error messages, open the file using an XML or text editor and replace the "error_message" text with the target language text. Table 10–1 lists all of the error messages in the `om_errors.xml` file.

**Note:** After you have localized this file, make sure you provide an updated copy to Oracle Global Support to assist you in any issues that may arise.

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>-20001</td>
<td>Order satisfied no rule. Cannot start process.</td>
</tr>
<tr>
<td>-20002</td>
<td>Order status and order state have not been changed.</td>
</tr>
<tr>
<td>-20004</td>
<td>Starting position for the process is not defined. Call support.</td>
</tr>
<tr>
<td>-20005</td>
<td>There is no creation task. Call support.</td>
</tr>
<tr>
<td>-20006</td>
<td>Workflow thread does not exist. The order may have been moved by another user. Refresh your worklist and try again.</td>
</tr>
<tr>
<td>-20007</td>
<td>Order is locked. Try again later.</td>
</tr>
<tr>
<td>-20008</td>
<td>Current history detail record not found for the order. Call support.</td>
</tr>
<tr>
<td>-20009</td>
<td>Cannot remove node, which has children.</td>
</tr>
</tbody>
</table>
### Table 10–1 (Cont.) OSM Error Messages

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>-20011</td>
<td>Order to be updated is currently locked by another user.</td>
</tr>
<tr>
<td>-20012</td>
<td>Order not found.</td>
</tr>
<tr>
<td>-20014</td>
<td>Rule to be evaluated is not found. Call support.</td>
</tr>
<tr>
<td>-20015</td>
<td>System data is protected.</td>
</tr>
<tr>
<td>-20016</td>
<td>Value cannot be null.</td>
</tr>
<tr>
<td>-20017</td>
<td>Rule cannot be evaluated.</td>
</tr>
<tr>
<td>-20020</td>
<td>Operation affected too many rows.</td>
</tr>
<tr>
<td>-20022</td>
<td>Access denied: not enough privileges.</td>
</tr>
<tr>
<td>-20023</td>
<td>You cannot accept this order thread.</td>
</tr>
<tr>
<td>-20025</td>
<td>There are too many instances for the order node.</td>
</tr>
<tr>
<td>-20026</td>
<td>There are too few instances for the order node.</td>
</tr>
<tr>
<td>-20027</td>
<td>Parent order node does not exist.</td>
</tr>
<tr>
<td>-20028</td>
<td>Rule engine has already been started.</td>
</tr>
<tr>
<td>-20029</td>
<td>Rule engine has not been started.</td>
</tr>
<tr>
<td>-20030</td>
<td>Invalid process definition.</td>
</tr>
<tr>
<td>-20031</td>
<td>Order node cannot be found. Call support.</td>
</tr>
<tr>
<td>-20038</td>
<td>Mandatory check failed.</td>
</tr>
<tr>
<td>-20040</td>
<td>Order view for task cannot be found.</td>
</tr>
<tr>
<td>-20041</td>
<td>Process position cannot be found. Call support.</td>
</tr>
<tr>
<td>-20044</td>
<td>Reporting status cannot be found. Call support.</td>
</tr>
<tr>
<td>-20049</td>
<td>Status is not valid for task.</td>
</tr>
<tr>
<td>-20050</td>
<td>View node cannot be found.</td>
</tr>
<tr>
<td>-20051</td>
<td>Notification cannot be found.</td>
</tr>
<tr>
<td>-20054</td>
<td>Notification history cannot be found.</td>
</tr>
<tr>
<td>-20055</td>
<td>Notification is not active.</td>
</tr>
<tr>
<td>-20056</td>
<td>Time interval is not valid.</td>
</tr>
<tr>
<td>-20043</td>
<td>Jump record does not exist.</td>
</tr>
<tr>
<td>-20059</td>
<td>Error processing notification.</td>
</tr>
<tr>
<td>-20060</td>
<td>Node information does not match any node in the database.</td>
</tr>
<tr>
<td>-20061</td>
<td>New parent node information is not valid.</td>
</tr>
<tr>
<td>-20062</td>
<td>Remark cannot be found.</td>
</tr>
<tr>
<td>-20063</td>
<td>The remark cannot be changed.</td>
</tr>
<tr>
<td>-20064</td>
<td>Attachment cannot be found.</td>
</tr>
<tr>
<td>-20065</td>
<td>No orders are stored in the system.</td>
</tr>
<tr>
<td>-20067</td>
<td>No orders satisfy the purge criteria.</td>
</tr>
<tr>
<td>-20068</td>
<td>The wrong task has been supplied for the order.</td>
</tr>
<tr>
<td>-20069</td>
<td>Invalid state transition.</td>
</tr>
</tbody>
</table>
### Table 10–1  (Cont.) OSM Error Messages

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>-20070</td>
<td>Invalid order type.</td>
</tr>
<tr>
<td>-20071</td>
<td>Invalid order source.</td>
</tr>
<tr>
<td>-20072</td>
<td>Invalid process status.</td>
</tr>
<tr>
<td>-20073</td>
<td>Invalid state.</td>
</tr>
<tr>
<td>-20074</td>
<td>State is not valid for task.</td>
</tr>
<tr>
<td>-20075</td>
<td>Task not found.</td>
</tr>
<tr>
<td>-20076</td>
<td>Stop date/time must be greater than start date/time.</td>
</tr>
<tr>
<td>-20077</td>
<td>Shift violates schedule boundary.</td>
</tr>
<tr>
<td>-20078</td>
<td>Please change the shift boundaries not to overlap any existing shift.</td>
</tr>
<tr>
<td>-20079</td>
<td>Exception shift not found.</td>
</tr>
<tr>
<td>-20080</td>
<td>Parent region is not found.</td>
</tr>
<tr>
<td>-20081</td>
<td>Cannot remove region, which has child regions.</td>
</tr>
<tr>
<td>-20082</td>
<td>Cannot remove region, which is attached to workgroup.</td>
</tr>
<tr>
<td>-20083</td>
<td>Cannot move to the next task. Cannot calculate expected completion time. Schedule is too short. Ask OSM administrator to extend the calendar.</td>
</tr>
<tr>
<td>-20084</td>
<td>Cannot move to the next task. Cannot calculate expected completion time. Cannot find the schedule.</td>
</tr>
<tr>
<td>-20085</td>
<td>Invalid parameter when you call internal function.</td>
</tr>
<tr>
<td>-20086</td>
<td>DST start/stop date is not correctly specified in om_workgroup table.</td>
</tr>
<tr>
<td>-20087</td>
<td>Can not find workgroup during the building of DST date.</td>
</tr>
<tr>
<td>-20088</td>
<td>Invalid calendar DST start/stop day of the month in the workgroup settings.</td>
</tr>
<tr>
<td>-20089</td>
<td>Incorrect value in DST week settings in the workgroup definition.</td>
</tr>
<tr>
<td>-20090</td>
<td>Missing parameter &quot;oms_timezone&quot; in OM_PARAMETER table.</td>
</tr>
<tr>
<td>-20091</td>
<td>The node is used as a coordinator node for this thread.</td>
</tr>
<tr>
<td>-20092</td>
<td>Parameter Stop Date must be greater than Current System Date.</td>
</tr>
<tr>
<td>-20093</td>
<td>Parameter Completion Date Before should be less than Current System Date.</td>
</tr>
<tr>
<td>-20094</td>
<td>Calendar can be generated only workgroup by workgroup not for all workgroups at once.</td>
</tr>
<tr>
<td>-20095</td>
<td>Parent thread is not found.</td>
</tr>
<tr>
<td>-20096</td>
<td>Partition boundary is too low. Increase the value of NEXT RANGE PARTITION BOUNDARY parameter.</td>
</tr>
<tr>
<td>-20097</td>
<td>Specific view is not assigned to a workgroup.</td>
</tr>
<tr>
<td>-20098</td>
<td>Specific task is not assigned to workgroup.</td>
</tr>
<tr>
<td>-20099</td>
<td>JMS message does not exists.</td>
</tr>
<tr>
<td>-20103</td>
<td>Event type is not recognized. Call support.</td>
</tr>
<tr>
<td>-20104</td>
<td>Increase value of job_queue_processes parameter in the init*.ora file.</td>
</tr>
<tr>
<td>-20105</td>
<td>Job type is not valid. Call support.</td>
</tr>
<tr>
<td>-20106</td>
<td>Minimum running jobs should be greater than 0.</td>
</tr>
<tr>
<td>-20107</td>
<td>It can be only one notification job running at the same.</td>
</tr>
</tbody>
</table>
About Application Server Strings

Application server strings are used in Java business application code in the OSM UI. These strings can be found in the `om_server_strings.xml` file in your `OSM_home/Database/dbinstall/install/OMS/main/model/utility/_table/_data` directory.

To edit these strings, open an XML or text editor and replace the "description" text with the target language text.

The strings shown in Table 10–2 are representative of the types of information displayed in the Worklist, Query, and Notifications views. For a full listing, see the `om_server_strings.xml` file.

### Table 10–2 Server Strings

<table>
<thead>
<tr>
<th>Class</th>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>order_history</td>
<td>NODEINST</td>
<td>Order Instance</td>
</tr>
<tr>
<td>order_history</td>
<td>FILTEREDNODE</td>
<td>Filtered Node</td>
</tr>
<tr>
<td>fixed_header</td>
<td>jdate_pos_started</td>
<td>&quot;Started&quot;</td>
</tr>
<tr>
<td>fixed_header</td>
<td>jexpected_duration</td>
<td>&quot;Expected Duration&quot;</td>
</tr>
</tbody>
</table>
### Table 10–2 (Cont.) Server Strings

<table>
<thead>
<tr>
<th>Class</th>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fixed_header</td>
<td>jexpected_start_date</td>
<td>&quot;Expected Start Date&quot;</td>
</tr>
<tr>
<td>fixed_header</td>
<td>jorder_creation_date</td>
<td>&quot;Order Creation Date&quot;</td>
</tr>
<tr>
<td>fixed_header</td>
<td>jord_compl_date_expected</td>
<td>&quot;Expected Order Completion Date&quot;</td>
</tr>
<tr>
<td>fixed_header</td>
<td>jorder_seq_id</td>
<td>&quot;Order ID&quot;</td>
</tr>
<tr>
<td>fixed_header</td>
<td>jorder_source</td>
<td>&quot;Source&quot;</td>
</tr>
<tr>
<td>fixed_header</td>
<td>jorder_state</td>
<td>&quot;State&quot;</td>
</tr>
<tr>
<td>fixed_header</td>
<td>jorder_type</td>
<td>&quot;Type&quot;</td>
</tr>
<tr>
<td>fixed_header</td>
<td>jprocess_description</td>
<td>&quot;Process&quot;</td>
</tr>
<tr>
<td>fixed_header</td>
<td>jreference_number</td>
<td>&quot;Ref. #&quot;</td>
</tr>
<tr>
<td>fixed_header</td>
<td>jrequesting_status</td>
<td>&quot;Process Status&quot;</td>
</tr>
<tr>
<td>fixed_header</td>
<td>jrequested_delivery_date</td>
<td>&quot;Requested Delivery Date&quot;</td>
</tr>
<tr>
<td>fixed_header</td>
<td>jtask_description</td>
<td>&quot;Task&quot;</td>
</tr>
<tr>
<td>fixed_header</td>
<td>juser</td>
<td>&quot;User&quot;</td>
</tr>
<tr>
<td>notification_fixed_header</td>
<td>_NOTIFICATION_DESC</td>
<td>&quot;Notification Description&quot;</td>
</tr>
<tr>
<td>notification_fixed_header</td>
<td>_NOTIFICATION_TYPE</td>
<td>&quot;Notification Type&quot;</td>
</tr>
<tr>
<td>notification_fixed_header</td>
<td>_PRIORITY</td>
<td>&quot;Priority&quot;</td>
</tr>
<tr>
<td>notification_fixed_header</td>
<td>_TIMESTAMP</td>
<td>&quot;Notification Timestamp&quot;</td>
</tr>
<tr>
<td>GUIMessage</td>
<td>ALT</td>
<td>&quot;View Remark(s)&quot;</td>
</tr>
<tr>
<td>fixed_header</td>
<td>jcompletion_date_expected_at_task</td>
<td>&quot;Expected Task Completion Date&quot;</td>
</tr>
<tr>
<td>fixed_header</td>
<td>jorder_completion_date</td>
<td>&quot;Completed Date&quot;</td>
</tr>
<tr>
<td>fixed_header</td>
<td>jnum_remarks</td>
<td>&quot;Number of Remarks&quot;</td>
</tr>
<tr>
<td>fixed_header</td>
<td>jorder_hist_seq_id</td>
<td>&quot;Order History ID&quot;</td>
</tr>
<tr>
<td>fixed_header</td>
<td>jtask_workgroup</td>
<td>&quot;Workgroups&quot;</td>
</tr>
</tbody>
</table>

### About OSM Process Definition Data

This section lists the process definition data used in OSM database tables. The data is contained in various files located in your `OSM_home/Database/dbinstall/install/OMS/main/model` directory.

- om_application_function
- om_attribute_type
- om_order_data_dictionary
- om_order_remarks_type
- om_order_type_category
- om_process
To edit these tables, open the file using an XML or text editor and replace the "description" text with the target language text.

**Note:** Do not localize the column names. Column names are not externally visible, and are necessary for the internal operation of OSM.

After you localize any of these files, make sure you provide an updated copy to Oracle Global Support, so they can assist you with any future issues that may arise.

**om_application_function**

Table 10–3 lists the application functions defined for OSM. The file is located in your $OSM_{home}/Database/dbinstall/install/OMS/main/model/user/privilege/_table/_data$ directory. Localize the values in the right column.

<table>
<thead>
<tr>
<th>app_function_mnemonic</th>
<th>app_function_description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reference_number_modification</td>
<td>Reference Number Modification</td>
</tr>
<tr>
<td>priority_modification</td>
<td>Order Priority Modification</td>
</tr>
<tr>
<td>online_reports</td>
<td>Online Reports</td>
</tr>
<tr>
<td>worklist_viewer</td>
<td>Worklist Viewer</td>
</tr>
<tr>
<td>task_assignment</td>
<td>Task Assignment</td>
</tr>
<tr>
<td>exception_processing</td>
<td>Exception Processing</td>
</tr>
<tr>
<td>search_viewer</td>
<td>Search Viewer</td>
</tr>
<tr>
<td>create_versioned_orders</td>
<td>Create Versioned Orders</td>
</tr>
</tbody>
</table>

**om_attribute_type**

Table 10–4 lists the attribute types used by OSM. The file is located in your $OSM_{home}/Database/dbinstall/install/OMS/main/model/ordertemplate/_table/_data$ directory. Localize the values in the description column.

<table>
<thead>
<tr>
<th>attribute_type</th>
<th>attribute_description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CN</td>
<td>Container</td>
</tr>
<tr>
<td>CT</td>
<td>Code Table</td>
</tr>
<tr>
<td>CY</td>
<td>Currency</td>
</tr>
</tbody>
</table>
om_order_data_dictionary

The data dictionary contains one element, which is a dictionary element for the root node in the OSM master order template. This file is located in your OSM_home/Database/dbinstall/install/OMS/main/model/ordertemplate/_table/_data directory. Localize the text that appears in the business_name column.

<table>
<thead>
<tr>
<th>Database Column</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>business_name</td>
<td>ROOT</td>
</tr>
</tbody>
</table>

om_order_remarks_type

This file is located in your OSM_home/Database/dbinstall/install/OMS/main/model/ordertemplate/_table/_data directory. Localize the text that appears in the remarks_type_description column.

<table>
<thead>
<tr>
<th>Database Column</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>remarks_type_description</td>
<td>Default</td>
</tr>
</tbody>
</table>

om_order_type_category

This file is located in your OSM_home/Database/dbinstall/install/OMS/main/model/ordertemplate/_table/_data directory. Localize the text that appears in the order_type_category descr column.

<table>
<thead>
<tr>
<th>Database Column</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>order_type_category_descr</td>
<td>Unknown</td>
</tr>
</tbody>
</table>

om_process

This file is located in your OSM_home/Database/dbinstall/install/OMS/main/model/process/_table/_data directory. Localize the text that appears in the process_id_description column.
Table 10–8  om_process

<table>
<thead>
<tr>
<th>Database Column</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>process_id_description</td>
<td>Creation Process</td>
</tr>
<tr>
<td></td>
<td>N/A</td>
</tr>
</tbody>
</table>

om_process_status

This file is located in your OSM_home/Database/dbinstall/install/OMS/main/model/process/_table/_data directory. Localize the text that appears in the process_status_description column.

Table 10–9  om_process_status

<table>
<thead>
<tr>
<th>Database Column</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>process_status_description</td>
<td>Possible values are:</td>
</tr>
<tr>
<td></td>
<td>• N/A</td>
</tr>
<tr>
<td></td>
<td>• True</td>
</tr>
<tr>
<td></td>
<td>• False</td>
</tr>
<tr>
<td></td>
<td>• Submit</td>
</tr>
<tr>
<td></td>
<td>• Delete</td>
</tr>
<tr>
<td></td>
<td>• Next</td>
</tr>
<tr>
<td></td>
<td>• Back</td>
</tr>
<tr>
<td></td>
<td>• Cancel</td>
</tr>
<tr>
<td></td>
<td>• Finish</td>
</tr>
<tr>
<td></td>
<td>• Back</td>
</tr>
</tbody>
</table>

om_responsibility

This file is located in your OSM_home/Database/dbinstall/install/OMS/main/model/user/privilege/_table/_data directory. Localize the text that appears in the responsibility_description column.

Table 10–10  om_responsibility

<table>
<thead>
<tr>
<th>Database Column</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>responsibility_description</td>
<td>System</td>
</tr>
</tbody>
</table>

om_rule_def

This file is located in your OSM_home/Database/dbinstall/install/OMS/main/model/rule/_table/_data directory. Localize the text that appears in the rule_description and rule_comment columns.

Null Rule is the rule that always evaluates to true.

Table 10–11  om_rule_def

<table>
<thead>
<tr>
<th>Database Column</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule_description</td>
<td>Null Rule</td>
</tr>
<tr>
<td>rule_comment</td>
<td>Null Rule</td>
</tr>
</tbody>
</table>
**om_state**

This file is located in your OSM_home/Database/dbinstall/install/OMS/main/model/process/_table/_data directory. Localize the text that appears in the state_description column.

