Oracle® Communications Order and Service Management

Developer's Guide

Release 7.4.1
F30318-02
May 2021

Oracle Communications Order and Service Management Developer's Guide, Release 7.4.1

F30318-02

Contents

3 Using the OSM XML API

9 Using XPath Functions

A Automation and Compensation Examples

B AutomationMap.xml File

C Automation: Start to Finish

Preface

This document provides information about the following areas of Oracle Communications Order and Service Management (OSM) that can be customized:

- Web services
- Extensible Markup Language (XML) Application Programming Interface (API)
- Automation
- Security Callback
- Behaviors
- Custom menu items and action items
- Localization of OSM

This document also provides a process example of provisioning a Plain Old Telephone Service (POTS) customer using unbundled local loop to illustrate various types of customization.

Audience

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

- System interfaces
- 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

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.

1

Introduction

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 *OSM Modeling Guide*.

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 *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 "Using OSM Order Management Web Services".

OSM Automation

OSM automation provides the primary interface for *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 *OSM Modeling Guide*.

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 "Using OSM Security Callback".

The OSM XML API

The OSM XML API is deprecated for all uses except the following:

- Customizing the appearance or functioning of a task when customization using behaviors or OSM Java server pages does not satisfy all of your requirements.
- Using from within an automation plug-in when necessary because the Web Services API and the OSM automation functionality do not meet your requirements.

For information about the OSM XML API, see "Using the OSM XML API".

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 "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 processes involves modifying OSM XML files.

This topic is further explored in "Localizing OSM".

Logging with ODL (Traditional OSM Only)

Oracle recommends that you use Oracle Diagnostic Logging (ODL), which is used by most Oracle Fusion Middleware applications, 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 *OSM Modeling Guide*.

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** file, 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 *OSM Modeling Guide* 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:

https://www.w3schools.com/xml/xpath_intro.asp

An XQuery tutorial is available at:

https://www.w3schools.com/xml/xquery_intro.asp

> **Note:**
>
> In OSM, XQuery statements are limited to a maximum of 4000 characters.

About the OSM SDK

> **Note:**
>
> In this book, “traditional OSM” refers to the traditional way of installing and maintaining an OSM environment and “OSM cloud native” refers to OSM deployed in a cloud native environment.

A number of directories within the SDK are referenced in procedures throughout this guide. For traditional OSM, if you selected **Custom** installation, then you can choose both the **SDK** and the **Samples** to be installed optionally. You can find the SDK inside the *OSM_Home* directory, where OSM is installed.

> **Note:**
>
> You can install only the SDK on a Windows, UNIX, or Oracle Linux machine by running the OSM Installer again and choosing **Custom**.

For OSM cloud native, the SDK is a separate artifact that is available in the download pack.

2

Using OSM Order Management Web Services

This chapter describes Oracle Communications Order and Service Management (OSM) order management 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 Order Management Web Services

The OSM Web Services provide the primary interface for in-bound order operations such as creating, updating, or canceling an order. OSM 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

- CreateOrderBySpecification
- CreateOrder
- FindOrder
- GetOrder
- UpdateOrder
- SuspendOrder
- ResumeOrder
- CancelOrder
- AbortOrder
- FailOrder

- ResolveFailure
- RetryOrder

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 OSM 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.

Determining Request and Response Queues To Use

The queues you should use depend on whether your implementation is in a WebLogic Server cluster or in a single server.

For more information about the specifics of the queues that are created, see the discussion of OSM installed components in *OSM System Administrator's Guide*.

Queues in a WebLogic Server Cluster

There are two queues created for requests. In a WebLogic Server cluster, you can use one or both of the queues. Following are the considerations to use to help you decide:

- The **oms_ws_cluster_requests** queue is designed with optimization for processing requests relating to updating or retrieving existing orders in a WebLogic Server cluster. This queue can also handle requests to create new orders, if desirable to simplify integration with upstream systems.
- The **oms_ws_requests** queue can be used for new order creation requests, and this avoids some overhead of the oms_ws_cluster_requests queue. This queue checks for an <OrderId> element which will not be present in a new order creation request. If order updates and new order requests come from two different interfaces, then it is a good idea to send new order requests to the oms_ws_requests queue.
- If there are a larger number of order updates than new order creation requests, then it makes sense to process all requests using the oms_ws_cluster_requests queue.

For responses, a WebLogic Server cluster environment uses the **oms_ws_cluster_responses** and **oms_ws_cluster_correlates** queues, which set the JMSCorrelationID properly and forward the message to the destination specified in the ReplyTo JMS property of the message.

Queues in a Single-Server WebLogic Server Environment (Traditional OSM Only)

In a single-server environment, like a development environment, you must always direct requests to the oms_ws_requests queue. Responses are forward the message to the destination specified in the ReplyTo JMS property of the message without the need for other queues.

Sending OSM Web Service Requests to a WebLogic Server Cluster (Traditional OSM Only)

If your web services client connects to OSM using Oracle WebLogic Server, and if your WebLogic Server instance for OSM is a cluster, the WSDL generated by WebLogic Server identifies the endpoint using the address of the first managed server and ignores the addresses of all other managed servers.

To ensure that the addresses of all managed servers are used, include code in your client to override the endpoint.

Example 2-1 demonstrates how to override the default endpoint and include all of the endpoints.

Example 2-1 Sample Code to Override the Endpoint Address for a Cluster

```
Stub stub = (Stub) port;
stub._setProperty(WlsProperties.READ_TIMEOUT, 1000000);
stub._setProperty(WLStub.JMS_TRANSPORT_JNDI_URL, t3://
ip_address1:port1,ip_address2:port2,ipaddressn:portn");
```

In the example, *ip_address1* is the IP address of the first managed server and *port1* is the port of that server, *ip_address2* is the IP address of the second managed server and *port2* is the port of that server, and so on for all of your managed servers. As in the example, separate each IP address from its port by a colon and separate the address information for the servers by commas.

Accessing the WSDL Files

OSM Web Services are part of the OSM installation. The OSM WSDL files and the supporting schema files (XSD files) are located in the **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

http://*server:port*/OrderManagement/diagnostic/wsapi for web service operations used for diagnosing problem orders.

where:

- *server* is the specific server (traditional OSM deployments) on which the application is deployed. In OSM cloud native, the base hostname to access this instance is *instance*.*project*.**osm**.**org**.
- *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 2-2.

Example 2-2 Message Header

```
<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 2-3.

Example 2-3 Message Body

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

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

White Space in Message Text

OSM trims off the white space to the right of the beginning of a text block and to the left of the end of a text block. For example, if you send an update with the following field:

```
<osmc:street index="1414682666685">   190   Attwell Drive  </osmc:street>
```

the response message returns having removed the white space at the beginning and the end of the text block:

```
<osmc:street index="1414682666685">190   Attwell Drive</osmc:street>
```

Testing OSM Web Services

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 Order Management Web Service Operations." To learn more about order states and their transitions, see *OSM Concepts.*

Web Services Sample

The OSM SDK provides a Web Service sample that demonstrates how OSM Web Services are called. The sample is available in the **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 Order Management Web Service Operations

The remaining sections of this chapter describes each OSM 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

Unless parameters require additional explanation, the parameters that are defined by each web service are not provided in this documentation. 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 and OSM Web Service Client Error Processing

OSM Web Service operations sent over JMS or HTTPS to OSM may fail for various reasons, such as a local error or exception on the OSM server, incorrect syntax, invalid permissions, and so on. The OSM Web Service client, such as a CRM communicating to OSM in the COM role, must monitor returning response messages from OSM for any fault types that indicate whether the operation succeeded or failed. If the OSM Web Service operation request fails it is the responsibility of the OSM Web Service client to track and resubmit the failed request after troubleshooting the problem.

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 are not provided in this documentation. However, several request and response examples are provided, which you can use to help you create or understand other web service requests and responses. See "Order Management 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:

- OSM 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 XML API Functionality 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 (query task) 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:

AmendmentFilter
Retrieves the amendment information associated with this order, if the **LevelOfDetail** child element is set to **AmendmentSummary**. If it is not specified, no amendment summary is 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 GetOrder response message specified in the **View** parameter. This filtering functionality improves OSM performance, especially when the order with the multi-instance data is a large order.

- **Condition**: An XPath 1.0 expression against the order data defined by the **View** parameter. OSM returns only the instances of the order data selected by the expression, not the other instances of the element. All other parent or sibling elements are returned.

  For example, in a situation where a customer has multiple <address> 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 instance of the <address> element (city, postal code, and street). The other 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>
  ```

  In the example, any sibling elements of <subscriber_info>, or sibling elements of <address> (except for the other instances of the <address> element) would be returned.

  When you are using an order condition that includes an element that is using a distributed order template, you should include the namespace of the data element in the condition. For example:

```
<OrderDataFilter>
   <Condition>
      /ControlData/OrderItem[@type='{OrderItemNamespace}OrderItemName' and
@LineId='1']
   </Condition>
</OrderDataFilter>
```

There can be as many **<Condition>** child elements as required. When there are more than one **<Condition**> elements, each condition is evaluated and applied independently of the other conditions to the sections of the order data respectively.

RemarkFilter
Controls how remarks and attachments are returned.

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

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 (query task) 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.

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 (query task) 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** parameter.
  - **Condition**: An XPath 1.0 expression against the order data defined by the **ResponseView** parameter. OSM returns only the instances of the order data selected by the expression, not the other instances of the element. All other parent or sibling elements are returned.

    For example, in a situation where a customer has multiple <address> 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 instance of the <address> element (city, postal code, and street). The other 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>
    ```

    For example, any sibling elements of <subscriber_info>, or sibling elements of <address> (except for the other instances of the <address> element) would be returned.

    When you are using an order condition that includes an element that is using a distributed order template, you should include the namespace of the data element in the condition. For example:

    ```
    <OrderDataFilter>
       <Condition>
          /ControlData/OrderItem[@type='{OrderItemNamespace}OrderItemName'
    and @LineId='1']
       </Condition>
    </OrderDataFilter>
    ```

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 **SDK/Samples/WebService**.

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 resolves all failed tasks within an order or a collection of order components for a given order. The operation causes all tasks to transition back to the corresponding normal execution mode such as do, redo and undo from failed-do, failed-redo, or failed undo. The operation also causes the task to return to the task state it had been in before failing (normally the accepted or a custom task state).

If you use the failed order state, then this operations also causes an order that is currently failed to transition back to the order state prior to entering the current failed order state.

You can grant permissions for this operation by editing the Manage Order Fallout transaction for the failed, amending, canceling, in progress, suspended, or waiting for revision states in the order life-cycle policy associated with the order's specification in Design Studio.

Preconditions

- The current state of the specified order must be one of the following:
  - open.not_running.failed
  - open.not_running.suspended

  - open.not_running.waitinforrevision
  - open.running.in_progress
  - *open.running.amending*
  - open.running.canceling
- 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

All tasks on the order or on specific order components of the order that are in a failed execution mode transition to the corresponding normal execution mode in the state the task had been in before failing. For example, an order with a task in the failed-undo mode in the accepted state would transition back to the normal undo mode in the state the task had been in when it had failed.

If this operation is run when the order is in the failed sate, then the order is successfully transitioned to its previous state.

RetryOrder

This operation retries all failed tasks within an order or a collection of order components for a given order. The operation causes all tasks to transition back to the corresponding normal execution modes such as do, redo and undo from failed-do, failed-redo, or failed-undo. The operation also causes tasks to return to the received state.

You can grant permissions for this operation by editing the Manage Order Fallout transaction for the failed, amending, canceling, in progress, suspended, or waiting for revision states in the order life-cycle policy associated with the order's specification in Design Studio.

Preconditions

- The current state of the specified order must be one of the following:
  - open.not_running.failed
  - open.not_running.suspended
  - open.not_running.waitinforrevision
  - open.running.in_progress
  - *open.running.amending*
  - open.running.canceling
- 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

All tasks on the order or on specific order components of the order that are in a failed execution mode transition to the corresponding normal execution mode in the received state. For example, an order with a task in the Failed-Undo mode in the accepted state would transition to the Undo mode in the received state and another task in the Failed-Do mode in the assigned state would transition to the Do mode in the received 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 diagnoses operations are defined in the **OrderManagementDiag.wsdl**.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 OSM Web Service operation defines. The information is presented through an example that is applicable to all operations.

Order Management WSDL File

Example 2-4 is an excerpt from the **OrderManagementWS.wsdl** file that shows how a typical OSM Web Service operation is defined.

Example 2-4 WSDL Operation Definition

```
<wsdl:operation name="CreateOrderBySpecification">
              <wsdl:input message="prov:CreateOrderBySpecificationRequest">
              </wsdl:input>
              <wsdl:output message="prov:CreateOrderBySpecificationResponse">
              </wsdl:output>
              <wsdl:fault name="InvalidOrderSpecificationFault"
   message="prov:CreateOrder_faultMsg">
              </wsdl:fault>
              <wsdl:fault name="TransactionNotAllowedFault"
   message="prov:CreateOrder_faultMsg1">
 </wsdl:fault>
              <wsdl:fault name="InvalidOrderDataFault"
   message="prov:CreateOrder_faultMsg3">
              </wsdl:fault>
       </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 **OrderManagmentWS.wsdl** file, and the corresponding XSD file is **OrderManagementWS.xsd**.

Order Management XSD File

This section describes how to navigate the XSD files. The request and response structures defined in the XSD are used by the OSM 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.

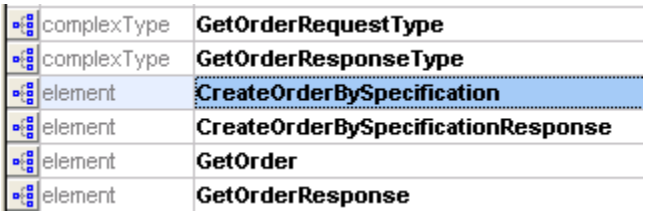
XMLSpy offers several ways to view XML files. XSD files containing large structures can be very difficult to read. The examples provided in this section show how to view XSD files using the Schema/WSDL Design view, which allows you to view the top level structures and then expand and collapse them as 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)

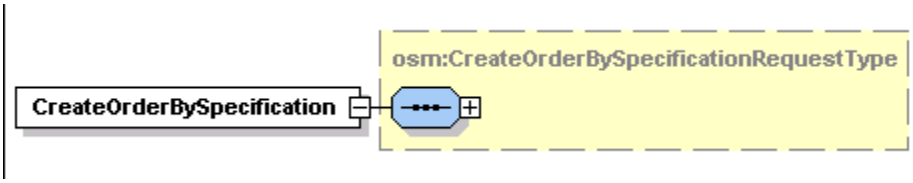
Figure 2-1 shows a portion of the **OrderManagmentWS.xsd** file in the Schema/WSDL Design view, as it appears when first opened. This is the top level of the view, which lists all simpleType, complexType, and elements that are defined in the file.

Figure 2-1 Schema/WSDL Design View



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 2-2 shows the result of clicking the grey box located to the left of CreateOrderBySpecification.

Figure 2-2 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 2-3 and Figure 2-4 show the result of this action.

Figure 2-3 Further Expanded Structure

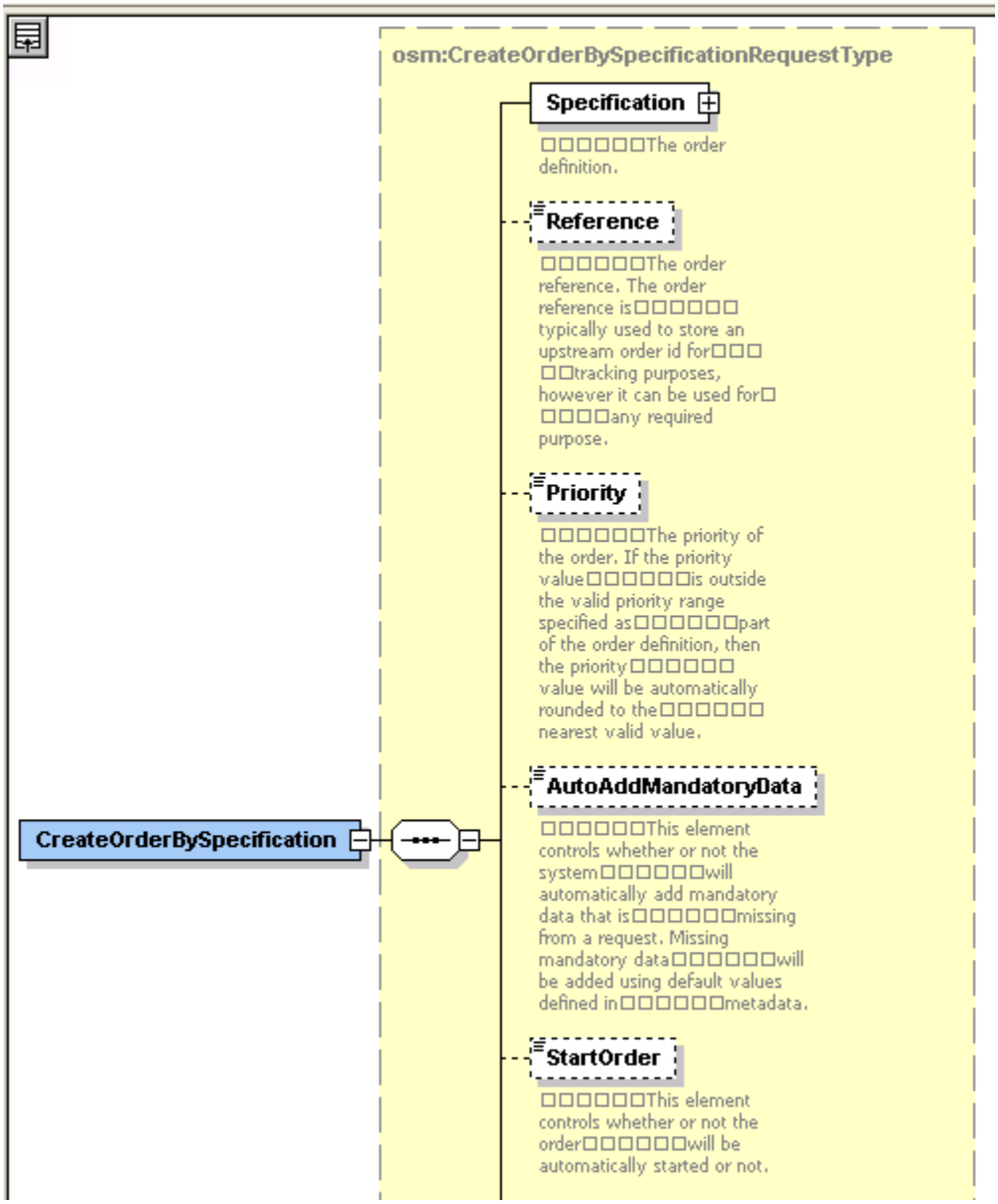
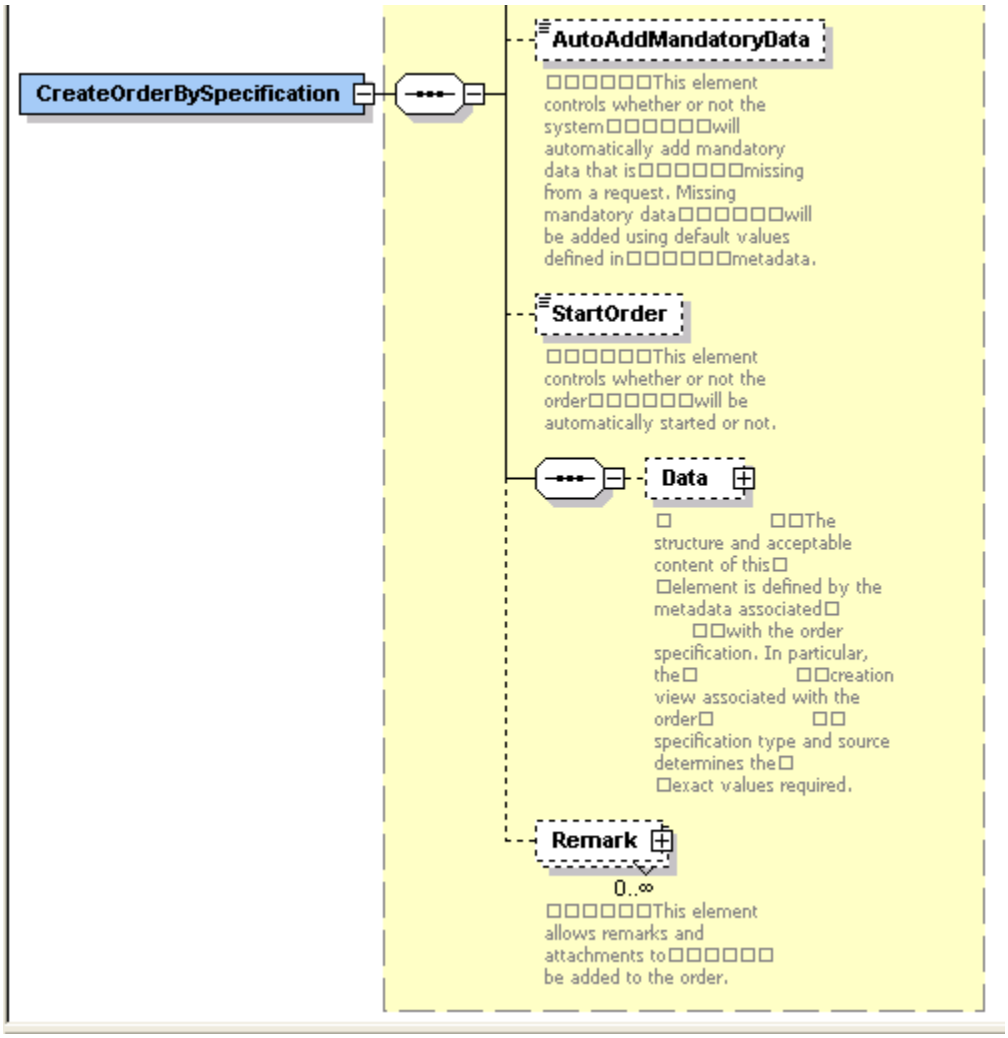


Figure 2-4 Further Expanded Structure (continued)

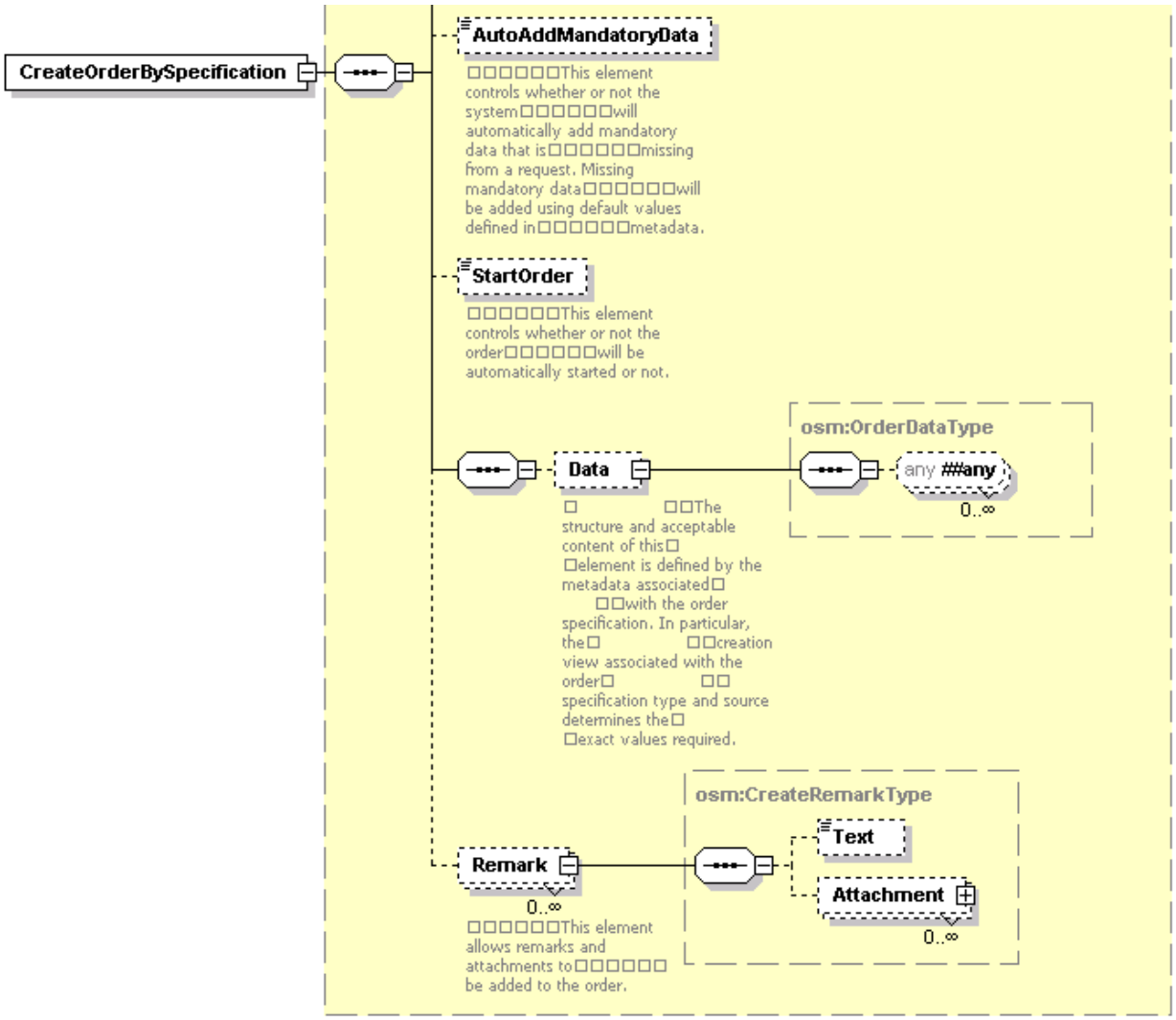


From this level, you can see that CreateOrderBySpecificationRequestType defines:

- 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 2-5 shows the result of this action.

Figure 2-5 Further Expansion of Data and Remark Elements



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 2-3. 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 2-6 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 2-6 Schema/WSDL Design View

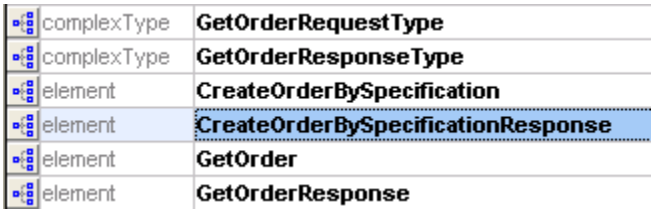
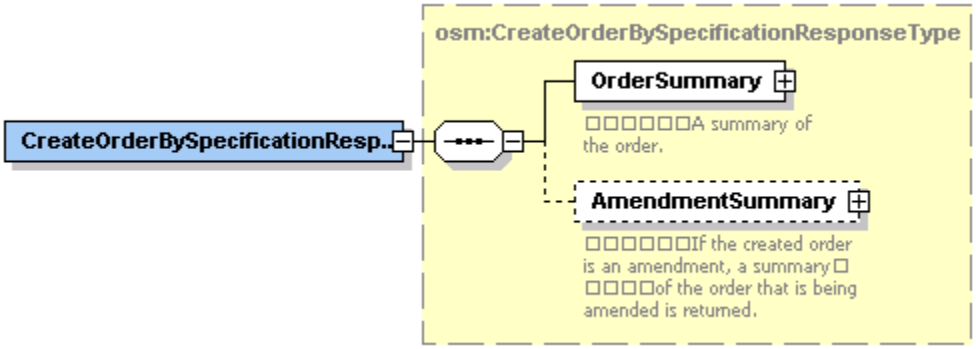


Figure 2-7 shows the expected response defined by CreateOrderBySpecificationResponse, which can be expanded even further.

Figure 2-7 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.

Order Management 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 2-5 CreateOrderBySpecificationRequest

```
<?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-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:CreateOrderBySpecification>
         <ws:Specification>
            <ws:Cartridge>
               <ws:Name>view_framework_demo</ws:Name>
               <!--Optional:-->
               <ws:Version>1.0</ws:Version>
            </ws:Cartridge>
            <ws:Type>vf_demo</ws:Type>
            <ws:Source>web</ws:Source>
         </ws:Specification>
         <!--Optional:-->
         <ws:Reference>test message</ws:Reference>
         <!--Optional:-->
         <ws:Priority>5</ws:Priority>
         <!--Optional:-->
```

```
         <ws:AutoAddMandatoryData>true</ws:AutoAddMandatoryData>
         <!--Optional:-->
         <ws:StartOrder>true</ws:StartOrder>
         <!--Optional:-->
         <ws:Data>
<_root>
             <account_information>
               <amount_owing>553</amount_owing>
             </account_information>
           </_root>
         </ws:Data>
<!--Zero or more repetitions:-->
<ws:Remark>
            <!--Optional:-->
            <ws:Text>Test Remark</ws:Text>
            <!--Zero or more repetitions:-->
            <ws:Attachment>
               <!--Optional:-->
               <ws:Name>readme.txt</ws:Name>
               <!--You have a CHOICE of the next 3 items at this level-->
               <ws:swaRefMimeContent>cid:first</ws:swaRefMimeContent>
               <!--ws:base64BinaryContent>?</ws:base64BinaryContent>
               <ws:hexBinaryContent>?</ws:hexBinaryContent-->
     </ws:Attachment>
 </ws:Remark>
<ws:Remark>
            <!--Optional:-->
            <ws:Text>Test Remark</ws:Text>
            <!--Zero or more repetitions:-->
            <ws:Attachment>
               <!--Optional:-->
               <ws:Name>test2.txt</ws:Name>
               <!--You have a CHOICE of the next 3 items at this level-->
               <ws:swaRefMimeContent>cid:second</ws:swaRefMimeContent>
               <!--ws:base64BinaryContent>?</ws:base64BinaryContent>
               <ws:hexBinaryContent>?</ws:hexBinaryContent-->
     </ws:Attachment>
 </ws:Remark>
         </ws:CreateOrderBySpecification>
   </soapenv:Body>
</soapenv:Envelope>
```

Response Example

Example 2-6 CreateOrderBySpecificationResponse

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
<soapenv:Header/>
<soapenv:Body>
<ws:CreateOrderBySpecificationResponse xmlns:ws="http://xmlns.oracle.com/
communications/ordermanagement">
<ws:OrderSummary>
<ws:Id>202</ws:Id>
<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:State>open.not_running.not_started</ws:State>
<ws:Reference>test message</ws:Reference>
<ws:Priority>5</ws:Priority>
</ws:OrderSummary>
</ws:CreateOrderBySpecificationResponse>
</soapenv:Body>
```

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.

> **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.

Example 2-7 CreateOrderBySpecificationRequest - Setting Up Reference Nodes

```
<ord:CreateOrderBySpecification>
  <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:CreateOrderBySpecification>
```

GetOrder Examples

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

Request and Response Example

Example 2-9 shows a GetOrderRequest.

Example 2-9 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.

Example 2-10 shows the GetOrderResponse returned for the GetOrderRequest in Example 2-9.

Example 2-8 GetOrder Request

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ord="http://xmlns.oracle.com/communications/ordermanagement">
   <soapenv:Header/>
   <soapenv:Body>
      <ord:GetOrder>
         <ord:OrderId>?</ord:OrderId>
         <!--Optional:-->
         <ord:View>?</ord:View>
         <!--Optional:-->
         <ord:OrderDataFilter>
            <!--Zero or more repetitions:-->
            <ord:Condition>?</ord:Condition>
         </ord:OrderDataFilter>
         <!--Optional:-->
         <ord:RemarkFilter>
            <!--Optional:-->
            <ord:RetrieveRemarks>false</ord:RetrieveRemarks>
            <!--Zero or more repetitions:-->
            <ord:AttachmentFilter>
               <!--Optional:-->
               <ord:FileNameMatch>.*</ord:FileNameMatch>
               <!--Optional:-->
               <ord:MinSize>0</ord:MinSize>
               <!--Optional:-->
               <ord:MaxSize>4</ord:MaxSize>
               <!--Optional:-->
               <ord:Format>inlineBase64Binary</ord:Format>
            </ord:AttachmentFilter>
         </ord:RemarkFilter>
         <!--Optional:-->
         <ord:AmendmentFilter>
            <ord:LevelOfDetail>AmendmentSummary</ord:LevelOfDetail>
         </ord:AmendmentFilter>
         <!--Optional:-->
```

```
         <ord:LifecycleEventFilter>
            <!--Optional:-->
            <ord:RetrieveLifecycleEvents>false</ord:RetrieveLifecycleEvents>
         </ord:LifecycleEventFilter>
      </ord:GetOrder>
   </soapenv:Body>
</soapenv:Envelope>
```

Example 2-9 GetOrderRequest

```
<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-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>Username</wsse:Username>

<wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
username-token-profile-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 2-10 GetOrderResponse

```
<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></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:city index="1414682666687">TO</osmc:city>
                 <osmc:postal_code index="1414682666686">M9W6H8</
osmc:postal_code>
                 <osmc:street index="1414682666685">190 Attwell Drive</
osmc:street>
              </osmc:address>
              <osmc:address index="1414682666692">
                 <osmc:city index="1414682666693">TO</osmc:city>
                 <osmc:postal_code index="1414682666694">A1B2Z7</
osmc:postal_code>
                 <osmc:street index="1414682666695">55 Updated St</osmc:street>
              </osmc:address>
              <osmc:address index="1414682666696">
                 <osmc:city index="1414682666697">TO</osmc:city>
                 <osmc:postal_code index="1414682666698">A1B2Z7</
osmc:postal_code>
                 <osmc:street index="1414682666699">56 Updated St</osmc:street>
              </osmc:address>
              <osmc:primary_phone_number index="1414682666689">1111111111</
osmc:primary_phone_number>
              <osmc:name index="1414682666688">John Doe</osmc:name>
           </osmc:subscriber_info>
           <osmc:adsl_service_details index="1414682666690">
              <osmc:bandwidth index="1414682666691">3</osmc:bandwidth>
           </osmc:adsl_service_details>
        </osmc:_root>
     </Data>
  </GetOrderResponse>
 </env:Body>
</env:Envelope>
```

Request and Response Example with OrderDataFilter

Example 2-11 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 2-12 shows the GetOrderResponse returned for the GetOrderRequest in Example 2-11.