<table>
<thead>
<tr>
<th>Database Column</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>state_description</td>
<td>Possible values are:</td>
</tr>
<tr>
<td></td>
<td>■ N/A</td>
</tr>
<tr>
<td></td>
<td>■ Received</td>
</tr>
<tr>
<td></td>
<td>■ Accepted</td>
</tr>
<tr>
<td></td>
<td>■ Assigned</td>
</tr>
<tr>
<td></td>
<td>■ Completed</td>
</tr>
</tbody>
</table>

**om_state_category**

This file is located in your OSM_home/Database/dbinstall/install/OMS/main/model/process/_table/_data directory. Localize the text that appears in the state_category_description column.

<table>
<thead>
<tr>
<th>Database Column</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>state_category_description</td>
<td>Possible values are:</td>
</tr>
<tr>
<td></td>
<td>■ System</td>
</tr>
<tr>
<td></td>
<td>■ User defined</td>
</tr>
</tbody>
</table>

**om_status_category**

This file is located in your OSM_home/Database/dbinstall/install/OMS/main/model/process/_table/_data directory. Localize the text that appears in the status_category_description column.

<table>
<thead>
<tr>
<th>Database Column</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>status_category_description</td>
<td>Possible values are:</td>
</tr>
<tr>
<td></td>
<td>■ System</td>
</tr>
<tr>
<td></td>
<td>■ User defined</td>
</tr>
</tbody>
</table>

**om_task**

file is located in your OSM_home/Database/dbinstall/install/OMS/main/model/process/_table/_data directory. Localize the text that appears in the task_description column.
### About Generic Preferences

This section lists the generic user preferences in the OSM database tables. The file is located in your `OSM_home/Database/dbinstall/install/OMS/main/model/user/preference/_table/_data` directory. Localize the text that appears in the label column.

#### om_view_order_node_label

This file is located in your `OSM_home/Database/dbinstall/install/OMS/main/model/ordertemplate/_table/_data` directory. Localize the text that appears in the label column.

<table>
<thead>
<tr>
<th>Database Column</th>
<th>Value</th>
</tr>
</thead>
</table>
| task_description | Possible values are:  
  ■ Null Task  
  ■ N/A  
  ■ Recycle |

#### om_generic_mnemonic

Localize the text that appears in the description column.

<table>
<thead>
<tr>
<th>CLASS Column</th>
<th>Description</th>
</tr>
</thead>
</table>
| worklist_filter | Possible values are:  
  ■ Accepted by current user  
  ■ Assigned to current user  
  ■ Received  
  ■ Suspended  
  ■ Completed |
### Table 10–17  (Cont.) om_generic_mnemonic

<table>
<thead>
<tr>
<th>CLASS Column</th>
<th>Description</th>
</tr>
</thead>
</table>
| colour       | Possible values are:  
  - Black  
  - White  
  - Gray  
  - Silver  
  - Red  
  - Green  
  - Blue  
  - Yellow  
  - Purple  
  - Olive  
  - Navy  
  - Aqua  
  - Lime  
  - Maroon  
  - Teal  
  - Fuschia |
| order_list   | Possible values are:  
  - Worklist  
  - Search List  
  - Notification List |
| user_attr    | Possible values are:  
  - Email address  
  - User Local Timezone  
  - Enable Worklist Filter |
| dynamic_filter | Possible values are:  
  - Worklist filter field mnemonic  
  - Worklist filter condition  
  - Worklist filter lower value  
  - Worklist filter upper value  
  - Worklist filter display lower value  
  - Worklist filter display upper value |
| login_screen | Possible values are:  
  - System Default  
  - Worklist  
  - Home  
  - Query  
  - About |
Localizing the Task Web Client

The Task Web client is a Web-based application that provides information dynamically and a set of interactions that the end-user can perform on this information. The information appears as HTML pages, but server-side technologies produce the HTML output and provide user interaction.

For information about the Task Web client architecture, see *OSM Task Web Client User’s Guide*.

You can localize the following Task Web client elements:

- **Text Strings**: Every non-database driven text message, including labels for fields and buttons.
- **Page titles**
- **Hyperlink text**
- **Images**: You can replace standard images with localized versions; you can also localize the alternate text for each image.
- **Dates and Date/Times**: You can display all date and date/time values in the locally preferred format.
- **Numerics**: You can display all numeric values in the locally preferred format.
- **Currency format**
- **Error messages**
- **Log messages**

The Task Web client uses the Java language, which has built-in support for localization of dates and times, as well as string sorting capabilities. The Task Web client uses the locale of the operating system upon which it runs in order to determine regional date and time settings.

When a user connects to OSM using their web browser, the OSM server attempts to determine their preferred locale. If no customization can be found for their preferred locale, the default locale (English) appears.

All modern Web-browsers support an Accept-Language header which indicates locale preferences. Multiple language preferences can be specified, however OSM uses the top one listed to locate localization resources.

To localize the Task Web client:

1. Unpack the *osm.ear* file as described in *OSM System Administrator’s Guide*.
2. Localize and customize the files in the *oms.ear/resources* directory as described in the following sections:

   - **Localizing Date, Time and Currency Formats**
   - **Localizing Text and Error Messages**
   - **Localizing Page Titles**
   - **Localizing Image References**
   - **Inserting New Images**
   - **Editing the First Day of the Week**
   - **Editing the Task Web Client Gantt Chart**
   - **Editing the Boolean Data Element Values**
- Editing the Number of Records Displayed in the Worklist
- Editing and Replacing Task Web Client Icons

3. Pack the oms.ear file and redeploy it to the WebLogic server as described in OSM System Administrator’s Guide.

Task Web Client Localization Resource Bundles

After you unpack the oms.ear file as described in OSM System Administrator’s Guide, the localizable resources for the Task Web client are contained in the OSM_home/SDK/Customization/resources/resources directory.

This directory contains the resources.properties file. This file contains all localizable strings and image references for the Task Web client. Several other properties files are included in the resources directory as samples, including:

- resources_cs.properties (localized for the Czech Republic)
- resources_zh.properties (localized for China)
- resources_en.properties

To create a new localization property file:

1. Copy the resources.properties file and create a new file using the following naming convention:

   resources_locale.properties

   where locale is a locale code. For example, resources_ja.properties for Japan.

2. For each parameter in the resources_locale.properties file, provide a replacement value in the new locale. If you do not provide a replacement value, the Task Web client uses the default value.

Localizing Date, Time and Currency Formats

To differentiate between 2:00:00 pm EST and 2:00:00 pm CST, modify the resource.properties file by changing the format.datetime.input setting to include time zone.

To specify this, you must:

1. Open the resources.properties file.
2. Specify date time input and display format. For example:

   format.datetime.input=MM/dd/yy hh:mm:ss a
   format.datetime.display=MM/dd/yy hh:mm:ss a

<table>
<thead>
<tr>
<th>Table 10–18 Localizing Data Formats</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Formats</td>
</tr>
<tr>
<td>format.encoding</td>
</tr>
<tr>
<td>format.currency</td>
</tr>
<tr>
<td>format.date.input</td>
</tr>
<tr>
<td>format.date.display</td>
</tr>
<tr>
<td>format.datetime.input</td>
</tr>
<tr>
<td>format.datetime.display</td>
</tr>
</tbody>
</table>
You must edit the mask.date and mask.time properties together. You cannot edit one property and leave the other one empty.

The available values for the properties include the following.

**Table 10–19 Properties Values**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
<th>Presentation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>year</td>
<td>Number</td>
<td>2005</td>
</tr>
<tr>
<td>M</td>
<td>month in year</td>
<td>Text &amp; Number</td>
<td>July &amp; 07</td>
</tr>
<tr>
<td>d</td>
<td>day in month</td>
<td>Number</td>
<td>10</td>
</tr>
<tr>
<td>h</td>
<td>hour in am/pm (1~12)</td>
<td>Number</td>
<td>12</td>
</tr>
<tr>
<td>H</td>
<td>hour in day (0~23)</td>
<td>Number</td>
<td>0</td>
</tr>
<tr>
<td>m</td>
<td>minute in hour</td>
<td>Number</td>
<td>30</td>
</tr>
<tr>
<td>s</td>
<td>second in minute</td>
<td>Number</td>
<td>55</td>
</tr>
<tr>
<td>E</td>
<td>day in week</td>
<td>Text</td>
<td>Tuesday</td>
</tr>
<tr>
<td>D</td>
<td>day in year</td>
<td>Number</td>
<td>189</td>
</tr>
<tr>
<td>w</td>
<td>week in year</td>
<td>Number</td>
<td>27</td>
</tr>
<tr>
<td>a</td>
<td>am/pm marker</td>
<td>Text</td>
<td>PM</td>
</tr>
<tr>
<td>'</td>
<td>escape for text</td>
<td>Delimiter</td>
<td>N/A</td>
</tr>
<tr>
<td>“</td>
<td>single quote</td>
<td>Literal</td>
<td>‘</td>
</tr>
</tbody>
</table>

The available values for the mask.currency include the following.

**Table 10–20 Mask Currency Values**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
<th>Presentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>zeros show as absent</td>
<td>Number</td>
</tr>
<tr>
<td>0</td>
<td>zeros show as 0</td>
<td>Number</td>
</tr>
<tr>
<td>,</td>
<td>the locale-specific grouping separator</td>
<td>Text</td>
</tr>
<tr>
<td>-</td>
<td>the locale-specific negative prefix</td>
<td>Text</td>
</tr>
<tr>
<td>;</td>
<td>separates positive number format from optional negative number</td>
<td>Text</td>
</tr>
<tr>
<td>'</td>
<td>escape for text format from optional negative number</td>
<td>Delimiter</td>
</tr>
<tr>
<td>other</td>
<td>all other symbols appear as entered</td>
<td>Text</td>
</tr>
</tbody>
</table>

You must change the on-window text of the HTML pages produced by the OSM UI, and, if necessary, indicate the character set encoding.
Localizing Text and Error Messages

If text and error messages (that is, messages in the resource file that begin with text or error), contain parameterized values, for example: {0}, {1}, {2}, and so on, ensure the localized message uses the same parameterized values.

Localizing Page Titles

Page titles in the resource file begin with page.

Localizing Image References

Image references consist of two parameters:

- image.name.alt: The HTML alternate text to display for the image
- image.name.src: The location of the image file in oms.war (see "Inserting New Images").

Inserting New Images

All images that display in the Task Web client are contained in the oms.war file packed in the oms.ear file. You can insert new images anywhere inside oms.war as long as your reference the correct location in the locale’s resource.properties file. Oracle recommends that you create a directory for each localization and name the directory images_locale, where locale is a code for the location.

Editing the First Day of the Week

By default, the Task Web client date and time calendar assumes Sunday is the first day of the week. To change the default first day of the week, edit the calendar.js file and follow the instructions provided.

---

Note: When displaying lists, the OSM Administrator application always displays Sunday as the first day of the week.

---

Editing the Task Web Client Gantt Chart

You can customize the Gantt chart’s width, height, and property panel width by modifying the following properties in the GanttChart.jsp file:

- var chartMaximumWidth = 1024;
- var chartMaximumHeight = 1000;
- var propertyPanelWidth = 200;

Editing the Boolean Data Element Values

The Task Web client displays boolean data element values in drop-down lists in several views. You can change the Boolean data element values the Task Web client displays by editing the Boolean display value fields in the resources.properties file.

For example:

```plaintext
# Boolean display values
text.boolean.yes=Yes
text.boolean.no=No
```
Editing the Number of Records Displayed in the Worklist

You can change the number of records displayed in the Worklist view from the default by editing the `oms-config.xml` file.

See the chapter on configuring OSM with `oms-config.xml` in *OSM System Administrator’s Guide* for detailed instructions on accessing and modifying the `oms-config.xml` file.

To change the number of records displayed in the Worklist view:

1. Open the `oms-config.xml` file for editing.
2. Search for `max_worklist_rows`, and update the `<oms-parameter-value>` tag with the new value.
3. Save and close the `oms-config.xml` file.

Editing and Replacing Task Web Client Icons

You can edit or replace the icon graphics that appear in the Task Web client. To do this, replace the graphical content of the existing icon files with your own, customized graphics.

When creating or editing an icon graphic, ensure you maintain the file name. For example, if you replace the graphical content of the `about.gif` file, ensure you name the resulting file `about.gif`.

Oracle recommends that you only customize or replace existing icon graphics if the original graphic introduces ambiguities or errors when you localize the Task Web client.

All of the OSM icon files are located in the `OSM_home/SDK/Customization/osm-war/images` directory.

Localizing the Order Management Web Client

As with the Task Web client, all language-specific text exposed by the Order Management Web client is localizable.

To localize the Order Management Web client:

1. Unpack the `oms.ear` file as described in *OSM System Administrator’s Guide*.
2. Localize the files located in the `OSM_home/SDK/Customization/resources/xliff` directory.

The Order Management Web client defines one XLIFF file per localizable application page.

3. Register the default and supported locales:
   a. Open the following file for editing:
      
      `OSM_home/SDK/Customization/osmwebui/WEB-INF/faces-config.xml`
      
   b. Search for the `<local-config>` tag.
   c. Specify the default and supported locales as follows:
      
      `<locale-config>
       <default-locale>locale1</default-locale>
       <supported-locale>locale2</supported-locale>
      </locale-config>`
where \textit{locale1} is the default locale you want to register and \textit{locale2} is a different locale you want to register. See Example 10–4.

d. Save and close the file.


5. Pack and redeploy the \texttt{oms.ear} file as described in OSM System Administrator’s Guide.

Visit the OASIS Web site to learn about the XLIFF standard:

\url{http://docs.oasis-open.org/xliff/xliff-core/xliff-core.html}

Example 10–1 shows a simple XLIFF example.

\textbf{Example 10–1} \textit{XLIFF}

\begin{verbatim}
<?xml version="1.0" encoding="UTF-8" ?>
<xliff version="1.1" xmlns="urn:oasis:names:tc:xliff:document:1.1">
  <file source-language="en" original="i18n.view.i18nTestBundle" datatype="xml">
    <body>
      <trans-unit id="MESSAGE">
        <source>Hello world!</source>
        <target/>
        <note>The message to display</note>
      </trans-unit>
    </body>
  </file>
</xliff>
\end{verbatim}

This file specifies a single localizable token (\textit{MESSAGE}) as well as the text with which to replace the token (\texttt{Hello world!}) for the specified language (\texttt{en}, which is English). The \texttt{note} element provides a description of the context in which the token is used. As with the \texttt{resources.locale.properties} file, an XLIFF file may contain parameters. However, unlike the numeric parameters in \texttt{resources.locale.properties}, the parameters in the Order Management Web client XLIFF files are named. For example:

\begin{verbatim}
<source>Change Reference for Order \{ORDER_ID\}</source>
\end{verbatim}

Building on Example 10–1, the XLIFF file might look like Example 10–2.

\textbf{Example 10–2} \textit{XLIFF with \texttt{<source>}}

\begin{verbatim}
<?xml version="1.0" encoding="UTF-8" ?>
<xliff version="1.1" xmlns="urn:oasis:names:tc:xliff:document:1.1">
  <file source-language="en" original="i18n.view.i18nTestBundle" datatype="xml">
    <body>
      <trans-unit id="MESSAGE">
        <source>{SALUTATION} {RECIPIENT}!</source>
        <target/>
        <note>The message to display</note>
      </trans-unit>
    </body>
  </file>
</xliff>
\end{verbatim}
As with the `resources.locale.properties` file, each XLIFF file is localized by creating a new file with the language code and (optionally) country code appended. Building on Example 10–2, "XLIFF with `<source>`", if the contents are found in a default bundle file named `applicationBundle.xlf`, a version localized to the French language would be named `applicationBundle_fr.xlf`, as shown in Example 10–3.

**Example 10–3 XLIFF Language set to French**

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<xliff version="1.1" xmlns="urn:oasis:names:tc:xliff:document:1.1">
  <file source-language="fr" original="i18n.view.i18nTestBundle_fr" datatype="xml">
    <body>
      <trans-unit id="MESSAGE">
        <source>Bonjour le monde!</source>
        <target/>
        <note>The message to display</note>
      </trans-unit>
    </body>
  </file>
</xliff>
```

For more information about the `faces.config.xml` file, see *Oracle Fusion Middleware Web User Interface Developer’s Guide for Oracle Application Development Framework*.

Example 10–4 shows an example `faces.config.xml` with the language set to French and English.

**Example 10–4 faces.config.xml Language set to French and English**

```xml
<?xml version="1.0" encoding="windows-1252"?>
<faces-config version="1.2" xmlns="http://java.sun.com/xml/ns/javaee">
  <application>
    <default-render-kit-id>oracle.adf.rich</default-render-kit-id>
    <locale-config>
      <default-locale>en</default-locale>
      <supported-locale>en</supported-locale>
      <supported-locale>fr</supported-locale>
    </locale-config>
  </application>
</faces-config>
```

### Changing the Order Management Web Client Logo Image and Text

To change the Order Management Web Client logo image and text:

1. Go to the `OSM_home/SDK/Customization/osmwebui/images` folder.
2. Change any images with the Oracle logo. Do not change the image names or file types.
   
   For example:
   
   - **splash.jpg**: This image is used in the Order Management Web client login and about screens and contains the Oracle logo at the top left corner.
   - **splash.bmp**: This is the Order Management Web client background and contains the Oracle logo at the top left corner.
3. Open `OSM_home/SDK/Customization/resources/xliff/oracle/communications/ordermanagement/ui/order3columnBundle.xlf` using a text editor.
4. Change the text associated with the customized Oracle logo images.

For example:

```xml
<xml version="1.0" encoding="windows-1252" ?
<xlliff version="1.1" xmlns="urn:oasis:names:tc:xlliff:document:1.1">
<file source-language="en"
original="oracle.communications.ordermanagement.ui.order3columnBundle"
datatype="xml">
<body>
<trans-unit id="OSM_APP_NAME">
<source>custom_name</source>
<target/>
</trans-unit>
<trans-unit id="HOME_PAGE_TITLE">
<source>custom_name2</source>
<target/>
</trans-unit>
<trans-unit id="DENIED_SEARCH_MSG">
<source>You are not authorized to perform search</source>
<target/>
</trans-unit>
<trans-unit id="ERROR_MSG_SUBJECT">
<source>Error</source>
<target/>
</trans-unit>
</body>
</file>
</xlliff>
```

where `custom_name` is the text associated `splash.jpg` and `custom_name2` is the text associated with `splash.bmp`.

5. Save and close the file.
This chapter describes how to use XPath functions when modeling orders in Oracle Communications Order and Service Management (OSM).

**About XPath Functions**

The XPath language provides for a core library of functions that deal with:

- node sets
- strings
- Booleans
- numbers

The following are some examples of XPath functions:

- Determine the number of articles written by Mr. Jones:
  
  ```xml
  count(/journal/article[author/last="Jones"])
  ```

- Find all authors whose last name begins with Mc:
  
  ```xml
  /journal/article[starts-with(author/last,"Mc")]
  ```

In addition to the core XPath functions defined by the XPath standard, a number of extended functions are also supported with OSM. These extended functions provide additional functionality that is useful to create behaviors, but does not conform to the XPath standard.

---

**Note:** OSM supports XPath 1.0.

---

The XPath function library is divided into four groups, each of which is described in more detail, below:

1. **Node set functions** - for working with node-sets, either the implicit current node set or one passed as a parameter.
2. **String functions**: For working with strings and include type coercions.
3. **Boolean functions**: For working with Booleans, including type coercions.
4. **Number functions**: For working with numbers, including type coercions.

**Node Set Functions**

The following describes the Node Set functions.
Node Set Functions

number last()

Returns the index of the last item of the current node set.

Example - /journal/article[last()]

number position()

Returns the index of the current item in the current node set.

Example - /journal/article[position()<3]

number count(node-set)

Returns the number of items in the argument node set.

Example - count(/journal/article)

node-set id(object)

Returns the elements with the ID specified.

Example - id("article.1")/author/last

string local-name(node-set?)

Returns the non-namespace portion of the node name of either a node set passed as a parameter or the current node in the current node set.

Example - local-name(/wj:journals)
Example - /journal/*[local-name()= "article"]

string namespace-uri(node-set?)

Returns the namespace URI of the node name of either a node set passed as a parameter or the current node in the current node set.

Example - namespace-uri(/wj:journals)
Example - /journal/*:*[namespace-uri()="http://werken.com/werken-journal/"]

string name(node-set?)

Returns the complete textual node name of either a node set passed as a parameter or the current node in the current node set.

Example - name(/journal)
Example - [name()="soap:Envelope"]

node-set evaluate(string)

Returns the node set resulting from the Xpath expression defined by the provided argument. Allows XPath expressions to be dynamically created. The argument is converted to a string as if by a call to the string function.

Example - evaluate('/GetOrder.Response/*')
node-set match(node-set, string)

Returns the node set that matches a regular expression pattern.

Example - match('GetOrder.Response/*, 'blur[f]+le[0-9]')

node-set instance(string)

Returns the content of the named XML instance.

Note: This function is only available to XPath expressions within behaviors.

The argument is converted to a string as if by a call to the string function. This string, along with the user’s preferred language is matched against instance elements that are within scope of the containing document. If a match is located this function returns a node-set containing the content of the root element node (also called the document element node) of the referenced instance data. In all other cases, an exception is thrown and an error is displayed.

Example: For instance data corresponding to the following XML:

<instance name="order_form" lang="en" xsi:type="inlineInstanceType">
    <orderForm>
        <shipTo>
            <firstName>John</firstName>
        </shipTo>
    </orderForm>
</instance>

The following expression selects the firstName node (assuming the logged-in the user’s preferred language is English, or that English is the default system language).

Note: The instance function returns an element node, effectively replacing the left most location step from the path:

instance('order_form')/shipTo/firstName

String Functions

The following describes the String functions.

string string(object?)

Converts an object (possibly the current context node) to its string value.

Example - /journal/article/author[string() = 'Jones']

string concat(string, string, string*)

Concatenates two or more strings together.

Example - concat(author/salutation, ', ', author/last)

string starts-with(string, string)

Determines if the first argument starts with the second argument string.
string contains(string, string)

Determines if the first argument contains the second argument string.

Example - /journal/article[contains(title, 'XPath')]

string substring-before(string, string)

Retrieves the substring of the first argument that occurs before the first occurrence of the second argument string.

Example - substring-before(/journal/article[1]/date, '/')

string substring-after(string, string)

Retrieves the substring of the first argument that occurs after the first occurrence of the second argument string.

Example - substring-after(/journal/article[1]/date, '/')

string substring(string, number, number?)

Retrieves the substring of the first argument starting at the index of the second number argument, for the length of the optional third argument.

Example - substring('Jones', 3)

number string-length(string?)

Determines the length of a string, or the current context node coerced to a string.

Example - /journal/article[string-length(author/last) > 9]

string normalize-space(string?)

Retrieves the string argument or context node with all space normalized, trimming white space from the ends and compressing consecutive white space elements to a single space.

Example - normalize-space(/journal/article[1]/content)

string translate(string, string, string)

Retrieves the first string argument augmented so that characters that occur in the second string argument are replaced by the character from the third argument in the same position.

Example - translate('bob', 'abc', 'ZXY')='XoX'

string lower-case(string?)

Retrieves the string argument or context node with all characters converted to lower case.

Example - lower-case('Foo')='foo'
String Functions

String upper-case(string?)
Retrieves the string argument or context node with all characters converted to upper case.

Example - upper-case('Foo')='FOO'

String ends-with(string, string)
Determines if the first argument ends with the second argument string.

Example - /journal/article[ends-with(title, 'Advanced')]

Boolean Functions
The following describes the Boolean functions.

Boolean boolean(object)
Converts the argument to a Boolean value.

Example - boolean(/journal/article/author/last.[.='Jones'])

Boolean not(boolean)
Negates the boolean value.

Example - not(/journal/article/author/last.[.='Jones'])

Boolean true()
The Boolean value is true.

Boolean false()
The Boolean value is false.

Boolean boolean-from-string(string)
Returns true if the required parameter string is true, 1, or Yes. In all other conditions, false is returned.

Example - boolean-from-string(../pay_entire_amount)

Object if(boolean,object,object)
Evaluates the first parameter as a Boolean, returning the second parameter when true, otherwise the third parameter.

Example - if(/journal/article/author/last.[.='Jones'], 'Match found', 'No match')
number number(object?)

Converts the argument or context node to a number value.

Example - /journal[number(year)=2003]

number sum(node-set)

Sums the node set value.

Example - sum(/journal/article/author/age)

number floor(number)

Returns the largest integer that is not greater than the number argument.

Example - floor(100.5)=100

number ceiling(number)

Returns the smallest integer that is not less than the number argument.

Example - ceiling(100.5)=101

number round(number)

Rounds the number argument.

Example - ceiling(100.3)=100

number avg(node-set)

Returns the arithmetic average of the string-values conversion of each node in the argument node-set to a number. The sum is computed with sum(), and divided with div() by the value computed with count(). If the parameter is an empty node-set, the return value is NaN.

Example - avg(/journal/article/author/age)

number min(node-set)

Returns the minimum value that results from converting the string-values of each node in argument node-set to a number. The minimum is determined with the < operator. If the parameter is an empty node-set, or if any of the nodes evaluate to NaN, the return value is NaN.

Example - min(/journal/article/author/age)

number max(node-set)

Returns the maximum value that results from converting the string-values of each node in argument node-set to a number. The maximum is determined with the < operator. If the parameter is an empty node-set, or if any of the nodes evaluate to NaN, the return value is NaN.

Example - max(/journal/article/author/age)
number count-not-empty(node-set)

Returns the number of non-empty nodes in argument node-set. A node is considered non-empty if it is convertible into a string with a greater-than zero length.

Example - count-not-empty(/journal/article/author/middle)

XPath 1.0 Reference

Complete XPath reference information is available at the World Wide Web Consortium (W3C) web site:

http://www.w3.org/TR/xpath

The abbreviated XPath highlights below are reproduced with permission from Mulberry Technologies, Inc. (http://www.mulberrytech.com).

Location Paths [XPath §2]

Optional ‘/’, zero or more location steps, separated by ‘/’

Location Paths [XPath §2.1]

Axis specifier, node test, zero or more predicates

Axis Specifiers [XPath §2.2]

ancestor:: ancestor-or-self:: attribute:: child::
descendant:: descendant-or-self:: following:: following-sibling::
namespace:: parent:: preceding:: preceding-sibling::
self::

Node Tests [XPath §2.]

name node() prefix:name text() *
* comment() prefix:* processing-instruction() processing-instruction(literal)

Abbreviated Syntax for Location Paths

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>(nothing)</td>
<td>child::</td>
</tr>
<tr>
<td>@</td>
<td>attribute::</td>
</tr>
<tr>
<td>//</td>
<td>/descendant-or-self::node()/</td>
</tr>
<tr>
<td>.</td>
<td>self::node()</td>
</tr>
<tr>
<td>..</td>
<td>parent::node()</td>
</tr>
<tr>
<td>/</td>
<td>Node tree root</td>
</tr>
</tbody>
</table>

Predicate [XPath §2.4]

[expr]
Variable Reference [XPath §3.7]

\$qname

XPath

http://www.w3.org/TR/xpath

XPath Operators

Parentheses may be used for grouping.

Node-sets [XPath §3.3]

\( [\text{expr}] / // \)

Booleans [XPath §3.4]

\(<=, <, >=, >=, !=, and, or\)

Numbers [XPath §3.5]

\(-\text{expr}, *, \text{div}, \text{mod}, +, -\)

Node Types [XPath §5]

<table>
<thead>
<tr>
<th>Root</th>
<th>Processing Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element</td>
<td>Comment</td>
</tr>
<tr>
<td>Attribute</td>
<td>Test</td>
</tr>
<tr>
<td>NameSpace</td>
<td></td>
</tr>
</tbody>
</table>

Object Types [§11.1, XPath §1]

<table>
<thead>
<tr>
<th>Type</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>True or False</td>
</tr>
<tr>
<td>number</td>
<td>Floating-point number</td>
</tr>
<tr>
<td>string</td>
<td>UCS characters</td>
</tr>
<tr>
<td>node-set</td>
<td>Set of nodes selected by a path</td>
</tr>
</tbody>
</table>

XPath Core Function Library

XPath core functions:

Node Set Functions [XPath §4.1]

\begin{itemize}
  \item number \texttt{last()}
  \item number \texttt{position()}
  \item number \texttt{count(node-set)}
  \item node-set \texttt{id(object)}
  \item string \texttt{local-name(node-set)}
  \item string \texttt{namespace-uri(node-set)}
  \item string \texttt{name(node-set)}
\end{itemize}
String Functions [XPath §4.2]

string string(object?)
string concat(string, string, string*)
string starts-with(string, string)
string contains(string, string)
string substring-before(string, string)
string substring-after(string, string)
string substring(string, number, number?)
number string-length(string)
string normalize-space(string?)
string translate(string, string, string)

Boolean Functions [XPath §4.3]

boolean boolean(object)
boolean not(boolean)
boolean true()
boolean false()

Number Functions [XPath §4.4]

number number(object?)
number sum(node-set)
number floor(number)
number ceiling(number)
number round(number)

OSM Behavior XPath Functions

OSM Behavior XPath Functions:

Node Set Functions

string matrix-concat(node-set, node-set, node-set?)
node-set evaluate(string)
node-set instance(string?) [Declarative Rules Only]
node-set match(node-set?, string)

String Functions

string lower-case(string?)
string upper-case(string?)
string ends-with(string, string)

Boolean Functions

boolean boolean-from-string(string)
object if(boolean, object, object)

Number Functions

number avg(node-set)
number min(node-set)
number max(node-set)
number count-not-empty(node-set)
This chapter describes an example of a competitive local exchange carrier (CLEC) using Oracle Communications Order and Service Management (OSM) to provision a plain-old-telephone-service (POTS) customer using unbundled local loop.

About the Process Example

Figure 12–1 depicts a sample OSM process. An incumbent local exchange carrier (ILEC) customer wants to replace an ILEC service with a CLEC POTS service. These are typically residential or small office or home office (SOHO) customer with a small number of phone lines. (A larger business customer would generally have equipment installed at the customer premise to support a large number of connections and use higher bandwidth connections from the CLEC central office (CO) to the customer site). The CLEC CO contains the switch to be used for connection to the customer. This example assumes that the CLEC also has existing presence and capacity at the ILEC CO that currently serves the customer.

The CLEC uses the ILEC local loop that connects the ILEC CO to the customer. The CLEC uses Digital Loop Carrier (DLC) technology that uses a higher bandwidth connection (typically 1 or more DS-1 connections, DS-3, and so on) from the CLEC switch to the DLC equipment (at the ILEC CO) that provides individual DS-0 ports for connection to the copper loop to the customer.

Figure 12–1 Overview of OSM Process
Figure 12–3 through Figure 12–5 show a graphical representation of the OSM process flow described in this document. The symbols used are described in Figure 12–2. The tasks are arranged to identify which group is performing the task, Customer Care or OSM. The task numbers correspond to the section that describes the task.

**Figure 12–2  OSM Process Flow Legend**

- **OSM Task**
- **Connector**
- **OSM Rule Task (decision)**
- **Terminator**
- **External System (outside OSM)**

**Process Flow Diagram**

**Figure 12–3  OSM Flow**

[Diagram of OSM process flow with task numbers and descriptions]
Figure 12–4  OSM Flow (continued)

Customer Care

Figure 12–5  OSM Flow (continued)

Customer Care
About the Tasks Defined in the Process Example

The following sections describe the tasks that make up the process of providing a new CLEC customer with a POTS service.

Before starting the process, information must be gathered from the customer. This is typically done by a Customer Service Representative (CSR) or Sales person. The information is generally entered into an order entry system by the CSR.

The Order Entry System also assigns an order number for the customer order that must be used to reference the activities taking place in the Integrated Suite components. The customer may order more than one service that requires separate processing. These must be identified and service processes started as necessary to implement the required services. In this case assume that the only service requested is POTS, and that only a single line is contained within each order initiated in the suite (OSM).

In this example, each task that is described, contains the following sections:

- **Task Activities**: The activities taking place in the task.
- **Input Data**: Data that is provided to the task. This is identified as data groups, as defined in Appendix B, “Automation: Start to Finish”. Data groups include any subgroups. Input data is read-only.
- **Output Data**: Data that is generated in the task and provided to subsequent tasks.
- **Completion**: The options for task completion and the tasks that follow.

The data format is described as A/N (Alpha/Numeric), or N (Numeric) with the number of characters, fixed format types such as Date or Phone, or Lookup, where a lookup table is used to set the value.

The source of data is the task that generated the data, or, within the generating task description.

**Initiate Order**

To begin activities, an order must be entered into OSM or there must be an interface between the Order Entry System (OES) and OSM that allows the information to be automatically passed into OSM.

**Activities**

Enter the data into OSM and initiate the process. Otherwise, the data is entered automatically into OSM, from OES, and the process is initiated.

**Input Data**

None.

**Output Data**

<table>
<thead>
<tr>
<th>OSM Business Name</th>
<th>Description</th>
<th>Format</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSM Order ID</td>
<td>OSM assigned order number</td>
<td>9 A/N</td>
<td>OSM</td>
</tr>
<tr>
<td>Customer Order Number</td>
<td>The order number assigned by the order entry system</td>
<td>12 A/N</td>
<td>OES</td>
</tr>
</tbody>
</table>
### Table 12–2  Output Data: Customer Details Group

<table>
<thead>
<tr>
<th>OSM Business Name</th>
<th>Description</th>
<th>Format</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer Name</td>
<td>Customer name</td>
<td>30 A/N</td>
<td>OES - Customer</td>
</tr>
<tr>
<td>Customer ID Number</td>
<td>Assigned by OE system</td>
<td>12 A/N</td>
<td>OES</td>
</tr>
</tbody>
</table>

### Table 12–3  Output Data: Service Address Group

<table>
<thead>
<tr>
<th>OSM Business Name</th>
<th>Description</th>
<th>Format</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Street address</td>
<td>Street address number</td>
<td>40 A/N</td>
<td>OES - Customer</td>
</tr>
<tr>
<td>City</td>
<td>City name</td>
<td>25 A/N</td>
<td>OES - Customer</td>
</tr>
<tr>
<td>State or Prov Code</td>
<td>State abbreviation</td>
<td>2 A</td>
<td>OES - Customer</td>
</tr>
<tr>
<td>Zip or Postal Code</td>
<td>Postal code</td>
<td>10 A/N</td>
<td>OES - Customer</td>
</tr>
<tr>
<td>Additional Description</td>
<td>Additional address description</td>
<td>50 A/N</td>
<td>OES - Customer</td>
</tr>
</tbody>
</table>

### Table 12–4  Output Data: Contact Details Group

<table>
<thead>
<tr>
<th>OSM Business Name</th>
<th>Description</th>
<th>Format</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contact Name</td>
<td>A contact in case a problem is encountered</td>
<td>30 A/N</td>
<td>OES - Customer</td>
</tr>
<tr>
<td>Contact Phone Number</td>
<td>Contact phone number</td>
<td>Phone number</td>
<td>OES - Customer</td>
</tr>
<tr>
<td>Contact Cell Phone</td>
<td>Contact cell phone</td>
<td>Phone number</td>
<td>OES - Customer</td>
</tr>
</tbody>
</table>

### Table 12–5  Output Data: Service Details Group

<table>
<thead>
<tr>
<th>OSM Business Name</th>
<th>Description</th>
<th>Format</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Existing phone number</td>
<td>Existing phone number</td>
<td>Phone number</td>
<td>OES - Customer</td>
</tr>
<tr>
<td>Port Existing Number?</td>
<td>Port existing number</td>
<td>Lookup[Y - Yes N - No]</td>
<td>OES - Customer</td>
</tr>
<tr>
<td>New Number</td>
<td>If required, blank otherwise</td>
<td>Phone number</td>
<td>OES - Number Assignment System</td>
</tr>
<tr>
<td>Current Access Provider</td>
<td>Current provider of local phone service [name or code number]</td>
<td>20 A/N</td>
<td>OES - Customer</td>
</tr>
<tr>
<td>Current Account number</td>
<td>Customer account number with ILEC</td>
<td>C20 A/N</td>
<td>OES - Customer</td>
</tr>
<tr>
<td>ILEC CO CLLI</td>
<td>Location code of ILEC CO where local loop terminates (blank if not known)</td>
<td>11 A/N</td>
<td>OES</td>
</tr>
<tr>
<td>LD Provider</td>
<td>Long distance provider (to set PIC in switch) [code number]</td>
<td>6 A/N</td>
<td>OES - Customer</td>
</tr>
<tr>
<td>Service Type</td>
<td>Business/Residential POTS [service code]</td>
<td>Lookup[B - Business R - Residential]</td>
<td>OES - Customer</td>
</tr>
</tbody>
</table>
### Table 12–5  (Cont.) Output Data: Service Details Group

<table>
<thead>
<tr>
<th>OSM Business Name</th>
<th>Description</th>
<th>Format</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Expedite/Normal</td>
<td>Lookup[E - Expedite N - Normal]</td>
<td>OES - Customer</td>
</tr>
<tr>
<td>Due Date</td>
<td>Required in service date</td>
<td>Date</td>
<td>OES - Customer</td>
</tr>
</tbody>
</table>

### Table 12–6  Output Data: Service Feature Group

<table>
<thead>
<tr>
<th>OSM Business Name</th>
<th>Description</th>
<th>Format</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service Feature 1.12</td>
<td>Features required (as detailed below)</td>
<td>12 entries, each consisting of: Feature identifier: LookupFeature parameter 1:12 A/NFeature Parameter 2:12 A/N</td>
<td>OES - Customer</td>
</tr>
</tbody>
</table>

### Table 12–7  Feature List

<table>
<thead>
<tr>
<th>Feature Name</th>
<th>Description</th>
<th>Lookup Code</th>
<th>Feature Parameter 1</th>
<th>Feature Parameter 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Three way calling</td>
<td>Allows a customer to add a third party to and existing conversation without operator assistance</td>
<td>3WC</td>
<td>Not Used</td>
<td>Not Used</td>
</tr>
<tr>
<td>Call Waiting</td>
<td>Informs a busy station user by a burst of tone that another call is waiting. The busy station may answer the new call by placing the original call on hold and flashing the switch hook or going on hook.</td>
<td>CW</td>
<td>Not Used</td>
<td>Not Used</td>
</tr>
<tr>
<td>Call Forwarding -Variable</td>
<td>Calls to the line are forwarded to another network address when this feature is subscribed to and the customer has activated it. An access code and the forwarding number is dialed to activate the feature.</td>
<td>CFV</td>
<td>Call Forward DN The number to which the call is forwarded</td>
<td>Call Forward Rings Interval (Number of seconds of ringing before call is forwarded)</td>
</tr>
<tr>
<td>Call Forwarding -Busy</td>
<td>When the subscriber line is busy, the call is forwarded to a designated line.</td>
<td>CFB</td>
<td>Call Forward DN The number to which the call is forwarded</td>
<td>Call Forward Rings Interval (Number of seconds of ringing before call is forwarded)</td>
</tr>
<tr>
<td>Call Forwarding -Don't Answer</td>
<td>When the subscriber does not answer, calls are forwarded to designated number.</td>
<td>CFD</td>
<td>Call Forward DN The number to which the call is forwarded</td>
<td>Call Forward Rings Interval (Number of seconds of ringing before call is forwarded)</td>
</tr>
<tr>
<td>Remote activation of call forwarding</td>
<td>Allows a subscriber with CFV to activate/deactivate the feature from a line other than their own.</td>
<td>CFR</td>
<td>Remote Activation Pin (Pin to access this feature from a remote location)</td>
<td>Not Used</td>
</tr>
</tbody>
</table>
About the Tasks Defined in the Process Example

Additional data may be gathered in the OES concerning the customer that is not required in the suite unless OSM is performing some order entry functions. This information may include information such as customer billing address, additional contacts and phone numbers, pricing information, and so on.

Completion
The "ILEC CO CLLI Identified? (Rule Task)" task runs.

**ILEC CO CLLI Identified? (Rule Task)**

The following sections describe ILEC CO CLLI Identifiers.

**Task Activities**
Check the ILEC CO CLLI data element. If it has an entry, complete the task with the status True. Otherwise, complete the task with the status False.

**Input Data**
Service Details Group

**Output Data**
None

**Completion**
True: The "Assign Port" task runs.
False: The "Identify ILEC CO" task runs.

**Identify ILEC CO**
The ILEC CO, which is connected to the customer, is generally determined using the NPA-NXX portion of the customer's telephone number or by the Postal or Zip code. The purpose of this task is to identify the ILEC CO if it was not provided in the order information.

### Table 12–7 (Cont.) Feature List

<table>
<thead>
<tr>
<th>Feature Name</th>
<th>Description</th>
<th>Lookup Code</th>
<th>Feature Parameter 1</th>
<th>Feature Parameter 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer Changeable Speed Calling - 1 Digit</td>
<td>Allows the customer to maintain the list, short</td>
<td>SC1</td>
<td>List Size</td>
<td>Not Used</td>
</tr>
<tr>
<td>Customer Changeable Speed Calling - 2 Digit</td>
<td>Allows the customer to maintain the list, long</td>
<td>SC2</td>
<td>List Size</td>
<td>Not Used</td>
</tr>
<tr>
<td>Caller ID - Number</td>
<td>Shows calling numbers on display.</td>
<td>CNU</td>
<td>Not Used</td>
<td>Not Used</td>
</tr>
<tr>
<td>Caller ID - Name</td>
<td>Gives the Caller Id feature with Name</td>
<td>CNA</td>
<td>Not Used</td>
<td>Not Used</td>
</tr>
<tr>
<td>Caller ID - Blocking (Call Blocking)</td>
<td>Blocks outgoing call information</td>
<td>CBL</td>
<td>Not Used</td>
<td>Not Used</td>
</tr>
</tbody>
</table>

Additional data may be gathered in the OES concerning the customer that is not required in the suite unless OSM is performing some order entry functions. This information may include information such as customer billing address, additional contacts and phone numbers, pricing information, and so on.
Task Activities
1. Perform a lookup into a database that matches phone numbers or postal codes to the ILEC CO (or switch) CLLI.
2. Enter the CLLI into the data element ILEC CO CLLI.
3. Complete the task.

Input Data
- Customer Order Number
- Customer Details Group
- Contact Details Group
- Service Details Group

Output Data
Service Details Group

Table 12–8  Output Data for Identify ILEC CO Task

<table>
<thead>
<tr>
<th>OSM Business Name</th>
<th>Description</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>ILEC CO CLLI</td>
<td>Location code of ILEC CO where local loop terminates</td>
<td>11 A/N</td>
</tr>
</tbody>
</table>

Completion
The "ILEC CO CLLI Identified? (Rule Task)" task runs.

Assign Port
This task interacts with UIM to perform the necessary functions to locate and assign a port for the required service.

Task Activities
1. Find the location with a common language location identifier (CLLI) attribute equal to the value of ILEC CO CLLI.
2. If it is not found, this indicates no presence at CO, or perhaps an error in the CLLI. Set the data element No Presence to True.
3. Complete the task with a Problem status. Otherwise, based on the Service Type, determine the required signal type, type of port, and the equipment to use for the search.
4. Find the port of the required type in the appropriate equipment whose Status attribute is Available.
5. If no port is found, set the data element No Capacity to True.
6. Complete the task with a Problem status. Otherwise, set the port Status attribute to Assigned.
7. Store the Customer ID Number in the port attribute Customer ID.
8. Store the Customer Order Number in the port attribute Order ID.
9. Store the OSM Order Number in the port attribute OSM Order ID.
10. Store the New, if entered, or existing phone number in the port attribute Phone
   No.

11. Determine the termination point of the port on the IDF (site, rack, panel, port
   identifiers) and store it in the data element Termination Point.

12. Obtain the LEN/OE for the port and store in the data element Port OE or LEN.

13. Obtain the identifier (CLLI) for the switch controlling the port and store in the
   data element Switch ID.

14. Obtain the Location Routing Number (LRN) for the switch controlling the port
    and store in the data element Switch LRN.

15. Complete the task with the status Success.

**Input Data**
- OSM Order ID
- Customer Order ID
- Customer Details Group

**Output Data**
Assignment Details

**Table 12–9  Output Data for Assign Port Task**

<table>
<thead>
<tr>
<th>OSM Business Name</th>
<th>Description</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Presence</td>
<td>No presence at ILEC CO</td>
<td>Lookup[1 - True 0 - False]</td>
</tr>
<tr>
<td>No Capacity</td>
<td>No ports available</td>
<td>Lookup[1 - True 0 - False]</td>
</tr>
<tr>
<td>Port OE or LEN</td>
<td>ID of port assigned</td>
<td>40 A/N</td>
</tr>
<tr>
<td>Termination Point</td>
<td>Identifier of IDF port where port is terminated</td>
<td>40 A/N</td>
</tr>
<tr>
<td>Switch ID</td>
<td>Identifier of the switch controlling the port</td>
<td>11 A/N</td>
</tr>
<tr>
<td>Switch LRN</td>
<td>LRN of the switch for LNP</td>
<td>Phone number</td>
</tr>
</tbody>
</table>

**Completion**

**Success**: The "Send LSR to ILEC" task runs.

**Problem**: The "Notify Customer of Delay" task runs.

**Notify Customer of Delay**
This task deals with notifying a customer of an order delay.

**Task Activities**
1. Determine the cause of the problem and the impact on customer delivery.
2. Alert Network Engineering to the problem.
3. Contact the customer for a decision to continue with the order.
4. If the customer cannot wait, complete the task with the status Cancel.
5. If the customer can wait, enter the **Updated Due Date**.
6. Complete the task with the status **Wait**.

**Input Data**
- Customer Order Number
- Customer Details Group
- Contact Details Group
- Service Details Group
- Assignment Details Group

**Output Data**
Updated value
Service Details Group

### Table 12–10  Output Data for Notify Customer of Delay Task

<table>
<thead>
<tr>
<th>OSM Business Name</th>
<th>Description</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Updated Due Date</td>
<td>Revision Required in service date</td>
<td>Date</td>
</tr>
</tbody>
</table>

**Completion**

**Cancel**: The "**Cancel Order**" task runs.

**Wait**: The "**Wait for Engineering**" task runs.

**Cancel Order**

This task deals with cancelling orders.

**Task Activities**

1. Enter the data element **Reason for Cancellation**.
2. Enter the data element **Cancellation Date**.
3. Complete the task.

**Input Data**
- Customer Order Number
- Customer Details Group
- Contact Details Group
- Service Details Group
- Assignment Details Group

**Output Data**
Cancellation Details Group

### Table 12–11  Output Data for Cancel Order Task

<table>
<thead>
<tr>
<th>OSM Business Name</th>
<th>Description</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cancellation Date</td>
<td>Cancellation date</td>
<td>Date</td>
</tr>
</tbody>
</table>
About the Tasks Defined in the Process Example

Wait for Engineering
The following sections describe the parameters which require engineering intervention.

Task Activities
1. Wait for an update from engineering indicating that the presence/capacity is available.
2. When available, enter the date into the data element Engineering Complete Date.
3. Complete task.

Input Data
- Customer Order Number
- Customer Details Group
- Contact Details Group
- Service Details Group
- Assignment Details Group

Output Data
Assignment Details Group

<table>
<thead>
<tr>
<th>OSM Business Name</th>
<th>Description</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Engineering Complete Date</td>
<td>Engineering completion date</td>
<td>Date</td>
</tr>
</tbody>
</table>

Completion
The "Assign Port" task runs.

Send LSR to ILEC
To acquire the use of the local loop, local service request (LSR) forms must be sent to the ILEC. There are a number of possible forms that may be used depending on the type of request. For an unbundled loop request, three forms must be sent to the ILEC, the LSR, EU, and LS as described below: Some of the data required to fill the forms is available from OSM and some data is provided by the person filling the forms.

Task Activities
1. Enter data into three Local Service Request (LSR) forms:
   - Loop Service Request Form (LSR)
About the Tasks Defined in the Process Example

- End User Information Form (EU)
- Loop Service Form (LS)

2. Transmit the forms to the ILEC.
3. Complete the task.

**Input Data**
- Customer Order Number
- Customer Details Group
- Contact Details Group
- Service Details Group
- Assignment Details Group

**Output Data**
LSR Details Group

**Table 12–13 Output Data for Send LSR to ILEC Task**

<table>
<thead>
<tr>
<th>OSM Business Name</th>
<th>Description</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSR Sent Date</td>
<td>Date that LSR was sent to ILEC</td>
<td>Date</td>
</tr>
</tbody>
</table>

**Completion**
The "Receive LSR Confirmation" task runs.

**Receive LSR Confirmation**

A response to the LSR request and the LSR Confirmation (LSC), is returned by the ILEC.

**Task Activities**
1. Wait for LSR response from ILEC.
2. Receive the response and handle any problems that can be resolved. For example, schedules.
3. If there are any problems, complete with status of the LSR with Problem. Otherwise, enter the data from the LSC.
4. Complete with the status of Success.

**Input Data**
- Customer Order Number
- Customer Details Group
- Contact Details Group
- Service Details Group
- Assignment Details Group
- LSR Details Group
Output Data
LSR Details Group

Table 12–14  Output Data for Receive LSR Confirmation Task

<table>
<thead>
<tr>
<th>OSM Business Name</th>
<th>Description</th>
<th>Format</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSR Number</td>
<td>Local Service Request Number (provider)</td>
<td>18 A/N</td>
<td>LSC Field # 6</td>
</tr>
<tr>
<td>Provider Order Number</td>
<td>Order Number (provider)</td>
<td>20 A/N</td>
<td>LSC Field # 7</td>
</tr>
<tr>
<td>Confirmation Date</td>
<td>Confirmation Date Sent</td>
<td>Date</td>
<td>LSC Field # 11</td>
</tr>
<tr>
<td>Provider Rep</td>
<td>Provider Contact Representative</td>
<td>15 A/N</td>
<td>LSC Field # 12</td>
</tr>
<tr>
<td>Provider Rep. Phone</td>
<td>Telephone Number (of provider’s rep)</td>
<td>Phone number</td>
<td>LSC Field # 15</td>
</tr>
<tr>
<td>Provider Due Date</td>
<td>Due Date</td>
<td>Date</td>
<td>LSC Field # 19</td>
</tr>
<tr>
<td>Provider Estimated Date</td>
<td>Estimated Due Date</td>
<td>Date</td>
<td>LSC Field # 28</td>
</tr>
<tr>
<td>Cut over time</td>
<td>Frame Due Time (cutover time)</td>
<td>10 A/N</td>
<td>LSC Field # 18</td>
</tr>
<tr>
<td>Provider Bill Date</td>
<td>Effective Bill Date</td>
<td>Date</td>
<td>LSC Field # 20</td>
</tr>
<tr>
<td>LSR Problem Identifier</td>
<td>Reason Code</td>
<td>2 A/N</td>
<td>LSC Field # 26</td>
</tr>
<tr>
<td>LSR Problem Detail</td>
<td>Reason Code Jeopardy Detail</td>
<td>60 A/N</td>
<td>LSC Field # 27</td>
</tr>
<tr>
<td>Provider Designer</td>
<td>Network Service Provider (NSP) Design/Engineering Contact</td>
<td>15 A/N</td>
<td>LSC Field # 31</td>
</tr>
<tr>
<td>Provider Designer Phone</td>
<td>Telephone Number</td>
<td>Phone number</td>
<td>LSC Field # 32</td>
</tr>
<tr>
<td>Provider Circuit ID (ECCKT)</td>
<td>Exchange Company Circuit ID</td>
<td>41 A/N</td>
<td>LSC Field # 48</td>
</tr>
<tr>
<td>Provider Loop Order Number</td>
<td>Loop Order Number</td>
<td>20 A/N</td>
<td>LSC Field # 56</td>
</tr>
</tbody>
</table>

Completion
LSR Problem: The "Handle LSR Problem" task runs.
Success: The "Send LNP Porting Request" task runs.

Handle LSR Problem

The following sections describe handling LSR problems.

Task Activities
1. Handle any problems with the LSR.
2. Update any LSR data fields as required.
3. Complete the task.

Input Data
- Customer Order Number
- Customer Details Group
- Contact Details Group
- Service Details Group
- Assignment Details Group
- LSR Details Group
About the Tasks Defined in the Process Example

**Output Data**
Updated fields as required.

**Completion**
The "LNP Required? (Rule Task)" runs.

**LNP Required? (Rule Task)**
The following sections describe the rule task that determines whether LNP is required.

**Task Activities**
If the Port Existing Number is Y, complete the task with a True status. Otherwise, complete the task with a False status.

**Input Data**

**Table 12–15 Output Data for LNP Required Task**

<table>
<thead>
<tr>
<th>OSM Business Name</th>
<th>Description</th>
<th>Format</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port existing number?</td>
<td>Port existing phone number</td>
<td>Lookup[Y - Yes N - No]</td>
<td>Initiate order</td>
</tr>
</tbody>
</table>

**Output Data**
None

**Completion**
True: The "Send LNP Porting Request" task runs.
False: The "Activate Port and Services on CLEC Switch" runs.

**Send LNP Porting Request**
If the LNP is available and required, a porting request is generated and sent to the NPAC (Number Portability Accounting Center).

**Task Activities**
1. Generate a porting request to the Number Portability Accounting Center (NPAC) using the data provided and manually generated data.
2. Transmit the request to NPAC.
3. Complete the task.

**Input Data**
- Customer Order Number
- Customer Details Group
- Contact Details Group
- Service Details Group

**Output Data**
LNP Details Group
About the Tasks Defined in the Process Example

Table 12–16  Output Data for Send LNP Porting Request Task

<table>
<thead>
<tr>
<th>OSM Business Name</th>
<th>Description</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>LNP Sent Date</td>
<td>Date that LNP request was sent to NPAC</td>
<td>Date</td>
</tr>
</tbody>
</table>

Completion
The "Receive LNP Confirmation" task runs.

Receive LNP Confirmation
A response to the LNP request is returned by the NPAC.

Task Activities
1. Wait for the LNP response from NPAC.
2. Receive the response.
3. If there are any problems, complete with a status of **LNP Problem**. Otherwise, enter the data from the NPAC response.
4. Complete with a status of **Success**.

Input Data
- Customer Order Number
- Customer Details Group
- Contact Details Group
- Service Details Group
- Assignment Details Group
- LNP Details Group

Output Data
LNP Details Group

Table 12–17  Output Data for Receive LNP Confirmation Task

<table>
<thead>
<tr>
<th>OSM Business Name</th>
<th>Description</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>LNP Reference Number</td>
<td>LNP request reference number from NPAC</td>
<td>18 A/N</td>
</tr>
<tr>
<td>LNP Confirmation Date</td>
<td>Confirmation date</td>
<td>Received</td>
</tr>
</tbody>
</table>

Completion
**LNP Problem**: The "Handle LNP Problem" task runs.

**Success**: The "Activate Port and Services on CLEC Switch" task runs.

Handle LNP Problem
The following sections describe handling LNP problems.

Task Activities
1. Handle any problems with the LNP request.
2. Update any data fields as required.
3. Complete the task.

**Input Data**
Same as previous for reference purposes.
- Customer Order Number
- Customer Details Group
- Contact Details Group
- Service Details Group
- LNP Details Group

**Output Data**
Updated fields as required.

**Completion**
The "Activate Port and Services on CLEC Switch" task runs.

**Activate Port and Services on CLEC Switch**
The required switch activations are performed.

**Task Activities**
1. Get required activation information.
2. Activate switch.
3. If activation is successful, store the date in the data element **Activation Date**.
4. Complete the task with a status **Success**.
5. If activation unsuccessful, store the error information in the data element **Activation Error Info**.
6. Complete the task with a status **Activation Failed**.

**Input Data**
- Customer Order Number
- Service Details Group
- Assignment Details Group

**Output Data**
Activation Details Group

<table>
<thead>
<tr>
<th><strong>Table 12–18</strong></th>
<th><strong>Output Data for Activate Port and Services on CLEC Switch Task</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>OSM Business Name</td>
<td>Description</td>
</tr>
<tr>
<td>Activation Date</td>
<td>Date activation was completed</td>
</tr>
<tr>
<td>Activation Error Info</td>
<td>Activation error returned from ASAP or IP Service Activator</td>
</tr>
</tbody>
</table>
Completion
Success: The "Check for Loop Availability" task runs.
Activation Failed: The "Handle Activation Problem" task runs.

Handle Activation Problem

The following sections describe handling activation problems.

Task Activities
1. Work with the activation system to determine a reason for the activation failure.
2. Correct the problem.
3. Complete the task.

Input Data
- Customer Order Number
- Service Details Group
- Assignment Details Group
- Activation Details Group

Output Data
Updated data fields as required.

Completion
The "Activate Port and Services on CLEC Switch" task runs.

Check for Loop Availability

When the loop is due to be switched over to the ILEC, it may be done within a time span of many hours. In order to minimize interruption to the customer and ensure that the changeover is successfully completed, the line is checked for connection beginning at the earliest time the changeover is expected.

Task Activities
1. Setup the test equipment as necessary to check for loop completion.
2. If it is not detected in a specific time following the expected date, contact the ILEC to investigate the problem.
3. When the loop is available, complete the task.

Input Data
- Customer Order Number
- Customer Details Group
- Contact Details Group
- Service Details Group
- Assignment Details Group
- LSR Details Group
About the Tasks Defined in the Process Example

- Activation Details Group

Output Data
Completion Details Group

Table 12–19  Output Data for Check for Loop Availability Task

<table>
<thead>
<tr>
<th>OSM Business Name</th>
<th>Description</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop Available Date</td>
<td>Date the loop was found to be available</td>
<td>Date</td>
</tr>
</tbody>
</table>

Completion
The "Update UIM" task runs.

Update UIM
The following sections describe updating UIM.

Task Activities
1. Access the assigned port attributes, update the attribute Status to Active, and store the ECCKT in the port attribute Provider Circuit ID.
2. Complete the task.

Input Data
- Customer Order Number
- Service Details Group
- Assignment Details Group
- LSR Details Group

Output Data
None

Completion
The "Order Complete" task runs.

Send Activation Notice to NPAC
The following sections describe sending notices to NPAC.

Task Activities
1. Send a message to the NPAC confirming that the LNP change was activated.
2. Store the date of sending the activation notice in the data element LNP Activation Date.
3. Complete task.

Input Data
- Customer Order Number
- Customer Details Group
About the Tasks Defined in the Process Example

- Service Details Group
- LNP Details Group

Output Data
LNP Details Group

Table 12–20  Output Data for Send Activation Notice to NPAC Task

<table>
<thead>
<tr>
<th>OSM Business Name</th>
<th>Description</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>LNP Activation Date</td>
<td>Date LNP activation sent</td>
<td>Date</td>
</tr>
</tbody>
</table>

Completion
The "Order Complete" task runs.

Update Other Systems
The following sections describe updating other systems.

Task Activities
1. Notify external systems:
   - Telephone directory producer (may be the ILEC).
   - Directory assistance provider (may be the ILEC).
   - 911?
2. Notify other interested CLEC systems that the service is active. These may include:
   - Billing - start billing customer
   - Accounting - receives charges for local loop
3. When activities are finished, complete the task with a status Success.

Input Data
- Customer Order Number
- Customer Details Group
- Contact Details Group
- Service Details Group
- Assignment Details Group
- LSR Details Group
- LNP Details Group
- Activation Details Group
- Completion Details Group

Output Data
Completion Details Group
About Port Configuration Data

Table 12–21  Output Data for Update Other Systems Task

<table>
<thead>
<tr>
<th>OSM Business Name</th>
<th>Description</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Update Completion Date</td>
<td>System update completion date</td>
<td>Date</td>
</tr>
</tbody>
</table>

Completion
The "Order Complete" task runs.

Order Complete
Notify Customer Service that the order is complete.

Task Activities
1. Review data as desired.
2. Enter the date into the data element Order Completion Date.

Input Data
- Customer Order Number
- Customer Details Group
- Contact Details Group
- Service Details Group
- Assignment Details Group
- LSR Details Group
- LNP Details Group
- Activation Details Group
- Completion Details Group

Output Data
Completion Details Group

Table 12–22  Output Data for Order Complete Task

<table>
<thead>
<tr>
<th>OSM Business Name</th>
<th>Description</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order Completion Date</td>
<td>Order completion date</td>
<td>Date</td>
</tr>
</tbody>
</table>

Completion
Complete: The order is now completed.

About Port Configuration Data
The following sections describe the port configuration data.

Physical Port
- Status
  - Available
- Assigned
- Reserved
- Active

- Customer ID
- OSM Order ID
- Order ID
- Phone Number
- Provider Circuit ID

**Functional Switch**

- LRN
- Switch ID

**About OSM Data Summary, Grouping**

The following is a summary of all OSM data elements organized by groups as used in OSM. If a group name is indented, the group is a subgroup of the previous group. The Process defined in OSM is Phone Line Activation and the order template is POTS/Order Entry. Views are defined to match the groups.

**Table 12–23 OSM Data Summary**

<table>
<thead>
<tr>
<th>OSM Business Name</th>
<th>Description</th>
<th>Format</th>
<th>Source Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSM Order ID</td>
<td>OSM assigned order number</td>
<td>9 A/N</td>
<td>OSM</td>
</tr>
<tr>
<td>Customer Order Number</td>
<td>The order number assigned by the order entry system</td>
<td>12 A/N</td>
<td>Initiate Order</td>
</tr>
</tbody>
</table>

**Table 12–24 OSM Data Summary: Customer Details Group**

<table>
<thead>
<tr>
<th>OSM Business Name</th>
<th>Description</th>
<th>Format</th>
<th>Source Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer Name</td>
<td>Customer name</td>
<td>30 A/N</td>
<td>Initiate Order</td>
</tr>
<tr>
<td>Customer ID Number</td>
<td>Assigned by OE system</td>
<td>12 A/N</td>
<td>Initiate Order</td>
</tr>
</tbody>
</table>

**Table 12–25 OSM Data Summary: Service Address Group**

<table>
<thead>
<tr>
<th>OSM Business Name</th>
<th>Description</th>
<th>Format</th>
<th>Source Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service Address:Street address</td>
<td>Service Address:Street address</td>
<td>40 A/N</td>
<td>Initiate Order</td>
</tr>
<tr>
<td>City</td>
<td>City</td>
<td>25 A/N</td>
<td>Initiate Order</td>
</tr>
<tr>
<td>State or Prov Code</td>
<td>State or Prov Code</td>
<td>2 A</td>
<td>Initiate Order</td>
</tr>
<tr>
<td>Zip or Postal Code</td>
<td>Zip or Postal Code</td>
<td>10 A/N</td>
<td>Initiate Order</td>
</tr>
<tr>
<td>Additional Description</td>
<td>Additional Description</td>
<td>50 A/N</td>
<td>Initiate Order</td>
</tr>
</tbody>
</table>
**Table 12–26 OSM Data Summary: Contact Details Group**

<table>
<thead>
<tr>
<th>OSM Business Name</th>
<th>Description</th>
<th>Format</th>
<th>Source Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contact Name</td>
<td>A contact in case a problem is encountered</td>
<td>30 A/N</td>
<td>Initiate Order</td>
</tr>
<tr>
<td>Contact Phone Number</td>
<td>Contact Phone Number</td>
<td>Phone</td>
<td>Initiate Order</td>
</tr>
<tr>
<td>Contact Cell Phone</td>
<td>Contact Cell Phone</td>
<td>Phone</td>
<td>Initiate Order</td>
</tr>
</tbody>
</table>

**Table 12–27 OSM Data Summary: Service Details Group**

<table>
<thead>
<tr>
<th>OSM Business Name</th>
<th>Description</th>
<th>Format</th>
<th>Source Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Existing phone number</td>
<td>Existing phone number</td>
<td>Phone</td>
<td>Initiate Order</td>
</tr>
<tr>
<td>Port existing number?</td>
<td>Port existing number?</td>
<td>Lookup[Y - Yes N - No]</td>
<td>Initiate Order</td>
</tr>
<tr>
<td>New Number</td>
<td>New Number</td>
<td>12 A/N</td>
<td>Initiate Order</td>
</tr>
<tr>
<td>Current Access Provider</td>
<td>Current provider of local phone service [name or code number]</td>
<td>20 A/N</td>
<td>Initiate Order</td>
</tr>
<tr>
<td>Current Account number</td>
<td>Customer account number with ILEC</td>
<td>20 A/N</td>
<td>Initiate Order</td>
</tr>
<tr>
<td>ILEC CO CLLI</td>
<td>Location code of ILEC CO where local loop terminates</td>
<td>11 A/N</td>
<td>Initiate Order/Identify ILEC CO</td>
</tr>
<tr>
<td>LD Provider</td>
<td>Long distance provider (to set PIC in switch) [code number]</td>
<td>6 A/N</td>
<td>Initiate Order</td>
</tr>
<tr>
<td>Service Type</td>
<td>Business/Residential POTS [service code]</td>
<td>Lookup[B - Business R - Residential]</td>
<td>Initiate Order</td>
</tr>
<tr>
<td>Priority</td>
<td>Expedite/Normal</td>
<td>Lookup[E - Expedite N - Normal]</td>
<td>Initiate Order</td>
</tr>
<tr>
<td>Due date</td>
<td>Required in service date</td>
<td>Date</td>
<td>Initiate Order</td>
</tr>
<tr>
<td>Updated Due Date</td>
<td>Revised Required in service date</td>
<td>Date</td>
<td>Notify Customer of Delay</td>
</tr>
</tbody>
</table>

**Table 12–28 OSM Data Summary: Service Features Group**

<table>
<thead>
<tr>
<th>OSM Business Name</th>
<th>Description</th>
<th>Format</th>
<th>Source Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service Feature 1.12</td>
<td>Features required</td>
<td>Up to 12 entries, each consisting of: Feature identifier: Lookup [see definition below]Feature parameter 1:12 A/NFeature Parameter 2:12 A/N</td>
<td>Initiate Order</td>
</tr>
</tbody>
</table>

**Table 12–29 OSM Data Summary: Assignment Details Group**

<table>
<thead>
<tr>
<th>OSM Business Name</th>
<th>Description</th>
<th>Format</th>
<th>Source Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Presence</td>
<td>No presence at ILEC CO</td>
<td>Lookup [0 - False 1 - True]</td>
<td>Assign Port</td>
</tr>
</tbody>
</table>
About OSM Data Summary, Grouping

Table 12–29 (Cont.) OSM Data Summary: Assignment Details Group

<table>
<thead>
<tr>
<th>OSM Business Name</th>
<th>Description</th>
<th>Format</th>
<th>Source Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Capacity</td>
<td>Insufficient port capacity</td>
<td>Lookup [0 - False 1 - True]</td>
<td>Assign Port</td>
</tr>
<tr>
<td>Engineering Complete Date</td>
<td>Engineering Complete Date</td>
<td>Date</td>
<td>Wait for Engineering</td>
</tr>
<tr>
<td>Port OE or LEN</td>
<td>ID of port assigned</td>
<td>40 A/N</td>
<td>Assign Port</td>
</tr>
<tr>
<td>Termination Point</td>
<td>Identifier of IDF port where port is terminated</td>
<td>40 A/N</td>
<td>Assign Port</td>
</tr>
<tr>
<td>Switch ID</td>
<td>Identifier of the switch controlling the port</td>
<td>11 A/N</td>
<td>Assign Port</td>
</tr>
<tr>
<td>Switch LRN</td>
<td>LRN of the switch for LNP</td>
<td>Phone</td>
<td>Assign Port</td>
</tr>
</tbody>
</table>

Table 12–30 OSM Data Summary: LSR Details Group

<table>
<thead>
<tr>
<th>OSM Business Name</th>
<th>Description</th>
<th>Format</th>
<th>Source Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSR Sent Date</td>
<td>Date that LSR was sent to ILEC</td>
<td>Date</td>
<td>Send LSR</td>
</tr>
<tr>
<td>LSR Number</td>
<td>Local Service Request Number (provider)</td>
<td>18 A/N</td>
<td>Receive LSR Confirmation</td>
</tr>
<tr>
<td>Provider Order Number</td>
<td>Order Number (provider)</td>
<td>20 A/N</td>
<td>Receive LSR Confirmation</td>
</tr>
<tr>
<td>Confirmation Date</td>
<td>Confirmation Date Sent</td>
<td>Date</td>
<td>Receive LSR Confirmation</td>
</tr>
<tr>
<td>Provider Rep</td>
<td>Provider Contact Representative</td>
<td>15 A/N</td>
<td>Receive LSR Confirmation</td>
</tr>
<tr>
<td>Provider Rep. Phone</td>
<td>Telephone Number (of provider’s rep)</td>
<td>Phone</td>
<td>Receive LSR Confirmation</td>
</tr>
<tr>
<td>Provider Due Date</td>
<td>Due Date</td>
<td>Date</td>
<td>Receive LSR Confirmation</td>
</tr>
<tr>
<td>Provider Estimated Date</td>
<td>Estimated Due Date</td>
<td>Date</td>
<td>Receive LSR Confirmation</td>
</tr>
<tr>
<td>Cutover time</td>
<td>Frame Due Time (cutover time)</td>
<td>10 A/N</td>
<td>Receive LSR Confirmation</td>
</tr>
<tr>
<td>Provider Bill Date</td>
<td>Effective Bill Date</td>
<td>Date</td>
<td>Receive LSR Confirmation</td>
</tr>
<tr>
<td>LSR Problem Identifier</td>
<td>Reason Code</td>
<td>2 A/N</td>
<td>Receive LSR Confirmation</td>
</tr>
<tr>
<td>LSR Problem Detail</td>
<td>Reason Code Jeopardy Detail</td>
<td>60 A/N</td>
<td>Receive LSR Confirmation</td>
</tr>
<tr>
<td>Provider Designer</td>
<td>Network Service Provider (NSP) Design/Engineering Contact</td>
<td>15 A/N</td>
<td>Receive LSR Confirmation</td>
</tr>
<tr>
<td>Provider Designer Phone</td>
<td>Telephone Number</td>
<td>Phone</td>
<td>Receive LSR Confirmation</td>
</tr>
<tr>
<td>Provider Circuit ID (ECCKT)</td>
<td>Exchange Company Circuit ID</td>
<td>41 A/N</td>
<td>Receive LSR Confirmation</td>
</tr>
<tr>
<td>Provider Loop Order Number</td>
<td>Loop Order Number</td>
<td>20 A/N</td>
<td>Receive LSR Confirmation</td>
</tr>
</tbody>
</table>
Table 12–31  OSM Data Summary: LNP Details Group

<table>
<thead>
<tr>
<th>OSM Business Name</th>
<th>Description</th>
<th>Format</th>
<th>Source Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>LNP Sent Date</td>
<td>Date that LNP request was sent to NPAC</td>
<td>Date</td>
<td>Send LNP</td>
</tr>
<tr>
<td>LNP Reference Number</td>
<td>LNP request reference number from NPAC</td>
<td>18 A/N</td>
<td>Receive LNP Confirmation</td>
</tr>
<tr>
<td>LNP Confirmation Date</td>
<td>Confirmation Received date</td>
<td>Date</td>
<td>Receive LNP Confirmation</td>
</tr>
</tbody>
</table>

Table 12–32  OSM Data Summary: Activation Details Group

<table>
<thead>
<tr>
<th>OSM Business Name</th>
<th>Description</th>
<th>Format</th>
<th>Source Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activation Date</td>
<td>Date activation was completed</td>
<td>Date</td>
<td>Activate Port.</td>
</tr>
<tr>
<td>Activation error info</td>
<td>Activation error returned from ASAP or IP Service Activator</td>
<td>60 A/N</td>
<td>Activate Port.</td>
</tr>
</tbody>
</table>

Table 12–33  OSM Data Summary: Completion Details Group

<table>
<thead>
<tr>
<th>OSM Business Name</th>
<th>Description</th>
<th>Format</th>
<th>Source Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop Available Date</td>
<td>Date loop was found to be available</td>
<td>Date</td>
<td>Check for Loop Availability</td>
</tr>
<tr>
<td>LNP Activation Date</td>
<td>LNP activation sent date</td>
<td>Date</td>
<td>Send Activation Notice to NPAC</td>
</tr>
<tr>
<td>System Update Completion Date</td>
<td>System Update Completion Date</td>
<td>Date</td>
<td>Update Other Systems</td>
</tr>
<tr>
<td>Order Completion Date</td>
<td>Order Completion Date</td>
<td>Date</td>
<td>Order Complete</td>
</tr>
</tbody>
</table>

Table 12–34  OSM Data Summary: Cancellation Details Group

<table>
<thead>
<tr>
<th>OSM Business Name</th>
<th>Description</th>
<th>Format</th>
<th>Source Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cancellation Date</td>
<td>Cancellation Date</td>
<td>Date</td>
<td>Cancel Order</td>
</tr>
<tr>
<td>Reason for Cancellation</td>
<td>Reason for Cancellation</td>
<td>160 A/N</td>
<td>Cancel Order</td>
</tr>
</tbody>
</table>

Feature Identifier Lookup Definition

Table 12–35 lists the feature identifier lookup definitions.

Table 12–35  Feature Identifier Lookup Definition

<table>
<thead>
<tr>
<th>Code</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>3WC</td>
<td>Three way calling</td>
</tr>
<tr>
<td>CW</td>
<td>Call Waiting</td>
</tr>
<tr>
<td>CFV</td>
<td>Call Forward Variable</td>
</tr>
<tr>
<td>CFB</td>
<td>Call Forward Busy</td>
</tr>
<tr>
<td>CFD</td>
<td>Call Forward Don’t Answer</td>
</tr>
<tr>
<td>Code</td>
<td>Definition</td>
</tr>
<tr>
<td>------</td>
<td>--------------------</td>
</tr>
<tr>
<td>CFR</td>
<td>Call Forward Remote</td>
</tr>
<tr>
<td>SC1</td>
<td>Speed Call 1 Digit</td>
</tr>
<tr>
<td>SC2</td>
<td>Speed Call 2 Digit</td>
</tr>
<tr>
<td>CNU</td>
<td>Caller ID Number</td>
</tr>
<tr>
<td>CNA</td>
<td>Caller ID Name</td>
</tr>
<tr>
<td>CBL</td>
<td>Caller ID Block</td>
</tr>
</tbody>
</table>
This appendix provides examples of the generated \textit{automationMap.xml} file for Oracle Communications Order and Service Management (OSM).

\textbf{Note:} This appendix assumes that you have read "Using Automation".

After you have defined the automated task or automated notification, and defined the automation for it, a successful build of the project automatically generates the \textit{automationMap.xml} file. The file is placed in the \texttt{cartridgeName/cartridgeBuild/automation} directory, which is only visible from the Java perspective.

This file is a direct result of the automation definition, as shown in the following examples. The field names, and the data defaulted or entered for the field, on the various tabs of the Properties window directly relate the XML elements and attributes, and their data values, defined in the \textit{automationMap.xml} file.

### AutomationMap.xml Examples for Automated Tasks

This section provides various examples of generated \textit{automationMap.xml} files. The examples include predefined and custom automations defined for automated tasks. In the XML, an automated task is defined by the \texttt{<taskAutomator>} element.

#### XSLTSender Internal Event Receiver

This example reflects an automated task with an automation defined as XSLTSender, and as an internal event receiver. Specifics of the automation definition include:

- Automated Task name: MyTask
- Automation name: MyXSLTSenderIntAutomation
- XSLT file name: C:\myWorkingDirectory\myXslt.xslt

\textbf{Example A–1} \textit{XSLTSender Internal Event Receiver}

\begin{verbatim}
<taskAutomator>
  <pluginJndiName>MyTask.MyTask.MyXsltSenderIntAutomation</pluginJndiName>
  <ejbName>MyTask.MyTask.MyXsltSenderIntAutomation</ejbName>
  <className>com.mslv.oms.automation.plugin.XSLTSender</className>
  <runAs>automation</runAs>
  <cartridgeNamespace>samplecart</cartridgeNamespace>
  <cartridgeVersion>1.0.0</cartridgeVersion>
</taskAutomator>
\end{verbatim}
Notes Common to All Examples

- `<pluginJndiName>` and `<ejbName>` are based on the **EJB Name** field, located on the Properties view **Details** tab.
- `<className>` is based on the **Action** field selection, located on the Add Automation window.
- `<runAs>` is based on the **Run As** field, located on the Properties view **Details** tab.

Notes on Example

- `<receive>` type is based on the **External Event Receiver** check box, located on the Add Automation window. Because this example defines an internal event receiver, the elements are based on information defined on the Properties view **Internal Event Receiver** tab. `<mnemonic>` is based on the task name.

- `<implement>` type is based on the automation plug-in you are implementing. Because this example implements XSLTSender, the `<to>` and `<sendNullMessage>` elements are generated. These elements are not present when the implementation is for an automator.
  - The `<to>` elements are based on information defined on the Properties view **Router** tab, **To** sub-tab. This tab is present only when the automation is XSLTSender or XQuerySender.
  - The `<script>` elements are based on information defined on the Properties view **XSLT** tab. This tab is present only when the automation is XSLTSender or XSLTAutomator.

XSLTSender External Event Receiver

This example reflects an automated task with an automation defined as XSLTSender, and as an external event receiver. Specifics of the automation definition include:

- Automated Task name: MyTask
- Automation name: MyXSLTSenderExtAutomation
- XSLT file name: C:\myWorkingDirectory\myXslt.xslt
Example A-2  XSLTSender External Event Receiver