Example 2-11 GetOrderRequest with OrderDataFilter

```
<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-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>Username</wsse:Username>

<wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
username-token-profile-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:OrderDataFilter>
            <ord:Condition>/subscriber_info/address[street='190 Attwell Drive']</
ord:Condition>
        </ord:OrderDataFilter>
        </ord:GetOrder>
   </soapenv:Body>
</soapenv:Envelope>
```

Example 2-12 GetOrderResponse with OrderDataFilter

```
<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></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:city index="1414682666687">TO</osmc:city>
                 <osmc:postal_code index="1414682666686">M9W6H8</
osmc:postal_code>
                 <osmc:street index="1414682666685">190 Attwell Drive
                 </osmc:street>
              </osmc:address>
              <osmc:primary_phone_number index="1414682666689">1111111111
              </osmc:primary_phone_number>
              <osmc:name index="1414682666688">John Doe</osmc:name>
           </osmc:subscriber_info>
           <osmc:adsl_service_details index="1414682666690">
              <osmc:bandwidth index="1414682666691">3</osmc:bandwidth>
           </osmc:adsl_service_details>
        </osmc:_root>
     </Data>
  </GetOrderResponse>
```

```
 </env:Body>
</env:Envelope>
```

Response Example - Order with Distributed Order Template Elements and Transformed Order Items

Example 2-13 Partial GetOrderResponse containing Distributed Order Template Data

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Header/>
  <env:Body>
    <GetOrderResponse xmlns="http://xmlns.oracle.com/communications/
ordermanagement">
      <OrderSummary>
        <Id>20</Id>
        <Specification>
           <Cartridge>
             <Name>OsmCentralOMExample-Solution</Name>
             <Version>4.0.0.0.0</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>2014-04-11T10:03:32.495-07:00</ExpectedStartDate>
        <ExpectedDuration>PT0S</ExpectedDuration>
        <ExpectedOrderCompletionDate>2014-04-11T10:03:32.495-07:00
        </ExpectedOrderCompletionDate>
        <Priority>5</Priority>
      </OrderSummary>
      <Data>
        <osmc:_root index="0"
xmlns:osmc="urn:oracle:names:ordermanagement:cartridge:OsmCentralOMExample-
Solution:4.0.0.0.0:view:OsmCentralOMExampleQueryTask">
          <osmc:OrderHeader index="1">
            <osmc:numSalesOrder index="2">Order1397235767310</osmc:numSalesOrder>
            <osmc:typeOrder index="3">Add</osmc:typeOrder>
          </osmc:OrderHeader>
          <osmc:CustomerDetails index="11">
            <osmc:nameLocation index="13">Location1</osmc:nameLocation>
            <osmc:typeAddress index="23">Building</osmc:typeAddress>
          </osmc:CustomerDetails>
          <osmc:AccountDetails index="24">
            <osmc:numAccount index="25">TEL1234</osmc:numAccount>
            <osmc:status index="26">Existing</osmc:status>
            <osmc:corporate index="27">PoC</osmc:corporate>
            <osmc:inscrState index="30">232,232,232,232</osmc:inscrState>
            <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="{http://www.oracle.com/otm/cso}TransformedOrderLineType"
```

```
index="1397235812899" referencedIndex="1397235811129" xmlns:ct160="http://
www.oracle.com/otm/cso" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
                    <ct135:LineId index="1397235812853" xmlns:ct135="http://
xmlns.oracle.com/communications/studio/ordermanagement/transformation">CSO_2</
ct135:LineId>
                    <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:QoS index="1397235812852">Data</ct264:QoS>
                      <ct264:UploadSpeed index="1397235812850">3.072</
ct264:UploadSpeed>
                    </ct160:dynamicParams>
                    <ct160:Recognition
index="1397235812846">{OracleComms_Model_BroadbandInternet/
4.0.0.0.0}SA_Provision_BroadbandInternetSpec</ct160:Recognition>
                    <ct160:LineName
index="1397235812847">SA_Provision_BroadbandInternetSpec [Add]</ct160:LineName>
                    <ct160:FulfillmentPattern index="1397235812855">{http://
oracle.communications.ordermanagement.unsupported.centralom}Service.Broadband</
ct160:FulfillmentPattern>
                    <ct160:Action index="1397235812854">Add</ct160:Action>
                  </osmc:orderItemRef>
                </osmc:transformedOrderItem>
                <osmc:componentKey
index="1397235812889">ProvisioningFunction.DSLProvisioningSystem_Region2.WholeOrd
er</osmc:componentKey>
                <osmc:orderItem index="1397235812892">
                  <osmc:orderItemRef
xsi:type="ct211:CustomerOrderItemSpecificationType" type="{http://
oracle.communications.ordermanagement.unsupported.centralom}CustomerOrderItemSpec
ificationType" index="1397235812893" referencedIndex="1397235811126"
xmlns:ct211="http://oracle.communications.ordermanagement.unsupported.centralom"
xmlns: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>
                    <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:serviceActionCode>Add</im:serviceActionCode>
                        <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: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>
          <osmc:OrderItem xsi:type="ct211:CustomerOrderItemSpecificationType"
type="{http://
oracle.communications.ordermanagement.unsupported.centralom}CustomerOrderItemSpec
ificationType" index="1397235811126" xmlns:ct211="http://
oracle.communications.ordermanagement.unsupported.centralom" xmlns: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>
            <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:serviceActionCode>Add</im:serviceActionCode>
                <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>
            [...]
            <osmc:TransformedOrderItem xsi:type="ct160:TransformedOrderLineType"
type="{http://www.oracle.com/otm/cso}TransformedOrderLineType"
index="1397235811129" xmlns:ct160="http://www.oracle.com/otm/cso"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
              <ct135:LineId index="1397235812853" xmlns:ct135="http://
xmlns.oracle.com/communications/studio/ordermanagement/transformation">CSO_2</
ct135:LineId>
              <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:QoS index="1397235812852">Data</ct264:QoS>
                <ct264:UploadSpeed index="1397235812850">3.072</
ct264:UploadSpeed>
              </ct160:dynamicParams>
              <ct160:Recognition
index="1397235812846">{OracleComms_Model_BroadbandInternet/
4.0.0.0.0}SA_Provision_BroadbandInternetSpec</ct160:Recognition>
              <ct160:LineName
index="1397235812847">SA_Provision_BroadbandInternetSpec [Add]</ct160:LineName>
              <ct160:FulfillmentPattern index="1397235812855">{http://
oracle.communications.ordermanagement.unsupported.centralom}Service.Broadband</
ct160:FulfillmentPattern>
              <ct160:Action index="1397235812854">Add</ct160:Action>
            </osmc:TransformedOrderItem>
            <osmc:MappingContext index="1397235812856">
              <osmc:ProviderFunction index="1397235812857">
                <osmc:namespace index="1397235812858">OracleComms_Model_Base/
4.0.0.0.0</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_BroadbandInt
ernet_Primary_---g--+U--+R---QI0kDkw</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 2-14 UpdateOrderRequest

```
<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: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 2-15 UpdateOrderRequest: Update nodes

```
<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:View>enter_payment_details</ws:View>
      <WS:UpdatedNodes>
        <_root>
          <ControlData>
             <Functions>
                <FulfillBillingFunction>
                   <orderItem>
                   <ExternalFulfillmentState>COMPLETED</ExternalFulfillmentState>
                       <orderItemRef>
                          <serviceName>C_GSM_ADD_SUB</serviceName>
                          <LineId>987654</LineId>
                      </orderItemRef>
                   </orderItem>
               </FulfillBillingFunction>
             </Functions>
          </ControlData>
        </_root>
       </ws:UpdatedNodes>
       <ws:ExternalFulfillmentStates>
         <ws:OrderItemOrderComponentFulfillmentState>
           <ws:ExternalFulfillmentState>COMPLETED</ws:ExternalFulfillmentState>
           <ws:OrderComponentIndex>1234</ws:OrderComponentIndex>
           <ws:OrderItemIndex>456789</ws:OrderItemIndex>
         </ws:OrderItemOrderComponentFulfillmentState>
       </ws:ExternalFulfillmentStates>
    </ws:UpdateOrder>
  </soapenv:Body>
</soapenv:Envelope>
```

Example 2-16 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>41</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 2-17 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='111222333']
[@type='{http://
oracle.communications.ordermanagement.unsupported.centralom}CustomerOrderItemSpec
ificationType']/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 2-18 shows the standard response message returned from an UpdateOrderRequest.

Example 2-19 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 2-20 shows the UpdateOrderResponse to the UpdateOrderRequest in Example 2-19.

Example 2-18 UpdateOrderResponse

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

Example 2-19 UpdateOrderRequest with ResponseView and OrderDataFilter

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ord="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>admin</wsse:Username>
     <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-username-token-profile-1.0#PasswordText">admin333</wsse:Password>
   </wsse:UsernameToken>
  </wsse:Security>
 </soapenv:Header>
 <soapenv:Body>
   <ord:UpdateOrder>
   <ord:OrderId>10</ord:OrderId>
   <ord:View>demo_query</ord:View>
   <ord:ResponseView>demo_query</ord:ResponseView>
   <ord:OrderDataFilter>
      <ord:Condition>/subscriber_info/address[street='112 Update Drive']</
ord:Condition>
   </ord:OrderDataFilter>
   <ord:UpdatedNodes>
    <_root>
     <subscriber_info>
      <address>
        <city>TO</city>
        <postal_code>A1A1A1</postal_code>
        <street>112 Update Drive</street>
      </address>
     </subscriber_info>
    </_root>
   </ord:UpdatedNodes>
  </ord:UpdateOrder>
 </soapenv:Body>
</soapenv:Envelope>
```

Example 2-20 UpdateOrderResponse with ResponseView and OrderDataFilter Applied

```
<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:adsl_service_details>
    </osmc:_root>
```

```
  </UpdateOrderResponse>
   </Data>
 </env:Body>
</env:Envelope>
```

SuspendOrder Examples

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

Request Example

Example 2-21 SuspendOrderRequest

```
<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: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 2-22 SuspendOrderResponse

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Header/>
   <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 2-23 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-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:ResumeOrder>
         <ws:OrderId>1176</ws:OrderId>
      </ws:ResumeOrder>
   </soapenv:Body>
</soapenv:Envelope>
```

Response Example

Example 2-24 ResumeOrderResponse

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Header/>
   <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 2-25 CancelOrderRequest

```
<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:CancelOrder>
         <ws:OrderId>1316</ws:OrderId>
         <!--Optional:-->
      </ws:CancelOrder>
   </soapenv:Body>
</soapenv:Envelope>
```

Response Example

Example 2-26 CancelOrderResponse

```
<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>
```

RetryOrder and ResolveFailure Examples

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

> **Note:**
>
> The structure of the RetryOrder and ResolveFailure operations are identical apart from the operation name itself.

Request Example Whole Order

Example 2-27 RetryOrderRequest

```
<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:RetryOrder>
         <ws:OrderId>18</ws:OrderId>
         <ws:Reason>1307</ord:Reason>
      </ws:RetryOrder>
   </soapenv:Body>
</soapenv:Envelope>
```

Response Example Whole Order

Example 2-28 RetryOrderResponse

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Header/>
   <soapenv:Body>
      <ws:RetryOrderResponse xmlns:ws="http://xmlns.oracle.com/communications/
ordermanagement">
         <ws:OrderId>18</ws:OrderId>
      </ws:RetryOrderResponse>
   </soapenv:Body>
</soapenv:Envelope>
```

Request Example Order Component

Example 2-29 ResolveFailureRequest

```
<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:ResolveFailure>
         <ws:OrderId>18</ws:OrderId>
         <ws:Reason>1307</ord:Reason>
         <ws:OrderComponent>
            <ws:OrderComponentId>

ADD_SUB_OPT.OrderProcessingSystemA.OrderProcessingDemoGranularity.1
               </ws:OrderComponentId>
         </ws:OrderComponent>
      </ws:ResolveFailure>
   </soapenv:Body>
</soapenv:Envelope>
```

Response Example Order Component

Example 2-30 ResolveFailureResponse

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Header/>
   <soapenv:Body>
```

```
      <ws:ResolveFailureResponse xmlns:ws="http://xmlns.oracle.com/
communications/ordermanagement">
         <ws:OrderId>18</ws:OrderId>
      </ws:ResolveFailureResponse>
   </soapenv:Body>
</soapenv:Envelope>
```

GetOrderProcessHistory Examples

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

GetOrderProcessHistory Requests

Example 2-31 GetOrderProcessHistory Requests by CompensationID

```
<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 2-32 GetOrderProcessHistory Requests by Perspective (Original)

```
<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 2-33 GetOrderProcessHistory Requests by Perspective (Latest)

```
<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 2-34 GetOrderProcessHistory Response by CompensationID

```
<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>
         <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</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>
                  <Links/>
               </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>
               <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>2015-07-23T13:40:03.849-07:00</EndDate>
            <User>omsadmin</User>
            <ActualDuration>P51DT5H16M15.849S</ActualDuration>
         </Item>
         <Links>
            <Link>
               <Source>1105</Source>
               <Target>1409</Target>
            </Link>
         </Links>
      </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 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>
         <Links/>
      </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>2015-06-02T08:24:04.000-07:00</EndDate>
      <User>omsadmin</User>
      <ActualDuration>PT21.000S</ActualDuration>
      <Scope>
         <Item xsi:type="ord:WorkItemType">
```

```
            <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>
         <Links/>
      </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>
         <Links>
            <Link>
               <Source>602</Source>
               <Target>704</Target>
            </Link>
         </Links>
      </Scope>
   </Item>
   <Links>
      <Link>
         <Source>2</Source>
         <Target>102</Target>
      </Link>
```

```
                  <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>
               </Links>
            </ProcessHistory>
         </ord:GetOrderProcessHistoryResponse>
   </env:Body>
</env:Envelope>
```

Example 2-35 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>
               <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>
            <Link>
               <Source>2</Source>
               <Target>102</Target>
            </Link>
            <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>
         </Links>
      </ProcessHistory>
   </ord:GetOrderProcessHistoryResponse>
</env:Body>
</env:Envelope>
```

Example 2-36 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>
                  <Links/>
               </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>
                        <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>P51DT23H55M19.207S</ActualDuration>
         </Item>
         <Links>
            <Link>
               <Source>1105</Source>
               <Target>1409</Target>
            </Link>
         </Links>
      </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 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>
         <Links/>
      </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>2015-06-02T08:24:04.000-07:00</EndDate>
      <User>omsadmin</User>
      <ActualDuration>PT21.000S</ActualDuration>
      <Scope>
         <Item xsi:type="ord:WorkItemType">
            <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>
         <Links/>
      </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>
         <Links>
            <Link>
               <Source>602</Source>
               <Target>704</Target>
            </Link>
         </Links>
      </Scope>
   </Item>
   <Links>
      <Link>
         <Source>2</Source>
         <Target>102</Target>
      </Link>
      <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>
            </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 2-37 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 2-38 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 2-39 GetOrderCompensationPlan

```
<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 2-40 GetOrderCompensationPlanResponse

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
   <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>
            <CompenationWorkItem>self</CompenationWorkItem>
            <FlowItemId>102</FlowItemId>
         </ActiveItem>
         <ActiveItem>
            <TaskName>CollectionsFunction_CollectionsSI</TaskName>
            <ExecutionMode>redo</ExecutionMode>
            <CompenationWorkItem>self</CompenationWorkItem>
            <FlowItemId>402</FlowItemId>
         </ActiveItem>
         <ActiveItem>
            <TaskName>configureBillingTask</TaskName>
            <ExecutionMode>redo</ExecutionMode>
            <CompenationWorkItem>self</CompenationWorkItem>
            <FlowItemId>1005</FlowItemId>
         </ActiveItem>
         <PendingItem>
            <TaskName>cdiTask</TaskName>
            <ExecutionMode>redo</ExecutionMode>
            <CompenationWorkItem>self</CompenationWorkItem>
            <FlowItemId>1508</FlowItemId>
            <WaitsFor>
               <TaskName>SyncCustomerFunction_CustomerSystemSI</TaskName>
               <ExecutionMode>redo</ExecutionMode>
               <PositionedInFlow>before</PositionedInFlow>
               <CompenationWorkItem>self</CompenationWorkItem>
               <FlowItemId>1509</FlowItemId>
            </WaitsFor>
         </PendingItem>
         <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>
       </ord:GetCompensationPlanResponse>
    </env:Body>
 </env:Envelope>
```

3
Using the OSM XML API

This chapter provides overview information about the Oracle Communications Order and Service Management (OSM) Extensible Markup Language (XML) Application Programming Interface (API). It is assumed that the programmer has user-level knowledge of Windows and UNIX operating systems.

About Using the XML API

The OSM XML API enables you to:

- **Create, retrieve, and update orders.**
- **Transition orders**: Move an order through the various states and tasks of a process. This includes moving an order to a new process through exception processing.
- **Search for the next order at task**: Search for the next available order at a given task.
- **Copy orders**: Copy the data of an existing order to produce a new order.
- **Suspending and resuming orders**: Temporarily halt all provisioning activity on an order and then release the suspended order back into the system for provisioning.
- **Add attachments and remarks**: Add remarks along with optional attachments to orders. After you add the remark, it is stamped with the time, task, and state of the order at that time.
- **Query**: Retrieve a list of orders through two query functions: a predefined query that lists orders of interest to an external agent, and a generalized query that provides external agents a means to define their own query criteria.
- **Retrieve the order data and process history**: Retrieve the history of an order as it moves through the process.
- **Retrieve user information**: Retrieve the user's name and description, as well as the name and description of each workgroup to which they belong.

You can use the XML API for the following purposes:

- Customizing the appearance or functioning of a task when customization using behaviors or OSM Java server pages does not satisfy all of your requirements.
- Using from within an automation plug-in when necessary because the Web Services API and the OSM automation functionality do not meet your requirements.

  You can use the XML API functions from the Automation Framework when running automation plug-ins by using the OSM Java OrderContext class processXMLRequest method. Parts of XML API (mainly GetOrder.Response) appear in various places as a context document throughout the OSM model. For example, when an automated task transitions into the received state, the automation framework starts an automation plug-in associated with it and passes

the plug-in the TaskContext object. You can access the data associated to that TaskContext object using the GetOrder.Response XML API function call.

The XML API is deprecated for the following uses:

- External automation (for example, polling). Use the event-driven Automation Framework for this purpose.
- Integration with an upstream system. Use the Web Service API for this purpose.
- Task automation when the equivalent functionality exists in the Automation Framework. See "Localizing OSM" and the OSM Javadocs for more information about automation plug-ins.
- Processing manual tasks and order submission when the equivalent functionality exists within the OSM Task web client and the OSM Web Services. See *OSM Task Web Client User's Guide* and "Using OSM Order Management Web Services" for more information.

The following operations are deprecated for all uses, and are provided for backward compatibility only, since they are not in accord with current OSM direction:

- SetException.Request
- ListException.Request
- GetNextOrderAtTask.Request
- AddOrderThread.Request

Audience

This chapter is designed for developers familiar with XML 1.0 DOM level-1 and the HTTP transport mechanism for delivery of XML messages. You must ensure that an XML parser and DOM implementation is available on your platform.

About Using the OrderID, View, and OrderHistID

OSM assigns all internally processing orders an order ID (**OrderID**). You can use the OrderID to indicate which order you want to run the function against. You specify the data available to many of the XML API function calls with the **View** and **OrderHistID** parameters.

You configure the data available to a View when you create an order specification in Design Studio. In the order specification editor **Permissions** tab, **Query Task** subtab, you can select a default Query Task, which is a manual task that is not part of a process flow. All data that you define in the manual task editor **Task Data** tab represents the data available to XML API functions that use the View field.

OSM generates a **OrderHistID** every time an OSM task transitions from one state to another of performs a status transition from one task to another. You configure the data available to an OrderHistID when you create a manual or automated task in Design Studio. All data that you define in the manual or automated task editor Task Data tab represents the data available to the XML API functions that use the OrderHistID field.

Oracle recommends that you only use the XML API functions that reference View or OrderHistID in the limited way described in "About Using the XML API."

About Accessing the XML API

The OSM XML API is a programmatic interface for sending HTTP POST requests and receiving HTTP responses.

See the **SDK/XMLSchema/oms-xmlapi.xsd** schema for additional information about the OSM XML API.

The OSM XML API provides a single access point for API requests.

The SDK contains a sample HTML file (**SDK/Samples/xmlapi/testxmlapi.html**) that you can modify and use to send XML API requests via a web browser. This file is intended to be used only for quick testing and similar exploratory usage. It does not represent a proper integration with an external component and is not suitable for production environments. See the **SDK/Samples/xmlapi/README.txt** file for details about using the sample HTML file.

Making the HTTP connection for Traditional OSM Deployments

See the **SDK/Samples/xmlapi/README.txt** file for details about making the HTTP connection. See *OSM Installation Guide* for more information about HTTP and HTTPS hardware and software load balancing options for OSM WebLogic clusters.

Making the HTTP connection for OSM Cloud Native Deployments

See the **SDK/Samples/xmlapi/README.txt** file for details about making the HTTP connection. See "Chapter 2 Planning and Validating Your Cloud Environment" in *OSM Cloud Native Deployment Guide* for details about hostname resolution.

Logging In and Logging Out

Before using any API messages, you must login by supplying a valid user ID and password. If the login is successful, the following message is displayed:

```
Login for admin successful.
Oracle Order and Service Management - Version 7.4.x.y.z
```

For details about logging in and logging out of the OSM XML API, see the **SDK/Samples/xmlapi/README.txt** file.

Message Formats

OSM messages follow a simple format that allows for arbitrary data and metadata to be passed across a network.

Input XML Message Format

Each operation that can be requested of the XML API defines its own root element for an XML document with the format `command_name.Request`. Any required parameters are child elements of the operation request.

Example 3-1 Input XML Message Format

```
<command_name.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <parameter1 />
```

```
   <parameter2 />
   ... additional parameters ...
</command_name.Request>
```

Output XML Message Format

For each request operation, there is a corresponding response document with a root element in the form `command_name.Response`. Any returned data is a child element of the operation response.

If non-critical errors occur during processing of an operation request, they are children of a Warnings element, as shown in Example 3-2.

Example 3-2 Output XML Message Format With Warnings

```
<command_name.Response xmlns="urn:com:metasolv:oms:xmlapi:1">
   <result1 />
   <result2 />
   ... additional result elements ...
   <Warnings>
      <Warning code="1052" desc="SQL Warning">message</Warning>
   </Warnings>
</command_name.Response>
```

If errors occur that prevent a request from being processed, the XML API returns an error document with a root element of `command_name.Error`. The error(s) that occurred are children of the error document.

You must monitor returning response messages from OSM for any errors that indicate whether the request operation succeeded or failed. If the operation request fails it is the responsibility of the sender to track and resubmit the failed request after troubleshooting the problem.

Example 3-3 Output XML Message Format With Errors

```
<command_name.Error xmlns="urn:com:metasolv:oms:xmlapi:1">
   <Error code="300" desc="Request parameter error">message</Error>
</command_name.Error>
```

If an invalid document is received, the server returns the HTTP code 403: Forbidden.

When you create XML, element values must not contain the characters &, <, or >. The XML standard defines the following replacement text:

- & - &
- < - <
- > - >
- ' - '
- " - "

Date/Time Formats

The date/time format is the same for input and output messages. For any parameter with a date/time value, the format is:

yyyy-MM-dd`THH:`*mm*`:`*ss timezone*

where

- *yyyy* is the four-digit year
- *MM* is the two-digit month
- *dd* is the two digit day of the month
- *HH* is the two digit hours in 24-hour format
- *mm* is the two-digit minutes
- *ss* is the two-digit seconds
- *timezone* is a three-letter designation of the time zone

For example:

```
2013-08-05T14:06:05 EDT
```

White Space in Message Text

OSM keeps the white space to the right of the beginning of a text block and to the left of the end of a text block. For example, if you create or update an order with the following field:

```
<street>   190   Attwell Drive  <street>
```

OSM retains the white space.

Authentication

All requests are processed in the context of the privileges assigned to a user ID. Prior to using the messages of the XML API, a user ID must be authorized. An authorization servlet, based on a user ID and password, authenticates a user ID and provides a session ID HTTP cookie to be used by subsequent requests. If further security is required, the XML API can be deployed in an environment that supports HTTPS for secure transport.

The OSM XML API does not provide access to the administrative facilities of OSM. Before using this API, you must use the OSM Administrator to configure a user ID that establishes the security privileges for the external software. This determines the range of data that can be retrieved from OSM.

Reserved Mnemonics

The mnemonics in Table 3-1 are reserved and used to reference special systems values. If an order data element is created with the same mnemonic as a reserved mnemonic, the system functions correctly. The Worklist and Query response lists two elements with the same element name - one for the system value and one for the data element value.

Table 3-1 Reserved Mnemonics

Header	Mnemonic
Order Sequence ID	_order_seq_id
Order History Sequence ID	_order_hist_seq_id

Table 3-1 (Cont.) Reserved Mnemonics

Header	Mnemonic
State	_order_state
Execution Mode	_execution_mode
Task Mnemonic	_task_id
Order Source Mnemonic	_order_source
Order Type Mnemonic	_order_type
Order State	_current_order_state
Target Order State	_target_order_state
Reference Number	_reference_number
Priority	_priority
User	_user
Process Description	_process_description
Order Status	_process_status
Date Created	_date_pos_created
Date Started	_date_pos_started
Root node	_root
Number of Remarks	_num_remarks
Expected Order Completion Date	_compl_date_expected
NotificationID	_notif_id
Notification Description	_notif_desc
Notification Type	_notif_type
Notification Priority	_notif_priority
Notification Timestamp	_notif_time
Namespace	Namespace_namespace
Version	Version_version

XML API Functionality

The following list contains all currently available requests and their responses.

AddOrderThread

AddOrderThread lets you add sub process threads to a pending order. The order must reside in one of the sub processes.

> **Note:**
>
> AddOrderThread has been deprecated and is supported only for backward compatibility. Use amendment processing functionality instead.

Operation

AddOrderThread

Parameters

OrderID: The Order ID

Process: The process mnemonic to indicate where the sub process task resides

ProcessPosition: The process position mnemonic of the sub process task.

Add: A list of nodes to be added. The format of this should follow UpdateOrder. Request format. You may add multiple instances of the pivot nodes, multiple threads are created as a result.

Request Example with One Pivot Node Value

```
<AddOrderThread.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>1234</OrderID>
   <Process>process_mnemonic</Process>
   <ProcessPosition>process_position_mnemonic</ProcessPosition>
   <Add path="/client_info">
      <address>
         <street1>55 James St.</street1>
         <city>Washington</city>
         <state>DC</state>
         <country>USA</country>
         <zip>20002</zip>
      </address>
   </Add>
</AddOrderThread.Request>
```

In this case, `client_info` is the pivot node.

Response

```
<AddOrderThread.Response xmlns="urn:com:metasolv:oms:xmlapi:1"/>
```

Request Example with Multiple Pivot Nodes

```
<AddOrderThread.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>1234</OrderID>
   <Process>process_mnemonic</Process>
   <ProcessPosition>process_position_mnemonic</ProcessPosition >
   <Add path="/error">
      <code>1000</code>
      <code>2000</code>
   </Add>
</AddOrderThread.Request>
```

In this case, `/error/code` is the pivot node.

Response

```
<AddOrderThread.Response xmlns="urn:com:metasolv:oms:xmlapi:1"/>
```

Error Codes

200: Order data invalid

232: Order update failed

270: Transaction not allowed

302: Request parameter error

350: Pivot node data is not provided

351: Process position supplied is not a sub process task

352: No sub process task is currently pending

354: Process position not found

355: Pivot node not found

356: Cannot spawn threads for sub-process tasks that support sequential sub-processing

400: Not authorized

500: Internal error

> **Note:**
>
> See Table 3-3 for more information if you receive an error code that is not listed here.

Acknowledgments

A list of retrievable acknowledgments for a given notification.

Operation

Acknowledgments

Parameters

The request requires one of two possible parameters:

Notification: If the notification is supplied, all acknowledgments for that notification are returned.

Order ID: If the order ID is supplied, all acknowledgments for all notifications for that order ID are returned.

Namespace: The namespace mnemonic of order type/source.

Version: The version of the order type or source. If you do not indicate a version, OSM uses the default version.

The Notification and NotificationDescription elements are identical for all acknowledgments. If the request was for a Notification, the information is duplicated to keep consistency of the Acknowledgement element's content.

Request Example

```
<Acknowledgements.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <Notification>send_order_creation</Notification>
   <Namespace>DSL_Highspeedline</Namespace>
   <Version>1.1</Version>
</Acknowledgements.Request>
```

The response includes the NotificationID or OrderID supplied and zero or more Acknowledgement elements. Each Acknowledgement element includes the following:

Notification ID: The notification ID associated with this acknowledgment.

NotificationDescription: The description of the notification. If no description, it is left blank.

OrderHistID: The order history ID associated with the acknowledgment. If this is not a transition based notification, then OrderHistID is empty.

Time: The time the acknowledgment was created.

Author: The user ID who created the acknowledgment.

Comment: The comment included with the acknowledgment. If no comment is supplied, it is empty.

Action: A string with a value of one of:

- Activate: the acknowledgment was created when the notification was activated.
- Update: the acknowledgment was added to the notification.
- Deactivate: the acknowledgment deactivated the notification.

Response Example

```
<Acknowledgements.Response xmlns="urn:com:metasolv:oms:xmlapi:1">
   <NotificationID>3244</NotificationID>
   <Acknowledgement>
      <NotificationID>3244</NotificationID>
      <NotificationDescription>Poll every hour</NotificationDescription>
      <OrderHistID/>
      <Time>2000-10-30T14:44:33 EST</Time>
      <Author>OMS_160</Author>
      <Comment/>
      <Action>activate</Action>
   </Acknowledgement>
   <Acknowledgement>
      <NotificationID>3244</NotificationID>
      <NotificationDescription>Poll every hour</NotificationDescription>
      <OrderHistID/>
      <Time>2000-10-30T15:01:22 EST</Time>
      <Author>jdoe</Author>
      <Comment>Activated switch</Comment>
      <Action>deactivate</Action>
   </Acknowledgement>
</Acknowledgements.Response>
```

Error Codes

- 110: Order not found

- 190: Notification not found
- 302: Request parameter error
- 500: Internal error

> **Note:**
>
> See Table 3-3 for more information if you receive an error code that is not listed here.

AcknowledgeNotification

Adds an acknowledgment to a notification. A notification can be acknowledged any number of times until is deactivated. Once an acknowledgment with a request to deactivate the notification is received, the notification is no longer included in the list of notifications.

Operation

AcknowledgeNotification

Parameters

NotificationID: The unique identification number of the notification.

OrderID: The order ID associated with the notification. Omitted for system-based notifications.

OrderHistID: The order history ID associated with the notification. Omitted or polled notifications.

Comment: A string description to include with the acknowledgment.

Deactivate: A "true" or "false" value to deactivate the notification.

Request Example

```
<AcknowledgeNotification.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <NotificationID>3444</NotificationID>
   <OrderID>123</OrderID>
   <OrderHistID>5665</OrderHistID>
   <Comment>This is a string comment</Comment>
   <Deactivate>true</Deactivate>
</AcknowledgeNotification.Request>
```

Response Example

```
<AcknowledgeNotification.Response
xmlns="urn:com:metasolv:oms:xmlapi:1" />
```

Error Codes

- 190: Notification not found
- 302: Request Parameter Error
- 500: Internal error

> **Note:**
>
> See Table 3-3 for more information if you receive an error code that is not listed here.

AssignOrder

Assigns an order to a given user.

Operation

AssignOrder

Parameters

OrderID: The ID of the order to change.

OrderHistID: The order history ID.

User: The user ID to assign to the order.

Request Example

```
<AssignOrder.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>1234</OrderID>
   <OrderHistID>22334</OrderHistID>
   <User>jsmith</User>
</AssignOrder.Request>
```

The AssignOrder response includes the new Order History ID for the order.

Response Example

```
<AssignOrder.Response xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>1234</OrderID>
   <OrderHistID>33247</OrderHistID>
</AssignOrder.Response>
```

Error Codes

- 110: Order not found
- 251: Transition invalid
- 253: User not found
- 270: Transaction not allowed
- 302: Request parameter error
- 400: Not authorized
- 401: Database Connection Failed
- 500: Internal error

> **Note:**
>
> See Table 3-3 for more information if you receive an error code that is not listed here.

CancelOrder

Cancels an order as described below depending on the parameters supplied in the request:

- Cancels the pending tasks for an order and sets an exception status regardless of where it currently is in the process flow. All pending tasks are removed and the order goes to the location defined by the exception status - either a particular task, or stopped.

  Or

- Cancels an order by undoing all completed tasks and returning the order to the creation task.

Parameters

To cancel and set an exception:

OrderID: The ID of the order to cancel

Status: The exception status mnemonic to set

To cancel and undo completed tasks:

OrderID: The ID of the order to cancel

And one of the following:

Immediate: Force immediate cancellation of all completed tasks in the order.

GracePeriodExpiryDate: A period of time to allow tasks in the Accepted state time to complete.

Infinite: Wait indefinitely until all tasks in the Accepted state complete.

Optional parameters

EventInterval: If the cancellation is not immediate, you can set an interval for sending a jeopardy notification.

Reason: The reason for canceling the order.

Request Example 1: Cancel and Set Exception

```
<CancelOrder.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>1234</OrderID>
   <Status>status_memonic</Status>
</CancelOrder.Request>
```

Request Example 2: Cancel and Undo

```
<CancelOrder.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>1234</OrderID>
```

```
   <Immediate/>
   <Reason>Customer relocating services</Reason>
</CancelOrder.Request>
```

or

```
<CancelOrder.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>1234</OrderID>
   <GracePeriodExpiryDate>2006-10-10T11:10:10 EST</GracePeriodExpiryDate>
   <EventInterval>PT10S</EventInterval>
   <Reason>Customer relocating services</Reason>
</CancelOrder.Request>
```

or

```
<CancelOrder.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>1234</OrderID>
   <Infinite/>
   <Reason>Customer relocating services</Reason>
</CancelOrder.Request>
```

Response Example 1: Cancel and Set Exception

If the status mnemonic resulted in the order moving to a task, the OrderHistID has a value. If the order is stopped, then OrderHistID is empty.

```
<CancelOrder.Response xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>1234</OrderID>
   <OrderHistID/>
</CancelOrder.Response>
```

Example 2: Cancel and Undo

```
<CancelOrder.Response xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>1234</OrderID>
</CancelOrder.Response>
```

Error Codes

- 110: Order Not Found
- 255: Invalid Status Mnemonic
- 270: Transaction not allowed
- 302: Request Parameter Error
- 400: Not Authorized

> **Note:**
>
> See Table 3-3 for more information if you receive an error code that is not listed here.

CompleteOrder

Completes an order and supplies a status mnemonic for the order.

Operation

CompleteOrder

Parameters

OrderID: The ID of the order to change.

OrderHistID: The order history ID.

Status: The status mnemonic.