```xml
<taskAutomator>
  <pluginJndiName>MyTask.MyXsltSenderExtAutomation</pluginJndiName>
  <ejbName>MyTask.MyXsltSenderExtAutomation</ejbName>
  <className>com.mslv.cms.automation.plugin.XSLTSender</className>
  <runAs>automation</runAs>
  <cartridgeNamespace>samplecart</cartridgeNamespace>
  <cartridgeVersion>1.0.0</cartridgeVersion>
  <receive xsi:type="am:ExternalReceiver">
    <jmsSource>
      <from>
        <jndiName>MyTask.MyXsltSenderExtAutomation.jndiName</jndiName>
        <destinationType>javax.jms.Queue</destinationType>
      </from>
    </jmsSource>
    <correlation xsi:type="MessagePropertyCorrelation">
      <property>JMSCorrelationID</property>
    </correlation>
  </receive>
  <implement xsi:type="am:XsltSender">
    <to>
      <jndiName>MyTask.MyXsltSenderExtAutomation.JNDIName</jndiName>
      <destinationType>javax.jms.Queue</destinationType>
    </to>
    <am:sendNullMessage>true</am:sendNullMessage>
    <am:script>
      <am:file>C:\myWorkingDirectory\myXslt.xslt</am:file>
      <am:cache>
        <am:maxSize>50</am:maxSize>
        <am:timeout>15000</am:timeout>
      </am:cache>
    </am:script>
  </implement>
</taskAutomator>
```

Notes on Example

- `<receive>` type is based on the External Event Receiver check box, located on the Add Automation window. Because this example defines an external event receiver, the elements are based on information defined on the Properties view External Event Receiver tab.

- `<implement>` type is based on the automation plug-in you are implementing. Because this example implements XSLTSender, the `<to>` and `<sendNullMessage>` elements are generated. These elements are not present when the implementation is for an automator.
  - The `<to>` elements are based on information defined on the Properties view Router tab, To sub-tab. This tab is present only when the automation is XSLTSender or XQuerySender.
  - The `<script>` elements are based on information defined on the Properties view XSLT tab. This tab is present only when the automation is XSLTSender or XSLTAutomator.

XSLTAutomator Internal Event Receiver

This example reflects an automated task with an automation defined as XSLTAutomator, and as an internal event receiver. Specifics of the automation definition include:
■ Automated Task name: MyTask
■ Automation name: MyXSLTAutomatorIntAutomation
■ XSLT file name: C:\myWorkingDirectory\myXslt.xslt

Example A–3   XSLTAutomator Internal Event Receiver

```xml
<taskAutomator>
  <pluginJndiName>MyTask.MyTask.MyXsltAutomatorIntAutomation</pluginJndiName>
  <ejbName>MyTask.MyTask.MyXsltAutomatorIntAutomation</ejbName>
  <className>com.mslv.oms.automation.plugin.XSLTReceiver</className>
  <runAs>automation</runAs>
  <cartridgeNamespace>samplecart</cartridgeNamespace>
  <cartridgeVersion>1.0.0</cartridgeVersion>
  <receive xsi:type="am:InternalReceiver">
    <mnemonic>MyTask</mnemonic>
    <executionModes>do</executionModes>
  </receive>
  <implement xsi:type="am:XsltAutomator">
    <am:script>
      <am:file>C:\myWorkingDirectory\myXslt.xslt</am:file>
      <am:cache>
        <am:maxSize>50</am:maxSize>
        <am:timeout>15000</am:timeout>
      </am:cache>
    </am:script>
    <am:updateOrder>true</am:updateOrder>
  </implement>
</taskAutomator>
```

Notes on Example

■ `<className>` is based on the Action field selection, located on the Add Automation window. (XSLTReciever is the name of the class that represents XSLTAutomator. The name presentation in Oracle Communications Design Studio was intentional to avoid confusion: XSLTAutomator and XSLTSender both receive data, but in addition, XSLTSender can send a message.)

■ `<receive>` type is based on the External Event Receiver check box, located on the Add Automation window. Because this example defines an internal event receiver, the elements are based on information defined on the Properties view Internal Event Receiver tab. (<mnemonic> is based on the task name.)

■ `<implement>` type is based on the automation plug-in you are implementing. Because this example implements XSLTAutomator, the `<to>` and `<sendNullMessage>` elements are not generated. These elements are present when the implementation is for a sender.
  - The `<script>` elements are based on information defined on the Properties view XSLT tab. This tab is present only when the automation is XSLTSender or XSLTAutomator.

XSLTAutomator External Event Receiver

This example reflects an automated task with an automation defined as XSLTAutomator, and as an external event receiver. Specifics of the automation definition include:

■ Automated Task name: MyTask
Automation name: MyXSLTAutomatorExtAutomation

XSLT file name: C:\myWorkingDirectory\MyXslt.xslt

Example A–4  XSLTAutomator External Event Receiver

<taskAutomator>
  <pluginJndiName>MyTask.MyTask.MyXsltAutomatorExtAutomation</pluginJndiName>
  <ejbName>MyTask.MyTask.MyXsltAutomatorExtAutomation</ejbName>
  <className>com.mslv.cms.automation.plugin.XSLTReceiver</className>
  <runAs>automation</runAs>
  <cartridgeNamespace>samplecart</cartridgeNamespace>
  <cartridgeVersion>1.0.0</cartridgeVersion>
  <receive xsi:type="am:ExternalReceiver">
    <jmsSource>
      <from>
        <jndiName>MyTask.MyXsltAutomatorExtAutomation.jndiName</jndiName>
        <destinationType>javax.jms.Queue</destinationType>
      </from>
    </jmsSource>
    <correlation xsi:type="MessagePropertyCorrelation">
      <property>JMSCorrelationID</property>
    </correlation>
  </receive>
  <implement xsi:type="am:XsltAutomator">
    <am:script>
      <am:file>C:\myWorkingDirectory\myXslt.xslt</am:file>
      <am:cache>
        <am:maxSize>50</am:maxSize>
        <am:timeout>15000</am:timeout>
      </am:cache>
    </am:script>
    <am:updateOrder>true</am:updateOrder>
  </implement>
</taskAutomator>

Notes on Example

■ <className> is based on the Action field selection, located on the Add Automation window. (XSLTReceiver is the name of the class that represents XSLTAutomator. The name presentation in Design Studio was intentional to avoid confusion: XSLTAutomator and XSLTSender both receive data, but in addition, XSLTSender can send a message.)

■ <receive> type is based on the External Event Receiver check box, located on the Add Automation window. Because this example defines an external event receiver, the elements are based on information defined on the Properties view External Event Receiver tab.

■ <implement> type is based on the automation plug-in you are implementing. Because this example implements XSLTAutomator, the <to> and <sendNullMessage> elements are not generated. These elements are present when the implementation is for a sender.
  - The <script> elements are based on information defined on the Properties view XSLT tab. This tab is present only when the automation is XSLTSender or XSLTAutomator.
Custom Automation Internal Event Receiver

This example reflects an automated task with an automation defined as a custom automation plug-in, and as an internal event receiver. Specifics of the automation definition include:

- Automated Task name: InfoRequestAT
- Automation name: MyAutomationOnTheTaskAutomationTab
- Java class name: InfoRequest

Example A–5  Custom Automation Internal Event Receiver

```xml
<taskAutomator>
  <pluginJndiName>
    InfoRequestAT.InfoRequestAT.MyAutomationOnTheTaskAutomationTab
  </pluginJndiName>
  <ejbName>
    InfoRequestAT.InfoRequestAT.MyAutomationOnTheTaskAutomationTab
  </ejbName>
  <className>InfoRequest</className>
  <runAs>automation</runAs>
  <cartridgeNamespace>samplecart</cartridgeNamespace>
  <cartridgeVersion>1.0.0</cartridgeVersion>
  <receive xsi:type="am:InternalReceiver">
    <mnemonic>InfoRequestAT</mnemonic>
    <executionModes>do</executionModes>
  </receive>
</taskAutomator>
```

Notes on Example

- `<receive>` type is based on the External Event Receiver check box, located on the Add Automation window. Because this example defines an internal event receiver, the elements are based on information defined on the Properties view Internal Event Receiver tab. ( `<mnemonic>` is based on the task name.)
- Because this is a custom automation, the `<implement>` element is not generated. You are required to define this element in the XML Template field, located on the Custom Automation Plugin window.

Custom Automation External Event Receiver

This example reflects an automated task with an automation defined as a custom automation plug-in, and as an external event receiver. Specifics of the automation definition include:

- Automated Task name: InfoResponseAT
- Automation name: MyAutomationOnTheTaskAutomationTab
- Java class name: InfoResponse

Example A–6  Custom Automation External Event Receiver

```xml
<taskAutomator>
  <pluginJndiName>
    InfoResponseAT.InfoResponseAT.MyAutomationOnTheAutomationTab
  </pluginJndiName>
  <ejbName>
    InfoResponseAT.InfoResponseAT.MyAutomationOnTheAutomationTab
  </ejbName>
  <className>InfoResponse</className>
  <runAs>automation</runAs>
  <cartridgeNamespace>samplecart</cartridgeNamespace>
  <cartridgeVersion>1.0.0</cartridgeVersion>
  <receive xsi:type="am:InternalReceiver">
    <mnemonic>InfoResponseAT</mnemonic>
    <executionModes>do</executionModes>
  </receive>
</taskAutomator>
```
<ejbName>
<className>InfoResponse</className>
<runAs>automation</runAs>
<cartridgeNamespace>samplecart</cartridgeNamespace>
<cartridgeVersion>1.0.0</cartridgeVersion>
<receive xsi:type="am:ExternalReceiver">
  <jmsSource>
    <from>
      <jndiName>
        InfoResponseAT.MyAutomationOnTheAutomationTab.jndiName
      </jndiName>
      <destinationType>javax.jms.Queue</destinationType>
    </from>
  </jmsSource>
  <correlation xsi:type="MessagePropertyCorrelation">
    <property>JMSCorrelationID</property>
  </correlation>
</receive>
</taskAutomator>

Notes on Example

- `<receive>` type is based on the External Event Receiver check box, located on the Add Automation window. Because this example defines an external event receiver, the elements are based on information defined on the Properties view External Event Receiver tab.
- Because this is a custom automation, the `<implement>` element is not generated. You are required to define this element in the XML Template field, located on the Custom Automation Plugin window.

AutomationMap.xml Examples for Automated Notifications

This section provides various examples of generated automationMap.xml files. The examples include predefined and custom automations defined for automated notifications. In the XML, an automated notification is defined by the `<notificationAutomator>` element.

Automated notifications can only be defined as internal event receivers so there are no examples of external event receivers in this section. The examples are similar: The main differences are:

- The value of `<ejbName>` is based on the Design Studio entity for which the notification is defined. As a result, the value varies because different types of notifications are defined on different Design Studio entities.

Order Milestone-Based Notification

This example reflects an order milestone-based notification with an automation defined as a custom automation plug-in. Specifics of the automation definition include:

- Order name: OsmCartridgeOrder
- Automation name: MyAutomationOnTheOrderEventsTab
Example A–7  Order Milestone-Based

```
<notificationAutomator>
  <ejbName>
    OsmCartridgeOrder.OsmCartridgeOrder.MyAutomationOnTheOrderEventsTab
  </ejbName>
  <className>GenericNotif</className>
  <runAs>automation</runAs>
  <cartridgeNamespace>samplecart</cartridgeNamespace>
  <cartridgeVersion>1.0.0</cartridgeVersion>
  <receive xsi:type="am:InternalReceiver">
    <event xsi:type="OrderNotification">
      <orderSource>OsmCartridgeOrder</orderSource>
      <orderType>OsmCartridgeOrder</orderType>
      <milestone>completion</milestone>
    </event>
  </receive>
</notificationAutomator>
```

Task State-Based Notifications

This example reflects a task state-based notification with an automation defined as a custom automation plug-in. Specifics of the automation definition include:

- Automated Task name: InfoRequestAT
- Automation name: MyAutomationOnTheTaskEventsTab
- Java class name: GenericNotif

Example A–8  Task State-Based Notification / Task Event Tab

```
<notificationAutomator>
  <ejbName>
    InfoRequestAT.InfoRequestAT.MyAutomationOnTheTaskEventsTab
  </ejbName>
  <className>GenericNotif</className>
  <runAs>automation</runAs>
  <cartridgeNamespace>samplecart</cartridgeNamespace>
  <cartridgeVersion>1.0.0</cartridgeVersion>
  <receive xsi:type="am:InternalReceiver">
    <event xsi:type="TaskNotification">
      <mnemonic>InfoRequestAT</mnemonic>
      <state>completed</state>
    </event>
  </receive>
</notificationAutomator>
```

This example reflects another task state-based notification with an automation defined as a custom automation plug-in. Specifics of the automation definition include:

- Process name: OsmProcess
- Rule name: MyProcessTaskStateRule
- Automation name: MyAutomationOnTheProcessEventsTabForTaskState
- Java class name: GenericNotif
Example A–9  State-Based Notification / Process Event Tab

```
<notificationAutomator>
  <ejbName>
    MyAutomationOnTheProcessEventsTabForTaskState
  </ejbName>
  <className>GenericNotif</className>
  <runAs>automation</runAs>
  <cartridgeNamespace>samplecart</cartridgeNamespace>
  <cartridgeVersion>1.0.0</cartridgeVersion>
  <receive xsi:type="am:InternalReceiver">
    <event xsi:type="SystemNotification">
      <mnemonic>OsmProcess_MyProcessTaskStateRule</mnemonic>
    </event>
  </receive>
</notificationAutomator>
```

Task Status-Based Notification

This example reflects a task status-based notification with an automation defined as a custom automation plug-in. Specifics of the automation definition include:

- Process name: OsmProcess
- Rule name: MyProcessTaskStatusRule
- Automation name: MyAutomationOnTheProcessEventsTabForTaskStatus
- Java class name: GenericNotif

Example A–10  Task Status-Based Notification