Request Example

```
<CompleteOrder.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>1234</OrderID>
   <OrderHistID>33251</OrderHistID>
   <Status>submit</Status>
</CompleteOrder.Request>
```

Response Example

```
<CompleteOrder.Response xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>1234</OrderID>
   <OrderHistID/>
</CompleteOrder.Response>
```

Response Example for Amendment

```
<CompleteOrder.Response xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>7</OrderID>
   <OrderHistID/>
   <Amendment xmlns="um:com:metasolv:oms:xmlapi:1">
      <matchedOrderID>6</matchedOrderID>
      <Status>accepted</Status>
   </Amendment>
</CompleteOrder.Response>
```

Error Codes

- 110: Order not found
- 251: Transition invalid
- 255: Status mnemonic invalid
- 270: Transaction not allowed
- 302: Request parameter error
- 401: Database Connection Failed
- 500: Internal error

> **Note:**
>
> See Table 3-3 for more information if you receive an error code that is not listed here.

CopyOrder

You can create a new order populated with the data from an existing order. The old order is retrieved using the order creation template associated with the type and source of the new order. The existing order data that is visible in the new order creation template is inserted into the new order. Any data from the old order that does not map to the new order's creation template is not inserted into the new order.

Operation

CopyOrder

Parameters

OriginalOrderID: The order ID of the existing order to copy.

OrderType: The order type mnemonic for the new order.

OrderSource: The order source mnemonic for the new order.

Reference: The reference number for the new order.

Priority: An integer of 0-9 indicating the priority level of the order. 5 is the default priority.

Request Example

```
<CopyOrder.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OriginalOrderID>1235</OriginalOrderID>
   <OrderSource>source1</OrderSource>
   <OrderType>phone_transfer</OrderType>
   <Reference>AA-NEW-345</Reference>
   <Priority>5</Priority>
   <Namespace>DSL_Highspeedline</Namespace>
   <Version>1.1</Version>
</CopyOrder.Request>
```

The content of the response is the same as CreateOrder, except that CopyOrder.Response is the top level element.

Response Example

```
<CopyOrder.Response xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>1236</OrderID>
   <OrderHistID>23334</OrderHistID>
   <OrderSource>source1</OrderSource>
   <OrderType>phone_transfer</OrderType>
   <OrderState>open.not_running.not_started</OrderState>
   <State>received</State>
   <Reference>AA-NEW-345</Reference>
   <Priority>5</Priority>
   <Namespace>DSL_Highspeedline</Namespace>
   <Version>1.1</Version>
   <CopyRemarks>false</CopyRemarks>
</CopyOrder.Response>
```

Error Codes

- 150: Namespace/version not found.

- 152: Invalid namespace mnemonic.
- 153: No legacy data found. Namespace and Version need to be supplied.

> **Note:**
>
> See Table 3-3 for more information if you receive an error code that is not listed here.

Namespace: The namespace mnemonic of order type/source.

Version: The version of the order type or source.

Scenarios

Scenario 1: Namespace is supplied in the request, but no version. This condition forces the system to use the default version for the supplied namespace.

Scenario 2: Version is supplied in the request, but no namespace. This generates an error (152) message.

Scenario 3: The first cartridge is reserved for the migrated data. If namespace and version are not supplied, then this cartridge is used. If this legacy cartridge does not exist, an error (153) message is shown.

Scenario 4: If the combination of namespace and version does not exist, an error (150) message is shown.

CreateOrder

To create an order you must provide an order type and order source with a CreateOrder operation. The initial data for the order is provided based on the same structure as the order template. The root element of the order has the XML name of _root.

Operation

CreateOrder

Parameters

ParentOrderID: The parent order.

OrderType: The order type mnemonic.

OrderSource: The order source mnemonic.

View: The view (query task) assigned to the order.

Reference: A reference ID string.

Priority: An integer of 0-9 indicating the priority level of the order. 5 is the default priority.

Namespace: The namespace mnemonic of order type/source.

Version: The version of the order type or source.

_root: The root element of the order document.

Optional Parameters

AddMandatory - If *true*:

- If you delete a mandatory node, AddMandatory replaces the node and populates it with the default value.
- If the request 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.

- Order header element.
- If not explicitly set, defaults to 5.
- If the priority specified is above the maximum or below the minimum priority value for the order type/source, it is automatically rounded up or down accordingly.

> **Note:**
>
> If the priority is set outside the range of allowable priority values for the system (0-9) or is set to a non-numeric value, an error is thrown.

- CreateOrder response always returns the priority regardless of whether it is set or not.

Request Example

```
<CreateOrder.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderSource>source1</OrderSource>
   <OrderType>phone_transfer</OrderType>
   <Reference>3ab34</Reference>
   <Priority>5</Priority>
   <Namespace>DSL_Highspeedline</Namespace>
   <Version>1.1</Version>
   <_root>
      <client_info>
         <name>John Doe</name>
         <address>
            <street1>1211 Lakeview Dr.</street1>
            <city>New York</city>
            <state>NY</state>
            <country>USA</country>
            <zip>12345</zip>
         </address>
      <client_info>
      ... more data ...
   </_root>
```

```
   <AddMandatory>true</AddMandatory>
</CreateOrder.Request>
```

Response Example

The response includes the new order ID number, the initial state, the order source and type,

and the reference provided with the request. The response always returns a priority value whether it is set or not (defaults to 5).

```
<CreateOrder.Response xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>1234</OrderID>
   <OrderHistID>678</OrderHistID>
   <OrderSource>phone_transfer</OrderSource>
   <OrderType>source1</OrderType>
   <OrderState>open.not_running.not_started</OrderState>
   <State>accepted</State>
   <Reference>3ab34</Reference>
   <Priority>5</Priority>
   <Namespace>DSL_Highspeedline</Namespace>
   <Version>1.1</Version>
</CreateOrder.Response>
```

Error Codes

150: Namespace/version not found.

152: Invalid namespace mnemonic.

153: No legacy data found. Namespace and Version need to be supplied.

200: Order data invalid

> **Note:**
>
> See Table 3-3 for more information if you receive an error code that is not listed here.

FalloutTask

Initiates fallout from a particular task. This request requires an Order ID, Order History ID, and Fallout mnemonic.

Operation

FalloutTask

Parameters

OrderID: The ID of the order to change.

OrderHistID: The order history ID.

Fallout: The fallout mnemonic as defined in the metadata.

Reason: The reason for the fallout. This parameter is optional.

Request Example

```
<FalloutTask.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>501</OrderID>
   <OrderHistID>3010</OrderHistID>
   <Fallout>fallout_switch</Fallout>
   <Reason>Bad switch</Reason>
</FalloutTask.Request>
```

Response Example

The system returns the response with an accepted status to indicate the fallout has been accepted for processing.

```
<FalloutTask.Response xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>501</OrderID>
   <Status>accepted</Status>
</FalloutTask.Response>
```

Error Codes

- 110: Order not found
- 270: Transaction not allowed
- 302: Request parameter error
- 400: Not authorized
- 401: Database connection failed
- 419: The process exception is restricted?
- 439: Invalid fallout mnemonic
- 500: Internal ErrorSuspendOrder

> **Note:**
>
> See Table 3-3 for more information if you receive an error code that is not listed here.

FailOrder

The request either fails the order, or fails the current task. OSM fails the task when OrderHistID is provided in the request otherwise OSM fails the order. One request cannot be used to fail both task and order at the same time.

Operation

FailOrder

Parameters

OrderID: The ID of the order to fail.

OrderHistID: The order history ID.

Reason: The reason for the failure. This parameter is optional.

And one of the following:

Immediate: Force immediate failure of order or task.

Infinite: Wait indefinitely until all tasks on the order complete or become available.

Optional parameters

EventInterval: An event will be generated periodically while the order remains in grace period. This event acts as a warning to external systems than an order is in grace period and awaiting completion of accepted work items. This value controls the frequency that the event will be generated.

GracePeriodExpiry: A point in time, after which the grace period for completing accepted work items expires. After this time, the order will be transitioned regardless of whether or not there are outstanding work items. The grace period expiry date specified here must be within the limits imposed by the grace period expiry range specified in the order state policy.

Request Example

```
<FailOrder.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>501</OrderID>
   <OrderHistID>3010</OrderHistID>
   <Reason>BadData</Reason>
</FailOrder.Request>
```

Response Example

The system returns the response with an accepted status to indicate the fallout has been accepted for processing.

```
<FailOrder.Response xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>501</OrderID>
   <OrderHistID>3010</OrderHistID>
</FailOrder.Response>
```

Error Codes

- 110: Order not found
- 270: Transaction not allowed
- 302: Request parameter error
- 400: Not authorized
- 401: Database connection failed
- 419: The process exception is restricted?
- 439: Invalid fallout mnemonic
- 500: Internal ErrorSuspendOrder

> **Note:**
>
> See Table 3-3 for more information if you receive an error code that is not listed here.

GetNextOrderAtTask

Allows agents to retrieve an order at a given task and at a specific state. The order returned by GetNextOrderAtTask is the first order found in the OSM database that matches the request criteria. At least one state should be present in GetNextOrderAtTask.Request. Until an order is moved to a task or a state that does not match the request criteria, it remains to be returned by subsequent calls to GetNextOrderAtTask.

Operation

GetNextOrderAtTask

Parameters

OrderID: If specified, retrieves the next instance of a task on the specific order. This is an optional parameter.

Task: The mnemonic for the task.

ExecutionMode: If specified, value may be one of "do", "redo", or "undo". Retrieves the next instance of a task with the given execution mode. This is an optional parameter.

Accept: A value of "true" or "false" indicating if the XML API should accept the order for the user's ID.

State: A state for the task. This element can have multiple instances and the values indicate which states a task must be in for the order to be returned. Acceptable values are:

- **Assigned**: The task is in the Assigned state and is assigned to the current user's ID.
- **Received**: The task is in the Received state.
- **Accepted**: The task is in the Accepted state for the current user's ID.
- **Suspended**: The task is in the Suspended state.

Namespace: The namespace mnemonic of the order type/source.

Version: The version of the order type or source.

Request Example

```
<GetNextOrderAtTask.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>1234</OrderID>
   <Task>provision_switch</Task>
   <ExecutionMode>redo</ExecutionMode>
   <Accept>false</Accept>
   <State>received</State>
   <State>assigned</State>
   <State>accepted</State>
   <Namespace>DSL_Highspeedline</Namespace>
   <Version>1.1</Version>
</GetNextOrderAtTask.Request>
```

If an order matching the request criteria is found, the response has the same content as GetOrder.Response, except the top-level element is

GetNextOrderAtTask.Response. If no matching order is found, the response consists of the top level GetNextOrderAtTask.Response with no child elements.

Request Example

```
<GetNextOrderAtTask.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>1234</OrderID>
   <Task>provision_switch</Task>
   <ExecutionMode>redo</ExecutionMode>
   <Accept>false</Accept>
   <State>received</State>
   <State>assigned</State>
   <State>accepted</State>
   <Namespace>DSL_Highspeedline</Namespace>
   <Version>1.1</Version>
</GetNextOrderAtTask.Request>
```

If an order matching the request criteria is found, the response has the same content as GetOrder.Response, except the top-level element is GetNextOrderAtTask.Response. If no matching order is found, the response consists of the top level GetNextOrderAtTask.Response with no child elements.

Response Example

```
<GetNextOrderAtTask.Response xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>1234</OrderID>
   <OrderHistID>2333</OrderHistID>
   <Task>provision_switch</Task>
   <OrderSource>source1</OrderSource>
   <OrderType>phone_transfer</OrderType>
   <Workgroups>
      <Workgroup>workgroup1</Workgroup>
      <Workgroup>workgroup2</Workgroup>
   </Workgroups>
   <OrderState>open.running.in_progress</OrderState>
   <State>received</State>
   <ExecutionMode>do</ExecutionMode>
   <Reference>3ab34</Reference>
   <Priority>5</Priority>
   <Namespace>DSL_Highspeedline</Namespace>
   <Version>1.1</Version>
   <_root index="0">
      <client_info index="76578">
         <name index="76579">John Doe</name>
         <address index="76580">
            <street1 index="76581">1211 Lakeview Dr.</Street1>
            <city index="76582">New York</city>
            <state index="76583">NY</state>
            <country index="76584">USA</country>
            <zip index="76585">12345</zip>
         </address>
         <address index="80132">
            <street1 index="80133">20 Biz drv.</street1>
            <city index="80134">New York</city>
            <state index="80135">NY</state>
            <country index="80136">USA</country>
            <zip index="80137">12345</zip>
         </address>
      </client_info>
      ... more order data ...
   </_root>
```

```
   <HistoricalPerspective>
      <OrderHistID>52</OrderHistID>
      <_root index="0">
         <customer_ref index="1146675411084">Cust01</customer_ref>
         <shape index="1146675411085">circle</shape>
         <color index="1146675411086">blue</color>
         <pattern index="1146675411087">checkerboard</pattern>
      </_root>
      <Changes>
            <Update path="/color[@index='1146675411086']"
             oldValue="blue">green</Update>
      </Changes>
   </HistoricalPerspective>
</GetNextOrderAtTask.Response>
```

Error Codes

- 150: Namespace/version not found.
- 152: Invalid namespace mnemonic.
- 153: No legacy data found. Namespace and Version need to be supplied.
- 270: Transaction not allowed.

> **Note:**
>
> See Table 3-3 for more information if you receive an error code that is not listed here.

Scenarios

Scenario 1: Namespace is supplied in the request, but no version. This condition forces the system to use the default version for the supplied namespace.

Scenario 2: Version is supplied in the request, but no namespace. This generates an error (152) message.

Scenario 3: The first cartridge is reserved for the migrated data. If namespace and version are not supplied, then this cartridge is used. If this legacy cartridge does not exist, an error (153) message is shown.

Scenario 4: If the combination of namespace and version does not exist, an error (150) message is shown.

GetOrder

To retrieve the order data, you must provide the relevant IDs obtained from the worklist or another external source. If successful, the response includes the order data values.

To retrieve an order, you must provide the order ID and an Accept parameter indicating whether the order is updated. You may also provide an order history ID or view (query task) ID.

Operation

GetOrder

Parameters

OrderID: The ID of the order.

Accept: Determines whether an attempt was made to move the order to an Accepted state.

And one of the following:

OrderHistID: The history ID of the order.

> **Note:**
>
> You should only use contemporary, current, or historical data that you retrieve using the OrderHistID from a currently received or accepted task. Contemporary, current, or historical data retrieved using OrderHIstID for a task that is already complete may no longer be valid.

ViewID: The particular view (query task) associated with the order. You can obtain a list of valid ViewIDs for an order with the ListViews.Request.

OrderChangeId: Determines from which revision the historical and current OCM (Order Change Management) perspectives are to be constructed.

TaskExecutionHistory: Determines the execution history of a revision on a task. It contains details of the execution mode in which the task was executed in the revision, the OrderHistoryID of the task during the revision, and the OrderChangeID of the corresponding revision.

OrderDataFilter: Parent element for the **Condition** child element that specifies which order data to return in the GetOrder.Response.

- **Condition**: An XPath 1.0 expression against the order data defined by the view (query task). OSM returns only the instances of the order data selected by the expression, not the other instances of the element. All other parent or sibling elements are returned.

  For example, in a situation where a customer has multiple <address> 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 instance of the <address> element (city, postal code, and street). The other instances of the <address> element and their child elements (city, street, and postal code) are not returned.

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

  For example, any sibling elements of <subscriber_info>, or sibling elements of <address> (except for the other instances of the <address> element) would be returned.

  There can be as many **<Condition>** child elements as required. When there are more than one **<Condition**> elements, each condition is evaluated and applied independently of the other conditions to the sections of the order data respectively.

Request Example

```
<GetOrder.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>1234</OrderID>
   <Accept>true</Accept>
   <OrderHistID>34433</OrderHistID>
</GetOrder.Request>
```

When you are using an order condition that includes an element that is using a distributed order template, you should include the namespace of the data element in the condition. For example:

```
<OrderDataFilter>
   <Condition>
      /ControlData/OrderItem[@type='{OrderItemNamespace}OrderItemName' and
@LineId='1']
   </Condition>
</OrderDataFilter>
```

The order response includes the order source and type, the reference number, and its current state. If an order history ID is supplied in the request, the response returns the related task mnemonic. The order data is supplied under the _root element. The _root element and all _root child elements include index attributes to uniquely identify each element in any subsequent order updates.

The order response also shows the workgroup associated to the user who has accepted a task, suspended a task, or to whom a task has been assigned. When the task is in the received state, the response displays all workgroups that can possibly work on the task. When the task is in the accepted state, the response displays the user who accepted the task.

If there are remarks associated with the order, there is a Remarks element in the <Remarks> element. A Remark element has the following content:

RemarkID: A unique identifier for the remark.

Time: The time the remark was added.

Author: The user ID of the person who added the remark.

Text: The text of the remark.

TaskID: The task mnemonic of the order when the remark was added.

TaskType: The type of task. For example, creation, manual, automated, rule or delay.

OrderHistID: The order history ID for the order when the remark was created. If the remark was added without using an OrderHistID, the field is empty.

State: The state of the order when the remark was added.

ReadOnly: A "true" or "false" value indicating if the remark can still be modified. To modify, the current user must be the author of the remark and the remark cannot be older than a time specified by your administrator.

ProcessStatus: The process status of the order. Possible ProcessStatus values are taken from the reporting statuses defined in the Studio Process Editor, on the General subtab in the Properties window.

Attachments: A parent for the attachment information having zero or more Attachment elements.

Each attachment element contains:

AttachmentID: A unique identifier for the attachment.

FileName: The name of the file.

Response Example

```
<GetOrder.Response xmlns="urn:metasolv:oms:xmlapi_1">
   <OrderID>1234</OrderID>
   <OrderHistID>34433</OrderHistID>
   <Task>SampleTask1</OrderHistID>
   <OrderSource>source1</OrderSource>
   <OrderType>phone_transfer</OrderType>
   <Workgroups>
      <Workgroup>workgroup1</Workgroup>
      <Workgroup>workgroup2</Workgroup>
   </Workgroups>
   <OrderState>open.running.in_progress</OrderState>
   <State>received</State>
   <ExecutionMode>do</ExecutionMode>
   <Reference>3ab34</Reference>
   <ExpectedOrderCompletionDate>2009-03-10T00:00:00Z</
ExpectedOrderCompletionDate>
   <Priority>5</Priority>
   <ProcessStatus>n/a</ProcessStatus>
   <_rootindex="0">
      <client_info index="76577">
         <name index="76578">John Doe</name>
         <address index="76579">
            <street1 index="76580">1211 Lakeview Dr.</Street1>
            <city index="76581">New York</city>
            <state index="76582">NY</state>
            <country index="76583">USA</country>
            <zip index="76584">12345</zip>
         </address>
         <address index="80132">
            <street1 index="80133">20 Biz drv.</street1>
            <city index="80134">New York</city>
            <state index="80135">NY</state>
            <country index="80136">USA</country>
            <zip index="80137">12345</zip>
         </address>
      </client_info>
      ... more order data ...
   </_root>
   <Remarks>
      <Remark>
         <RemarkID>13444</RemarkID>
         <Date>2000-10-30T14:44:33 EST</Date>
         <Author>jdoe</Author>
         <TaskID>provision_switch</TaskID>
         <TaskType>manual</TaskType>
         <OrderHistID>34401</OrderHistID>
         <State>accepted</State>
         <Text>OSM completed</Text>
         <ReadOnly>true</ReadOnly>
         <Attachments>
```

```
            <Attachment>
               <AttachmentID>111324</AttachmentID>
               <FileName>provisioninfo.txt</FileName>
            </Attachment>
         </Attachments>
      </Remark>
      <Remark>
         <RemarkID>14322</RemarkID>
         <Date>2000-10-30T15:01:22 EST</Date>
         <Author>jdoe</Author>
         <TaskID>provision_switch</TaskID>
         <TaskType>manual</TaskType>
         <OrderHistID>34401</OrderHistID>
         <State>accepted</State>
         <Text>Switch activated</Text>
         <ReadOnly>false</ReadOnly>
         <Attachments/>
      </Remark>
   </Remarks>
</GetOrder.Response>
```

Request Example with OrderChangeId

This is an example of a GetOrder.Request in which the OrderChangeId is set to 123. This function sends a request to OSM to retrieve an order with OrderChangeId 123.

```
<GetOrder.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>1234</OrderID>
   <OrderHistID>34433</OrderHistID>
   <OrderChangeId>123</OrderChangeId>
</GetOrder.Request>
```

Response Example with OrderChangeId and TaskExecution History

This is an example of a GetOrder.Response in which the TaskExecutionHistory element contains details of order data revisions with corresponding OrderChangeIDs. The base order has an OrderChangeID of 0 and the revised OrderChangeId is indicated by 561.

```
<GetOrder.Response xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>2845</OrderID><OrderHistID>61926</OrderHistID>
   <Task>stask_a</Task>
   <State>received</State>
   <OrderChangeID>565</OrderChangeID>
   <OrderSource>OCMPerspectivesTestOrder</OrderSource>
   <Workgroups>
      <Workgroup>workgroup1</Workgroup>
      <Workgroup>workgroup2</Workgroup>
   </Workgroups>
   <OrderType>OCMPerspectivesTestOrder</OrderType>
   <OrderState>open.running.compensating.amending</OrderState>
   <ExecutionMode>redo</ExecutionMode> <Reference/><Priority>5</Priority>
   <Namespace>OCMPerspectivesTest</Namespace><Version>1.0.0</Version>
   <_rootindex="0">
      <data index="1271706877188">rev2</data>
   </_root>
   <HistoricalPerspective>
      <OrderHistID>61917</OrderHistID>
      <_root index="0">
         <data index="1271706877188">rev1</data>
      </_root>
```

```
      <Changes>
         <Update significant="true" path="/data[@index='1271706877188']
          "oldValue="rev1">rev2</Update>
      </Changes>
   </HistoricalPerspective>
   <CurrentPerspective>
      <_root index="0">
         <data index="1271706877188">rev2</data>
      </_root>
   </CurrentPerspective>
   <TaskExecutionHis tory>
      <Task>
         <OrderHistID>61917</OrderHistID>
         <ExecutionMode>redo</ExecutionMode>
         <OrderChangeID>561</OrderChangeID>
      </Task>
      <Task>
         <OrderHistID>61904</OrderHistID>
         <ExecutionMode>do</ExecutionMode>
         <OrderChangeID>0</OrderChangeID>
      </Task>
   </TaskExecutionHistory>
</GetOrder.Response>
```

Response Example with Distributed Order Template

This is a partial example of the response for an order in which the order items and their dynamic parameters properties are using the distributed order template.

```
<GetOrder.Response xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>15</OrderID>
   <OrderHistID>236809</OrderHistID>
   <Task>configureCustomerSystemTask</Task>
   <OrderSource>OsmCentralOMExampleOrder</OrderSource>
   <OrderType>OsmCentralOMExampleOrder</OrderType>
   <Workgroups>
      <Workgroup>workgroup1</Workgroup>
      <Workgroup>workgroup2</Workgroup>
   </Workgroups>
   <OrderState>open.running.in_progress</OrderState>
   <State>accepted</State>
   <ExecutionMode>do</ExecutionMode>
   <Reference>Order1397065147300</Reference>
   <RequestedDeliveryDate>2014-03-31T07:05:00 PDT</RequestedDeliveryDate>
   <ExpectedStartDate>2014-04-09T10:39:58 PDT</ExpectedStartDate>
   <ExpectedDuration>PT0S</ExpectedDuration>
   <ExpectedOrderCompletionDate>2014-04-09T10:39:58 PDT</
ExpectedOrderCompletionDate>
   <Priority>5</Priority>
   <Namespace>OsmCentralOMExample-Solution</Namespace>
   <Version>4.0.0.0.0</Version>
   <ProcessStatus>n/a</ProcessStatus>
   <_root index="0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <CustomerDetails index="11">
         <nameLocation index="13">Jangadeiros</nameLocation>
         <typeAddress index="23">Building</typeAddress>
      </CustomerDetails>
      <OrderHeader index="1">
         <numSalesOrder index="2">Order1397065147300</numSalesOrder>
         <typeOrder index="3">Add</typeOrder>
      </OrderHeader>
```

```
      <AccountDetails index="24">
         <numAccount index="25">TEL1234</numAccount>
         <status index="26">Existing</status>
         <corporate index="27">PoC</corporate>
         <category index="32">Corporate</category>
      </AccountDetails>
      <ControlData index="1397065199251">
         <Functions index="1397065199537">
         <SyncCustomerFunction index="1397065194936" instanceLocked="true">
               <componentKey index="1397065199567">
                  SyncCustomerFunction.CustomerSystem.WholeOrder</componentKey>
               <orderItem index="1397065199570">
                  <orderItemRef index="1397065199571"
referencedIndex="1397065194907" xmlns:ct234="http://oracle.osm.centralom"
xsi:type="ct234:CustomerOrderItemSpecificationType" type="{http://
oracle.osm.centralom}CustomerOrderItemSpecificationType">
                     <ct234:productSpec index="1397065199259"
xmlns:ct234="http://oracle.osm.centralom">Broadband Service Feature Class</
ct234:productSpec>
                     <ct234:filfillPatt index="1397065199260"
xmlns:ct234="http://oracle.osm.centralom">Service.Broadband</ct234:fulfillPatt>
                     <ct234:lineId index="1397065199254" xmlns:ct234="http://
oracle.osm.centralom">1</ct234:lineId>
                     <ct234:lineItemName index="1397065199263"
xmlns:ct234="http://oracle.osm.centralom">Brilliant Broadband [Add]</
ct234:lineItemName>
                     <ct234:requestedDeliveryDate index="1397065199257"
xmlns:ct234="http://oracle.osm.centralom">2014-03-31T07:05:00 PDT</
ct234:requestedDeliveryDate>
                     <ct234:lineItemPayload index="1397065199255"
xmlns:ct234="http://oracle.osm.centralom">
                        <im:salesOrderLine xmlns:im="http://xmlns.oracle.com/
InputMessage" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
                           <im:lineId>1</im:lineId>
                           <im:promotionalSalesOrderLineReference>1 </
im:promotionalSalesOrderLineReference>
                           <im:serviceId/>
                           <im:requestedDeliveryDate>
                              2014-03-31T07:05:00</im:requestedDeliveryDate>
                           <im:serviceActionCode>Add</im:serviceActionCode>
                           <im:serviceInstance>N</im:serviceInstance>
                           <im:serviceAddress>
                              <im:locationType>Street</im:locationType>
                              <im:typeAddress>Building</im:typeAddress>
                           </im:serviceAddress>
                           <im:itemReference>
                              <im:name>Brilliant Broadband</im:name>
                               <im:primaryClassificationCode>Broadband Service
Feature Class</im:primaryClassificationCode>
                              <im:specificationGroup/>
                           </im:itemReference>
                        </im:salesOrderLine>
                     </ct234:lineItemPayload>
                     <ct234:Recognition index="1397065199252"
xmlns:ct234="http://oracle.osm.centralom">Broadband Service Feature Class</
ct234:Recognition>
                     <ct234:dynamicParams
index="1397065199689" xmlns:ct234="http://oracle.osm.centralom"
xmlns:ct264="OracleComms_Model_Mobile/4.0.0.0.0"
xsi:type="ct264:SA_MobileMessagingCFSType" type="{OracleComms_Model_Mobile/
4.0.0.0.0}SA_MobileMessagingCFSType" xmlns:ct135="http://xmlns.oracle.com/
```

```
communications/studio/ordermanagement/transformation">
                        <ct264:MMSIncoming index="1397065199692"
xmlns:ct264="OracleComms_Model_Mobile/4.0.0.0.0">Yes</ct264:MMSIncoming>
                        <ct264:MMSOutgoing index="1397065199690"
xmlns:ct264="OracleComms_Model_Mobile/4.0.0.0.0">Yes</ct264:MMSOutgoing>
                    </ct234:dynamicParams>
                    <ct234:Action index="1397065199256" xmlns:ct234="http://
oracle.osm.centralom">Add</ct234:Action>
                    <ct234:ServiceInstance index="1397065199253"
xmlns:ct234="http://oracle.osm.centralom">N</ct234:ServiceInstance>
                </orderItemRef>
            </orderItem>

[...]
        </SyncCustomerFunction>
        </Functions>
        <OrderItem index="1397065194907" xmlns:ct234="http://
oracle.osm.centralom" xsi:type="ct234:CustomerOrderItemSpecificationType"
type="{http://oracle.osm.centralom}CustomerOrderItemSpecificationType">
            <ct234:productClass index="1397065199259" xmlns:ct234="http://
oracle.osm.centralom">Broadband Service Feature Class</ct234:productClass>
            <ct234:productSpec index="1397065199260" xmlns:ct234="http://
oracle.osm.centralom">Service.Broadband</ct234:productSpec>
            <ct234:lineId index="1397065199254" xmlns:ct234="http://
oracle.osm.centralom">1</ct234:lineId>
            <ct234:lineItemName index="1397065199263" xmlns:ct234="http://
oracle.osm.centralom">Brilliant Broadband [Add]</ct234:lineItemName>
            <ct234:requestedDeliveryDate index="1397065199257"
xmlns:ct234="http://oracle.osm.centralom">2014-03-31T07:05:00 PDT</
ct234:requestedDeliveryDate>
            <ct234:lineItemPayload index="1397065199255" xmlns:ct234="http://
oracle.osm.centralom">
               <im:salesOrderLine xmlns:im="http://xmlns.oracle.com/
InputMessage" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
                  <im:lineId>1</im:lineId>
                  <im:promotionalSalesOrderLineReference>1 </
im:promotionalSalesOrderLineReference>
                  <im:serviceId/>
                  <im:requestedDeliveryDate>
                     2014-03-31T07:05:00</im:requestedDeliveryDate>
                  <im:serviceActionCode>Add</im:serviceActionCode>
                  <im:serviceInstance>N</im:serviceInstance>
                  <im:serviceAddress>
                     <im:locationType>Street</im:locationType>
                     <im:typeAddress>Building</im:typeAddress>
                  </im:serviceAddress>
                  <im:itemReference>
                     <im:name>Brilliant Broadband</im:name>
                     <im:primaryClassificationCode>Broadband Service Feature
Class</im:primaryClassificationCode>
                     <im:specificationGroup/>
                  </im:itemReference>
               </im:salesOrderLine>
            </ct234:lineItemPayload>
            <ct234:Recognition index="1397065199252" xmlns:ct234="http://
oracle.osm.centralom">Broadband Service Feature Class</ct234:Recognition>
            <ct234:dynamicParams index="1397065199689" xmlns:ct234="http://
oracle.osm.centralom" xmlns:ct264="OracleComms_Model_Mobile/4.0.0.0.0"
xsi:type="ct264:SA_MobileMessagingCFSType" type="{OracleComms_Model_Mobile/
4.0.0.0.0}SA_MobileMessagingCFSType" xmlns:ct135="http://xmlns.oracle.com/
communications/studio/ordermanagement/transformation">
```

```
               <ct264:MMSIncoming index="1397065199692"
xmlns:ct264="OracleComms_Model_Mobile/4.0.0.0.0">Yes</ct264:MMSIncoming>
               <ct264:MMSOutgoing index="1397065199690"
xmlns:ct264="OracleComms_Model_Mobile/4.0.0.0.0">Yes</ct264:MMSOutgoing>
            </ct234:dynamicParams>
            <ct234:Action index="1397065199256" xmlns:ct234="http://
oracle.osm.centralom">Add</ct234:Action>
            <ct234:ServiceInstance index="1397065199253" xmlns:ct234="http://
oracle.osm.centralom">N</ct234:ServiceInstance>
         </OrderItem>
[...]
      </ControlData>
   </_root>
</GetOrder.Response>
```

Request and Response Example with OrderDataFilter

The following GetOrder.Request specifies the demo_query query task that specifies the data to return in the GetOrder.Response from an order with an order ID of 2.

```
<GetOrder.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>2</OrderID>
   <View>demo_query</View>
</GetOrder.Request>
```

The following response returned all the data specified by the demo_query query task including all address multi-instance nodes.

```
<GetOrder.Response>
   <OrderID>2</OrderID>
   <View>demo_query</View>
   <OrderSource>add_adsl_siebel</OrderSource>
   <OrderType>add_adsl_siebel</OrderType>
   <OrderState>open.running.in_progress</OrderState>
   <Reference>1112223333</Reference>
   <Priority>5</Priority>
   <Namespace>bb_ocm_demo</Namespace>
   <Version>1.0.0.0.0</Version>
   <ProcessStatus>n/a</ProcessStatus>
   <_root index="0">
      <subscriber_info index="1414783208019">
         <address index="1414783208020">
            <city index="1414783208023">MO</city>
            <postal_code index="1414783208022">A1A1A1</postal_code>
            <street index="1414783208021">Montreal Street</street>
         </address>
         <address index="1414783208024">
            <city index="1414783208027">OT</city>
            <postal_code index="1414783208026">B1B1B1</postal_code>
            <street index="1414783208025">Ottawa Street</street>
         </address>
         <address index="1414783208028">
            <city index="1414783208031">TO</city>
            <postal_code index="1414783208030">M9W6H8</postal_code>
            <street index="1414783208029">190 Drive</street>
         </address>
         <primary_phone_number index="1414783208033">1112223333
         </primary_phone_number>
         <name index="1414783208032">John Doe</name>
      </subscriber_info>
      <adsl_service_details index="1414783208034">
```

```
         <bandwidth index="1414783208035">3</bandwidth>
      </adsl_service_details>
   </_root>
</GetOrder.Response>
```

The following request uses the OrderDataFilter to filter out all sibling instances of the address multi-instance node. This functionality is particularly important in large orchestration orders when requesting order item information where an unfiltered GetOrder.Response message containing all the order data would negatively impact performance.

```
<GetOrder.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>2</OrderID>
   <View>demo_query</View>
   <OrderDataFilter>
      <Condition>/_root/subscriber_info/address[street='190 Drive']</Condition>
   </OrderDataFilter>
</GetOrder.Request>
```

The following response has filtered out two other instances of the address multi-instance node.

```
<GetOrder.Response>
   <OrderID>2</OrderID>
   <View>demo_query</View>
   <_root index="0">
      <subscriber_info index="1414682666683">
         <address index="1414682666696">
            <city index="1414682666697">TO</city>
            <postal_code index="1414682666698">A1B2Z7</postal_code>
            <street index="1414682666699">190 Drive</street>
         </address>
         <phone_number index="1414682666689">1111111111</phone_number>
         <name index="1414682666688">John Doe</name>
      </subscriber_info>
      <adsl_service_details index="1414682666690">
         <bandwidth index="1414682666691">3</bandwidth>
      </adsl_service_details>
   </_root>
</GetOrder.Response>
```

Error Codes

- 110: Order not found
- 232: Order update failed
- 270: Transaction not allowed
- 302: Request parameter error

> **Note:**
>
> See Table 3-3 for more information if you receive an error code that is not listed here.