```
<notificationAutomator>
  <ejbName>
    MyAutomationOnTheProcessEventsTabForTaskStatus
  </ejbName>
  <className>GenericNotif</className>
  <runAs>automation</runAs>
  <cartridgeNamespace>samplecart</cartridgeNamespace>
  <cartridgeVersion>1.0.0</cartridgeVersion>
  <receive xsi:type="am:InternalReceiver">
    <event xsi:type="SystemNotification">
      <mnemonic>OsmProcess_MyProcessTaskStatusRule</mnemonic>
    </event>
  </receive>
</notificationAutomator>
```

Order Data Changed Notification

This example reflects an order data changed notification with an automation defined as a custom automation plug-in. Specifics of the automation definition include:

- Order name: OsmCartridgeOrder
- Rule name: MyOrderNotificationRule
- Automation name: MyAutomationOnTheOrderNotificationTab
- Java class name: GenericNotif
**Example A-11  Order Data Changed Notification**

```
<notificationAutomator>
  <ejbName>
    OsmCartridgeOrder_MyOrderNotificationRule.OsmCartridgeOrder.
    MyAutomationOnTheOrderNotificationTab
  </ejbName>
  <className>GenericNotif</className>
  <runAs>automation</runAs>
  <cartridgeNamespace>samplecart</cartridgeNamespace>
  <cartridgeVersion>1.0.0</cartridgeVersion>
  <receive xsi:type="am:InternalReceiver">
    <event xsi:type="SystemNotification">
      <mnemonic>OsmCartridgeOrder_MyOrderNotificationRule</mnemonic>
    </event>
  </receive>
</notificationAutomator>
```

**Order Jeopardy Notification**

This example reflects an order jeopardy notification with an automation defined as a custom automation plug-in. Specifics of the automation definition include:

- **Order name:** OsmCartridgeOrder
- **Rule name:** MyOrderJepRule
- **Automation name:** MyAutomationOnTheOrderJepTab
- **Java class name:** GenericNotif

**Example A-12  Order Jeopardy Notification**

```
<notificationAutomator>
  <ejbName>
    OsmCartridgeOrder_MyOrderJepRule.OsmCartridgeOrder.MyAutomationOnTheOrderJepTab
  </ejbName>
  <className>GenericNotif</className>
  <runAs>automation</runAs>
  <cartridgeNamespace>samplecart</cartridgeNamespace>
  <cartridgeVersion>1.0.0</cartridgeVersion>
  <receive xsi:type="am:InternalReceiver">
    <event xsi:type="SystemNotification">
      <mnemonic>OsmCartridgeOrder_MyOrderJepRule</mnemonic>
    </event>
  </receive>
</notificationAutomator>
```

**Task Jeopardy Notification**

This example reflects a task jeopardy notification with an automation defined as a custom automation plug-in. Specifics of the automation definition include:

- **Automated Task name:** InfoRequestAT
- **Rule name:** MyTaskJepRule
- **Automation name:** MyAutomationOnTheTaskJepTab
- **Java class name:** GenericNotif
**Example A–13  Task Jeopardy Notification**

```xml
<notificationAutomator>
  <ejbName>
    InfoRequestAT_MyTaskJepRule.InfoRequestAT.MyAutomationOnTheTaskJepTab
  </ejbName>
  <className>GenericNotif</className>
  <runAs>automation</runAs>
  <cartridgeNamespace>samplecart</cartridgeNamespace>
  <cartridgeVersion>1.0.0</cartridgeVersion>
  <receive xsi:type="am:InternalReceiver">
    <event xsi:type="SystemNotification">
      <mnemonic>InfoRequestAT_MyTaskJepRule</mnemonic>
    </event>
  </receive>
</notificationAutomator>
```

**Generated Entity-Specific XML Files**

Design Studio also generates a separate XML file per Design Studio entity that defines an automation. Entity-specific XML files are also placed in the `cartridgeName/cartridgeBuild/automation` directory within the cartridge, which is only visible from the Java perspective; the directory path and files are not visible from the Studio Design perspective.

The entity-specific XML file names are dependent upon the Design Studio entity name that defines the automation and upon the type of event, resulting in the file name being `DesignStudioEntityName_EventType.xml`. For a task event, `EventType` is represented as "automation" in the file name, and for a notification event, `EventType` is represented as "notification_automation" in the file name.

For example, a Design Studio entity named MyAutomatedTask that defines a task event generates a file named `MyAutomatedTask_automation.xml`. Similarly, a Design Studio entity named MyOrder that defines an order notification event generates a file named `MyOrder_notification_automation.xml`.

If multiple task events are defined per Design Studio entity, one XML file that defines all the task events defined for the entity is generated. If multiple notification events are defined per Design Studio entity, one XML file that defines all the notification events defined for the entity is generated.

The `automationMap.xml` file is a cumulative collection of the contents of these entity-specific XML files, which can be helpful if you should need to determine which mapping is which.
This appendix provides steps to define a basic automation in Oracle Communications Design Studio, starting with a new cartridge and finishing with triggering the automation in Oracle Communications Order and Service Management (OSM) following deployment of the cartridge to the OSM server. The information is presented in the form of high-level steps. For specific instructions on how to perform each individual step, see the Design Studio Platform Help and the Design Studio Modeling OSM Processes Help.

**Assumptions**

The steps presented in this appendix assume that you have the following applications installed:

- Eclipse
- OSM Plug-ins
- OSM Administrator
- OSM

**Getting Started**

This section describes creating a new cartridge in Design Studio and compiling the project, prior to defining the automation. This section provides information that is used regardless of the automation example.

The creation of a new cartridge results in the creation of an Order entity of the same name within the cartridge. For example, if you create a new cartridge `cartridgeName`, an Order entity is created within the cartridge named `cartridgeNameOrder`. On the Order editor Details tab, three fields must be defined:

- Life-cycle Policy
- Default Process
- Creation Task

Until these fields are defined, the following errors are present for an order:

- Order Model Error: Creation task is not defined for order `cartridgeNameOrder`.
- Order Model Error: Default process is not defined for order `cartridgeNameOrder`.

---

**Note:** This appendix assumes that you have read "Using Automation".
Order Model Error: No roles have been granted for Creation permissions for this order.

Order Model Error: Order Life-cycle Policy is not defined in workspace.

Order Model Error: Order cartridgeNameOrder has empty Order Template.

Order Model Error: There is no Permission defined for order cartridgeNameOrder.

The following steps walk you through creating a cartridge to resolve these errors:

1. Create a new cartridge.
2. Build the project.
3. Open the Problems view.
   You may see the errors listed above.
4. Create a Role.
   You must create a role first because every entity you create requires that permissions be set, which is done by assigning a role.
5. On the Role editor, select the appropriate permissions.
6. Save the role.
7. Create an Order Life-cycle Policy.
8. On the Order Life-cycle Policy editor Permissions tab, set permissions for the order life-cycle policy by assigning a role.
9. Save the order life-cycle policy.
10. Create a Process.
11. On the Process editor Permissions tab, set permissions for the process by assigning a role.
12. Save the process.
13. Create a Manual Task.
15. Save the task.
16. On the Order editor Order Template tab, define an order template.
   This can be done by defining elements in the Data Dictionary and adding them to the order template, or by importing an order template. For purposes of understanding a basic automation, you may just want to define a few fields, such as name, address, city, state.
17. On the Order editor Details tab, set the following fields:
   - Life-cycle Policy
     Select the life-cycle policy you created in step 7.
   - Default Process
     Select the process you created in step 10.
   - Creation Task
     Select the task you created in step 13.
18. On the Order editor Permissions tab, set permissions for the order by assigning a role.

19. Save the order.

20. Build the project.

Upon successful build, the Problems view shows that all errors are resolved.

Defining an Automated Task

At this point, you have a cartridge that defines an order within a project that compiles. This example is using an automated task to trigger the automation, so this section describes the high-level steps for defining an automated task:

---

**Note:** An automation can also be triggered by a notification.
---

1. Define an automated task.

2. On the Automated Task editor Permissions tab, set permissions for the process by assigning a role.

3. Save the automated task.

Writing the Custom Automation Plug-in

This section describes the high-level steps for writing a basic custom automation plug-in:

1. In the cartridgeName/src directory, create a new Java source file.

2. Write the Java code where the class extends AbstractAutomator.

3. This automation is being triggered by an automated task, so the Java code can expect the TaskContext object as an input parameter. Code something simple, such as:
   - Cast the TaskContext object to a local variable
   - Print out a data value that is available through the TaskContext object. (This example had an order template that defined name, address, city, state, and zip, which would be available through the TaskContext object.)
   - Complete the task by calling the method on the TaskContext object.

4. Compile the source file.

The resultant compiled class file now resides in the cartridgeName/out directory.

Defining the Custom Automation Plug-in

This section describes the high-level steps for creating a Custom Automation Plug-in that is the Design Studio entity representation of your custom automation plug-in.

1. Create a Custom Automation Plug-in:
   a. Provide a name for your Custom Automation Plug-in.
   b. Select your custom automation plug-in class name.
c. In the XML Template field, define the implementation of your custom automation plug-in using the `<implement>` element. See Appendix A, “AutomationMap.xml File” for examples of defined `<implement>` elements.

d. In the cartridgeName/customAutomation directory, create a corresponding schema file that defines the rules for the XML you defined in the XML Template field.

2. Save the Custom Automation Plug-in.

Defining the Automation

This section describes the high-level steps for defining the automation, which maps your automated task to your Custom Automation Plug-in.

1. Open the automated task editor for the automated task you created.

2. On the Automated Task editor Automation tab, define the automation:
   a. In the Name field, enter a name for your automation.
   b. In the Automation Type list, select your custom automation plug-in.
   c. For this example, let the Event Type default to the choice of Internal Event Receiver.

3. Save the Automated Task.

Defining the Process

This section describes the high-level steps for defining the process, which must include your automated task in order for the task to be initiated and trigger your automation.

1. Open the Process Editor.

2. Add your automated task to the process.

   For the project to compile, and for your automation to run, your process must define a Start node, your automated task, an End node, and statuses between the three.

3. Save the process.

Building the Cartridge

After you have completed these steps, you must build the cartridge project. A successful build of the project results in the generation of the automationMap.xml file.

Packaging and Deploying the Cartridge

This section describes the high-level steps for deploying the cartridge to the OSM server, including what must be done prior to deployment. For more information, see the Design Studio Help topic about packaging and deploying OSM cartridges.

1. Create an Environment Design Studio entity, which defines the connection information for the server hosting the OSM environment to which you plan to deploy your cartridge.

2. Deploy the cartridge to your OSM environment.
Triggering the Automation in OSM

The final step is to trigger the automation from within OSM; this can only occur after the cartridge is successfully deployed to the OSM server.

1. Within OSM, create an order based on the order template that you defined in your cartridge.

2. Save the order.

   This results in the order starting to process. The order process you defined, which includes the automated task you defined, starts processing: First, the creation task runs.

3. Complete the creation task.

   (This example defined the creation task as a manual task, so you must manually complete the creation task.)

   When the creation task is completed, the next task defined in the process is created, which is your automated task. The creation of the automated task sets the task state to Received, which triggers your automation to run.
The following pages contain a quick reference for Oracle Communications Order and Service Management (OSM) behaviors which you can print and keep as a work aid.

For comprehensive information on behaviors, see OSM Concepts.

**OSM Behavior Type Overview**

Table C–1 provides an overview of the OSM behaviors.

<table>
<thead>
<tr>
<th>Behavior Type Name</th>
<th>Order</th>
<th>Synopsis</th>
<th>Default</th>
<th>Applies To</th>
<th>Parent/Child Inheritance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calculate Behavior</td>
<td>1st</td>
<td>Calculates the value of the data instance node.</td>
<td>None</td>
<td>All value nodes.</td>
<td>Does not inherit.</td>
</tr>
<tr>
<td>Style Behavior: Appearance Facet</td>
<td>2nd</td>
<td>Specifies the appearance of a data instance node:</td>
<td>Data type specific.</td>
<td>Boolean and Lookup type value nodes.</td>
<td>Does not inherit.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ DEFAULT: the default appearance should be used</td>
<td>For Boolean type fields: Compact</td>
<td>Nodes with Lookup behaviors that have only one displayed column.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ FULL: all choices should be rendered at all times.</td>
<td>For Lookup type fields: Minimal</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ COMPACT: a fixed number of choices should be rendered, with scrolling facilities as needed.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ MINIMAL: a minimum number of choices should be rendered with a facility to temporarily render additional choices.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Style Behavior: CSS Style Facet</td>
<td>2nd</td>
<td>Specifies the HTML CSS style attributes of the data instance node and label.</td>
<td>None</td>
<td>All value and group nodes.</td>
<td>Does not inherit.</td>
</tr>
<tr>
<td>Style Behavior: CSS Class Facet</td>
<td>2nd</td>
<td>Specifies the HTML CSS Class name of the data instance node and label.</td>
<td>None</td>
<td>All value and group nodes.</td>
<td>Does not inherit.</td>
</tr>
<tr>
<td>Style Behavior: Newline Facet</td>
<td>2nd</td>
<td>Specifies whether a line-break is inserted before the node causing it to be displayed at the start of a new line.</td>
<td>False</td>
<td>All value nodes.</td>
<td>Does not inherit.</td>
</tr>
<tr>
<td>Style Behavior: Secret Facet</td>
<td>2nd</td>
<td>Ensures unauthorized users are now allowed to view the contents of nodes containing sensitive information.</td>
<td>True</td>
<td>All value nodes except for modifiable (read/write) lookups and boolean values.</td>
<td>Does not inherit.</td>
</tr>
</tbody>
</table>
### Table C–1 (Cont.) Behavior Type Overview

<table>
<thead>
<tr>
<th>Behavior Type Name</th>
<th>Order</th>
<th>Synopsis</th>
<th>Default</th>
<th>Applies To</th>
<th>Parent/Child Inheritance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Style Behavior: Layout Facet</td>
<td>2nd</td>
<td>Specifies the organization of a group’s child nodes into tabbed pages.</td>
<td>None</td>
<td>All group nodes.</td>
<td>Does not inherit.</td>
</tr>
<tr>
<td>Style Behavior: Location Facet</td>
<td>2nd</td>
<td>Specifies the tabbed page that this group will be placed in.</td>
<td>None</td>
<td>All group nodes.</td>
<td>Does not inherit.</td>
</tr>
<tr>
<td>Information Behavior</td>
<td>3rd</td>
<td>Specifies the label, hint, and help information for the data instance node.</td>
<td>None</td>
<td>All value and group nodes.</td>
<td>Does not inherit.</td>
</tr>
<tr>
<td>Relevant Behavior</td>
<td>4th</td>
<td>Indicates whether the data instance node is currently relevant. Data instance nodes with this property evaluating to false are not displayed in the view. If this property is False, other behaviors for this node are not evaluated.</td>
<td>True</td>
<td>All value and group nodes.</td>
<td>If any ancestor node evaluates to false, this value is treated as false. Otherwise, the local value is used.</td>
</tr>
<tr>
<td>Lookup Behavior</td>
<td>5th</td>
<td>Specifies a set of dynamic generated choices for the data instance node.</td>
<td>Static lookup values (if any) specified in the OSM Model data dictionary.</td>
<td>All value nodes that are of type lookup, number, or text.</td>
<td>Does not inherit.</td>
</tr>
<tr>
<td>Constraint Behavior: Attachment Facet</td>
<td>6th</td>
<td>Specifies a condition that needs to be satisfied for the associated order attachment content to be considered valid. NOTE: This facet is only supported through programmatic behavior implementations.</td>
<td>True</td>
<td>Attachment nodes.</td>
<td>Does not inherit.</td>
</tr>
<tr>
<td>Read Only Behavior</td>
<td>7th</td>
<td>Describes whether the value is restricted from changing. This behavior overrides the static read-only value specified in the OSM Model View Node.</td>
<td>Default specified by the static read-only value on the OSM Model View Node.</td>
<td>All value and group nodes.</td>
<td>If any ancestor node evaluates to true, this value is treated as true. Otherwise, the local value is used.</td>
</tr>
<tr>
<td>Event Behavior</td>
<td>8th</td>
<td>Specifies an action to perform when a given event occurs.</td>
<td>None</td>
<td>Value nodes.</td>
<td>Does not inherit.</td>
</tr>
<tr>
<td>Constraint Behavior</td>
<td>N/A</td>
<td>Specifies a condition that needs to be satisfied for the associated data instance node to be considered valid. If the condition is not satisfied (evaluates to false), then messages are displayed to the user.</td>
<td>True</td>
<td>All value and group nodes.</td>
<td>Does not inherit.</td>
</tr>
<tr>
<td>Data Instance Behavior</td>
<td>N/A</td>
<td>Defines a container in which instances can be declared. It has no affect on the user interface display of the element for which the behavior is defined.</td>
<td>None</td>
<td>All elements and structures</td>
<td>Children. (Applies to element relationships within a structure. This is different than the inheritance of behaviors between the data dictionary, order, and task levels.)</td>
</tr>
</tbody>
</table>
Common Behavior Elements

This section describes the syntax for declaring common behavior elements.

Annotation Element

```xml
<annotation>
  <documentation lang="NCName">...</documentation>
</annotation>
```

Description Element

```xml
<description>string</description>
```

Instance Element

```xml
<instance name="NCName" lang="NCName" xsi:type="inlineInstanceType|externalInstanceType">
  For inlineInstanceType, any valid XML document is allowed up to 4000 characters in length.
  For externalInstanceType, adapter followed by parameter*, and cache*
</instance>
```

Adapter Element [externalInstanceType]

```xml
<adapter>com.mslv.oms.view.rule.adapter.ObjectelAdapter
  |com.mslv.oms.view.rule.adapter.OrderAdapter
  |com.mslv.oms.view.rule.adapter.XMLAttachmentAdapter
  |com.mslv.oms.view.rule.adapter.XMLFileAdapter
  |javaClassNameType</adapter>
```

Parameter Element [externalInstanceType]

```xml
<parameter name="string">string-expr</parameter>
```

Cache Element

```xml
<cache>
  <scope>NONE|NODE|SYSTEM</scope>
  <timeout>positiveInteger</timeout>
  <maxSize>positiveInteger</maxSize>
</cache>
```

Expression Element