GetOrderAtTask

To retrieve the order data as it exists at a specified task, you must provide the relevant IDs obtained from the worklist or another external source. If successful, the response includes the order data values as they exist at the specified task.

Parameters

OrderID: The order's ID number

Task: A descriptive mnemonic for the task

Request Example

```
<GetOrderAtTask.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>1234</OrderID>
   <Task>SampleTaskl</Task>
</GetOrderAtTask.Request>
```

Response Example

```
<GetOrderAtTask.Response xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>204</OrderID>
   <OrderHistID>443</OrderHistID>
   <Task>SampleTaskl</Task>
   <OrderSource>xmlapi</OrderSource>
   <OrderType>xmlapi</OrderType>
   <OrderState>open.running.in_progress</OrderState>
   <State>received</State>
   <ExecutionMode>do</ExecutionMode>
   <Reference>get_order</Reference>
   <ExpectedOrderCompletionDate>2009-03-10T00:00:00Z</
ExpectedOrderCompletionDate>
   <Priority>5</Priority>
   <Namespace>vadim</Namespace>
   <Version>1.0</Version>
   <_root index="0">
      <ProcessStatus index="1">n/a</ProcessStatus>
      <order_origination index="2">
         <currency index="3">150.55</currency>
         <boolean index="4">Yes</boolean>
         <m_multiple_lines_text index="5">text1</m_multiple_lines_text>
         <m_multiple_lines_text index="6">text2</m_multiple_lines_text>
         <options index="7">#1</options>
         <phone index="8">1234567890</phone>
         <date index="9">2010-10-10T15:10:10 EDT</date>
         <numeric index="10">155.0</numeric>
         <nested_group index="11">
            <currency index="12">200.0</currency>
         </nested_group>
      </order_origination>
   </_root>
</GetOrderAtTask.Response>
```

Response Example with Distributed Order Template

This is a partial example of the response for an order in which the order items and their dynamic parameters properties are using the distributed order template.

```
<GetOrderAtTask.Response xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>15</OrderID>
   <OrderHistID>236809</OrderHistID>
   <Task>configureCustomerSystemTask</Task>
   <OrderSource>OsmCentralOMExampleOrder</OrderSource>
   <OrderType>OsmCentralOMExampleOrder</OrderType>
   <OrderState>open.running.in_progress</OrderState>
   <State>accepted</State>
   <ExecutionMode>do</ExecutionMode>
   <Reference>Order1397065147300</Reference>
   <ExpectedOrderCompletionDate>2014-04-09T10:39:58 PDT</
ExpectedOrderCompletionDate>
   <Priority>5</Priority>
   <Namespace>OsmCentralOMExample-Solution</Namespace>
   <Version>4.0.0.0.0</Version>
   <ProcessStatus>n/a</ProcessStatus>
   <_root index="0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <CustomerDetails index="11">
         <locationType index="12">Street</locationType>
         <typeAddress index="23">Building</typeAddress>
      </CustomerDetails>
      <OrderHeader index="1">
         <numSalesOrder index="2">Order1397065147300</numSalesOrder>
         <typeOrder index="3">Add</typeOrder>
      </OrderHeader>
      <AccountDetails index="24">
         <numAccount index="25">TEL1234</numAccount>
         <status index="26">Existing</status>
         <corporate index="27">PoC</corporate>
         <category index="32">Corporate</category>
      </AccountDetails>
      <ControlData index="1397065199251">
         <Functions index="1397065199537">
         <SyncCustomerFunction index="1397065194936" instanceLocked="true">
               <componentKey index="1397065199567">
                  SyncCustomerFunction.CustomerSystem.WholeOrder</componentKey>
               <orderItem index="1397065199570">
                  <orderItemRef index="1397065199571"
referencedIndex="1397065194907" xmlns:ct234="http://oracle.osm.centralom"
xsi:type="ct234:CustomerOrderItemSpecificationType" type="{http://
oracle.osm.centralom}CustomerOrderItemSpecificationType">
                     <ct234:productClass index="1397065199259"
xmlns:ct234="http://oracle.osm.centralom">Broadband Service Feature Class</
ct234:productClass>
                     <ct234:productSpec index="1397065199260"
xmlns:ct234="http://oracle.osm.centralom">Service.Broadband</ct234:productSpec>
                     <ct234:lineId index="1397065199254" xmlns:ct234="http://
oracle.osm.centralom">1</ct234:lineId>
                     <ct234:lineItemName index="1397065199263"
xmlns:ct234="http://oracle.osm.centralom">Brilliant Broadband [Add]</
ct234:lineItemName>
                     <ct234:requestedDeliveryDate index="1397065199257"
xmlns:ct234="http://oracle.osm.centralom">2014-03-31T07:05:00 PDT</
ct234:requestedDeliveryDate>
                     <ct234:lineItemPayload index="1397065199255"
xmlns:ct234="http://oracle.osm.centralom">
                        <im:salesOrderLine xmlns:im="http://xmlns.oracle.com/
InputMessage" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
                           <im:lineId>1</im:lineId>
                           <im:promotionalSalesOrderLineReference>1 </
im:promotionalSalesOrderLineReference>
```

```
                              <im:serviceId/>
                              <im:requestedDeliveryDate>
                                 2014-03-31T07:05:00</im:requestedDeliveryDate>
                              <im:serviceActionCode>Add</im:serviceActionCode>
                              <im:serviceInstance>N</im:serviceInstance>
                              <im:serviceAddress>
                                 <im:locationType>Street</im:locationType>
                                <im:typeAddress>Building</im:typeAddress>
                              </im:serviceAddress>
                              <im:itemReference>
                                 <im:name>Brilliant Broadband</im:name>
                                 <im:primaryClassificationCode>Broadband Service
Feature Class</im:primaryClassificationCode>
                                 <im:specificationGroup/>
                              </im:itemReference>
                           </im:salesOrderLine>
                        </ct234:lineItemPayload>
                        <ct234:Recognition index="1397065199252"
xmlns:ct234="http://oracle.osm.centralom">Broadband Service Feature Class</
ct234:Recognition>
                        <ct234:dynamicParams
index="1397065199689" xmlns:ct234="http://oracle.osm.centralom"
xmlns:ct264="OracleComms_Model_Mobile/4.0.0.0.0"
xsi:type="ct264:SA_MobileMessagingCFSType" type="{OracleComms_Model_Mobile/
4.0.0.0.0}SA_MobileMessagingCFSType" xmlns:ct135="http://xmlns.oracle.com/
communications/studio/ordermanagement/transformation">
                           <ct264:MMSIncoming index="1397065199692"
xmlns:ct264="OracleComms_Model_Mobile/4.0.0.0.0">Yes</ct264:MMSIncoming>
                           <ct264:MMSOutgoing index="1397065199690"
xmlns:ct264="OracleComms_Model_Mobile/4.0.0.0.0">Yes</ct264:MMSOutgoing>
                        </ct234:dynamicParams>
                        <ct234:Action index="1397065199256" xmlns:ct234="http://
oracle.osm.centralom">Add</ct234:Action>
                        <ct234:ServiceInstance index="1397065199253"
xmlns:ct234="http://oracle.osm.centralom">N</ct234:ServiceInstance>
                     </orderItemRef>
                  </orderItem>
[...]
         </SyncCustomerFunction>
         </Functions>
         <OrderItem index="1397065194907" xmlns:ct234="http://
oracle.osm.centralom" xsi:type="ct234:CustomerOrderItemSpecificationType"
type="{http://oracle.osm.centralom}CustomerOrderItemSpecificationType">
            <ct234:productClass index="1397065199259" xmlns:ct234="http://
oracle.osm.centralom">Broadband Service Feature Class</ct234:productClass>
            <ct234:productSpec index="1397065199260" xmlns:ct234="http://
oracle.osm.centralom">Service.Broadband</ct234:productSpec>
            <ct234:lineId index="1397065199254" xmlns:ct234="http://
oracle.osm.centralom">1</ct234:lineId>
            <ct234:lineItemName index="1397065199263" xmlns:ct234="http://
oracle.osm.centralom">Brilliant Broadband [Add]</ct234:lineItemName>
            <ct234:requestedDeliveryDate index="1397065199257"
xmlns:ct234="http://oracle.osm.centralom">2014-03-31T07:05:00 PDT</
ct234:requestedDeliveryDate>
            <ct234:lineItemPayload index="1397065199255" xmlns:ct234="http://
oracle.osm.centralom">
               <im:salesOrderLine xmlns:im="http://xmlns.oracle.com/
InputMessage" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
                  <im:lineId>1</im:lineId>
                  <im:promotionalSalesOrderLineReference>1 </
im:promotionalSalesOrderLineReference>
```

```
                  <im:serviceId/>
                  <im:requestedDeliveryDate>
                     2014-03-31T07:05:00</im:requestedDeliveryDate>
                  <im:serviceActionCode>Add</im:serviceActionCode>
                  <im:serviceInstance>N</im:serviceInstance>
                  <im:serviceAddress>
                     <im:locationType>Street</im:locationType>
                     <im:typeAddress>Building</im:typeAddress>
                  </im:serviceAddress>
                  <im:itemReference>
                     <im:name>Brilliant Broadband</im:name>
                     <im:primaryClassificationCode>Broadband Service Feature
Class</im:primaryClassificationCode>
                     <im:specificationGroup/>
                  </im:itemReference>
               </im:salesOrderLine>
            </ct234:lineItemPayload>
            <ct234:Recognition index="1397065199252" xmlns:ct234="http://
oracle.osm.centralom">Broadband Service Feature Class</ct234:Recognition>
            <ct234:dynamicParams index="1397065199689" xmlns:ct234="http://
oracle.osm.centralom" xmlns:ct264="OracleComms_Model_Mobile/4.0.0.0.0"
xsi:type="ct264:SA_MobileMessagingCFSType" type="{OracleComms_Model_Mobile/
4.0.0.0.0}SA_MobileMessagingCFSType" xmlns:ct135="http://xmlns.oracle.com/
communications/studio/ordermanagement/transformation">
               <ct264:MMSIncoming index="1397065199692"
xmlns:ct264="OracleComms_Model_Mobile/4.0.0.0.0">Yes</ct264:MMSIncoming>
               <ct264:MMSOutgoing index="1397065199690"
xmlns:ct264="OracleComms_Model_Mobile/4.0.0.0.0">Yes</ct264:MMSOutgoing>
            </ct234:dynamicParams>
            <ct234:Action index="1397065199256" xmlns:ct234="http://
oracle.osm.centralom">Add</ct234:Action>
            <ct234:ServiceInstance index="1397065199253" xmlns:ct234="http://
oracle.osm.centralom">N</ct234:ServiceInstance>
         </OrderItem>
[...]
      </ControlData>
   </_root>
</GetOrderAtTask.Response>
```

Error Codes

- 110: order not found
- 257: Invalid task mnemonic
- 302: request parameter error

GetOrderDataHistory

Provides a list of the creation, update, and deletion of data in an order. Only those data elements that are visible in the order's assigned view (query task) is described in the OrderDataHistory response.

Operation

GetOrderDataHistory

Parameters

OrderID - The order sequence ID.

And one of the following:

OrderHistID: The order history ID.

View: A view (query task) assigned to the order.

With the given parameters, OrderDataHistory returns the order data history for all data elements of the order. To request the data history of specific elements, any number of Field elements can be provided to restrict the returned data history to the specific elements. The Field element contains an attribute and path, which resolves to a mnemonic path (using '/' as separators) for the data element whose data history is requested. Index values can be used in the path to narrow the scope of elements returned. Some examples are:

```
<Field path="/group_node/value_node" />
```

Returns data history for the root node, and all instances of /group_node and all instances of /group_node/value_node.

```
<Field path="/group_node[@index='12234']/value_node" />
```

Returns data history for the root node, the group_node with index 12234, and all instances of /group_node/value_node having group_node with index 12234 as a parent.

```
<Field path="/group_node[@index='12234']/value_node[@index='23111']" />
```

Returns data history for the root node, the group_node with index 12234, and all instances of /group_node/value_node having the index 23111 and group_node with index 12234 as a parent.

```
<Field path="/group_node/value_node" namespace="DSL_Highspeedline " version
="1.1" />
```

Returns data history for the root node, and all instances of /group_node and all instances of /group_node/value_node for the cartridge namespace DSL_Highspeedline version 1.1.

Request Example

```
<GetOrderDataHistory.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>1234</OrderID>
   <ViewID>32222</ViewID>
</GetOrderDataHistory.Request>
```

or

```
<GetOrderDataHistory.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>1234</OrderID>
   <View>Order_Creation_view</View>
   <Field path="/group_node/value_node" namespace="DSL_Highspeedline"
    version ="1.1"/>
</GetOrderDataHistory.Request>
```

The response returns the OrderID and OrderHistID/ViewID provided in the request, in addition to order data history in Field elements.

Each Field element has the following attributes:

path: The mnemonic path for the data element with '/' separating mnemonics.

namespace: The namespace mnemonic of order type/source.

version: The version of the order type or source. If you do not indicate a version, OSM uses the default version.

Index: The index number for the data element.

ParentIndex: The index number of the data element's parent. If the data element is the root node, it has an empty parentIndex (parentIndex="").

Multiple instance data elements have the same path but different indexes.

Each Field element has a Change element for each modification made to a data element. Each Change element has the following attributes:

Action: The action, either create, update, or delete.

User: The user ID that performed the change.

Time: The time of the change.

If the Change element is for a value node and has an action of "create" or "update", the value supplied to the data appears as the text value of the Change element.

Response Example

```
<GetOrderDataHistory.Response xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>1234</OrderID>
   <OrderHistID>32222</OrderHistID>
   <Field path="/" namespace="DSL_Highspeedline " version ="1.1" index="1221"
    parentIndex="">
      <Change action="create" user="oms" time="2000-01-28T14:33:22 EST"/>
   </Field>
   <Field path="/client_info" namespace="DSL_Highspeedline " version ="1.1"
    index="1222" parentIndex="1221">
      <Change action="create" user="oms" time="2000-01-28T14:33:22 EST"/>
   </Field>
   <Field path="/client_info/phone" namespace="DSL_Highspeedline" version ="1.1"
    index="1223" parentIndex="1221">
      <Change action="create" user="oms" time="2000-01-28T14:33:22
EST">4169999999
      </Change>
      <Change action="update" user="jdoe" time="2000-01-28T14:35:23 EST">
       4168888888
      </Change>
   </Field>
   <Field path="/client_info/address" namespace="DSL_Highspeedline" version
="1.1"
    index="12552" parentIndex="1222">
      <Change action="create" user="oms" time="2000-01-28T14:33:22 EST"/>
   </Field>
   <Field path="/client_info/address/street" namespace="DSL_Highspeedline"
    version ="1.1" index="12553" parentIndex="12552">
      <Change action="create" user="oms" time="2000-01-28T14:33:22 EST">
       20 West St.
      </Change>
      <Change action="delete" user="oms" time="2000-01-28T15:21:45 EST" />
   </Field>
</GetOrderDataHistory.Response>
```

Error Codes

- 110: Order not found
- 302: Request parameter error
- 400: Not authorized
- 401: Database Connection Failed
- 500: Internal error

> **Note:**
>
> See Table 3-3 for more information if you receive an error code that is not listed here.

GetOrderProcessHistory

Provides a list of each task transition of an order and a summary of the total time an order has been in a process.

Operation

GetOrderProcessHistory

Parameter

OrderID: The order ID for the order.

Request Example

```
<GetOrderProcessHistory.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>1234</OrderID>
</GetOrderProcessHistory.Request>
```

The GetOrderProcessHistory response returns the Order ID provided, and a Summary and Transitions element.

The children of the Summary element are:

ExpectedCompletionDate: The expected completion date of the entire process.

ActualDuration: The sum of the duration of all transitions of the order in seconds.

StartDate: The date the order was started in the process.

CompleteDate: The date the order was completed.

The children of the Transitions element are zero or more Transition elements. Each Transition element has the following children:

TaskID: The task mnemonic.

TaskType: The task type; manual, automatic, and creation.

TaskDescription: The description of the task.

ExpectedCompletionDate: The expected completion date of the task in seconds.

ActualDuration: The actual duration of the task and state in seconds.

StartDate: The date/time the task and state was entered.

CompleteDate: The date and time the task and state was completed.

OrderHistID: The order history sequence ID of the order's task and state.

FromOrderHistID: The order history sequence ID of the previous task and state.

State: The state mnemonic of the order.

Status: The status mnemonic of the order.

TransitionType: There are two transition types "normal", indicating transition within the process or "exception" indicating the transition was to an exception processing transition.

User: The unique identifier of the user who performed the transition.

ParentTaskOrderHistID: If the transition is within a sub-process resulting from an order data based transition, this value indicates the order history ID of the parent process task. If the transition is not within a sub-process, this value is empty.

DataNodeIndex: If the transition or a previous transition resulted from order data, this is a correlation index for the transitions that followed from the order data based transition.

DataNodeMnemonic: If the transition is a sub-process and the result of an order data based task (creates sub-processes), this value contains the mnemonic path (with '/' separators) of the node on which the sub-processes tasks were created.

DataNodeValue: If the transition is based on order data and is a value node, this element provides the value of the order data.

Response Example

```
<GetOrderProcessHistory.Response xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>1234</OrderID>
   <Summary>
      <ExpectedCompletionDate>2000-03-25T14:33:22 EST</ExpectedCompletionDate>
      <ActualDuration>101000</ActualDuration>
      <StartDate>2000-01-28T14:33:22 EST</StartDate>
      <CompleteDate/>
   </Summary>
   <Transitions>
      <Transition>
         <TaskID>order_entry</TaskID>
         <TaskType>manual</TaskType>
         <TaskDescription>Order Entry Task</TaskDescription>
         <ExpectedCompletionDate>2000-02-15T14:33:22 EST</ExpectedCompletionDate>
         <ActualDuration>65</ActualDuration>
         <StartDate>2000-01-28T14:33:22 EST</StartDate>
         <CompleteDate>2000-01-28T14:34:27 EST </CompleteDate>
         <OrderHistID>12432</OrderHistID>
         <FromOrderHistID>12431</FromOrderHistID>
         <State>received</State>
         <Status/>
         <TransitionType>normal</TransitionType>
         <User>oms</User>
```

```
        <ParentTaskOrderHistID/>
        <DataNodeIndex/>
        <DataNodeMnemonic/>
        <DataNodeValue/>
      </Transition>
      ... more Transition elements ...
   </Transitions>
</GetOrderProcessHistory.Response>
```

Error Codes

- 110: Order not found
- 302: Request parameter error
- 400: Not authorized
- 401: Database Connection Failed
- 500: Internal error

> **Note:**
>
> See Table 3-3 for more information if you receive an error code that is not listed here.

GetOrderStateHistory

Provides a list of each order state transition and its duration.

Operation

GetOrderStateHistory

Parameter

OrderID: The order ID for the order.

Namespace

Version

Request Example

```
<GetOrderStateHistory.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>1234</OrderID>
</GetOrderStateHistory.Request>
```

The GetOrderStateHistory response returns the Order ID provided, and the following Transition elements:

OrderState: The state mnemonic of the order.

TransitionStartDate: The date/time the order state was entered.

TransitionCompletedDate: The date/time the order state was completed.

ActualDuration: The actual duration of the order state in seconds.

User: The unique identifier of the user who performed the transition.

Reason: The reason for the order state transition. Supplied by the system when an order is created (create order) or submitted (submit order). Optionally supplied by the user when an order is suspended or resumed.

Response Example

```
<GetOrderStateHistory.Response xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>161</OrderID>
   <Namespace>orderamendment</Namespace>
   <Version>1.0</Version>
   <OrderStates>
      <OrderState>
         <OrderState>open.not_running.not_started</OrderState>
         <TransitionStartDate>2006-04-03T14:08:59 EDT</TransitionStartDate>
         <TransitionCompletedDate>2006-04-03T14:19:21 EDT
         </TransitionCompletedDate>
         <ActualDuration>PT10M22.000S</ActualDuration>
         <User>oms</User>
         <Reason>create order</Reason>
      </OrderState>
      <OrderState>
         <OrderState>open.running.in_progress</OrderState>
         <TransitionStartDate>2006-04-03T14:19:32 EDT</TransitionStartDate>
         <TransitionCompletedDate>2006-04-03T14:19:36 EDT
         </TransitionCompletedDate>
         <ActualDuration>PT4.000S</ActualDuration>
         <User>oms</User>
         <Reason>submit order</Reason>
      </OrderState>
      <OrderState>
         <OrderState>open.not_running.suspended</OrderState>
         <TransitionStartDate>2006-04-03T14:19:36 EDT</TransitionStartDate>
         <TransitionCompletedDate>2006-04-03T14:19:36 EDT
         </TransitionCompletedDate>
         <ActualDuration>PT0.000S</ActualDuration>
         <User>oms</User>
         <Reason>customer requested hold on order</Reason>
      </OrderState>
   </OrderStates>
</GetOrderStateHistory.Response>
```

Error Codes

- 110: Order not found
- 302: Request parameter error
- 400: Not authorized
- 401: Database Connection Failed
- 500: Internal error

> **Note:**
>
> See Table 3-3 for more information if you receive an error code that is not listed here.

GetTaskStatuses

Provides a list of all statuses for a given task.

Operation

GetTaskStatuses

Parameter

Task - A task mnemonic.

Namespace

Version

Request Example

```
<GetTaskStatuses.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <Task>new_test_task</Task>
   <Namespace>DSL_Highspeedline</Namespace>
   <Version>1.1</Version>
</GetTaskStatuses.Request>
```

Response Example

```
<GetTaskStatuses.Response xmlns="urn:com:metasolv:oms:xmlapi:1">
   <Task>new_test_task</Task>
   <Namespace>DSL_Highspeedline</Namespace>
   <Version>1.1</Version>
   <Status>delete</Status>
   <Status>dickson1</Status>
   <Status>dickson2</Status>
   <Status>end</Status>
   <Status>false</Status>
   <Status>redirect</Status>
   <Status>submit</Status>
   <Status>true</Status>
   <Status>undo</Status>
</GetTaskStatuses.Response>
```

Error Codes

- 150: Namespace/version not found.
- 152: Invalid namespace mnemonic.
- 153: No legacy data found. Namespace and Version need to be supplied.

> **Note:**
>
> See Table 3-3 for more information if you receive an error code that is not listed here.

Scenarios

Scenario 1: Namespace is supplied in the request, but no version. This condition forces the system to use the default version for the supplied namespace.

Scenario 2: Version is supplied in the request, but no namespace. This generates an error (152) message.

Scenario 3: The first cartridge is reserved for the migrated data. If namespace and version are not supplied, then this cartridge is used. If this legacy cartridge does not exist, an error (153) message is shown.

Scenario 4: If the combination of namespace and version does not exist, an error (150) message is shown.

GetUserInfo

Provides information regarding the current user's ID, the mnemonic and description of all assigned workgroups, and all user-defined columns (flexible headers.)

Parameters

None

Request Example

```
<GetUserInfo.Request xmlns="urn:com:metasolv:oms:xmlapi:1" />
```

The response includes the following elements:

User: ID of the user currently logged in.

Workgroup: The workgroup mnemonic to which the user is assigned.

The Workgroup element has the following attribute:

Desc: Description of the workgroup.

FlexibleHeaders: A list of all flexible headers available to the user.

The FlexibleHeaders element has the following attribute:

namespace: The namespace mnemonic of order type/source.

version: The version of the order type or source. If you do not indicate a version, OSM uses the default version.

Desc: Description of the flexible header.

Response Example

```
<GetUserInfo.Response xmlns="urn:com:metasolv:oms:xmlapi:1">
   <User>jdoe</User>
   <Workgroup desc="Provisioning">provisioning</Workgroup>
   <Workgroup desc="Customer Service">customer_service</Workgroup>
   <FlexibleHeaders>
      <FlexibleHeader namespace="DSL_Highspeedline" version="1.1"
       desc="Name">customer.name</FlexibleHeader>
      <FlexibleHeader namespace="DSL_Highspeedline" version="1.1" desc=
       "Phone Number">customer.phone_number</FlexibleHeader>
```

```
   </FlexibleHeaders>
<GetUserInfo.Response>
```

Error Codes

- 401: Database Connection Failed
- 500: Internal error

> **Note:**
>
> See Table 3-3 for more information if you receive an error code that is not listed here.

ListExceptions

Provides a list of available exception statuses for a given order. The request message includes an Order ID and Order History ID.

Operation

ListExceptions

Parameters

OrderID: The ID of the order.

OrderHistID: The order history ID.

Request Example

```
<ListExceptions.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>3422</OrderID>
   <OrderHistID>4333</OrderHistID>
</ListExceptions.Request>
```

The response includes the Exceptions element with zero or more status elements. The value of a status element is the status mnemonic.

Response Example

```
<ListExceptions.Response xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>3422</OrderID>
   <OrderHistID>4333</OrderHistID>
   <Exceptions>
      <Status desc="Complete">complete</Status>
      <Status desc="Delete">delete</Status>
   </Exceptions>
</ListExceptions.Response>
```

Error Codes

- 110: Order not found
- 302: Request parameter error
- 400: Not authorized

- 401: Database Connection Failed
- 500: Internal Error

> **Note:**
>
> See Table 3-3 for more information if you receive an error code that is not listed here.

ListStatesNStatuses

Returns a list of states and statuses used to transition a given task.

The task status/state request consists of a ListStatesNStatuses operation with parameters indicating the order ID and order history ID.

Operation

ListStatesNStatuses

Parameters

OrderID: The ID of the order.

OrderHistID: The order history ID.

Request Example

```
<ListStatesNStatuses.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>12345</OrderID>
   <OrderHistID>67890</OrderHistID>
</ListStatesNStatuses.Request>
```

The response has parameters with the requested order ID, the order history ID, and the current state of the task. The list of possible states and statuses are listed under the TaskStatesNStatuses element. There are five possible children:

- **Received**: The task may be set to the Received state.
- **Accepted**: The task may be accepted by the current user. Tasks are automatically moved to the Accepted state when retrieved using GetOrder.Request with an Accept parameter of "true".
- **Assigned**: The task can be assigned to any user ID listed in the User children elements.
- **Suspended**: The task can be suspended by providing any of the state mnemonics listed as children.
- **Completed**: The task can be completed by providing any of the status mnemonics listed as children.

Response Example

```
<ListStatesNStatuses.Response xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>12345</OrderID>
   <OrderHistID>67890</OrderHistoryID>
   <State>Received</State>
```

```
   <TaskStatesNStatuses>
      <Accepted/>
      <Assigned>
         <User>jdoe</User>
         <User>rsmith</User>
      </Assigned>
      <Suspend>
         <waiting_on_provisioning/>
         <customer_info_incomplete/>
      </Suspend>
      <Completed>
         <submit/>
         <delete/>
      </Completed>
   </TaskStatesNStatuses>
</ListStatesNStatuses.Response>
```

Error Codes

- 110: Order not found
- 302: Request parameter error
- 400: Not authorized
- 401: Database Connection Failed
- 500: Internal error

> **Note:**
>
> See Table 3-3 for more information if you receive an error code that is not listed here.

ListViews

Lists the views (query tasks) associated to a user workgroup for a given order source and type. If there is no view associated with a user workgroup, a view will not be returned. You can associate views with workgroups by bringing up the query task in Design Studio and selecting the Default view. See the discussion of query tasks in the Design Studio online help. (Note also that workgroups are called roles in Design Studio.)

Operation

ListViews

Parameters

OrderType: The type of the order.

OrderSource: The source of the order.

Namespace

Version

Request Example

```
<ListViews.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderSource>source1</OrderSource>
   <OrderType>provisioning</OrderType>
   <Namespace>DSL_Highspeedline</Namespace>
   <Version>1.1</Version>
</ListViews.Request>
```

Response Example

```
<ListViews.Response xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderSource>source1</OrderSource>
   <OrderType>provisioning</OrderType>
   <Namespace>DSL_Highspeedline</Namespace>
   <Version>1.1</Version>
   <View desc="Create order" mnemonic="create_order_view">1223</View>
   <View desc="Provision number" mnemonic="create_order_view">3424</View>
</ListViews.Response>
```

Error Codes

- 150: Namespace/version not found.
- 152: Invalid namespace mnemonic.
- 153: No legacy data found. Namespace and Version need to be supplied.

> **Note:**
>
> See Table 3-3 for more information if you receive an error code that is not listed here.

Scenarios

Scenario 1: Namespace is supplied in the request, but no version. This condition forces the system to use the default version for the supplied namespace.

Scenario 2: Version is supplied in the request, but no namespace. This condition generates an error (152) message.

Scenario 3: The first cartridge is reserved for the migrated data. If namespace and version are not supplied, then this cartridge is used. If this legacy cartridge does not exist, an error (153) message is shown.

Scenario 4: If the combination of namespace and version does not exist, an error (150) message is shown.

ModifyRemark

A remark can be modified after it is created, either to change the text of the remark or to add or remove attachments. Only the user who created the initial remark has authorization to change it, and only within an administrator defined time interval. The time interval after creating a remark is specified in the oms-config.xml file of the OSM Task web client with the property name remark_change_timeout_hours.

Operation

ModifyRemark

Parameters

OrderID: The order ID associated with the remark.

OrderHistID: The order history ID associated with the remark. If the remark has no OrderHistID, this field can be omitted or empty.

RemarkID: The unique identifier for the remark.

Text: The replacement text for the remark.

AddAttachments: A list of FileName elements that specify file names for new attachments.

DeleteAttachments: A list of AttachmentID elements that specify attachments to remove from the repository. Invalid AttachmentID values are not reported as errors. When attachments are deleted, the associated file is deleted from the file repository.

Request Example

```
<ModifyRemark.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>1234</OrderID>
   <OrderHistID>12333</OrderHistID>
   <RemarkID>1333</RemarkID>
   <Text>This is the new text for the remark</Text>
   <AddAttachments>
      <FileName>newfile.txt</FileName>
      <FileName>moreInformation.doc</FileName>
   </AddAttachments>
   <DeleteAttachments>
      <AttachmentID>10222</AttachmentID>
   </DeleteAttachments>
</ModifyRemark.Request>
```

The response follows the same format as that of UpdateOrder when there is a new attachment. The AttachmentID elements must be used to construct the file name for storing the attachment.

The response has a Remark element with the following child elements:

- **RemarkID**: The unique ID for the remark, assigned by OSM.
- **Attachment**: Zero or more Attachment elements for each attachment. An Attachment element has the following child elements:
  - **AttachmentID**: The unique ID for the attachment, assigned by OSM. When adding the attachment with the WebLogic file (T3) service, use the file name *AttachmentID***.srv**, where *AttachmentID* is the value of the AttachmentID element on the response.
  - **FileName**: The name of the file specified for the attachment.

Response Example

```
<ModifyRemark.Response xmlns="urn:com:metasolv:oms:xmlapi:1">
   <Remark>
      <RemarkID>1333</RemarkID>
```

```
      <Attachment>
         <AttachmentID>12222</AttachmentID>
         <FileName>newfile.txt</FileName>
      </Attachment>
      <Attachment>
         <AttachmentID>12223</AttachmentID>
         <FileName>moreInformation.doc</FileName>
      </Attachment>
   </Remark>
</ModifyRemark>
```

Error Codes

- 160: Remark not found
- 260: Remark cannot be modified
- 270: Transaction not allowed
- 302: Request parameter error
- 401: Database Connection Failed
- 500: Internal error

> **Note:**
>
> See Table 3-3 for more information if you receive an error code that is not listed here.

Notifications

A list of current retrievable notifications.

Operation

Notifications

Parameters

None

The returned information includes the following information for each notification:

_notif_id: An integer unique identification number for the notification.

_notif_desc: A string description of the notification

_notif_type: Either the string 'poll' for a polled notification or 'process' for a transitional notification.

_notif_priority: An integer indicating the priority level of the notification.

_notif_time: An XML API datetime representing the time the notification was generated.

_order_seq_id: An integer unique identification number of the order for which the notification was generated. If order ID does not exist, an empty value appears.

_order_hist_seq_id: An integer ID number of the order history ID for which the notification was generated.

_order_type: The mnemonic of the order's type. If there is no order, then an empty value appears.

_order_source: The mnemonic of the order's source. If there is no order, then an empty value appears.

_date_pos_created: An XML API datetime representation of the time the order was created. If there is no order, then an empty value appears.

_reference_number: The order's reference string. If there is no order, then an empty value appears.

_priority: An integer indicating the priority level of the order.

_requested_delivery_date: The date when the order is requested to be delivered. If an order contains multiple requested dates, for example because multiple order components have individual requested dates, then the requested date is interpreted to be the one selected for calculation of "expected start date."

_user: The surname currently associated with the order. If there is no order, then an empty value appears.

_ProcessStatus: The process status of the order.

_expected_completion_date: The date that OSM expects order processing to complete.

_expected_duration: The amount of time OSM determines it will take to complete the order. OSM calculates this value from durations given in the OSM model. OSM selects the durations based on the details of a specific order.

_expected_start_date: The date that OSM determines that the order should begin executing the order to meet its requested delivery date. This date is calculated by considering the expected duration. OSM only returns this parameter for orders that use orchestration plans. If an order does not use an orchestration plan, it is not returned.

_namespace: The namespace of the order type/source

_version: The version of the order type/source

In addition, for each flexible header assigned to the user, an instance of the following parameter is returned:

_header: This element has an attribute named **mnemonic_path** which contains the path of the flexible header. The value of the **_header** element is the value of the flexible header converted into a string. Table 3-2 lists the formats for data types that require formatting to be converted into a string:

Table 3-2 Formatting for Text Representation of Data Types

Primitive Type	Format
dateTime	yyyy-MM-ddThh:mm:ss time zone (for example 2013-10-30T14:33:22 EST)
date	yyyy-MM-dd

Table 3-2 (Cont.) Formatting for Text Representation of Data Types

Primitive Type	Format
Boolean	Yes or No

The notifications retrieved for a given user ID consist of those assigned to the user or any of the user's workgroups still in an active state. The maximum number of notifications returned in one request is defined by the **max_notification_rows** property in the oms-config.xml.

Request Example

```
<Notifications.Request xmlns="urn:com:metasolv:oms:xmlapi:1" />
```

If there are no notifications for a user ID, the response contains only the Header element.

Response Example

```
<Notifications.Response xmlns="urn:com:metasolv:oms:xmlapi:1">
   <Header>
      <_notif_id desc="Notification ID"/>
      <_notif_desc desc="Notification Description/>
      <_notif_type desc="Notification Type"/>
      <_notif_priority desc="Priority" />
      <_notif_time desc="Notification Timestamp"/>
      <_order_seq_id desc="Order ID"/>
      <_order_hist_seq_id desc="Order History ID"/>
      <_order_source desc="Source"/>
      <_order_type desc="Type"/>
      <_reference_number desc="Ref. #"/>
      <_priority desc="Priority" />
      <_date_pos_created desc="Order Creation Date"/>
      <_requested_delivery_date desc="Requested Order Delivery Date"/>
      <_expected_start_date desc="Expected Order Start Date"/>
      <_expected_duration desc="Expected Order Duration"/>
      <_compl_date_expected desc="Expected Order Completion Date"/>
      <_user desc="User" />
      <_process_status desc="Status"/>
      <_namespace desc="Namespace" />
      <_version desc="Version" />
      <customer.name desc="Customer Name"/>
      <customer.phone desc="Customer Phone"/>
   </Header>
   <Notification>
      <_notif_id>4567</_notif_id>
      <_notif_desc>Order transitioned</_notif_desc>
      <_notif_type>process</_notif_type>
      <_notif_priority>1</_notif_priority>
      <_notif_time>2000-10-30T14:33:22 EST</_notif_time>
      <_order_seq_id>2345</_order_seq_id>
      <_order_hist_seq_id>2333</_order_hist_seq_id>
      <_order_source>order_entry</_order_source>
      <_order_type>pots</_order_type>
      <_reference>AA-B3653F</_reference>
      <_priority>5</_priority>
      <_date_pos_created>2000-10-30T14:30:00 EST</_date_pos_created>
      <_requested_delivery_date>2000-10-30T14:30:00 EST</
```

```
_requested_delivery_date>
      <_expected_start_date>2000-10-20T14:30:00 EST</_expected_start_date>
      <_expected_duration>P10D</_expected_duration>
      <_compl_date_expected>2000-10-30T14:30:00 EST</_compl_date_expected>
      <_user>oms</_user>
      <_status/>
      <_namespace>DSL_Highspeedline</_namespace>
      <_version>1.1</_version>
      <customer.name>John Doe</customer.name>
      <customer.phone>4165555555</customer.phone>
   </Notification>
   <Notification>
      <_notif_id>4568</_notif_id>
      <_notif_desc>Pots Order Overdue</_notif_desc>
      <_notif_type>poll</_notif_type>
      <_notif_priority>1</_notif_priority>
      <_notif_time>2000-10-30T18:33:22 EST</_notif_time>
      <_order_seq_id>2346</_order_seq_id>
      <_order_hist_seq_id>2333</_order_hist_seq_id>
      <_order_source>order_entry</_order_source>
      <_order_type>pots</_order_type>
      <_reference>AA-B3653F</_reference>
      <_priority>5</_priority>
      <_date_pos_created>2000-10-30T14:30:00 EST</_date_pos_created>
      <_requested_delivery_date>2000-10-30T14:30:00 EST</
_requested_delivery_date>
      <_expected_start_date>2000-10-20T14:30:00 EST</_expected_start_date>
      <_expected_duration>P10D</_expected_duration>
      <_compl_date_expected>2000-10-30T14:30:00 EST</_compl_date_expected>
      <_user>oms</_user>
      <_status/>
      <_namespace>DSL_Highspeedline</_namespace>
      <_version>1.1</_version>
      <customer.name>John Doe</customer.name>
      <customer.phone>4165555555</customer.phone>
   </Notification>
   ... more notifications ...
</Notifications.Response>
```

Error Codes

- 500: Internal error

> **Note:**
>
> See Table 3-3 for more information if you receive an error code that is not listed here.

OrderTypesNSources

Orders have an associated source and type that is required prior to creating new orders. The available order type/source combinations are retrieved by requesting OrderTypesNSources. The mnemonics for the type/source pairs can be used to retrieve an order template and create a new order.

Operation

OrderTypesNSources

Parameters

Namespace

Version

Request Example

```
<OrderTypesNSources.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <Namespace>DSL_Highspeedline</Namespace>
   <Version>1.1</Version>
</OrderTypesNSources.Request>
```

Response Example

```
<OrderTypesNSources.Response mlns="urn:com:metasolv:oms:xmlapi:1">
   <TypeNSource namespace="DSL_Highspeedline" version="1.1">
      <Type mnemonic="phone_activation" category="Customer Service"
       desc="Phone Activation Order"/>
      <Source mnemonic="source1" desc="from front-end system"/>
   </TypeNSource>
   <TypeNSource namespace="DSL_Highspeedline" version="1.1" >
      <Type mnemonic="phone_transfer category="Customer Service"
       desc="Phone Number Transfer"/>
      <Source mnemonic="source1" desc="from front-end system"/>
   </TypeNSource>
</OrderTypesNSources.Response>
```

Error Codes

- 150: Namespace/version not found.
- 152: Invalid namespace mnemonic.
- 153: No legacy data found. Namespace and Version need to be supplied.

> **Note:**
>
> See Table 3-3 for more information if you receive an error code that is not listed here.

Scenarios

Scenario 1: Namespace is supplied in the request without a version. This forces the system to use the default version for the supplied namespace.

Scenario 2: Version is supplied in the request, but no namespace. This generates an error (152) message.

Scenario 3: The first cartridge is reserved for the migrated data. If namespace and version are not supplied, this cartridge is used. If this legacy cartridge does not exist, then an error (153) message is shown.

Scenario 4: If the combination of namespace and version does not exist an error (150) message is shown.

Scenario 5: If both namespace and version attributes are set to "*", the list of available order types and sources across all namespaces and versions is returned.

Scenario 6: If a namespace is supplied with version "*", the list of available order types and sources corresponding to the namespace across all of its existing versions is returned.

OrderViewTemplate

The OrderViewTemplate request provides the data type descriptions, such as minimum and maximum instances of lists, data types, and business names for data elements.

The OrderViewTemplate describes the structure of a particular order type. The types of order templates can be grouped into the following categories:

- **Order view creation template**: You must provide the mnemonics for a valid order type/source pair.
- **Order view template for a particular task in a process**: You must provide the order ID (_order_seq_id in the worklist) and the order history ID (_order_hist_seq_id in the worklist).
- **Order view templates for orders not in a process**: You must provide the order view (query task) template "view ID". You can obtain the list of valid view IDs for an order with a ListViews Request.
- **Order view templates for a task**: You must provide the order type/source and task mnemonic.

Operation

OrderViewTemplate

Parameters

For a creation template:

OrderSource: The order source mnemonic.

OrderType: The order type mnemonic.

For an in-process order:

OrderID: The sequence ID of an order (Orderdata/_order_seq_id from worklist).

OrderHistID: The history sequence ID of an order (Orderdata/_order_hist_seq_id from the worklist).

Namespace: The namespace mnemonic of order type/source.

Version: The version mnemonic of order type/source.

For an order not in a process:

OrderSource: The order source mnemonic.

OrderType: The order type mnemonic.

View: A view (query task) valid for a particular order type and source.

Namespace: The namespace mnemonic of order type/source.

Version: The version mnemonic of order type/source.

For a task order view template:

OrderSource: The order source mnemonic.

OrderType: The order type mnemonic.

Task: The task mnemonic.

Namespace: The namespace mnemonic of order type/source.

Version: The version mnemonic of order type/source.

Request Example

```
<OrderViewTemplate.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderSource>source1</OrderSource>
   <OrderType>phone_transfer</OrderType>
   <Namespace>DSL_Highspeedline</Namespace>
   <Version>1.1</Version>
</OrderViewTemplate.Request>
```

Response Example

```
<OrderViewTemplate.Response xmlns="urn:com:metasolv:oms:xmlapi:1">
<OrderType>phone_transfer</OrderType>
   <OrderSource>source1</OrderSource>
   <Namespace>DSL_Highspeedline</Namespace>
   <Version>1.1</Version>
   <_root>
      <client_info
       readOnly="false"
       desc="Client Information"
       minInstance="1"
       maxInstance="1">
         <name
          readOnly="false"
          desc="Name"
          minInstance="1"
          maxInstance="1"
          type="TX"
          len="30"/>
         <address
          readOnly="false"
          desc="Address"
          minInstance="1"
          maxInstance="2">
            <street1
             readOnly="false"
             desc="Street 1"
             minInstance="1"
             maxInstance="1"
             type="TX"
             length="50" />
            <street2
             readOnly="false"
             desc="Street 2"
```

```
                minInstance="0"
                maxInstance="1"
                type="TX"
                length="50" />
               <city
                readOnly="false"
                desc="City"
                minInstance="0"
                maxInstance="1"
                type="TX"
                length="25" />
               <state
                readOnly="false"
                desc="State"
                minInstance="1"
                maxInstance="1"
                type="LK" />
            </address>
         </client_info>
      </_root>
      <LookupTables>
         <state>
            <option desc="New York">NY</option>
            <option desc="California">CA</option>
            <option desc="New Jersey">NJ</option>
            ... etc ...
         </state>
      </LookupTables>
</OrderViewTemplate.Response>
```

The response includes the OrderViewTemplate beginning with a _root element. Each child is named with the mnemonic of that data element, along with the following attributes:

Desc: The business name of the element.

MinInstance: The minimum number of instances allowed (0-n). A field with minInstance > 0 is a mandatory field if the parent is defined in the order.

MaxInstance: The maximum number of instances allowed (1-n).

Mask: The mask associated with an element of type NM or TX.

ReadOnly: A true/false attribute indicating that the field cannot be modified.

Type: The data type of the element. The data type values are:

- **NM**: Numeric type
- **TX**: Text type
- **D**: Date type in the form: yyyy-mm-dd (for example, 2000-03-10)
- **DT**: Date/time type in the form: yyyy-mm-ddThh:mm:ss time zone (for example, 2000-03-10T14:43:00 EST)
- **PH**: Phone number type
- **YN**: Boolean type (Yes/No)
- **CY**: Currency type
- **LK**: Lookup type

Length: The maximum length of the element if the type is NM or TX.

The values for the lookup elements in the view template are rooted at the LookupTables element. The name of a lookup element matches the data elements of type LK in the order template.

Error Codes

- 150: Namespace/version not found.
- 152: Invalid namespace mnemonic.
- 153: No legacy data found. Namespace and Version need to be supplied.

> **Note:**
>
> See Table 3-3 for more information if you receive an error code that is not listed here.

Scenarios

Scenario 1: Namespace is supplied in the request, but no version. This forces the system to use the default version for the supplied namespace.

Scenario 2: Version is supplied in the request, but no namespace. This generates an error (152) message.

Scenario 3: The first cartridge is reserved for the migrated data. If namespace and version are not supplied, this cartridge is used. If this legacy cartridge does not exist, then an error (153) message is shown.

Scenario 4: If the combination of namespace and version does not exist, then an error (150) message is shown.

For an In-Process Order

```
<OrderViewTemplate.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>1234</OrderID>
   <OrderHistID>12333</OrderHistID>
</OrderViewTemplate.Request>
```

For an Order Not In a Process

```
<OrderViewTemplate.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderSource>source1</OrderSource>
   <OrderType>phone_transfer</OrderType>
   <ViewID>1123</ViewID>
   <Namespace>DSL_Highspeedline</Namespace>
   <Version>1.1</Version>
</OrderViewTemplate.Request>
```

or

```
<OrderViewTemplate.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderSource>source1</OrderSource>
   <OrderType>phone_transfer</OrderType>
   <View>phone_transfer</View>
   <Namespace>DSL_Highspeedline</Namespace>
```

```
   <Version>1.1</Version>
</OrderViewTemplate.Request>
```

For a Task Order View Template

```
<OrderViewTemplate.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderSource>source1</OrderSource>
   <OrderType>phone_transfer</OrderType>
   <Task>phone_transfer</Task>
   <Namespace>DSL_Highspeedline</Namespace>
   <Version>1.1</Version>
</OrderViewTemplate.Request>
```

Query

Queries the order header information and data elements of orders for inclusion in the Query.Response.

Operation

Query

Parameters

OrderID: An integer value with wildcards for the order sequence ID.

Reference: A string with wildcards for the reference number.

OrderType: An order type mnemonic.

OrderSource: An order source mnemonic.

Task: The task mnemonic.

CreatedDate: The date the order was created. Two attributes, "from" and "to", indicate the beginning and ending date or date/time for the query.

RequestedDeliveryDate: The date requested by the client for the order to be completed.

ExpectedStartDate: The date that OSM will begin processing an order.

ExpectedDuration: The amount of time the order is expected to take to complete processing.

ExpectedCompletionDate: The date that OSM expects the order to complete.

CompletedDate: The date the order was completed.

User: A user ID.

State: A task state mnemonic. The valid mnemonics are:

- Received
- Assigned
- Accepted
- Completed
- Any user-defined state mnemonics

Status: A task status mnemonic. The valid status mnemonics are those statuses defined in the OSM Administrator.

Priority: An integer indicating the priority level of the order.

OrderState: An order state mnemonic. The valid mnemonics are:

- open.not_running.not_started
- open.not_running.suspended
- open.not_running.waiting_for_revision
- open.not_running.cancelled
- open.running.in_progress
- open.running.compensating.amending
- open.running.compensating.cancelling
- closed.completed
- closed.aborted
- open.not_running.failed

TargetOrderState: Same mnemonics as OrderState.

ExecutionMode: An execution mode mnemonic. The valid mnemonics are:

- Do
- Redo
- Undo

FlexibleHeaders: A list of flexible headers available to the user. Can contain zero or more FlexibleHeader elements with the mnemonic paths of flexible headers to include in the output. If FlexibleHeaders is omitted, no flexible headers are returned.

SingleRow: This element forces OSM to display only a single row if a query returns more than one match per order.

OrderBy: The OrderBy element contains a list of fields on which to order the results. The order of the Field elements is significant: the results are ordered based on the first element, then the second element, and so on.

The order attribute must be either **descending** or **ascending**, otherwise an error results. The path attribute must belong to the element name of one of the fixed headers or one of the request's selected flexible headers. If the mnemonic path does not resolve to a valid field to sort by, error code 170: Header for mnemonic path not found is returned.

When sorting on fixed header elements, the sorted value for elements that are represented by an ID (that is, state) will be sorted based on the internal ID's value, not that of the final output.

For querying order data values, a 'Field' element is provided with the following format. If querying for equality:

```
<Field path="/client_info/address/city" namespace="DSL_Highspeedline"
version="1.1">Toronto</Field>
```

If querying for a range of values:

```
<Field path="/client_info/address/city" namespace="DSL_Highspeedline"
 version="1.1">
   <From>A*</From>
   <To>D*</To>
</Field>
```

The "path" attribute of the field is a sequence of data element mnemonics, separated by a slash, "/", indicating the mnemonic path of the data field to be matched against. Only the data elements that are assigned as flexible headers for the user ID can be queried in a Field element.

All of the elements have the following attributes:

namespace: The namespace mnemonic of order type/source. If you do not specify a namespace or specify *, all available namespaces are picked up.

version: The version of the order type or source. If you do not specify a version or specify *, all available versions are picked up.

You can use wildcard characters with the Order ID, Reference and Field elements whose path attribute resolves to a OSM text field (TX data type). Valid wildcard characters are:

- "*" indicates any number of characters
- "?" indicates a single character

If an additional element, SingleRow, is "true", an order is listed only once. If SingleRow is "false", the order is listed once for each data element it matches in the query request.

The Query request performs a logical "AND" of all provided parameters.

Request Example

```
<Query.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>123*</OrderID>
   <FlexibleHeaders>
      <FlexibleHeader namespace="DSL_Highspeedline" version="1.1"
       path="customer/name"/>
      <FlexibleHeader namespace="DSL_Highspeedline" version="1.1"
       path="customer/phone"/>
      <FlexibleHeader>customer/address</FlexibleHeader>
      <FlexibleHeader>customer/name</FlexibleHeader>
   </FlexibleHeaders>
</Query.Request>
```

or

```
<Query.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <FlexibleHeaders>
      <FlexibleHeader namespace="view_framework_demo" version="1.0.0.0.0"path="/
account_information/amount_owing"/>
   </FlexibleHeaders>
   <Field namespace="view_framework_demo" version="1.0.0.0.0" path="/
account_information/amount_owing">444</Field>
</Query.Request>
```

or

```
<Query.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>123*</OrderID>
```

```
   <Field path="/client_info/address/city" namespace="DSL_Highspeedline"
    version="1.1">Toronto</Field>
</Query.Request>
```

or

```
<Query.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>123*</OrderID>
   <OrderBy>
      <Field order="descending" namespace="DSL_Highspeedline" version="1.1">
       customer/phone </Field>
   </OrderBy>
</Query.Request>
```

or

```
<Query.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderState>open.not_running.not_started</OrderState>
</Query.Request>
```

Response Example

```
<Query.Response xmlns="urn:com:metasolv:oms:xmlapi:1">
  <Header>
    <_order_seq_id desc="Order ID"/>
    <_order_hist_seq_id desc="Order History ID"/>
    <_order_state desc="Task State"/>
    <_execution_mode desc="Execution Mode"/>
    <_task_id desc="Task Name"/>
    <_order_source desc="Source"/>
    <_order_type desc="Order Type"/>
    <_current_order_state desc="Order State"/>
    <_target_order_state desc="Target Order State"/>
    <_reference_number desc="Ref. #"/>
    <_priority desc="Priority"/>
    <_date_pos_created desc="Order Creation Date"/>
    <_requested_delivery_date desc="Requested Delivery Date"/>
    <_expected_start_date desc="Expected Start Date"/>
    <_expected_duration desc="Expected Duration"/>
    <_user desc="User"/>
    <_process_status desc="Process Status"/>
    <_date_pos_started desc="Started"/>
    <_compl_date_expected desc="Expected Order Completion Date"/>
    <_ord_completion_date desc="Completed Date"/>
    <_grace_period_completion_date desc="Expected Grace Period Completion"/>
    <_num_remarks desc="Number of Remarks"/>
    <_namespace desc="Namespace"/>
    <_version desc="Version"/>
    <_workgroups desc="Workgroups">
        <_workgroup/>
    </_workgroups>
  </Header>
  <Orderdata>
    <_order_seq_id>40</_order_seq_id>
    <_order_hist_seq_id>502</_order_hist_seq_id>
    <_order_state>received</_order_state>
    <_execution_mode>do</_execution_mode>
    <_task_id>CollectionsFunction_CollectionsSI</_task_id>
    <_order_source>OsmCentralOMExampleOrder</_order_source>
    <_order_type>OsmCentralOMExampleOrder</_order_type>
    <_current_order_state>open.not_running.waiting</_current_order_state>
    <_target_order_state/>
```

```
    <_reference_number>[do:1]$ref1436191716838</_reference_number>
    <_priority>5</_priority>
    <_date_pos_created>2015-07-06T07:08:36 PDT</_date_pos_created>
    <_requested_delivery_date>2015-08-05T10:08:36 PDT</_requested_delivery_date>
    <_expected_start_date>2015-07-09T10:08:36 PDT</_expected_start_date>
    <_expected_duration>P26D</_expected_duration>
    <_user/>
    <_process_status>n/a</_process_status>
    <_date_pos_started>2015-07-06T07:08:38 PDT</_date_pos_started>
    <_compl_date_expected>2015-08-04T10:08:36 PDT</_compl_date_expected>
    <_ord_completion_date/>
    <_grace_period_completion_date/>
    <_num_remarks>0</_num_remarks>
    <_namespace>OsmCentralOMExample-Solution</_namespace>
    <_version>4.0.0.0.0</_version>
    <_workgroups>
        <_workgroup/>
    </_workgroups>
  </Orderdata>
</Query.Response>
```

The query response uses a similar format as the XMLAPI Worklist message. It consists of a header element that contains descriptive elements for all columns returned. The columns consist of a number of fixed header elements (prefixed with _), followed by any flexible headers defined for the user in the OSM Administrator. Zero or more Orderdata elements follow the Header element with each one corresponding to the data for a particular order matched by the query criteria. If the request has "SingleRow" set to "false", an order appears once for each data element that was matched. If set to "true", it appears only once.

To retrieve an order matched in the query, the GetOrder.Request supports a parameter, "ViewID", which retrieves an order based on a particular view (query task). You can obtain a list of valid ViewIDs for an order source/type with ListViews.Request.

Error Codes

- 170: Header for mnemonic path not found
- 302: Request parameter error
- 400: Not authorized
- 401: Database connection failed
- 500: Internal Error
- 601: Deprecated parameter

> **Note:**
>
> See Table 3-3 for more information if you receive an error code that is not listed here.

ReceiveOrder

Moves an order to the Received state.

Operation

ReceiveOrder

Parameters

OrderID: The ID of the order to change.

OrderHistID: The order history ID.

Request Example

```
<ReceiveOrder.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>1234</OrderID>
   <OrderHistID>222334</OrderHistID>
</ReceiveOrder.Request>
```

Response Example

```
<ReceiveOrder.Response xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>1234</OrderID>
   <OrderHistID>33123</OrderHistID>
</ReceiveOrder.Response>
```

The ReceiveOrder response includes the new order history ID for the task.

Error Codes

- 110: Order not found
- 251: Transition invalid
- 270: Transaction not allowed
- 302: Request parameter error
- 401: Database Connection Failed
- 500: Internal error

> **Note:**
>
> See Table 3-3 for more information if you receive an error code that is not listed here.

ResolveFailure

Resolves failures from a particular task causing the task to transition from a failed execution mode to a corresponding normal execution mode. The task reverts to the state it had been in before the failure occurred. This request requires an Order ID and Order History ID.

Operation

ResolveFailure

Parameters

OrderID: The ID of the order to change.

OrderHistID: The order history ID.

Reason: The reason for the failure resolution. This parameter is optional.

Request Example

```
<ResolveFailure.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>18</OrderID>
   <OrderHistID>342</OrderHistID>
</ResolveFailure.Request>
```

Response Example

```
<ResolveFailure.Response xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>18</OrderID>
   <OrderHistID>342</OrderHistID>
</ResolveFailure.Response>
```

Error Codes

- 110: Order not found
- 270: Transaction not allowed
- 302: Request parameter error
- 400: Not authorized
- 401: Database connection failed
- 500: Internal ErrorSuspendOrder

> **Note:**
>
> See Table 3-3 for more information if you receive an error code that is not listed here.

ResumeOrder

Releases a suspended order back into the system and returns it to the state from which it was suspended. Can also be used to resubmit a canceled order back into the system.

Operation

ResumeOrder

Parameters

OrderID: The ID of the order to resume.

Reason: The reason why the order is being resumed. Optional.

Request Example

```
<ResumeOrder.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>1234</OrderID>
   <Reason>Customer requests resumption of order</Reason>
</ResumeOrder.Request>
```

Response Example

```
<ResumeOrder.Response xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>1234</OrderID>
</ResumeOrder.Response>
```

Error Codes

- 110: Order not found
- 251: Transition invalid
- 270: Transaction not allowed
- 302: Request parameter error
- 401: Database Connection Failed
- 500: Internal error

> **Note:**
>
> See Table 3-3 for more information if you receive an error code that is not listed here.

RetryTask

Retries a failed task causing the task to transition from a failed execution mode to a corresponding normal execution mode. The task reverts to the received state. This request requires an Order ID and Order History ID.

Operation

RetryTask

Parameters

OrderID: The ID of the order to change.

OrderHistID: The order history ID.

Reason: The reason retrying the task. This parameter is optional.

Request Example

```
<RetryTask.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>18</OrderID>
   <OrderHistID>342</OrderHistID>
</RetryTask.Request>
```

Response Example

```
<RetryTask.Response xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>18</OrderID>
   <OrderHistID>342</OrderHistID>
</RetryTask.Response>
```

Error Codes

- 110: Order not found
- 270: Transaction not allowed
- 302: Request parameter error
- 400: Not authorized
- 401: Database connection failed
- 500: Internal ErrorSuspendOrder

> **Note:**
>
> See Table 3-3 for more information if you receive an error code that is not listed here.

SetException

Sets the exception status for a given order. This request requires an Order ID, Order History ID, and Status mnemonic.

Operation

SetException

Parameters

OrderID: The ID of the order to change.

OrderHistID: The order history ID.

Status: The status mnemonic.

Request Example

```
<SetException.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>3422</OrderID>
   <OrderHistID>4333</OrderHistID>
   <Status>complete</Status>
</SetException.Request>
```

Response Example

```
<SetException.Response xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>3422</OrderID>
   <OrderHistID>4334</OrderHistID>
</SetException.Response>
```

The SetException response includes the new order history ID if the exception goes to a new task.

Error Codes

- 110: Order not found
- 256: Invalid exception status mnemonic
- 270: Transaction not allowed
- 302: Request parameter error
- 400: Not authorized
- 401: Database Connection Failed
- 419: The process exception is restricted
- 500: Internal ErrorSuspendOrder

> **Note:**
>
> See Table 3-3 for more information if you receive an error code that is not listed here.

SuspendOrder

Suspends an order, or a task, depending on the parameters supplied in the request.

Operation

SuspendOrder

Parameters

For an order:

OrderID: The ID of the order to suspend

And one of the following:

Immediate: Force immediate suspension of all tasks associated with the order.

GracePeriodExpiryDate: A period of time to allow tasks in the Accepted state time to complete.

Infinite: Wait indefinitely until all tasks in the Accepted state complete.

Optional parameters

EventInterval: If the suspension is not immediate, you can set an interval for sending a jeopardy notification.

Reason: The reason why the order is being suspended.

For a task:

OrderID: The ID of the order to change.

OrderHistID: The order history ID.

State: The user-defined state mnemonic.

Request Example 1: Order

```
<SuspendOrder.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>1234</OrderID>
   <Immediate/>
   <Reason>Customer requested hold on order</Reason>
</SuspendOrder.Request>
```

or

```
<SuspendOrder.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>1234</OrderID>
   <GracePeriodExpiryDate>2006-10-10T11:10:10 EST</GracePeriodExpiryDate>
   <EventInterval>PT10S</EventInterval>
   <Reason>Customer requested hold on order</Reason>
</SuspendOrder.Request>
```

or

```
<SuspendOrder.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>1234</OrderID>
   <Infinite/>
   <Reason>Customer requested hold on order</Reason>
</SuspendOrder.Request>
```

Request Example 2: Task

```
<SuspendOrder.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>1234</OrderID>
   <OrderHistID>22334</OrderHistID>
   <State>waiting_on_provisioning</State>
</SuspendOrder.Request>
```

Response Example 1: Order

```
<SuspendOrder.Response xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>1234</OrderID>
</SuspendOrder.Response>
```

Response Example 2: Task

```
<SuspendOrder.Response xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>1234</OrderID>
   <OrderHistID>33247</OrderHistID>
</SuspendOrder.Response>
```

The SuspendOrder response includes the new order history ID for the task.

Error Codes

- 110: Order not found
- 251: Transition invalid
- 254: State mnemonic invalid
- 270: Order could not be suspended
- 302: Request parameter error

- 401: Database Connection Failed
- 500: Internal error

> **Note:**
>
> See Table 3-3 for more information if you receive an error code that is not listed here.

TaskDescription

The TaskDescription operation retrieves the list of all available tasks.

Operation

TaskDescription

Parameters

Namespace: The namespace mnemonic of the order type/source (optional).

Version: The version of the order type or source (optional).

Request Example 1

```
<TaskDescription.Request xmlns="urn:com:metasolv:oms:xmlapi:1" />
```

Request Example 2

```
<TaskDescription.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <Namespace>DSL_Highspeedline</Namespace>
   <Version>1.1</Version>
</TaskDescription.Request>
```

Response Example

```
<TaskDescription.Response xmlns="urn:com:metasolv:oms:xmlapi:1">
   <Task mnemonic="activate_switch" TaskType="automatic" desc="Activate switch">
   <Task mnemonic="provision_number" TaskType="manual"
    desc="Provision Customer Number">
</TaskDescription.Response>
```

Error Codes

- 400: Not authorized
- 401: Database Connection Failed
- 500: Internal errorWorklist

> **Note:**
>
> See Table 3-3 for more information if you receive an error code that is not listed here.

The worklist data retrieved by the Client user ID is defined by the OSM Administrator. The Worklist is implicitly defined by the privileges of each workgroup to which the Client ID is assigned.

The worklist response consists of fixed header elements followed by the flexible headers as defined in the OSM Administrator.

UpdateOrder

An order update consists of an operation UpdateOrder, with the order ID and order history ID to identify the order. The request defines different ways to update the order such as UpdatedOrder, UpdatedNodes, Add, and Delete.

Operation

UpdateOrder

Parameters

This API defines the following parameter:

OrderID: The ID of the order to change.

OrderHistID: The order history ID.

ViewID: The id of the view (query task) that is used for the order update. This is a workgroup view that must be associated with one or more workgroups the requesting user is a member of for the definition of the order.

View: The name of the task (view) that is used for the order update. You must associate the task you want to update to a role (workgroup) in the Design Studio Order editor Permissions Query Task sub tab and set the task as the Default query task. You can associate only one role per task in the Order editor. The user submitting the UpdateOrder must be a member of this role.

ResponseView: An optional parameter that defines the name of the task (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 effected OrderItems and OrderComponents instances.

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

- **Condition**: An XPath 1.0 expression against the order data defined by the ResponseView. OSM returns only the instances of the order data selected by the expression, not the other instances of the element. All other parent or sibling elements are returned.

  For example, in a situation where a customer has multiple <address> 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 instance of the <address> element (city, postal code, and street). The other instances of the <address> element and their child elements (city, street, and postal code) are not returned.

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

For example, any sibling elements of <subscriber_info>, or sibling elements of <address> (except for the other instances of the <address> element) would be returned.

When you are using an order condition that includes an element that is using a distributed order template, you should include the namespace of the data element in the condition. For example:

```
<OrderDataFilter>
   <Condition>
      /ControlData/OrderItem[@type='{OrderItemNamespace}OrderItemName' and
@LineId='1']
   </Condition>
</OrderDataFilter>
```

NewReference: An optional new reference number for the order.

AddMandatory: This parameter is true if the mandatory fields defined in the order view (task) should be added into the order by this order update, otherwise this parameter should be false.

Priority: The priority that the order is set to by this order update.

A choice of:

- **UpdatedOrder**: Allows the user to update the order by supplying a complete order. The existing order is then updated (elements added, changed or deleted as necessary) to match the supplied order.
- **UpdatedNodes**: Allows the user to update the order by supplying only the nodes that should be added or updated. The nodes are supplied in the format of the existing order: The structure of the nodes (parents and children) must match the view (task) specification for the view being used. No deletes are performed using this approach.
- **Add**: Allows the user to update the order by adding new data in the form of a node to be added. The path attribute identifies the parent node under which to add the element.
- **Delete**: Allows the user to update the order by deleting existing data in the form of a node to be deleted. The path attribute identifies the node to delete.
- **Update**: Allows the user to update the order by updating existing data in the form of a node to be updated. The path attribute identifies the node to update.

AddRemark: A remark can be added to the order using an AddRemark parameter. The AddRemark element has the following elements:

- **Text**: The text for the new remark.
- **Attachments**: The parent element for FileName elements. The Attachments element can also have one of the following elements:
- **FileName**: The name of the file for a new attachment. If both the Text and Attachments elements are empty, a remark is not created.

ExternalFulfillmentStates: Allows you to set external fulfillment states instead of using an 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.

> **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.

Request Example

```
<UpdateOrder.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>702</OrderID>
   <View>Update Order View</View>
   <NewReference/>
   <Add path="/client_info">
      <address>
         <street1>55 James St.</street1>
         <city>Washington</city>
         <state>DC</state>
         <country>USA</country>
         <zip>45432</zip>
      </address>
   </Add>
   <Delete path="/client_info/address[@index='80132']" />
   <AddRemark>
      <Text>This is the text for the remark</Text>
      <Attachments>
         <FileName>provisioninfo.txt</FileName>
         <FileName>diagram.bmp</FileName>
      </Attachments>
   </AddRemark>
   <ExternalFulfillmentStates>
      <OrderItemOrderComponentFulfillmentState>
         <ExternalFulfillmentState>COMPLETED</ExternalFulfillmentState>
         <OrderComponentIndex>123</OrderComponentIndex>
         <OrderItemIndex>456</OrderItemIndex>
      </OrderItemOrderComponentFulfillmentState>
```

```
  </ExternalFulfillmentStates>
</UpdateOrder.Request>
```

If a remark is added to an order, remark information is returned in the UpdateOrder response. The response has a Remark element with the following child elements:

- **RemarkID**: The unique ID for the remark, assigned by OSM.
- **Attachment**: Zero or more Attachment elements for each attachment. An Attachment element has the following child elements:
  - **AttachmentID**: The unique ID for the attachment, assigned by OSM. When adding the attachment with the WebLogic file (T3) service, use the file name *AttachmentID***.srv**, where *AttachmentID* is the value of the AttachmentID element on the response.
  - **FileName**: The name of the file specified for the attachment.

Response Example

```
<UpdateOrder.Response xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>702</OrderID>
   <View>Update Order View</View>
   <Remark>
      <RemarkID>3224</RemarkID>
      <Attachment>
         <AttachmentID>10333</AttachmentID>
         <FileName>provisioninfo.txt</FileName>
      </Attachment>
      <Attachment>
         <AttachmentID>10334</AttachmentID>
         <FileName>diagram.bmp</FileName>
      </Attachment>
   </Remark>
</UpdateOrder.Response>
```

Request Example with ResponseView and OrderDataFilter

```
<UpdateOrder.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>15778</OrderID>
   <View>OsmCentralOMExampleQueryTask</View>
   <ResponseView>OsmCentralOMExampleQueryTask</ResponseView>
   <OrderDataFilter>
       <Condition>/CustomerDetails/typeCompl[@index='15']</Condition>
   <OrderDataFilter>
   <UpdatedNodes>
      <_root>
         <CustomerDetails>
            <typeCompl>floor</typeCompl>
         </CustomerDetails>
      </_root>
   </UpdatedNodes>
   <ExternalFulfillmentStates>
      <OrderItemOrderComponentFulfillmentState>
         <ExternalFulfillmentState>ExtFulfState1</ExternalFulfillmentState>
         <ExternalFulfillmentStateDescription>
         </ExternalFulfillmentStateDescription>
         <OrderComponentIndex>132490</OrderComponentIndex>
         <OrderItemIndex>1434565</OrderItemIndex>
      </OrderItemOrderComponentFulfillmentState>
    </ExternalFulfillmentStates>
</UpdateOrder.Request>
```

Response Example with ResponseView and OrderDataFilter

```
<UpdateOrder.Response xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderID>15778</OrderID>
   <View>Update Order View</View>
   <Data>
      <_root index="0">
         <CustomerDetails index="11">
            <typeCompl index="15">floor</typeCompl>
         </CustomerDetails>
      </_root>
   </Data>
</UpdateOrder.Response>
```

Error Codes

110: Order not found

200: Order data invalid

230: Order not accepted by user

232: Order update failed

270: Transaction not allowed

302: Request parameter error

400: Not authorized

401: Database Connection Failed

420: Not authorized to modify order priority

500: Internal error

> **Note:**
>
> See Table 3-3 for more information if you receive an error code that is not listed here.