```xml
<expression>boolean-expr</expression>
```

Declaring Behaviors in OSM XML Model

This section describes the syntax for declaring behaviors in OSM XML model.

Data Dictionary Level

```xml
<dataDictionary>
  element+
  <element name="nameType" xsi:type="booleanType
    |currencyType|dateType|dateTimeType|phoneType|groupType
    |textType|numericType|lookupType">
```
**Master Order Template Level**

```xml
<masterOrderTemplate>
  <dataNode+>
    <dataNode element="NCName">viewRule*, followed by dataNode*
    </dataNode>
  </dataNode>
</masterOrderTemplate>
```

**View Level**

```xml
&viewNode element="NCName"> editable?, minOccurs?, maxOccurs?, viewRule*, viewNode*
  <editable>boolean</editable>
  <minOccurs>unsignedInt</minOccurs>
  <maxOccurs>unsignedInt</maxOccurs>
</viewNode>
```

**Data Provider Overview**

Table C–2 provides an overview of the built-in and custom data providers. See "Using Data Providers to Retrieve Data" for details.

<table>
<thead>
<tr>
<th>Data Provider</th>
<th>Synopsis</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Custom</td>
<td>Uses data provided by a custom-defined Java class.</td>
<td>Implementation-defined</td>
</tr>
<tr>
<td>JDBC</td>
<td>Lets OSM query any JDBC database, then use the results within a behavior.</td>
<td>oms:dataSource, oms:sql, in:1 ... in:n?, out:1 ... out:n?</td>
</tr>
<tr>
<td>Objectel</td>
<td>Uses results of an Objectel Server Extension as an instance.</td>
<td>obj:extensionName, obj:jmsFactory?, obj:queue?, obj:allowErrorResponse?. Other parameters passed to Objectel</td>
</tr>
<tr>
<td>Order</td>
<td>Uses data from any OSM order as an external instance.</td>
<td>oms:OrderID, oms:View</td>
</tr>
<tr>
<td>Property File</td>
<td>Retrieves an external Java property file with a given name from the classpath.</td>
<td>oms:url</td>
</tr>
<tr>
<td>SOAP</td>
<td>Lets you open up OSM to web services, using the HTTP protocol.</td>
<td>soap.endpoint, soap.action?, soap.envelope, soap.body, soap.header?, oms:credentials.username?, oms:credentials.password?, oms:credentials.scope.host?, soap.allowErrorResponse</td>
</tr>
<tr>
<td>XML Attachment</td>
<td>Uses an XML attachment from any OSM order as an instance.</td>
<td>oms:OrderID, oms:FileName</td>
</tr>
<tr>
<td>XML File</td>
<td>Uses an XML file from any URL as an instance.</td>
<td>oms:url</td>
</tr>
<tr>
<td>XML Validation</td>
<td>Validates a provided XML instance document according to a user-defined schema. The document may be either a URL or an element. The schema may also be a URL or an element.</td>
<td>document, schema</td>
</tr>
</tbody>
</table>
Table C–3 provides an overview of the programmatic behavior implementation.

<table>
<thead>
<tr>
<th>Rule Type Name</th>
<th>Java Interface</th>
<th>Method Names</th>
<th>Parameter Types</th>
<th>Return Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Style Rule: Appearance Facet</td>
<td>com.mslv.oms.view.StyleRule</td>
<td>appearance_&lt;mnemonic&gt;</td>
<td>com.msvl.oms.view.rule.ViewRuleContext, org.w3c.dom.Node</td>
<td>int NOTE: Return value must be one of FULL_APPEARANCE, COMPACT_APPEARANCE, MINIMAL_APPEARANCE defined on the StyleRule interface</td>
</tr>
<tr>
<td>Rule Type Name</td>
<td>Java Interface</td>
<td>Method Names</td>
<td>Parameter Types</td>
<td>Return Types</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------------------------</td>
<td>-----------------------</td>
<td>---------------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Attachment Facet</td>
<td>com.mslv.oms.view.ConstraintRule</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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**
- **Task Compensation Strategy XQuery Expressions**

### 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
<?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: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>2d3237363033323437363930353735</corecom:EBMID>
    </corecom:EBMHeader>
    <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>
  </ProcessSalesOrderFulfillmentEBM>
</ws:CreateOrder>
```
<ProcessSalesOrderFulfillment/>
<corecom:Identification xmlns:corecom="http://xmlns.oracle.com/EnterpriseObjects/Core/Common/V2"/>
<corecom:BusinessComponentID schemeID="SALESORDER_ID" schemeAgencyID="COMMON">3433393931323333323339373135353138</corecom:BusinessComponentID>
<corecom:ID schemeID="SALESORDER_ID" schemeAgencyID="SEBL_01">ScenarioA2</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:Status>
<corecom:BusinessUnitReference xmlns:corecom="http://xmlns.oracle.com/EnterpriseObjects/Core/Common/V2"/>
<corecom:ID schemeID="ORGANIZATION_ID" schemeAgencyID="SEBL_01">0-R9NH</corecom:ID>
<corecom:BusinessUnitIdentification/>
</corecom:BusinessUnitReference>
<corecom:CustomerPartyAccountIdentification/>
<corecom:BusinessComponentID schemeID="CUSTOMERPARTY_ACCOUNTID" schemeAgencyID="COMMON">2d353537333130353233303536343833</corecom:BusinessComponentID>
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:

  ```java
  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:

  ```java
  declare namespace automator =
  "java:oracle.communications.ordermanagement.automation.plugin.ScriptSenderContextInvocation";
  ...
  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($outboundMessage, '_wls_mimehdrContent_Type', 'text/xml; charset=UTF-8'),
    ```
  - Use outboundMessage to set up the OSM Web Service URI JMS message property:
    ```
    outboundMessage:setStringProperty($outboundMessage, 'URI', '/oms/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:
    ```
    ```
  - 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:
    ```
    ```

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:

```xml
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.

```xml
declare namespace automator = "java:oracle.communications.ordermanagement.automation.plugin.ScriptSenderContextInvocation";
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($outboundMessage, '_wls_mimehdrContent_Type',
'text/xml; charset="utf-8"'),
outboundMessage:setStringProperty($outboundMessage, 'URI', '/osm/wsapi'),
outboundMessage:setStringProperty($outboundMessage, 'Ora_OSM_COM_OrderId',
'/oms:GetOrder.Response/oms:OrderID'),
outboundMessage:setJMSCorrelationID($outboundMessage, concat($order/oms:_
root/oms:messageXmlData/ebo:ProcessSalesOrderFulfillmentEBM/ebo:DataArea/ebo:Proce
ssSalesOrderFulfillment/corecom:Identification/corecom:ID/text(),'-COM')),
log:info($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.xs
d* 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-profi
le-1.0#PasswordText">passw0rd</wsse:Password>
  </wsse:UsernameToken>
  <wsse:Security>
  </wsse:Security>
</soapenv:Header>
  <soapenv:Body>
    <ord:CreateOrder

xmlns:ebo=http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/ProvisioningOrder/V1

"}>
  <ebo:DataArea>

  </ebo:DataArea>

  <corecom:Process
xmlns=http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/ProvisioningOrder/V1"
xmlns:corecom=http://xmlns.oracle.com/EnterpriseObjects/Core/Common/V2"
Functions"
xmlns:xref="http://www.oracle.com/XSL/Transform/java/oracle.tip.xref.xpath.XrExPa
thFunctions" xmlns:oms=com:metasolv:oms:xmlapi:1"
xmlns:provord="http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/ProvisioningOrde
r/V1*/

  <provord:ProcessProvisioningOrder
xmlns=http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/ProvisioningOrder/V1"
Functions"
xmlns:xref="http://www.oracle.com/XSL/Transform/java/oracle.tip.xref.xpath.XrExPa
thFunctions" xmlns:oms=com:metasolv:oms:xmlapi:1"
xmlns:provord="http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/ProvisioningOrde
r/V1*/


<corecom:SalesOrderReference>
    <corecom:SalesOrderIdentification>
    </corecom:SalesOrderIdentification>
</corecom:SalesOrderReference>

<provord:RequestedDeliveryDateTime>2010-07-16T08:24:38Z</provord:RequestedDeliveryDateTime>

<provord:TypeCode>SALES ORDER</provord:TypeCode>


<provord:FulfillmentModeCode>DELIVER</provord:FulfillmentModeCode>

<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>


    <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:Identification>
        <provord:OrderQuantity>1</provord:OrderQuantity>
        <provord:ServicePointCode/>
    </provord:ProvisioningOrderLine>
}
Predefined Automation Plug-ins

for $y in $x/cord:Attribute
  return
  <corecom:Specification>
    <corecom:ServiceActionCode>
    </corecom:ServiceActionCode>
  </corecom:Specification>

<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:

```java
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:

```java
declare namespace automator = "java:oracle.communications.ordermanagement.automation.plugin.ScriptReceiverContextInvocation";
... declare variable $automator external;
```

Oracle recommends that you use the standard Apache log class. Always declare this class along with the $log java binding.

```java
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:

```java
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. For more information about structuring order update code, see "Using OrderDataUpdate Elements to Pass Order Modification Data".
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 XML API 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
declare namespace oms="urn:com:metasolv:oms:xmlapi:1";
declare namespace automator = "java:oracle.communications.orderrmanagement.automation.plugin.ScriptReceiverContext";
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

      ...
    </OrderDataUpdate>
  )

Note: For large orders, Oracle recommends that you use the OSM Java API processXMLRequest method in the OrderContext class to run XML API UpdateOrder.Request instead of using an OrderDataUpdate statement. See "Using GetOrder and UpdateOrder API Functions in Large Orders".

---
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. For more information, see "External XQuery Automator" and "Internal XQuery Sender".

---

**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. For more information, see "External XQuery Automator" and "Internal XQuery Sender".

---

**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 Sender

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:

```xml
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:

```xml
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.

```xml
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.

```xml
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:
    ```xml
    <xsl:variable name="outboundMessage"
    select="java:setStringProperty($outboundMessage, '_wls_mimehdrContent_Type', 'text/xml; charset=UTF-8')"/>
    ```
  - Use outboundMessage to set up the OSM Web Service URI JMS message property:
    ```xml
    <xsl:variable name="outboundMessage"
    select="java:setStringProperty($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:
    ```xml
    <xsl:variable name="outboundMessage"
    ```
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:

```xml
<xsl:variable name="void"
```

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:

```xml
<xsl:template match="/">
  <xsl:variable name="order" select="oms:GetOrder.Response"/>
  <xsl:variable name="othervariable" select="$order/oms_root/oms:orderid"/>
</xsl:template>
```

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.ScriptSenderContextInvocation"
    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/BB0/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/BB0/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="/"
    xalan: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="utf-8"')"/>
<xsl:variable name="outboundMessage" select="java:setStringProperty($outboundMessage, 'URI', '/osm/wsapi')"/>
<xsl:variable name="log" select="java:info($log,concat('Sending Service Order for COM order: ', $order/oms:OrderID))"/>
<xsl:call-template name="sendSomOrder"/>
</xsl:template>
</xsl:stylesheet>
<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-wsseecurity-secext-1.0.xsd" soapenv:mustUnderstand="1">
      <wsse:UsernameToken xmlns:wsu = "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wsseecurity-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 xmlns:ebo="http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/ProvisioningOrder/V1">
      <ebo:ProcessProvisioningOrderEBM xmlns:ebo="http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/ProvisioningOrder/V1">
          xmlns:provord="http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/ProvisioningOrder/V1">
              xmlns:provord="http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/ProvisioningOrder/V1">
              <corecom:SalesOrderReference>
                <corecom:SalesOrderIdentification>
                </corecom:SalesOrderIdentification>
              </corecom:SalesOrderReference>
              <provord:RequestedDeliveryDateTime>2010-07-16T08:24:38Z</provord:RequestedDeliveryDateTime>
              <provord:TypeCode>SALES ORDER</provord:TypeCode>
              <provord:FulfillmentModeCode>DELIVER</provord:FulfillmentModeCode>
              <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>
<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: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: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($x/@name,'')}</corecom:Name>
  <corecom:ClassificationCode listID="PermittedTypeCode"></corecom:ClassificationCode>
  <corecom:ClassificationCode listID="BillingProductTypeCode"></corecom:ClassificationCode>
  <corecom:ClassificationCode listID="FulfillmentItemCode">{concat($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:Specification>
      }
    }
  </corecom:SpecificationGroup>

  <corecom:PrimaryClassificationCode>{concat($x/@name,'')}</corecom:PrimaryClassificationCode>
  <corecom:ServiceInstanceIndicator>true</corecom:ServiceInstanceIndicator>
</corecom:ItemReference>

<provord:ProvisioningOrderLineSpecificationGroup>
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:

  ```xml
  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:

  ```xml
  xmlns:automator="java:oracle.communications.ordermanagement.automation.plugin.ScriptReceiverContextInvocation"
  ```
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"
```

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"
```

```xml
<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. For more information about structuring order update code, see "Using OrderDataUpdate Elements to Pass Order Modification Data".

**Note:** For large orders, Oracle recommends that you use the OSM Java API processXMLRequest method in the OrderContext class to run XML API UpdateOrder.Request instead of using an OrderDataUpdate statement. See "Using GetOrder and UpdateOrder API Functions in Large Orders".

- 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 XML API 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
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" xmlns="http://www.metasolv.com/OMS/OrderDataUpdate"
xmns:xsl="http://www.w3.org/1999/XSL/Transform"
xmns:java="http://xml.apache.org/xslt/java"
xmns:xalan="http://xml.apache.org/xslt"
xmns:oms="urn:com:metasolv:oms:xmlapi:1"
xmlns:automator="java:oracle.communications.ordermanagement.automation.plugin.ScriptReceiverContextInvocation"
xmlns:context="java:com.mslv.oms.automation.TaskContext"
xmlns:log="java:org.apache.commons.logging.Log"
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:stylesheet>
```

```xml
<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 $taskData/oms:_root/oms:ControlData/oms:Functions/*[fn:position()=1] else ()"/>
  <OrderDataUpdate
    <xsl:for-each select="su:ParentLineId">
      <xsl:variable name="parent" select="."/>
```
<ExternalFulfillmentState>{$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. For more information, see "External XSLT Automator" and "Internal XSLT Sender".

*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. For more information, see "External XSLT Automator" and "Internal XSLT Sender".

*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
- Jeopardy Automators

**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".
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 a 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>
    </orderNotification>
```

**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";
```
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 expected 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.ScriptReceiverContextInvocation";
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)
Predefined Automation Plug-ins

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:

```java
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.

```java
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.ScriptSenderContextInvocation";
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>
    }
```

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)
};
```
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:
  ```java
generate com.mslv.oms.sample.atm_frame;
```

- Import statements required for this custom automation plug-in. For example:
  ```java
  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:
  ```java
  public class MyPlugin extends AbstractAutomator {
  ```

- The required run method, as dictated by the parent class, AbstractAutomator.
  ```java
  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.
  ```java
  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.

```java
String taskName = taskContext.getTaskMnemonic();
```

Add any require business logic.

```java
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.

```java
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);
        }
    }
}
```

## Internal Custom Java Sender

A basic internal custom Java sender has the following characteristics:

- **The name of the custom automation package.** For example:

  ```java
  package com.mslv.oms.sample.atm_frame;
  ```

- **Import statements required for this custom automation plug-in.** For example:

  ```java
  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:

  ```java
  public class MyPlugin extends AbstractSendAutomator {
  ```

- **The required run method, as dictated by the parent class, AbstractSendAutomator.**

  ```java
  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.

```java
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.
  ```java
  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.
  ```java
  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.

```java
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();
Custom Java Automation Plug-ins

- 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.

```java
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.

```java
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.

```java
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`.

```java
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("";
            osmUpdate.append("<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();
        }
    }
```
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.

```java
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". 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.

```java
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(osmUpdate.toString());
```

catch(Exception e) {
  System.out.println(e.getMessage());
}
```

return omsUpdate.toString();
```
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" +
            xmlns: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<z_site_data.getLength();i++) {
                Element e = (Element)z_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(osmUpdate.toString()); }
        catch(Exception e) {
            System.out.println(e.getMessage()); }
    return osmUpdate.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"
 xmlns: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>
</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:

```xml
<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.

```xml
<OrderDataUpdate>
  <Delete path="/client_info/address[@index='80132']/city" />
  <Update path="/client_info/address[@index='76579']/city">Newark</Update>
</OrderDataUpdate>
```
<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
<?xml version="1.0" encoding="UTF-8"?>

<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>
```
Examples of Sending Messages to External Systems

The following example shows a custom automation plug-in that sends data to an external system.

```java
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" 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><feType>PP</feType><portType>$FEP</portType><selectionMethod>LOAD_BALANCE</selectionMethod><portSelectionAttribName>AG2ObjectID</portSelectionAttribName><portSelectionAttribValue>189508</portSelectionAttribValue><portSelectionAttribName>AG2PortLabel</portSelectionAttribName><portSelectionAttribValue>F-31-OC-48</portSelectionAttribValue><portUpdateAttribName/><portUpdateAttribValue/>
            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.

```java
String xmlRequest = "<Message
    type="ni" 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><feType>PP</feType><portType>$FEP</portType><selectionMethod>LOAD_BALANCE</selectionMethod><portSelectionAttribName>AG2ObjectID</portSelectionAttribName><portSelectionAttribValue>189508</portSelectionAttribValue><portSelectionAttribName>AG2PortLabel</portSelectionAttribName><portSelectionAttribValue>F-31-OC-48</portSelectionAttribValue><portUpdateAttribName/><portUpdateAttribValue/>
```

Examples of Handling Responses from External Systems

The following example shows a custom automation plug-in that handles and processes response messages from an external system.

```java
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.

```java
public void sendMessage(Message originalMessage) {
    try {
        // Set up the JMS connections
        QueueConnectionFactory connectionFactory =
            (QueueConnectionFactory)jndiCtx.lookup("ConnectionFactory");
        QueueConnection queueConnection = connectionFactory.createQueueConnection();
        QueueSession queueSession = queueConnection.createQueueSession(false, 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());

---

**Task Compensation Strategy 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:

```xquery
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 extract the OSM_home/SDK/osm7.2.x-javadocs.zip OSM Java docs (where OSM_home is the location of the OSM software and x is the software build numbers).

- **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).

```xquery
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:

```xquery
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)
```