Scenarios

Scenario 1: Namespace is supplied in the request, but no version. This condition forces the system to use the default version for the supplied namespace.

Scenario 2: Version is supplied in the request, but no namespace. This generates an error (152) message.

Scenario 3: The first cartridge is reserved for the migrated data. If namespace and version are not supplied, then this cartridge is used. If this legacy cartridge does not exist, an error (153) message is shown.

Scenario 4: If the combination of namespace and version does not exist, an error (150) message is shown.

Worklist

Returns order data from the worklist.

Operation

Worklist

Parameters

FlexibleHeaders: A list of flexible headers available to the user. It can contain zero or more flexibleheader elements with the mnemonic paths of flexible headers to include in the output. If FlexibleHeaders is omitted, all available flexible headers are returned.

FlexibleHeader: This parameter is a sub-parameter of **FlexibleHeaders** and has the following attributes:

- `namespace` (mandatory if you use the **FlexibleHeader** element of **Worklist.Request**)
- `version` (optional)

OrderBy: The **OrderBy** element contains a list of fields on which to order the results. The order of the Field elements is significant: the results are ordered based on the first element, then the second element, and so on.

The order attribute must be either **descending** or **ascending**, otherwise an error results. The path attribute must belong to the element name of one of the fixed headers or one of the request's selected flexible headers. If the mnemonic path does not resolve to a valid field to sort by, error code 170: Header for mnemonic path not found is returned.

When sorting on fixed header elements, the sorted value for elements that are represented by an ID (that is, state) will be sorted based on the internal ID's value, not that of the final output.

Field: A value on the order, either **Fixed** or **Flexible**. This element takes the following mandatory attribute:

- `Order` (you must specify either `ascending` or `descending` as the argument.)

FilterStates: A list of the states on which the Task web client filters the Worklist. For example, if **FilterStates** only contains `accepted`, then the Worklist displays only those tasks that are in the Accepted state.

OrderState: A state for the order. This element can have multiple instances and the values indicate which states an order must be in to be returned. Acceptable values are: "Amending", "Cancelled", "Cancelling", "Completed", "In Progress", "No state", "Not Started", "Suspended", and any user-defined state states.

State: A state for the task. This element can have multiple instances and the values indicate which states a task must be in to be returned. Acceptable values are:

- **Assigned**: The task is in the Assigned state and is assigned to the current user's ID.
- **Received**: The task is in the Received state.
- **Accepted**: The task is in the Accepted state for the current user's ID.

- **Suspended**: The task is in the Suspended state.

Namespace: The namespace mnemonic of order type/source.

Version: The version mnemonic of order type/source.

UsePreferences: The element uses user-preferences from the Task web client to determine how to filter and sort the Worklist.

Request Example

```
<Worklist.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <FlexibleHeaders>
      <FlexibleHeader namespace="OrderAmendment" version="1.0"
       path="customer/phone"/>
      <FlexibleHeader namespace="OrderAmendment" version="1.0"
       path="customer/name"/>
   </FlexibleHeaders>
</Worklist.Request>
```

or

```
<Worklist.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <OrderBy>
      <Field order="ascending" path="masks_group/m_numeric_999"/>
   </OrderBy>
</Worklist.Request>
```

or

```
<Worklist.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
   <Namespace>sp_nead</Namespace>
   <Version>2.1.2</Version>
   <OrderBy>
      <Field order="descending">_order_seq_id</Field>
   </OrderBy>
</Worklist.Request>
```

Response Example

```
<Worklist.Response xmlns="urn:metasolv-com:oms:xmlapi_1">
   <Header>
      <_order_seq_id desc="Order ID" />
      <_order_hist_seq_id desc="Order History ID" />
      <_order_state desc="State" />
      <_execution_mode desc="Execution Mode" />
      <_task_id desc="Task" />
      <_order_source desc="Source" />
      <_order_type desc="Type" />
      <_current_order_state desc="Order State" />
      <_target_order_state desc="Target Order State" />
      <_reference_number desc="Ref. #" />
      <_priority desc="Priority" />
      <_user desc="User" />
      <_process_status desc="Process Status" />
      <_date_pos_started desc="Started" />
      <_requested_delivery_date desc="Requested Order Delivery Date"/>
      <_expected_start_date desc="Expected Order Start Date"/>
      <_expected_duration desc="Expected Order Duration"/>
      <_compl_date_expected desc="Expected Order Completion Date" />
      <_num_remarks desc="Number of Remarks" />
      <_namespace desc="Namespace" />
```

```
      <_version desc="Version" />
      <customer_phone desc="Customer Phone Number" />
      <customer_name desc="Customer Name" />
   </Header>
   <Orderdata>
      <_order_seq_id>390</_order_seq_id>
      <_order_hist_seq_id>7822</_order_hist_seq_id>
      <_order_state>received</_order_state>
      <_execution_mode>do</_execution_mode>
      <_task_id>assign_port</_task_id>
      <_order_source>order_entry</_order_source>
      <_order_type>pots</_order_type>
      <_current_order_state>open.not_running.not_started
      </_current_order_state>
      <_target_order_state />
      <_reference_number>SEP-27-0-1</_reference_number>
      <_priority>5</_priority>
      <_user>oms</_user>
      <_process_status/>
      <_date_pos_started>2001-10-29T10:30:24 EST</_date_pos_started>
      <_requested_delivery_date>2001-10-29T10:30:14 EST</
_requested_delivery_date>
      <_expected_start_date>2001-10-19T10:30:14 EST</_expected_start_date>
      <_expected_duration>P10D</_expected_duration>
      <_compl_date_expected>2001-10-29T10:30:16 EST</_compl_date_expected>
      <_num_remarks>0</_num_remarks>
      <_namespace>OrderAmendment</_namespace>
      <_version>1.0</_version>
      <customer_phone>4165551212</customer_phone>
      <customer_name>John Doe</customer_name>
   </Orderdata>
   <Orderdata>
      <_order_seq_id>391</_order_seq_id>
      <_order_hist_seq_id>7718</_order_hist_seq_id>
      <_order_state>accepted</_order_state>
      <_execution_mode>do</_execution_mode>
      <_task_id>assign_port</_task_id>
      <_order_source>order_entry</_order_source>
      <_order_type>pots</_order_type>
      <_current_order_state>open.running.in_progress</_current_order_state>
      <_target_order_state />
      <_reference_number>SEP-27-0-2</_reference_number>
      <_priority>5</_priority>
      <_user>oms</_user>
      <_process_status/>
      <_date_pos_created>2001-10-29T10:31:16 EST</_date_pos_created>
      <_requested_delivery_date>2001-10-29T10:30:14 EST</
_requested_delivery_date>
      <_expected_start_date>2001-10-19T10:30:14 EST</_expected_start_date>
      <_expected_duration>P10D</_expected_duration>
      <_compl_date_expected>2001-10-29T10:30:14 EST</_compl_date_expected>
      <_num_remarks>0</_num_remarks>
      <_namespace>OrderAmendment</_namespace>
      <_version>1.0</_version>
      <customer_phone>4165552121</customer_phone>
      <customer_name>Frank Smith</customer_name>
   </Orderdata>
</Worklist.Response>
```

Error Codes

- 170: Header for mnemonic path not found
- 400: Not authorized
- 401: Database connection failed
- 500: Internal error
- 601: Deprecated parameter

> **Note:**
>
> See Table 3-3 for more information if you receive an error code that is not listed here.

Warning and Error Code Descriptions

Any request can produce warnings as a side effect of accessing the OSM database. The warning element supplies the code and message of any non-fatal warnings that occurred while processing a request. Any changes to the data by the request are still committed. For more information on the cause of the warning codes, consult your Oracle DBA.

Error Codes represent request errors in the XML API that prevent further processing of a request. If an error is returned by a request, data changes are not sent to the database.

Table 3-3 lists the error codes and their descriptions.

Table 3-3 Error Code Descriptions

Error Code	Description
100: Order type/source not found	The order type/source does not exist, or is not available to the user.
110: Order not found	The order does not exist, or is not available to the user.
120: Order template not found	The order template does not exist, or is not available to the user.
160: Remark not found	The remark could not be found.
150: Namespace/version not found.	The namespace or version does not exist or is not available to the user.
152: Invalid namespace mnemonic.	The order cannot be completed with the given namespace mnemonic.
153: No legacy data found. Namespace and Version need to be supplied.	Because the legacy data could not be found you must specify a valid Namespace and Version.
170: Header for mnemonic path not found	The header for a given mnemonic path does not exist, or is not available to the user.
190: Notification not found	If the notification ID does not exist, is no longer active, or is not assigned to the user ID or user ID's workgroup.

Table 3-3 (Cont.) Error Code Descriptions

Error Code	Description
200: Order data invalid	The format of the order data is not correct. The message details the error location. For example, a message detail, such as: "com.mslv.oms.handler.OrderDataInvalidException: Could not convert input value: value_for_x for field data_field_x", may indicate that value_for_x exceeds the field's acceptable length set in OSM Administrator.
230: Order not accepted by user	An attempt to update an order was made without first retrieving the order with an Accept parameter of "true".
232: Order update failed	The order could not be updated due to a data format error. The message details the reason for failure.
250: Mandatory check failed	A mandatory field was not given a value when attempting to create, assign, complete, or suspend an order.
251: Transition invalid	The order cannot be transitioned to that state. Use ListStatesNStatuses.Request to get a list of valid states.
252: Unable to accept order	When retrieving an order for update, the order cannot be accepted by the current user.
253: User not found	The order cannot be assigned to the given user ID.
254: Invalid state mnemonic	The order cannot be suspended with the given state mnemonic. **Note**: Only user-defined states are valid. To complete or assign an order, you must use the appropriate request.
255: Invalid status mnemonic	The order cannot be completed with the given status mnemonic.
256: Invalid exception status mnemonic	You used an invalid exception status mnemonic when raising the process exception.
257: Invalid Task Mnemonic	The supplied task mnemonic could not be found.
260: Remark cannot be modified	The time interval in which a created remark can be modified has elapsed, or the user trying to change the remark is not the user who created the remark. The remark can no longer be modified.
270: Transaction not allowed	The requested transaction is not allowed. This error is returned in situations where the transaction has been disabled, or the user or workgroup is not authorized to perform the transaction. It can also be returned when attempting to suspend/resume an order that is already suspended/resumed.
300: Request unknown	The request type could not be identified from the root element of the XML document of the message.
302: Request parameter error	A parameter for the request is missing or invalid. The message details the parameter in question.
350: Pivot node data is not provided	The order cannot proceed because no pivot node is indicated.

Table 3-3 (Cont.) Error Code Descriptions

Error Code	Description
351: Process position supplied is not a sub process task.	The indicated process position is not a sub process task. To add a sub process thread, the order must reside in one of the sub processes.
352: No sub process task is currently pending.	There are no sub-process tasks to which you can add a thread.
354: Process position not found.	The indicated process position does not exist or is inaccessible by the order.
355: Pivot node not found.	The indicated pivot node does not exist or is inaccessible by the order.
356: Cannot spawn threads for sub-process tasks that support sequential sub-processing.	You cannot add a sub-process thread to a sub-process that supports sequential sub-process.
400: Not authorized	The user is not authorized to make the request.
401: Database Connection Failed.	The XML API cannot connect to OSM.
419: The process exception is restricted.	A process exception cannot be raised, because it is restricted.
420: Not authorized to modify order priority.	The user does not have the necessary privilege to modify order priority.
500: Internal error.	An internal application error has occurred. The message details further information.
601: Deprecated parameter	The parameter identified in the warning details has been deprecated. The details specify an applicable replacement parameter.

Document Type Definitions (DTD)

Document Type Definitions are markup declarations that describe the syntax for a class of documents. The DTD is declared within the document type declaration production of the XML file. The markup declarations can be in an external subset (a special kind of external entity), in an internal subset directly within the XML file, or both. The DTD for a document consists of both subsets taken together. The following is a list of the OSM DTDs.

AddOrderThread

The AddOrderThread XML API is used to implement sub-process creation, (also known as process forking), is implemented by the AddOrderThread XML API.

> **Note:**
>
> AddOrderThread has been deprecated and is supported only for backward compatibility. Use amendment processing functionality instead.

Request Example

```
<!-- add_order_thread.dtd -->
<!ELEMENT AddOrderThread.Request(OrderID, Process, ProcessPosition, View, Add)>
<!ELEMENT OrderID(#PCDATA)>
<!ELEMENT Process(#PCDATA)>
<!ELEMENT ProcessPosition(#PCDATA)>
<!ELEMENT View(#PCDATA)>
<!-- The contents of the Add element cannot be described in a dtd
Let children := mnemonic for order elements from the template -->
<!ELEMENT Add(children+)>
<!ATTLIST Add path CDATA #REQUIRED>
```

Response Example

```
<!-- add_order_thread.dtd -->
<!ENTITY % warning SYSTEM "warning.dtd">
%warning;
<!ELEMENT AddOrderThread.Response(warnings?)>
```

AssignOrder

The AssignOrder XML API is used to assign an order.

Request Example

```
<!-- assign_order_request.dtd -->
<!ELEMENT AssignOrder.Request(OrderID, OrderHistID, User)>
<!ELEMENT OrderID(#PCDATA)>
<!ELEMENT OrderHistID(#PCDATA)>
<!ELEMENT User(#PCDATA)>
```

Response Example

```
<!-- assign_order_response.dtd -->
<!ENTITY % warning SYSTEM "warning.dtd">
%warning;
<!ELEMENT AssignOrder.Response(OrderID, OrderHistID, Warnings?)>
<!ELEMENT OrderID(#PCDATA)>
<!ELEMENT OrderHistID(#PCDATA)>
```

CompleteOrder

The CompleteOrder XML API is used to complete an order.

Request Example

```
<!-- complete_order_request.dtd -->
<!ELEMENT CompleteOrder.Request(OrderID, OrderHistID, Status)>
<!ELEMENT OrderID(#PCDATA)>
<!ELEMENT OrderHistID(#PCDATA)>
<!ELEMENT Status(#PCDATA)>
```

Response Example

```
<!-- complete_order_response.dtd -->
<!ENTITY % warning SYSTEM "warning.dtd">
%warning;
<!ELEMENT CompleteOrder.Response(OrderID, OrderHistID, Warnings?)>
```

```
<!ELEMENT OrderID(#PCDATA)>
<!ELEMENT OrderHistIDEMPTY>
```

CopyOrder

The CopyOrder XML API is used to copy an order.

Request Example

```
<!-copy_order_request.dtd -->
<!ELEMENT CopyOrder.Request(OriginalOrderID, OrderType,
OrderSource, Reference, Namespace?, version?)>
<!ELEMENT OriginalOrderID(#PCDATA)>
<!ELEMENT OrderType(#PCDATA)>
<!ELEMENT OrderSource(#PCDATA)>
<!ELEMENT Reference(#PCDATA)>
<!ELEMENT Namespace(#PCDATA)>
<!ELEMENT Version(#PCDATA)>
```

Response Example

```
<!-- copy_order_response.dtd -->
<!ENTITY % warning SYSTEM "warning.dtd">
%warning;
<!ELEMENT CopyOrder.Response(OrderID, OrderHistID, OrderSource,
OrderType, OrderState, State, Reference, Priority, Warnings?, Namespace,
Version)>
<!ELEMENT OrderID(#PCDATA)>
<!ELEMENT OrderHistID(#PCDATA)>
<!ELEMENT OrderSource(#PCDATA)>
<!ELEMENT OrderType(#PCDATA)>
<!ELEMENT OrderState(#PCDATA)>
<!ELEMENT State(#PCDATA)>
<!ELEMENT Reference(#PCDATA)>
<!ELEMENT Priority(#PCDATA)>
<!ELEMENT Namespace(#PCDATA)>
<!ELEMENT Version(#PCDATA)>
```

CreateOrder

The CreateOrder XML API is used to create an order.

Request Example

```
<!-- create_order_request.dtd -->
<!ELEMENT CreateOrder.Request(OrderSource, OrderType,
 Reference, _root, Namespace?, Version?)>
<!ELEMENT OrderSource(#PCDATA)>
<!ELEMENT OrderType(#PCDATA)>
<!ELEMENT Reference(#PCDATA)>
<!ELEMENT Priority(#PCDATA)>
<!-- The contents of the _root element cannot be described in a dtd
       Let children := the mnemonics for order elements from the template -->
<!ELEMENT _root(children*)>
<!ELEMENT children(#PCDATA | children*)>
<!ELEMENT Namespace(#PCDATA)>
<!ELEMENT Version(#PCDATA)>
```

Response Example

```
<!-- create_order_response.dtd -->
<!ENTITY % warning SYSTEM "warning.dtd">
%warning;
<!ELEMENT CreateOrder.Response(OrderID, OrderHistID,
OrderSource, OrderType, OrderState, State, Reference, Priority, Warnings?,
Namespace?, Version?)>
<!ELEMENT OrderID(#PCDATA)>
<!ELEMENT OrderHistID(#PCDATA)>
<!ELEMENT OrderSource(#PCDATA)>
<!ELEMENT OrderType(#PCDATA)>
<!ELEMENT OrderState(#PCDATA)>
<!ELEMENT State(#PCDATA)>
<!ELEMENT Reference(#PCDATA)>
<!ELEMENT Priority(#PCDATA)>
<!ELEMENT Namespace(#PCDATA)>
<!ELEMENT Version(#PCDATA)>
```

Error

If an error occurs during processing for each of the following request DTDs, the following information is returned.

```
<!-- error.dtd -->
<!-- let command_name := the command name of the request
     that originated the error -->
<!ELEMENT command_name.Error(Error+)>
<!ELEMENT Error(#PCDATA)>
<!ATTLIST Error codeCDATA #REQUIRED>
<!ATTLIST Error descCDATA #REQUIRED>
```

GetOrder

The GetOrder XML API is used to retrieve an order from OSM.

Request Example

```
<!-- get_order_request.dtd -->
<!ELEMENT GetOrder.Request(OrderID, (OrderHistID | ViewID))>
<!ELEMENT OrderID(#PCDATA)>
<!ELEMENT OrderHistID(#PCDATA)>
<!ELEMENT ViewID(#PCDATA)>
<!ELEMENT Accept(#PCDATA)>
```

Response Example

```
<!-- get_order_response.dtd -->
<!ENTITY % warning SYSTEM "warning.dtd">
%warning;
<!ELEMENT GetOrder.Response(OrderID, (OrderHistID | ViewID,
OrderSource, OrderType, OrderState, State, ExecutionMode,
Reference, Priority, _root, Remarks, Warnings?)>
<!ELEMENT OrderID(#PCDATA)>
<!ELEMENT OrderHistID(#PCDATA)>
<!ELEMENT ViewID(#PCDATA)>
<!ELEMENT OrderSource(#PCDATA)>
<!ELEMENT OrderType(#PCDATA)>
<!ELEMENT Workgroups(#PCDATA)>
```

```
<!ELEMENT Workgroup(#PCDATA)>
<!ELEMENT OrderState(#PCDATA)>
<!ELEMENT State(#PCDATA)>
<!ELEMENT ExecutionMode(#PCDATA)>
<!ELEMENT Reference(#PCDATA)>
<!ELEMENT Priority(#PCDATA)>
<!-- The contents of the _root element cannot be described in a dtd
       Let children := the mnemonics for order elements from the template -->
<!ELEMENT _root(children*)>
<!ELEMENT children(#PCDATA | children*)>
<!ELEMENT Remarks(Remark*)>
<!ELEMENT Remark(RemarkID, Date, Author, TaskID,
TaskType, OrderHistID, State, Text,
ReadOnly, Attachments)>
<!ELEMENT RemarkID(#PCDATA)>
<!ELEMENT Date(#PCDATA)>
<!ELEMENT Author(#PCDATA)>
<!ELEMENT TaskID(#PCDATA)>
<!ELEMENT TaskType(#PCDATA)>
<!ELEMENT State(#PCDATA)>
<!ELEMENT Text(#PCDATA)>
<!ELEMENT ReadOnly(#PCDATA)>
<!ELEMENT Attachments(Attachment*)>
<!ELEMENT Attachment(AttachmentID, FileName)>
<!ELEMENT AttachmentID(#PCDATA)>
<!ELEMENT FileName(#PCDATA)>
```

GetNextOrderAtTask

The GetNextOrderAtTask XML API is used to retrieve the next order.

Request Example

```
<!-get_next_order_at_task_request.dtd -->
<!ELEMENT GetNextOrderAtTask.Request(TaskID, Accept, State+, Namespace?,
Version?)>
<!ELEMENT TaskID(#PCDATA)>
<!ELEMENT Accept(#PCDATA)>
<!ELEMENT State(#PCDATA)>
<!ELEMENT Namespace(#PCDATA)>
<!ELEMENT Version(#PCDATA)>
```

Response Example

```
<!-get_next_order_at_task_response.dtd -->
<!ENTITY % warning SYSTEM "warning.dtd">
%warning;
<!ELEMENT GetNextOrderAtTask.Response((OrderID, OrderHistID, OrderSource,
OrderType, OrderState, State, ExecutionMode, Reference, Priority, _root,
Warnings?, Namespace?, Version?) | EMPTY)>
<!ELEMENT OrderID(#PCDATA)>
<!ELEMENT OrderHistID(#PCDATA)>
<!ELEMENT OrderSource(#PCDATA)>
<!ELEMENT OrderType(#PCDATA)>
<!ELEMENT Workgroups(#PCDATA)>
<!ELEMENT Workgroup(#PCDATA)>
<!ELEMENT OrderState(#PCDATA)>
<!ELEMENT State(#PCDATA)>
<!ELEMENT ExecutionMode(#PCDATA)>
<!ELEMENT Reference(#PCDATA)>
```

```
<!ELEMENT Priority(#PCDATA)>
<!-- The contents of the _root element cannot be described in a dtd
       Let children := the mnemonics for order elements from the template -->
<!ELEMENT _root(children*)>
<!ELEMENT children(#PCDATA | children*)>
<!ELEMENT Namespace(#PCDATA)>
<!ELEMENT Version(#PCDATA)>
```

GetOrderDataHistory

The GetOrderDataHistory XML API is used to retrieve order data history for an order.

Request Example

```
<!-order_data_history_resquest.dtd -->
<!ELEMENT GetOrderDataHistory.Request(OrderID, (OrderHistID |
ViewID), Field*)>
<!ELEMENT OrderID(#PCDATA)>
<!ELEMENT OrderHistID(#PCDATA)>
<!ELEMENT ViewID(#PCDATA)>
<!ELEMENT FieldEMPTY>
<!ATTLIST Field pathCDATA #REQUIRED>
```

Response Example

```
<!-order_data_history_response.dtd -->
<!ENTITY % warning SYSTEM "warning.dtd">
%warning;
<!ELEMENT GetOrderDataHistory.Response(OrderID, (OrderHistID |
ViewID), Field*)>
<!ELEMENT OrderID(#PCDATA)>
<!ELEMENT OrderHistID(#PCDATA)>
<!ELEMENT ViewID(#PCDATA)>
<!ELEMENT Field(Change+)>
<!ATTLIST Field pathCDATA #REQUIRED>
<!ATTLIST Field indexCDATA #REQUIRED>
<!ATTLIST Field parentIndexCDATA #REQUIRED>
<!ELEMENT Change(#PCDATA | EMPTY)>
<!ATTLIST Change action(create | update | delete) #REQUIRED>
<!ATTLIST Change userCDATA #REQUIRED>
<!ATTLIST Change timeCDATA #REQUIRED>
```

GetOrderProcessHistory

The GetOrderProcessHistory XML API is used to retrieve the process history for an order.

Request Example

```
<!-order_process_history_request.dtd -->
<!ELEMENT GetOrderProcessHistory.Request(OrderID)>
<!ELEMENT OrderID(#PCDATA)>
```

Response Example

```
<!-order_process_history_response.dtd -->
<!ENTITY % warning SYSTEM "warning.dtd">
%warning;
<!ELEMENT GetOrderProcessHistory.Response(OrderID, Summary,
```

```
Transitions)>
<!ELEMENT OrderID(#PCDATA)>
<!ELEMENT Summary(ExpectedDuration,
ActualDuration,
StartDate,
CompleteDate)>
<!ELEMENT ExpectedDuration(#PCDATA)>
<!ELEMENT ActualDuration(#PCDATA)>
<!ELEMENT StartDate(#PCDATA)>
<!ELEMENT CompleteDate(#PCDATA)>
<!ELEMENT Transitions(Transition*)>
<!ELEMENT Transition(TaskID, TaskType,
TaskDescription,
ExpectedDuration,
ActualDuration,
StartDate,
CompleteDate,
OrderHistID,
FromOrderHistID, State,
Status, TransitionType, user, ParentTaskOrderHistID, DataNodeIndex,
DataNodeMnemonic,
DataNodeValue)>
<!ELEMENT TaskID(#PCDATA)>
<!ELEMENT TaskType(#PCDATA)>
<!ELEMENT TaskDescription(#PCDATA)>
<!ELEMENT ExpectedDuration(#PCDATA)>
<!ELEMENT ActualDuration(#PCDATA)>
<!ELEMENT StartDate(#PCDATA)>
<!ELEMENT CompleteDate(#PCDATA)>
<!ELEMENT OrderHistID(#PCDATA)>
<!ELEMENT FromOrderHistID(#PCDATA)>
<!ELEMENT State(#PCDATA)>
<!ELEMENT Status(#PCDATA)>
<!ELEMENT TransitionType(#PCDATA)>
<!ELEMENT User(#PCDATA)>
<!ELEMENT SubProcessParentTaskOrderHistID(#PCDATA)>
<!ELEMENT DataNodeIndex(#PCDATA)>
<!ELEMENT DataNodeMnemonic(#PCDATA)>
<!ELEMENT DataNodeValue(#PCDATA)>
```

GetOrderStateHistory

The GetOrderStateHistory XML API is used to retrieve the order state history for an order.

Request Example

```
<!-- get_order_state_history.dtd -->
<!ELEMENT GetOrderStateHistory.Request(OrderID)>
<!ELEMENT OrderID(#PCDATA)>
```

Response Example

```
<!-- get_order_state_history.dtd -->
<!ELEMENT OrderID(#PCDATA)>
<!ELEMENT OrderState(#PCDATA)>
<!ELEMENT TransitionStartDate(#PCDATA)>
<!ELEMENT TransitionCompletedDate(#PCDATA)>
<!ELEMENT ActualDuration(#PCDATA)>
```

```
<!ELEMENT User(#PCDATA)>
<!ELEMENT Reason(#PCDATA)>
```

GetUserInfo

The GetUserInfo XML API is used to retrieve user information.

Request Example

```
<!-user_info_request.dtd -->
<!ELEMENT GetUserInfo.RequestEMPTY>
```

Response Example

```
<!-user_info_response.dtd -->
<!ENTITY % warning SYSTEM "warning.dtd">
%warning;
<!ELEMENT GetUserInfo.Response(User, Workgroup*FlexibleHeaders)>
<!ELEMENT User(#PCDATA)>
<!ATTLIST User descCDATA #IMPLIED>
<!ELEMENT Workgroup(#PCDATA)>
<!ATTLIST Workgroup desc CDATA #IMPLIED>
<!ELEMENT FlexibleHeaders((FlexibleHeader*)>
<!ELEMENT FlexibleHeader(#PCDATA)>
<!ATTLIST FlexibleHeader descCDATA #REQUIRED>
```

ListExceptions

The ListExceptions XML API is used to retrieve exceptions.

Request Example

```
<!-- list_exceptions_request.dtd -->
<!ELEMENT ListExceptions.Request(OrderID, OrderHistID)>
<!ELEMENT OrderID(#PCDATA)>
<!ELEMENT OrderHistID(#PCDATA)>
```

Response Example

```
<!-- list_exceptions_response.dtd -->
<!ENTITY % warning SYSTEM "warning.dtd">
%warning
<!ELEMENT ListExceptions.Response(OrderID, OrderHistID,
 Exceptions, Warnings?)>
<!ELEMENT OrderID(#PCDATA)>
<!ELEMENT OrderHistID(#PCDATA)>
<!ELEMENT Exceptions(Status*)>
<!ELEMENT Status(#PCDATA)>
<!ATTLIST Status descCDATA #REQUIRED>
```

ListStatesNStatuses

The ListStatesNStatuses XML API is used to retrieve the states and statuses.

Request Example

```
<!-- task_state_status_request.dtd -->
<!ELEMENT ListStatesNStatuses.Request(OrderID, OrderHistID)>
```

```
<!ELEMENT OrderID(#PCDATA)>
<!ELEMENT OrderHistID(#PCDATA)>
```

Response Example

```
<!-task_state_status_response.dtd -->
<!ENTITY % warning SYSTEM "warning.dtd">
%warning;
<!ELEMENT ListStatesNStatuses.Response(OrderID, OrderHistID,
TaskStatesNStatuses,
Warnings?)>
<!ELEMENT OrderID(#PCDATA)>
<!ELEMENT OrderHistID(#PCDATA)>
<!ELEMENT TaskStatesNStatuses(Accepted?, Assigned?, Suspend?, Completed?)>
<!ELEMENT AcceptedEMPTY>
<!ELEMENT Assigned(User+)>
<!ELEMENT User(#PCDATA)>
<!-- The content of the Suspend element cannot be described in a dtd
        Let userstates := mnemonics for user defined states -->
<!ELEMENT Suspend(userstates+)>
<!-- The content of the Completed element cannot be described in a dtd
        Let userstatuses := mnemonics for user defined statuses -->
<!ELEMENT Completed(userstatuses+)>
```

ListViews

The ListViews XML API is used to retrieve views.

Request Example

```
<!-- views_request.dtd -->
<!ELEMENT ListViews.Request(OrderSource, OrderType, Namespace?, Version?)>
<!ELEMENT OrderSource(#PCDATA)>
<!ELEMENT OrderType(#PCDATA)>
<!ELEMENT Namespace(#PCDATA)>
<!ELEMENT Version(#PCDATA)>
```

Response Example

```
<!-- views_response.dtd -->
<!ENTITY % warning SYSTEM "warning.dtd">
%warning;
<!ELEMENT ListViews.Response(OrderSource, OrderType, View*, Warnings?,
Namespace?, Version?)>
<!ELEMENT OrderSource(#PCDATA)>
<!ELEMENT OrderType(#PCDATA)>
<!ELEMENT View(#PCDATA)>
<!ATTLIST View desc    CDATA #REQUIRED>
<!ELEMENT Namespace(#PCDATA)>
<!ELEMENT Version(#PCDATA)>
```

ModifyRemark

The ModifyRemark XML API is used to modify remarks for an order.

Request Example

```
<!-modify_remark_request.dtd -->
<!ELEMENT ModifyRemark.Request(OrderID, OrderHistID, RemarkID, Text?,
Attachments?, DeleteAttachments?)>
```

```
<!ELEMENT OrderID(#PCDATA)>
<!ELEMENT OrderHistID(#PCDATA)>
<!ELEMENT RemarkID(#PCDATA)>
<!ELEMENT Text(#PCDATA)>
<!ELEMENT Attachments(FileName*)>
<!ELEMENT FileName(#PCDATA)>
<!ELEMENT DeleteAttachments(AttachmentID*)>
<!ELEMENT AttachmentID(#PCDATA)>
```

Response Example

```
<!-modify_remark_response.dtd -->
<!ENTITY % warning SYSTEM "warning.dtd">
%warning;
<!ELEMENT ModifyRemark.Response(Remark?, Warnings?)>
<!ELEMENT Remark(RemarkID, Attachment*)>
<!ELEMENT RemarkID(#PCDATA)>
<!ELEMENT Attachment(AttachmentID, FileName)>
<!ELEMENT AttachmentID(#PCDATA)>
<!ELEMENT FileName(#PCDATA)>
```

OrderTypeNSource

The OrderTypeNSource XML API is used for the order type and source.

Request Example

```
<!-- order_source_type_request.dtd -->
<!ELEMENT OrderTypesNSources.RequestEMPTY, Namespace?, Version?>
<!ELEMENT Namespace(#PCDATA)>
<!ELEMENT Version(#PCDATA)>
```

Response Example

```
<!-- order_source_type_request.dtd -->
<!ENTITY % warning SYSTEM "warning.dtd">
%warning;
<!ELEMENT OrderTypesNSources.Response(TypeNSource*, Warnings?, Namespace?,
Version?)>
<!ELEMENT TypeNSource(Source, Type)>
<!ELEMENT SourceEMPTY>
<!ATTLIST Source mnemonicCDATA #REQUIRED>
<!ATTLIST Source descCDATA #IMPLIED>
<!ELEMENT TypeEMPTY>
<!ATTLIST Type mnemonicCDATA #REQUIRED>
<!ATTLIST Type categoryCDATA #IMPLIED>
<!ATTLIST Type descCDATA #IMPLIED>
<!ELEMENT Namespace(#PCDATA)>
<!ELEMENT Version(#PCDATA)>
```

OrderViewTemplate

The OrderViewTemplate XML API is used for the order view template.

Request Example

```
<!-- order_template_request.dtd -->
<!ELEMENT OrderViewTemplate.Request((OrderSource, OrderType) | (OrderID,
OrderHistID) | (OrderSource, OrderType, ViewID), Namespace?, Version?)>
<!ELEMENT OrderSource(#PCDATA)>
```

```
<!ELEMENT OrderType(#PCDATA)>
<!ELEMENT OrderID(#PCDATA)>
<!ELEMENT OrderHistID(#PCDATA)>
<!ELEMENT ViewID(#PCDATA)>
<!ELEMENT Namespace(#PCDATA)>
<!ELEMENT Version(#PCDATA)>
```

Response Example

```
<!--order_template_response.dtd -->
<!ENTITY % warning SYSTEM "warning.dtd">
%warning;
<!ELEMENT OrderViewTemplate.Response(((OrderSource, OrderType) |
(OrderID, OrderHistID) | (OrderSource, OrderType, ViewID)),
 _root, LookupTables?, Warnings?, Namespace?, Version?)>
<!ELEMENT OrderSource(#PCDATA)>
<!ELEMENT OrderType(#PCDATA)>
<!ELEMENT OrderID(#PCDATA)>
<!ELEMENT OrderHistID(#PCDATA)>
<!ELEMENT ViewID(#PCDATA)>
<!-- The children of _root cannot be described in a dtd.
        Let children := the mnemonic of each node in the order template -->
<!ELEMENT _root(children*)>
<!ELEMENT childrenEMPTY>
<!ATTLIST children descCDATA #REQUIRED>
<!ATTLIST children minInstanceCDATA "0">
<!ATTLIST children maxInstanceCDATA "1">
<!ATTLIST children typeCDATA (NM | TX | DT | D | PH | YN | CY | LK) "TX">
<!ATTLIST children readOnly(true | false) "false">
<!-- the mask attribute is only applicable if type = NM or TX and there is a
mask defined -->
<!ATTLIST children maskCDATA #IMPLIED>
<!-- the len attribute is only applicable if type = NM or TX and a mask is not
defined -->
<!ATTLIST children lenCDATA #IMPLIED>
<!-- The children of LookupTables cannot be described in a dtd.
        Let lkchildren := the element name of children[type="LK"]
<!ELEMENT LookupTables(lkchildren+)>
<!ELEMENT lkchildren(option+)>
<!ELEMENT option(#PCDATA)>
<!ATTLIST option descCDATA #REQUIRED>
<!ELEMENT Namespace(#PCDATA)>
<!ELEMENT Version(#PCDATA)>
```

Query

The Query XML API is used to query an order.

Request Example

```
<!-- query_request.dtd -->
<!ELEMENT Query.Request(OrderID?, Reference?, Priority? OrderType?,
 OrderSource?, SingleRow?, TaskID?, CreatedDate?, CompletedDate?, Field*)*,
FlexibleHeaders?, Namespace?, Version?)>
<!ELEMENT OrderID(#PCDATA)>
<!ELEMENT Reference(#PCDATA)>
<!ELEMENT Priority(#PCDATA)>
<!ELEMENT OrderType(#PCDATA)>
<!ELEMENT OrderSource(#PCDATA)>
<!ELEMENT SingleRow(#PCDATA)>
```

```
<!ELEMENT TaskID(#PCDATA)>
<!ELEMENT OrderState(#PCDATA)>
<!ELEMENT TargetState(#PCDATA)>
<!ELEMENT ExecutionMode(#PCDATA)>
<!ELEMENT CreatedDateEMPTY>
<!ATTLIST CreatedDate fromCDATA #IMPLIED>
<!ATTLIST CreatedDate toCDATA #IMPLIED>
<!ELEMENT CompletedDateEMPTY>
<!ELEMENT CompletedDate fromCDATA #IMPLIED>
<!ELEMENT CompletedDate toCDATA #IMPLIED>
<!ELEMENT Field(#PCDATA | From, To)>
<!ATTLIST Field pathCDATA #REQUIRED>
<!ELEMENT From(#PCDATA)>
<!ELEMENT To(#PCDATA)>
<!ELEMENT FlexibleHeaders(FlexibleHeader*)>
<!ELEMENT FlexibleHeader(#PCDATA)>
<!ELEMENT Namespace(#PCDATA)>
<!ELEMENT Version(#PCDATA)>
```

Response Example

```
<!-- query_response.dtd -->
<!ENTITY % warning SYSTEM "warning.dtd">
%warning
<!ELEMENT Query.Response(Header, Orderdata*, Warnings?, Namespace?, Version?)>
<!-- The contents of Header cannot be described in a dtd.
       Let children := the mnemonic path of all flexible headers for the user
-->
<!ELEMENT Header(_order_seq_id, _order_hist_seq_id,
_date_pos_created, _date_pos_started, _task_id,
_order_type,_order_source, _order_state,
_process_description, _reporting_status,
_reference_number, _priority, _user, _num_remarks, children*)>
<!-- Let headerchild := each of the children of Header -->
<!ELEMENT headerchildEMPTY>
<!ATTLIST headerchild descCDATA #IMPLIED>

<!-- The contents of Orderdata cannot be described in a dtd.
       Let orderdatachildren := the mnemonic path of all flexible headers for
the user -->
<!ELEMENT Orderdata(_order_seq_id, _order_hist_seq_id,
_date_pos_created, _date_pos_started,
_task_id, _order_type, _order_source,
_order_state, _process_description,
_reporting_status, _reference_number, _user,
_num_remarks,orderdatachildren*)>
<!-- Let ordedatachild := each of the children of Orderdata -->
<!ELEMENT orderdatachild(#PCDATA)>
<!ELEMENT Namespace(#PCDATA)>
<!ELEMENT Version(#PCDATA)>
```

ResumeOrder

The ResumeOrder XML API is used to resume an order.

Request Example

```
<!-- resume_order_request.dtd -->
<!ELEMENT ResumeOrder.Request(OrderID, Reason)>
```

```
<!ELEMENT OrderID(#PCDATA)>
<!ELEMENT Reason(#PCDATA)>
```

Response Example

```
<!-- resume_order_response.dtd -->
<!ENTITY % warning SYSTEM "warning.dtd">
%warning;
<!ELEMENT ResumeOrder.Response(OrderID, Reason, Warnings?)>
<!ELEMENT OrderID(#PCDATA)>
<!ELEMENT Reason(#PCDATA)>
```

SetException

The SetException XML API is used to set an exception.

Request Example

```
<!-- set_exception_request.dtd -->
<!ELEMENT SetException.Request(OrderID, OrderHistID, Status)>
<!ELEMENT OrderID(#PCDATA)>
<!ELEMENT OrderHistID(#PCDATA)>
<!ELEMENT Status(#PCDATA)>
```

Response Example

```
<!-- set_exception_response.dtd -->
<!ELEMENT SetException.Response(OrderID, OrderHistID, Warnings?)>
<!ELEMENT OrderID(#PCDATA)>
<!ELEMENT OrderHistID(#PCDATA)>
```

SuspendOrder

The SuspendOrder XML API is used to suspend an order.

Request Example

```
<!-- suspend_order_request.dtd -->
<!ELEMENT SuspendOrder.Request(OrderID, Reason, OrderHistID, State)>
<!ELEMENT OrderID(#PCDATA)>
<!ELEMENT Reason(#PCDATA)>
<!ELEMENT OrderHistID(#PCDATA)>
<!ELEMENT State(#PCDATA)>
```

Response Example

```
<!-- suspend_order_response.dtd -->
<!ENTITY % warning SYSTEM "warning.dtd">
%warning;
<!ELEMENT SuspendOrder.Response(OrderID, OrderHistID, Warnings?)>
<!ELEMENT OrderID(#PCDATA)>
<!ELEMENT OrderHistID(#PCDATA)>
```

TaskDescription

The TaskDescription XML API is used for the task description.

Request Example

```
<!-- task_description_request.dtd -->
<!ELEMENT TaskDescription.RequestEMPTY, Namespace?, Version?>
<!ELEMENT Namespace(#PCDATA)>
<!ELEMENT Version(#PCDATA)>
```

Response Example

```
<!-- task_description_response.dtd -->
<!ENTITY % warning SYSTEM "warning.dtd">
%warning;
<!ELEMENT TaskDescription.Response(Task, Warnings?, Namespace?, Version?)>
<!ELEMENT TaskEMPTY>
<!ATTLIST Task mnemonicCDATA #REQUIRED>
<!ATTLIST Task taskTypeCDATA (automatic | manual | creation | rule | delay)
#REQUIRED>
<!ATTLIST Task descCDATA #REQUIRED>
<!ELEMENT Namespace(#PCDATA)>
<!ELEMENT Version(#PCDATA)>
```

UpdateOrder

The UpdateOrder XML API is used to update an order.

Request Example

```
<!-- update_order_request.dtd -->
<!ELEMENT UpdateOrder.Request(OrderID, ViewID, NewReference?, NewPriority? Add*,
Delete*, Update*, AddRemark?)>
<!ELEMENT OrderID(#PCDATA)>
<!ELEMENT OrderHistID(#PCDATA)>
<!ELEMENT NewReference(#PCDATA)>
<!ELEMENT NewPriority(#PCDATA)>
<!-- The contents of the Add element cannot be described in a dtd
        Let children := mnemonic for order elements from the template -->
<!ELEMENT Add(children*)>
<!ATTLIST Add pathCDATA #REQUIRED>
<!ELEMENT DeleteEMPTY>
<!ATTLIST Delete pathCDATA #REQUIRED>
<!-- The contents of the Updateelement cannot be described in a dtd
        Let children := mnemonic for order elements from the template -->
<!ELEMENT Update(children* | #PCDATA)>
<!ATTLIST Update pathCDATA #REQUIRED>
<!ELEMENT AddRemark(Text, Attachments?)>
<!ELEMENT Text(#PCDATA)>
<!ELEMENT Attachments(FileName*)>
<!ELEMENT FileName(#PCDATA)>
```

Response Example

```
<!-- update_order_response.dtd -->
<!ENTITY % warning SYSTEM "warning.dtd">
%warning;
<!ELEMENT UpdateOrder.Response(Remark?, Warnings?)>
<!ELEMENT Remark(RemarkID, Attachment*)>
<!ELEMENT RemarkID(#PCDATA)>
<!ELEMENT Attachment(AttachmentID, FileName)>
```

```
<!ELEMENT AttachmentID(#PCDATA)>
<!ELEMENT FileName(#PCDATA)>
```

Warning

The warning DTD is included by all response documents.

```
<!-- warning.dtd -->
<!ELEMENT Warnings(Warning+)>
<!ELEMENT Warning(#PCDATA)>
<!ATTLIST Warning codeCDATA #REQUIRED>
<!ATTLIST Warning descCDATA #REQUIRED>
```

Worklist

The Worklist XML API is used for worklists.

Request Example

```
<!-- worklist_request.dtd -->
<!ELEMENT Worklist.Request(FlexibleHeaders?, Namespace?, Version?)>
<!ELEMENT FlexibleHeaders(FlexibleHeader*)>
<!ELEMENT FlexibleHeader(#PCDATA)>
<!ELEMENT Namespace(#PCDATA)>
<!ELEMENT Version(#PCDATA)>
```

Response Example

```
<!-- worklist_response.dtd -->
<!ENTITY % warning SYSTEM "warning.dtd">
%warning;
<!ELEMENT Worklist.Response(Header, OrderData*, Warnings?, Namespace?, Version?)>
<!-- The contents of Header cannot be described in a dtd.
       Let children := the mnemonic path of all flexible headers specified in
the request -->
<!ELEMENT Header(_order_seq_id, _order_hist_seq_id, _date_pos_created,
  _date_pos_started, _task_id, _order_type, _order_source, _order_state,
  _execution_mode, _process_description, _current_order_state,
  _target_order_state, _reporting_status, _reference_number, _priority,
  _user, _num_remarks,children*)>
<!-- Let headerchild := each of the children of Header -->
<!ELEMENT headerchildEMPTY>
<!ATTLIST headerchild descCDATA #IMPLIED>

<!-- The contents of Orderdata cannot be described in a dtd.
       Let orderdatachildren := the mnemonic path of all flexible headers
specified in the request -->
<!ELEMENT Orderdata(_order_seq_id, _order_hist_seq_id, _date_pos_created,
  _date_pos_started, _task_id, _order_type, _order_source, _order_state,
  _execution_mode, _process_description,_current_order_state,
_target_order_state,
  _reporting_status, _reference_number, _priority, _user,_num_remarks,
  orderdatachildren*)>
<!-- Let ordedatachild := each of the children of Orderdata -->
<!ELEMENT orderdatachild(#PCDATA)>
<!ELEMENT Namespace(#PCDATA)>
<!ELEMENT Version(#PCDATA)>
```

4

Using OSM Security Callback

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, see the Javadocs located in the OSM SDK at **/SDK/osm7.***w.x.y.z***-javadocs.zip** (where *w.x.y.z* represents the specific version numbers for OSM). See *OSM Installation Guide* for more information about installing the OSM SDK for traditional OSM.

```
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 Collection<String> checkOrderSummaryAccess
                            (UserSummaryInfo userInfo,
                             Collection<OrderSummaryInfo> ordersInfo
```

```
                                    ) throws OrderViewAccessNotAllowedException;
}
```

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 SDK in the **SDK/Samples/SecurityCallback** directory.

```
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 acessing 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
     *             This embeds any custom code application exceptions. The
exception to differentiate unexpected exception that may be
     *             occurring in custom code.
     *
```

```
     * @see
com.mslv.oms.security.OrderViewAccessProvider#checkOrderViewDetail(java.lang.Stri
ng, java.lang.String, java.lang.String,
     *      java.lang.String, java.lang.String, java.lang.String,
java.lang.String)
     */
    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 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 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.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());

            }
            final Map<String, String> flexHeadersWithValues =
order.getFlexibleHeaders();
            if (LOG.isInfoEnabled()) {
                LOG.info("FlexibleHeaders in the form {(MnemonicPath=Value)} " +
flexHeadersWithValues);
            }

        }

        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. See Unpacking the oms.ear File for more information about unpacking the **oms.ear** file.

2. Register the callback:

   - For traditional OSM, register through the **oms-config.xml** file.

     ```
     <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.

- For OSM cloud native, specify the following:

  ```
  omsConfig:
      com.mslv.oms.security.OrderViewAccessProvider:
  callbackexamples.MyViewAccessCallback
  ```

  See the "Configuring Parameters" section in "Chapter 6 Creating Your Own OSM Cloud Native Instance" of *OSM Cloud Native Deployment Guide* for further details on modifying oms-config parameters.

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. See Packing the oms.ear File for more information about packing the **oms.ear** file.
6. Do one of the following:
   - For traditional OSM, redeploy the **oms.ear**.
   - For OSM cloud native, rebuild the OSM container image. See the "Deploying Entities to an OSM WebLogic Domain" section in "Chapter 7 Extending the WebLogic Server Deploy Tooling (WDT) Model" in *OSM Cloud Native Deployment Guide*.

5

Using Custom Menu Items and Actions

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. Refresh the server cache in the Administration area of the OSM Order Management web client (or an Ant task in the Cartridge Development Kit) to trigger a reload of the configuration file.

The name and location of the custom menu action file are configurable parameters. For traditional OSM, these are available in the **oms-config.xml** file. In OSM cloud native, these can be set in the specification files. See the "Configuring Parameters" section in "Chapter 6 Creating Your Own OSM Cloud Native" Instance of *OSM Cloud Native Deployment Guide* for details on working with osm-config parameters in cloud native.

A working model, which includes a sample configuration file, Javascript file, and ReadMe, is available in the **SDK/Samples/CustomMenuAndAction** directory.

About the Model Definition

The definition of the model must follow the XML schema **menuAction.xsd** located in the **SDK/XMLImportExport/models** directory. The `action` and `menuItem` elements are described below.

Action Definition

Table 5-1 lists the action elements.

Table 5-1 Action Elements

Element	Description
name	The name of the action referenced by the menu item.
xsi:type	There are three types of actions: javascriptActionType - Defines a Javascript function as part of the <implementation> 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 <implementation> element. orderContextActionType - Similar to javascriptActionType, 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 <implementation> contains a call to that function, pass orderContext as a parameter to the function. uriActionType - Forwards you to the supplied URI, which opens in a new window in the browser. The URI is supplied as part of the <implementation> element.
description	The description of the action that appears on the context menu when no menu item description is supplied.
hint	The tool tip associated with the action.
icon	The icon associated with the action. Icons must be packed as part of the **oms.ear** file (oms.ear/oms.war/images).
implementation	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.
uri	The path to a local directory, web page address, or any point of content.

OrderContext and Orders

An **orderContextActionType** 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 5-2 shows the method calls that can be made on the **orderContext** and **order objects**.

Table 5-2 orderContext and Orders

Object	Methods	Description
orderContext	getOrders()	Call this method to get the selected orders.
order	getOrderId()	Call this method to get the order ID for the order.
order	getOrderHistId()	Call this method to get the order history ID for the order.

Table 5-2 (Cont.) orderContext and Orders

Object	Methods	Description
order	getOrderTypeId()	Call this method to get the order type ID for the order.
order	getOrderSourceId()	Call this method to get the order source ID for the order.
order	getState()	Call this method to get the state of the order.

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.

```
<action name="get_worklist_through_xml_api" xsi:type="javascriptActionType">
    <description>Get worklist though 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 5-3 shows the elements of the menu item definition.

Table 5-3 Menu Item Elements

Attribute	Description
name	The name of the menu item (internal reference only).
description	The description of the menu item that appears on the context menu.
enabled	Set to **true()**, **true**, **yes**, or **y** (case-insensitive) to enable the menu item, or set to anything else to disable it.
visible	Set to **true()**, **true**, **yes**, or **y** (case-insensitive) to make the menu item visible, or set to anything else to make the menu item invisible.
displayStyle	The display style of the menu item on the context menu, either **ICON**, or **TEXT**, or both (**ICON TEXT**). References the action icon and/or description.
action	Reference to the action being called.

Sample Menu Item Definition

```
<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 (Traditional OSM Only)" 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

Setting Up the oms-config.xml File (Traditional OSM Only)

All three methods of environment configuration require that you set up the **oms-config.xml** 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-value>
</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-value>
</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 overwritten 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).

Working with oms-config Parameters in OSM Cloud Native

In OSM cloud native, all oms-config parameters can be updated in the specification files. The parameter name and value can be set in either the shape, instance, or project specification files. For more details, see *OSM Cloud Native Deployment Guide*.

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 (Traditional OSM Only)."
4. Do one of the following:
   - For traditional OSM, deploy the **oms.ear** file that contains your **oms-config.xml** changes to the environment.
   - For OSM cloud native, create or update your OSM instance with the new oms config parameters. See the "Configuring Parameters" section in "Chapter 6 Creating Your Own OSM Cloud Native Instance" of *OSM Cloud Native Deployment Guide*.

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 (Traditional OSM Only)."
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 (Traditional OSM Only)." In the example, you would copy the files to the **resources** directory for your cartridge.
5. Build and deploy the cartridge.
6. Do one of the following:
   - For traditional OSM, deploy the **oms.ear** file that contains your **oms-config.xml** changes to the environment.
   - For OSM cloud native, create or update your OSM instance with the new oms config parameters. See the "Configuring Parameters" section in "Chapter 6 Creating Your Own OSM Cloud Native Instance" of *OSM Cloud Native Deployment Guide*.

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 (Traditional OSM Only)" 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"/>
   ```

   > **Note:**
   >
   > File re-write is not supported in OSM cloud native.

7. Build and deploy the cartridge.
8. Do one of the following:
   - For traditional OSM, deploy the **oms.ear** file that contains your **oms-config.xml** changes to the environment.
   - For OSM cloud native, create or update your OSM instance with the new oms config parameters. See the "Configuring Parameters" section in "Chapter 6 Creating Your Own OSM Cloud Native Instance" of *OSM Cloud Native Deployment Guide*.

Verifying the Changes

1. If OSM was running when you made the changes to set up the environment, refresh the server cache in the Administration area of the OSM Order Management web client 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.

6
Using Automation

This chapter describes the Oracle Communications Order and Service Management (OSM) automation framework, 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 notifications at the order level, you write automation plug-ins. The automation framework runs instances of 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 instance 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.

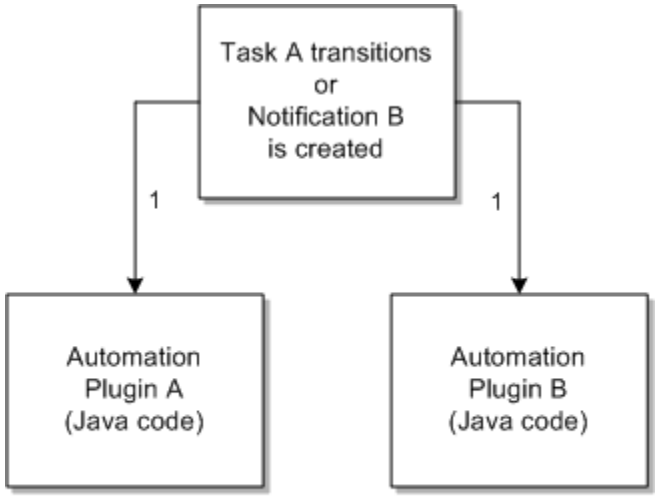
1. After processing the request, the external system copies the properties from the incoming request to the outgoing response.
2. The external system sends the response message to the reply to queue based on the replyTo JMS property in the request.
3. The automation framework routes the response from 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 6-1 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 6-1 Automation Flow



1 – Task A Transitions to the Received State or Notification B is trigger based on a notification event.

About Sender and Automator Automation Types

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.

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.

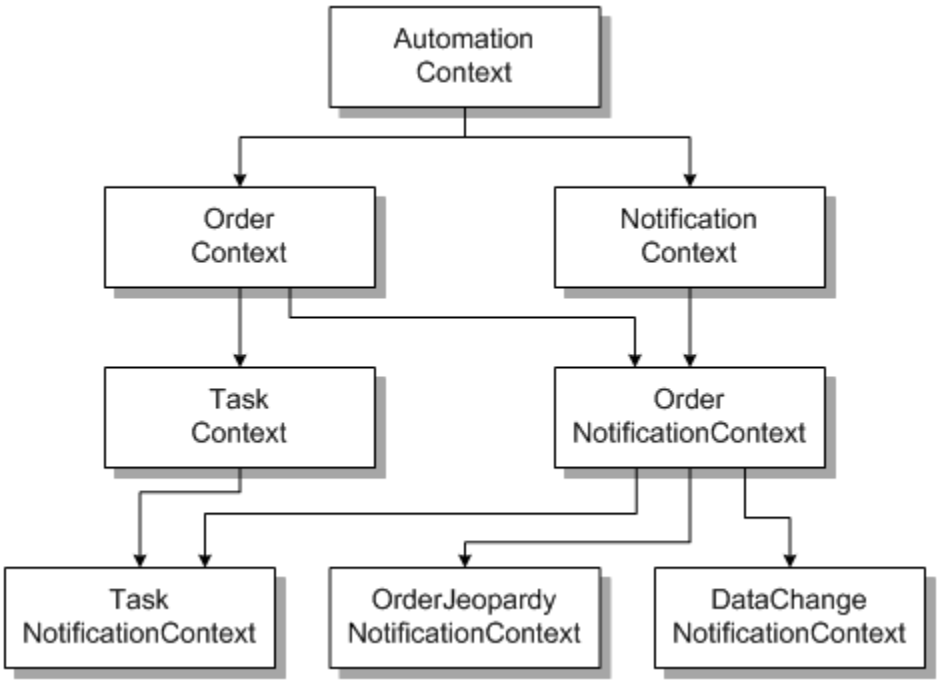
When you create custom automations, you can access these contexts from the OSM Java API **com.mslv.automation.oms.AutomationContext** class which is the parent class of **OrderContext** which is in turn the parent of the **TaskContext**. These are either parent or sibling classes for all the other contexts. 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 6-2 shows the class hierarchy stemming from the AutomationContext.

Figure 6-2 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 6-1 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 6-1 Context Objects Passed To Plug-in

Automation Plug-in Trigger	Design Studio Definition Location	OSM Event	OSM Event Type	Context Object Passed To Plug-in
Automated task	Task editor, Automation tab	Task state transitions to *Received*	Task Event	TaskContext
Order milestone-based event notification	Order editor, Events tab	Order reaches specified milestone	Order Notification Event	OrderNotificationContext
Task state-based event notification	Task editor, Events tab	Task reaches specified state	Task Notification Event	TaskNotificationContext
Task state-based event notification	Process editor, Events tab on Properties view of a task in the process	Task reaches specified state, then data condition specified by rule evaluates to true.	Task Notification Event	TaskNotificationContext
Task status-based event notification	Process editor, Events tab on Properties view of a status in the process	Task reaches specified status, then data condition specified by rule evaluates to true.	Task Notification Event	TaskNotificationContext
Order data changed event notification	Order editor, Notifications tab	Specified order data changes.	Order Notification Event	OrderDataChangeNotificationContext
Order jeopardy notification	Order Jeopardy editor	The timer conditions for the jeopardy have been reached.	System Notification Event	OrderJeopardyNotificationContext
Order jeopardy notification	Order editor, Jeopardy tab	At polling, data condition defined by rule evaluates to true.	System Notification Event	OrderNotificationContext
Task jeopardy notification	Task editor, Jeopardy tab	At polling, data condition defined by rule evaluates to true.	System Notification Event	If the task-level jeopardy condition **Multiple events per Task instance** is set, then TaskNotificationContext is passed. Otherwise OrderNotificationContext is passed.

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

About Internal and External Events that Trigger Automations

You must also define where you expect the sender or automator plug-in to receive its data when you set the plug-in **Event Type**, which specifies whether the plug-in instance 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 **oms_events** message queue that the OSM installer creates during the installation process. OSM maps order priority 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 or you can use the **oms_events** queue that OSM uses for internal message processing.
  - 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.

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 Queues, Correlation, and Property Selectors

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.

> **Note:**
>
> For JMS messages, Oracle recommends that you **do not** use the JMS prefix for **custom** headers. Reserve the JMS prefix for predefined JMS headers, for example, JMSCorrelationID, JMSMessageID, JMSPriority, and so on. Using the JMS prefix in custom headers can cause problems.

OSM Request and Response Message Queues

When configuring OSM automation plug-in requests, you must create request queues that external systems consume OSM request messages from. You can configure the JMS settings for these queues based on the order processing requirements of the solution. For example, request queues often require different retry, pause, and resume settings when external systems are down. As such, it is important to have specific request queues configured to support the various JMS message consumption scenarios for each external systems.

For returning responses messages, you can create response queues. The benefits of creating new response queues is that you can configure the JMS settings as the solution requires. Optionally, you can also use the predefined **oms_events** queue (using the **mslv/oms/server_name/internal/jms/events** JNDI) that OSM uses for internal message processing. The benefits for using oms_events exclusively for all response messages include:

- Design Studio requires less time to build cartridges because the oms_events queue is internal to the oms.ear file. Design Studio does not need to generate a message-driven-bean and external automation ear file to listen on the external queue.
- You can more efficiently deploy and undeploy cartridges where there is no external automation ear file for the response queue.
- The OSM server consumes less memory when there is no external automation ear file for the response queue.

- OSM is better able to prioritize messages from different systems when there is only one queue. OSM can observe message priority uniformly across all messages within the queue.

However, if you use oms_events, you cannot configure the JMS settings, such as the pause, retry, or resume settings because these settings are already optimized to process internal OSM messages. Being able to configure these JMS settings can be important in production systems when configuring error queues or when stopping the JMS message flow to certain queues during upgrade or maintenance windows. You must weigh the advantages and disadvantages of using oms_events.

Correlating Requests from OSM to Responses from External Systems

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.

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, 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.

> **Note:**
>
> Internally, the activation task uses the OSS_API_CLIENT_ID property in the message property selector when listening for response message from Oracle Communications ASAP. Do not use this property in a non-activation task external automator (even if the activation task is not used in the solution) because this causes OSM to route the response message incorrectly.

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

About Automation Plug-in Communication Options

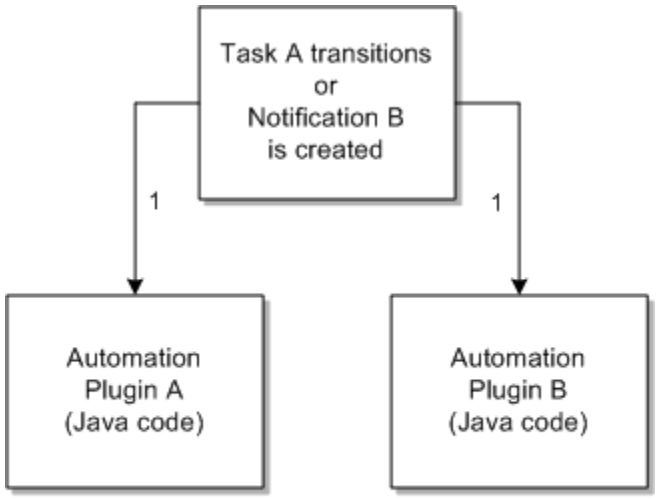
Automated tasks and the automation plug-ins they trigger can handle asynchronous or synchronous communication. Automated notifications and the automation plug-ins they trigger can handle asynchronous communication only because an automated notification can not be defined as external event receiver, so 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-level or task-level notifications because notifications do not require responses. You can also use this scenario with automated task plug-ins.

Figure 6-3 illustrates this scenario. In the figure, Automation Plug-ins A and B are internal event receivers/automators.

Figure 6-3 Automation Flow



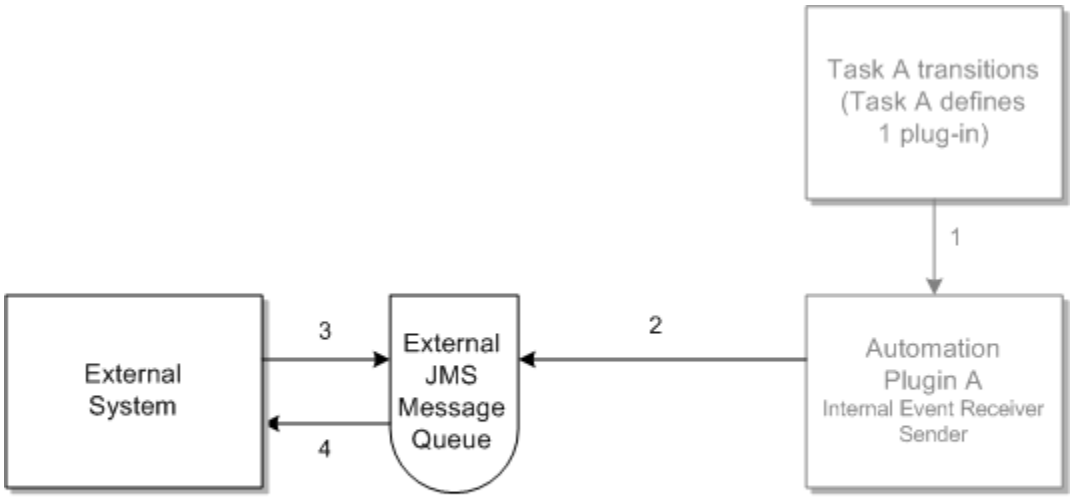
1 – Task A Transitions to the Received State or Notification B is trigger based on a notification event.

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 6-4 illustrates this scenario, which builds on Figure 6-1. In the figure, Automation Plug-in A is an internal event receiver/sender.

Figure 6-4 Automation Flow: Fire-and-Forget



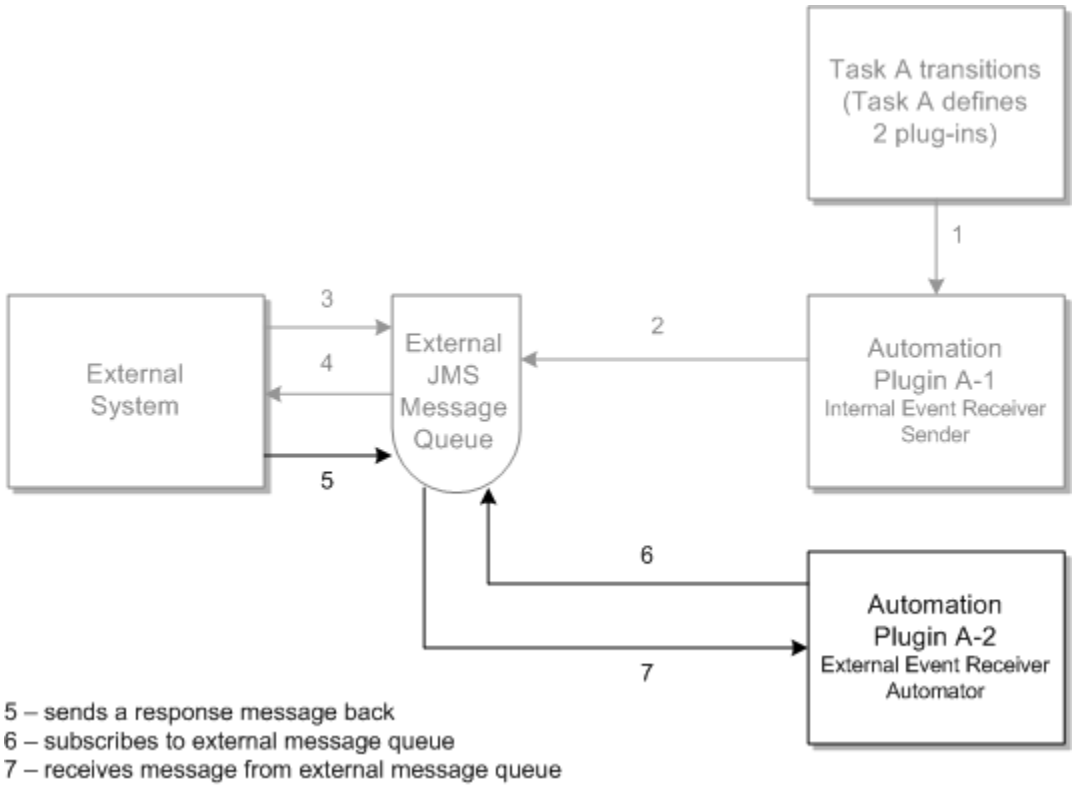
Synchronous Communication: Single Request and Response

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 (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 6-5 illustrates this scenario, which builds upon Figure 6-4. 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 6-5 Automation Flow: Simple Synchronous



Synchronous Communication: Multiple Requests and Responses

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

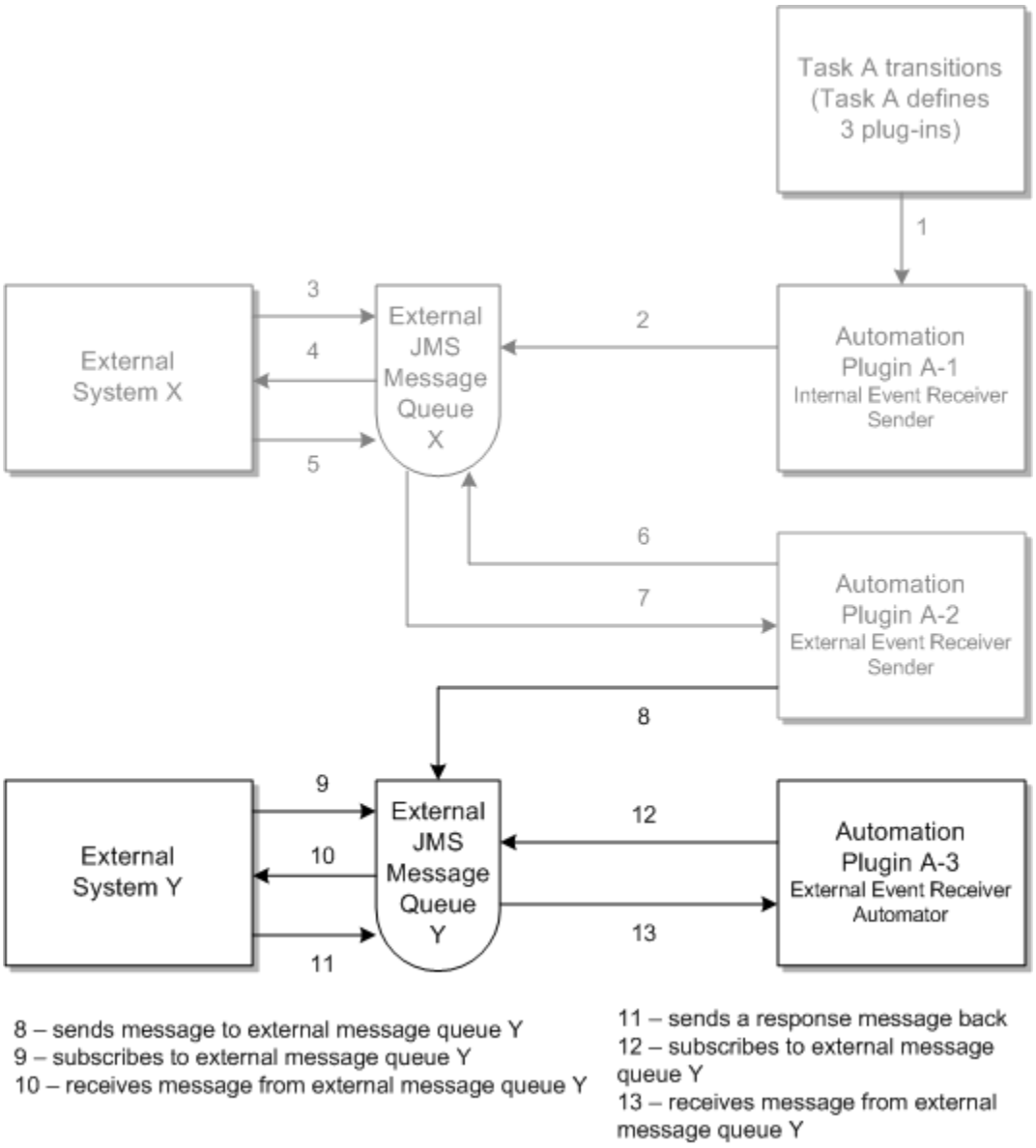
- You can define the first automation as an internal event receiver that extends AbstractSendAutomator. In this scenario, the input data is coming from OSM and being sent to an external system. The automation plug-in sends a synchronous message which expects a response back from the external system.
- 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 6-6 illustrates this scenario, which builds upon Figure 6-5. In the figure, Automation Plug-in A-1 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 6-6 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.

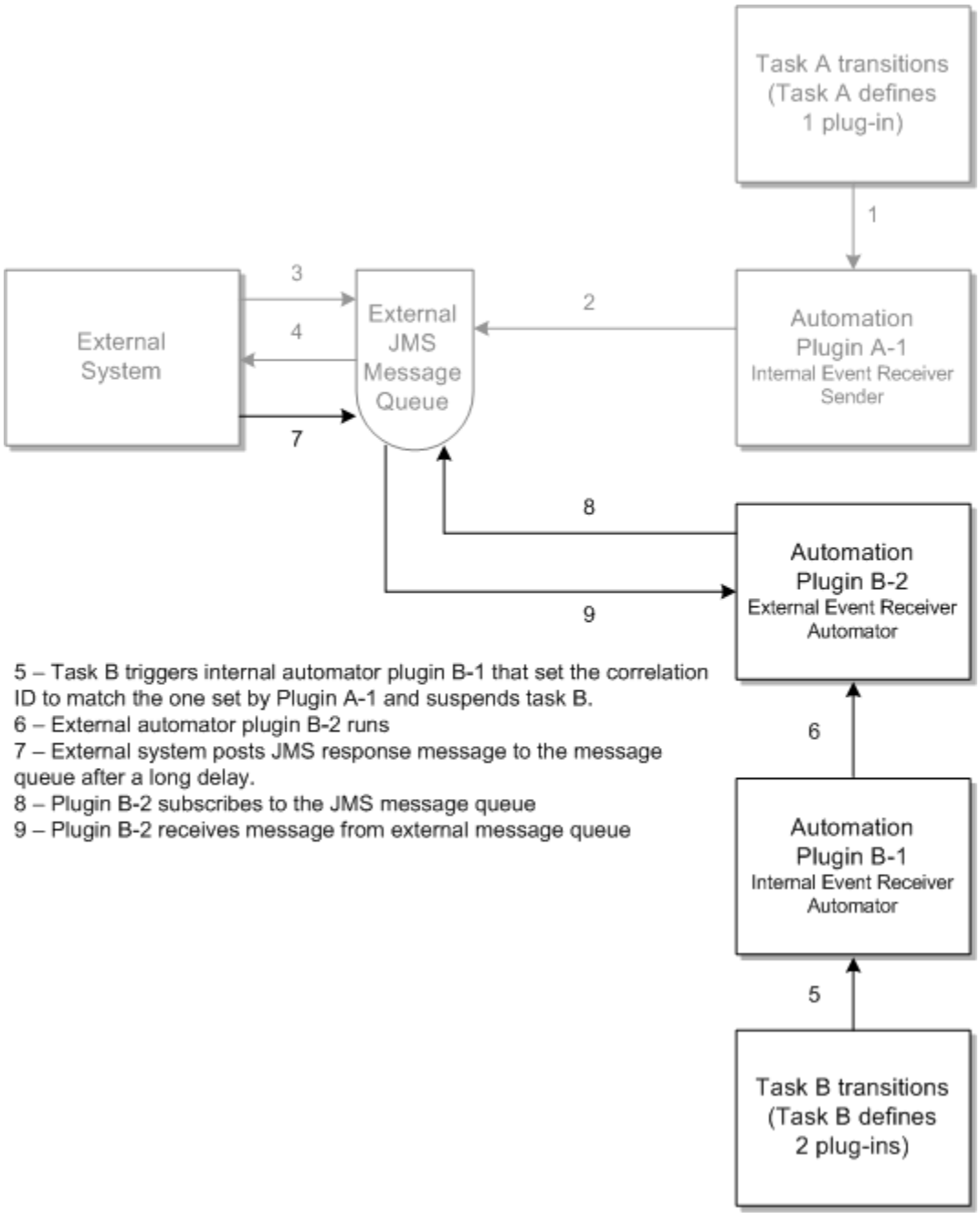
In this scenario, you must programmatically set the correlation ID for the outgoing message in the sending task. You cannot use the OSM auto-generated correlation ID functionality. For more information, see "Correlating Requests from OSM to Responses from External Systems."

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 6-7 illustrates this scenario, which is a variant of Figure 6-5. 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.

Figure 6-7 Automation Flow: Simple Asynchronous



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 6-6, 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 envelop, header, and body from the response message before storing data in this way. Having XML type data that includes the envelop, header, or body prevents OSM from sending any subsequent Web Service request messages because Web Service message envelops, 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 6-1 Stripping the Envelope, Header, and Body

```
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.

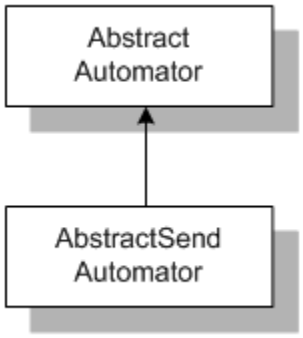
All custom automation plug-ins must extend one of the following automation classes, located in the **SDK/automation/automationdeploy_bin/automation_plugins.jar** file:

- 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 6-8. Both classes reside in the **com.mslv.automation.plugin** package.

Figure 6-8 Class Hierarchy



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 6-2. The `<implement>` element is defined in the *cartridgeName*/**customAutomation/automationMap.xsd** file, which is available with the creation of an OSM cartridge. See *OSM Modeling Guide* for more information.

Example 6-2 XML Template

```
<implement xsi:type="hw:customImplementation"
xmlns: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 6-2. The content of **helloWorld.xsd** is shown in Example 6-3.

Example 6-3 Schema for XML Template

```
<?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"
xmlns:Q1="http://www.oracle.com/OMS/AutomationMap/2001/11/23">
<import schemaLocation="automationMap.xsd"
  namespace="http://www.oracle.com/OMS/AutomationMap/2001/11/23">
</import>
<complexType name="customImplementation">
      <complexContent>
              <extension base="Q1:Implementation">
                    <sequence>
                          <element name="completionStatus" type="string"></
element>
                    </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 "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:

  ```
  public void run(String inputXML, AutomationContext automationContext)
  ```

- A custom automation plug-in that extends AbstractSendAutomator must define the method:

  ```
  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.

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:

```
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.

```
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 *".

```
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***/wlserver_10.3/server/lib/weblogic.jar**
- **SDK/automation/automationdeploy_bin/automation_plugins.jar**

> **Note:**
>
> 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 **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**.

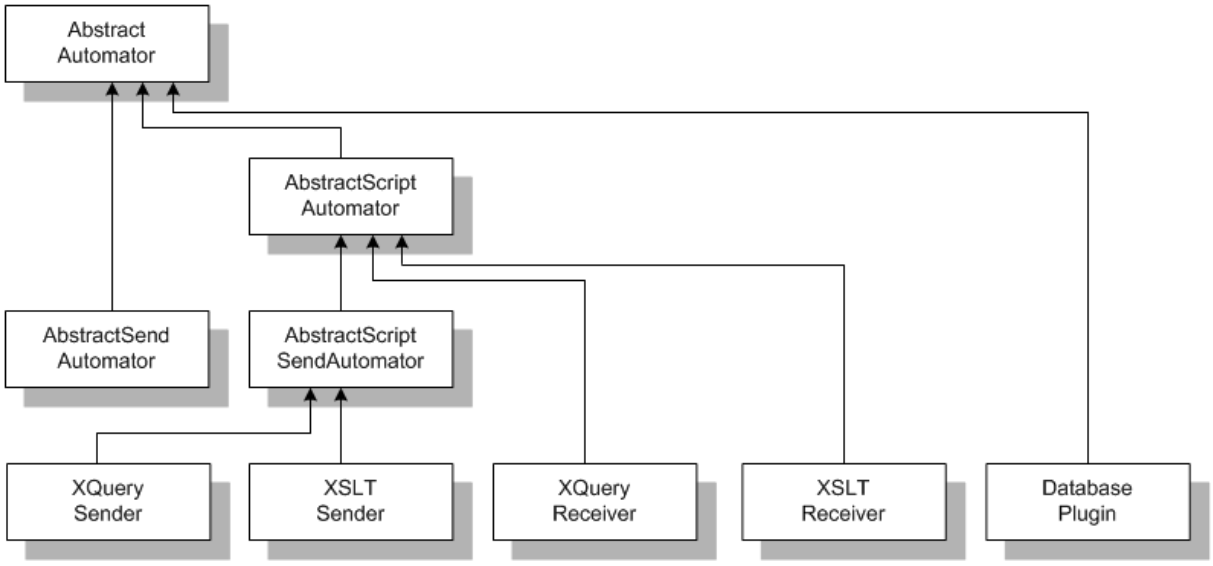
   The **Jar Selection** window opens.
5. Navigate to the location of the JAR file and double-click the JAR file to add it to the library list.

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 6-9.

Figure 6-9 Predefined Automation Plug-in 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 6-4:

Example 6-4 XSLT Java Call

```
<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"
      xmlns:sa="http://java.sun.com/products/oss/xml/ServiceActivation"
      xmlns:mslv-sa="http://www.oracle.com/oss/ServiceActivation/2003"
      xmlns:co="http://java.sun.com/products/oss/xml/Common"
      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:

- 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 6-4.

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 6-4.

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 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 6-4.

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 **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>[<arg1>, <arg2>, ...]}` or `{call <procedure-name>[<arg1>,<arg2>, ...]}`. Parameters are declared with the? character.

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

Example 6-5 SQL Statement

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

Example 6-6 Stored Procedure Call

```
<implementationxmlns="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 6-7 Transaction Definition

```
<implementation xmlns="http://www.oracle.com/Provisioning/database/
DatabasePlugin/2006/02/28" ...>
    <transaction isolationLevel="READ_COMMITTED"
scrollType="SCROLL_INSENSITIVE">
        <dataSource>test/dblugin/datasource</dataSource>
        <query>
            <sql>SELECT 'dummy' FROM dual</sql>
            <resultSet>
                <column number="1">/path/to/p6/field</column>
            </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 6-8 and the corresponding plug-in configuration in Example 6-9.

Example 6-8 OSM Order Data

```
<employees>
    <employee>
```

```
        <name>William</name>
        <job/>
    </employee>
    <employee>
        <name>Mary</name>
        <job/>
    </employee>
</employees>
```

Example 6-9 Plug-in Definition Using a Bind Path

```
<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 6-2 Bind Path Evaluation Behavior

XPath Result	Behavior
null	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.
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 6-4).
Object	The parameter value is set using the selected data based on the parameter's type (see Table 6-4).

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 6-3.

Table 6-3 Path Expression Evaluation Behavior

XPath Result	Behavior
null	The SQL parameter is set to null.
Node-set	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 6-4).
Object	The parameter value is set using the selected data based on the parameter's type (see Table 6-4).

type: Specifies the data type of the parameter, which are OSM specific.Valid values are: `boolean`, `currency`, `date`, `dateTime`, `numeric`, `phone`, and `text`.

Table 6-4 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 6-4 OSM Data Type to SQL Data Type Mapping

type Attribute Value	SQL Data Type[1]	Parameter Evaluation[2]
Boolean	Boolean	Evaluated according to `java.lang.Boolean.parseBoolean()` using the String value of the parameter. OSM values Yes and No are also supported.
currency	double	Evaluated according to `java.lang.Double.parseDouble()` using the String value of the parameter.
numeric	double	Evaluated according to `java.lang.Double.parseDouble()` using the String value of the parameter.
date	date	The String value of the parameter is expected to match the format `yyyy-MM-dd`.
dateTime	timestamp	The String value of the parameter is expected to match the format `yyyy-MM-dd'T'HH:mm:ss z`.
phone	string	Evaluated according to `java.lang.String.valueOf()`.
text	string	Evaluated according to `java.lang.String.valueOf()`.

1 where the parameter is set as java.sql.PreparedStatement.set*XXX*(#, *value*)

2 If the class of the parameter is directly assignable to the SQL data type, it is not first evaluated as a String. For example, if the type attribute value is `numeric` and the class of the parameter value is `java.lang.Number`, no String evaluation is required.

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 6-10.

Example 6-10 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 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 version="1.0"?>
<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]).
    ```

    See "About Automation Maps" for more information about the **automationMap.xml** file.

- ***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 OSM 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.

See "GetOrder" for more information about the OrderDataFilter element in the XML API GetOrder.Request. See "GetOrder" for more information about the OrderDataFilter in the GetOrder Web Service.

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 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.

See "UpdateOrder" for more information about the ResponseView, OrderDataFilter, and ExternalFulfillmentStates elements in the XML API UpdateOrder.Request. See "UpdateOrder" for more information about the OrderDataFilter in the UpdateOrder Web Service.

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.

At the task level:

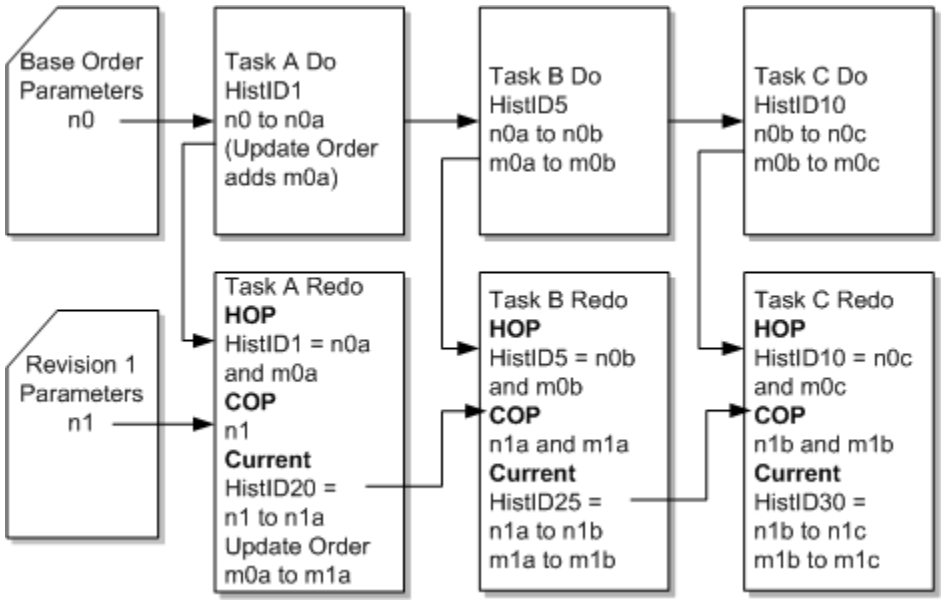
- Internal senders and automators configured in the **Automation** tab can run in do, redo, or undo in normal or fallout modes.
- Internal senders and automators configured in the **Events** tab can run in do, redo, or undo in normal mode.

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 normal or fallout execution modes, participates in task-level compensation analysis. Figure 6-10 illustrates how an update order run during the base order processing of Task A is included in the historical order perspective (HOP) of revision 1 Task A.

Figure 6-10 Update Orders in Task Compensation Analysis



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 order perspective (COP) or historical order perspectives (HOP). Nevertheless, OSM includes these data updates into the real-time order perspective (ROP) 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.

For more information about how perspectives work in change order management scenarios, see *OSM Modeling Guide*.

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 "GetOrder" 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:

```
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 Creating Automations in Design Studio

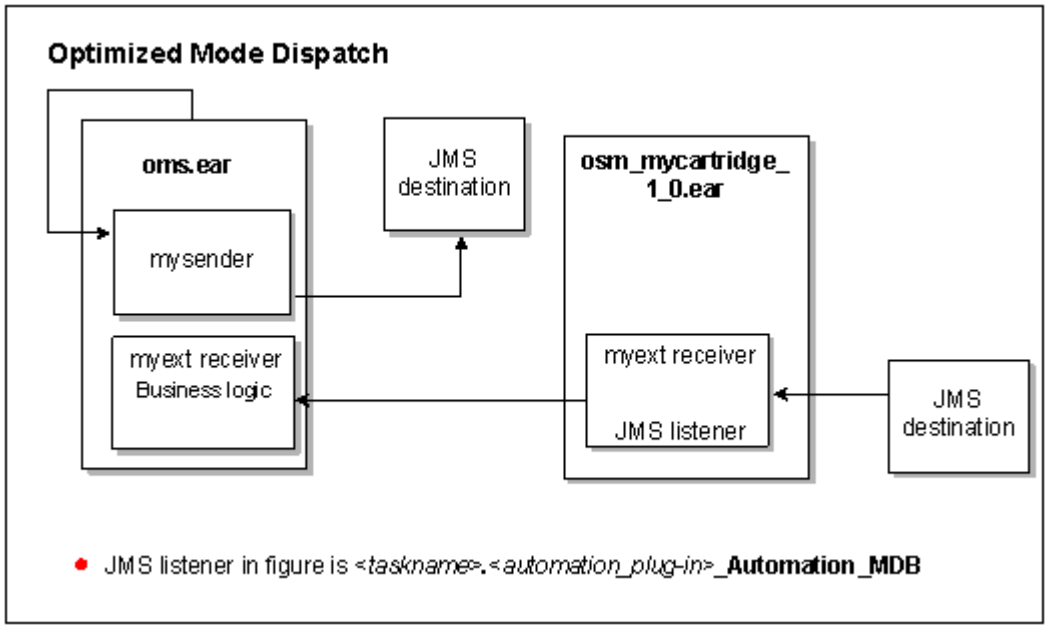
The following sections describe Design Studio tasks involved in creating automations.

About Building and Deploying Automation Plug-ins

Starting with OSM 7.3, OSM 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.

Figure 6-11 illustrates when automation plug-ins are built and deployed using Optimized mode, which is the only method available in OSM 7.3 and later. Internal event receiver type plug-ins run within the OSM application and do not require their own J2EE application. The figure also illustrates that 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.

Figure 6-11 Dispatch of Automation Plug-ins



External event receiver type automation plug-ins always require their own J2EE application in order to listen on a JMS destination. 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).

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 executed within the OSM application. The automation framework acts primarily to forward the message to OSM. 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).

> **Note:**
>
> Starting with OSM 7.2.4, automations are validated when they are deployed. Prior to that, errors like missing queues were only validated at run time. The deployment logs will provide information about any validation failures. Because the validation can cause the deployment to fail, once you have corrected the problem, you will need to redeploy the automation.

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.

For examples of generated XML for automations defined for automated tasks and automated notifications, see "AutomationMap.xml File." 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 6-5.

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:

Exception caught assembling plug-ins: "Parse/validation of automation map *cartridgeName*/**cartridgeBuild/automation/automationMap.xml** using schema *cartridgeName*/**customAutomation/automationMap.xsd** failed: Invalid text *fiftyPlusMnemonicValue* in element: mnemonic."

Table 6-5 Mnemonic Values

Automated Task or Automated Notification	<mnemonic> value
Automated task	*taskName*
Order milestone-based event notification	The <mnemonic> element is not generated for order milestone-based event notifications.
Task state-based event notification (task **Events** tab)	*taskName*
Task state-based event notification (process **Events** tab)	*processName_eventName*
Task status-based event notification	*processName_eventName*
Order data changed event notification	*orderName_eventNotificationName*
Order jeopardy notification	*orderName_jeopardyName*
Task jeopardy notification	*taskName_jeopardyName*

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 *OSM Modeling Guide* for more information.

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 *OSM Modeling Guide*.

Troubleshooting Automations

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 **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 **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 **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.

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
- Define the automation mapping in Design Studio
- Define the Custom Automation Plug-in in Design Studio
- Deploy the cartridge that contains *genericPlugin* to the OSM server

If *genericPlugin* is a custom automation plug-in, you can reuse the custom Java code by placing the Java source file in the cartridge **src** directory, compiling it, and selecting the class when defining the Custom Automation Plug-in. If *genericPlugin* is a predefined automation plug-in, you can select the predefined class when defining the

automation, and reuse your XSLT or XQuery files by copying them into the cartridge **resource** directory.

7

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.dms_12.1.3/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*.

8

Localizing OSM

This chapter provides information about 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 client and Task web client
- Configure regional settings for the Administration area of the OSM Order Management web client
- 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.

> **Note:**
>
> These steps assume you have followed the directions specified in the *OSM Installation Guide*, which include:
>
> - Installing Oracle WebLogic Server
> - Making a backup of your WebLogic Server installation
> - Installing OSM to create the database schema by selecting **Custom Installation and Database Schema**. This generates the **osm-db-model-l10n.jar** file needed for localization.
> - Making a backup of your OSM installation *after* performing these localization steps

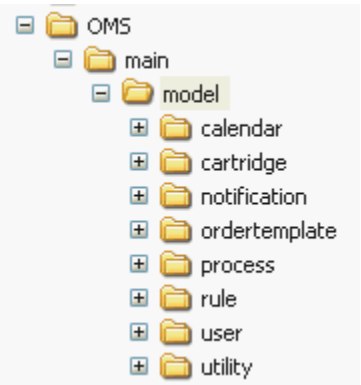
1. Uncompress the **Database/osm-db-installer-core/install/osm-db-model-l10n.jar** file.

   Set the environment variable PATH to *JDK_home***/bin**, then from a command prompt run the command:

   ```
   jar xvf osm-db-model-l10n.jar
   ```

   The uncompressed files are organized into the directories shown in Figure 8-1.

Figure 8-1 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

   OMS/main/model/calendar/_table/_data:

   - om_region.xml

- om_schedule.xml

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_order_remarks_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 **osm-db-model-l10n.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:

```
jar cvf osm-db-model-l10n.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.

   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," "Localizing the Order Management Web Client," and "Localizing the Order Lifecycle Management User Interface."

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 **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 **osm-db-model-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 **osm-db-model-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 **osm-db-model-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 and Fonts

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.

You must ensure that a font appropriate for the text you intend to use has been installed on the system.

Localization of Settings

OSM supports localized Windows regional settings: All number, currency, date, and time formats displayed in the Administration area of the OSM Order Management web client 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 database. So, 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 CHARACTER SET clause of the CREATE DATABASE statement. A complete list of character sets supported by Oracle is included in the Oracle documentation.

You can use the 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 v$nls_parameters view.
2. At the SQL*Plus prompt, enter the following:

```
SQL> select value from v$nls_parameters where parameter = 'NLS_CHARACTERSET';
```

Information similar to the following is returned:

```
VALUE
----------------------------------------
WE8ISO8859P1
```

You can use the CHARACTER SET clause of the ALTER DATABASE statement to change the character set for an existing database.

> **Note:**
>
> You must have SYSDBA system privileges.

To change the character set:

1. Initiate the database in restricted mode. For example, use the `SQL*Plus STARTUP RESTRICT` command.
2. Connect to server Manager (svrmgrl in UNIX or svrmgr30 on Windows XP).
3. At the prompt, enter:

```
ALTER DATABASE db1 CHARACTER SET WE8ISO8859P1;
```

> **Note:**
>
> The source character set must be a strict subset of the target character set.

NLS Environment

For the OSM Order Management web client Administration area 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 **HKEY_LOCAL_MACHINE/SOFTWARE/ORACLE**.

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:

```
NLS_LANG = 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/osm-db-installer-core/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 8-1 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 8-1 OSM Error Messages

Error Codes	Error Message
-20001	Order satisfied no rule. Cannot start process.
-20002	Order status and order state have not been changed.
-20004	Starting position for the process is not defined. Call support.
-20005	There is no creation task. Call support.
-20006	Workflow thread does not exist. The order may have been moved by another user. Refresh your worklist and try again.
-20007	Order is locked. Try again later.
-20008	Current history detail record not found for the order. Call support.
-20009	Cannot remove node, which has children.

Table 8-1 (Cont.) OSM Error Messages

Error Codes	Error Message
-20011	Order to be updated is currently locked by another user.
-20012	Order not found.
-20014	Rule to be evaluated is not found. Call support.
-20015	System data is protected.
-20016	Value cannot be null.
-20017	Rule cannot be evaluated.
-20020	Operation affected too many rows.
-20022	Access denied: not enough privileges.
-20023	You cannot accept this order thread.
-20025	There are too many instances for the order node.
-20026	There are too few instances for the order node.
-20027	Parent order node does not exist.
-20028	Rule engine has already been started.
-20029	Rule engine has not been started.
-20030	Invalid process definition.
-20031	Order node cannot be found. Call support.
-20038	Mandatory check failed.
-20040	Order view for task cannot be found.
-20041	Process position cannot be found. Call support.
-20044	Reporting status cannot be found. Call support.
-20049	Status is not valid for task.
-20050	View node cannot be found.
-20051	Notification cannot be found.
-20054	Notification history cannot be found.
-20055	Notification is not active.
-20056	Time interval is not valid.
-20043	Jump record does not exist.
-20059	Error processing notification.
-20060	Node information does not match any node in the database.
-20061	New parent node information is not valid.
-20062	Remark cannot be found.
-20063	The remark cannot be changed.
-20064	Attachment cannot be found.
-20065	No orders are stored in the system.
-20067	No orders satisfy the purge criteria.
-20068	The wrong task has been supplied for the order.
-20069	Invalid state transition.

Table 8-1 (Cont.) OSM Error Messages

Error Codes	Error Message
-20070	Invalid order type.
-20071	Invalid order source.
-20072	Invalid process status.
-20073	Invalid state.
-20074	State is not valid for task.
-20075	Task not found.
-20076	Stop date/time must be greater than start date/time.
-20077	Shift violates schedule boundary.
-20078	Please change the shift boundaries not to overlap any existing shift.
-20079	Exception shift not found.
-20080	Parent region is not found.
-20081	Cannot remove region, which has child regions.
-20082	Cannot remove region, which is attached to workgroup.
-20083	Cannot move to the next task. Cannot calculate expected completion time. Schedule is too short. Ask OSM administrator to extend the calendar.
-20084	Cannot move to the next task. Cannot calculate expected completion time. Cannot find the schedule.
-20085	Invalid parameter when you call internal function.
-20086	DST start/stop date is not correctly specified in om_workgroup table.
-20087	Can not find workgroup during the building of DST date.
-20088	Invalid calendar DST start/stop day of the month in the workgroup settings
-20089	Incorrect value in DST week settings in the workgroup definition
-20090	Missing parameter "oms_timezone" in OM_PARAMETER table
-20091	The node is used as a coordinator node for this thread.
-20092	Parameter Stop Date must be greater than Current System Date
-20093	Parameter Completion Date Before should be less than Current System Date
-20094	Calendar can be generated only workgroup by workgroup not for all workgroups at once
-20095	Parent thread is not found.
-20096	Partition boundary is too low. Increase the value of NEXT RANGE PARTITION BOUNDARY parameter.
-20097	Specific view is not assigned to a workgroup
-20098	Specific task is not assigned to workgroup.
-20099	JMS message does not exists.
-20103	Event type is not recognized. Call support.
-20104	Increase value of job_queue_processes parameter in the init*.ora file.

Table 8-1 (Cont.) OSM Error Messages

Error Codes	Error Message
-20105	Job type is not valid. Call support.
-20106	Minimum running jobs should be greater than 0
-20107	It can be only one notification job running at the same.
-20108	Task list is empty.
-20109	Specified pooler id is invalid.
-20110	Invalid type/source combination. Call support.
-20112	Cartridge already exists.
-20113	Cartridge can not be dropped. There are pending orders found.
-20114	Oracle runtime error.
-20121	Cannot modify cartridge - automation component is referring it.
-20122	Order type does not exist.
-20123	Order source does not exist.
-20124	Parent order not found for target node.
-20125	Cannot migrate the order header because it has activities in the source cartridge.
-20126	Character to number conversion error.
-20127	Unable to drop cartridge. There are pending orders referencing current cartridge.
-20128	Unable to drop cartridge. Drop oldest cartridge first.
-20129	Migration of schedule based tasks is not supported.
-20142	Operation is not allowed.
-20143	Summary extend date is invalid for existing summary interval.

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/osm-db-installer-core/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.

> **Note:**
>
> After you localize this file, make sure you provide an updated copy to Oracle Global Support, so that they can assist you with any future issues that may arise.

The strings shown in Table 8-2 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 8-2 Server Strings

Class	Key	Description
order_history	NODEINST	Order Instance
order_history	FITEREDNODE	Filtered Node
fixed_header	]date_pos_started	"Started"
fixed_header	]expected_duration	"Expected Duration"
fixed_header	]expected_start_date	"Expected Start Date"
fixed_header	]order_creation_date	"Order Creation Date"
fixed_header	]ord_compl_date_expected	"Expected Order Completion Date"
fixed_header	]order_seq_id	"Order ID"
fixed_header	]order_source	"Source"
fixed_header	]order_state	"State"
fixed_header	]order_type	"Type"
fixed_header	]process_description	"Process"
fixed_header	]reference_number	"Ref. #"
fixed_header	]reporting_status	"Process Status"
fixed_header	]requested_delivery_date	"Requested Delivery Date"
fixed_header	]task_description	"Task"
fixed_header	]user	"User"
notification_ fixed_header	_NOTIFICATION_ DESC	"Notification Description"
notification_ fixed_header	_NOTIFICATION_ TYPE	"Notification Type"
notification_ fixed_header	_PRIORITY	"Priority"
notification_ fixed_header	_TIMESTAMP	"Notification Timestamp"
GUIMessage	ALT	"View Remark(s)"
fixed_header	]completion_date_expected _at_task	"Expected Task Completion Date"
fixed_header	]ord_completion_date	"Completed Date"
fixed_header	]num_remarks	"Number of Remarks"
fixed_header	]order_hist_seq_id	"Order History ID"

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/osm-db-installer-core/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
- om_process_status
- om_responsibility
- om_rule_def
- om_state
- om_state_category
- om_status_category
- om_task
- om_view_order_node_label

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 8-3 lists the application functions defined for OSM. The file is located in your *OSM_home*/**Database/osm-db-installer-core/install/OMS/main/model/user/privelege/_table/_data** directory. Localize the values in the right column.

Table 8-3 om_application_function

app_function_mnemonic	app_function_description
reference_number_modification	Reference Number Modification
priority_modification	Order Priority Modification
online_reports	Online Reports
worklist_viewer	Worklist Viewer
task_assignment	Task Assignment
exception_processing	Exception Processing
search_viewer	Search Viewer

Table 8-3 (Cont.) om_application_function

app_function_mnemonic	app_function_description
create_versioned_orders	Create Versioned Orders

om_attribute_type

Table 8-4 lists the attribute types used by OSM. The file is located in your *OSM_home*/**Database/osm-db-installer-core/install/OMS/main/model/ordertemplate/_table/_data** directory. Localize the values in the description column.

Table 8-4 om_attribute_type

attribute_type	attribute_description
CN	Container
CT	Code Table
CY	Currency
DT	Date
GR	Group
LK	Lookup
NM	Numeric
PH	Phone
RF	Reference
TX	Text
YN	Boolean

om_order_data_dictionary

Table 8-5 describes the 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/osm-db-installer-core/install/OMS/main/model/ordertemplate/_table/_data** directory. Localize the text that appears in the business_name column.

Table 8-5 om_order_data_dictionary

Database Column	Value
business_name	ROOT

om_order_remarks_type

Table 8-6 describes the order remarks localization. This file is located in your *OSM_home*/**Database/osm-db-installer-core/install/OMS/main/**

model/ordertemplate/_table/_data directory. Localize the text that appears in the remarks_type_description column.

Table 8-6 om_order_remarks_type

Database Column	Value
remarks_type_description	Default

om_order_type_category

Table 8-7 describes the order type localization. This file is located in your *OSM_home***/Database/osm-db-installer-core/install/OMS/main/model/ordertemplate/_table/_data** directory. Localize the text that appears in the order_type_category_descr column.

Table 8-7 om_order_type_category

Database Column	Value
order_type_category_descr	Unknown

om_process

Table 8-8 describes the order process localization. This file is located in your *OSM_home***/Database/osm-db-installer-core/install/OMS/main/model/process/_table/_data** directory. Localize the text that appears in the process_id_description column.

Table 8-8 om_process

Database Column	Value
process_id_description	Creation Process N/A

om_process_status

Table 8-9 describes the process status localization. This file is located in your *OSM_home***/Database/osm-db-installer-core/install/OMS/main/model/process/_table/_data** directory. Localize the text that appears in the process_status_description column.

Table 8-9 om_process_status

Database Column	Value
process_status_description	Possible values are: • N/A • True • False • Submit • Delete • Next • Back • Cancel • Finish • Back

om_responsibility

Table 8-10 describes the responsibility localization. This file is located in your *OSM_home*/**Database/osm-db-installer-core/install/OMS/main/model/user/privelege/_table/_data** directory. Localize the text that appears in the responsibility_description column.

Table 8-10 om_responsibility

Database Column	Value
responsibility_description	System

om_rule_def

Table 8-11 describes the rule definition localization. This file is located in your *OSM_home*/**Database/osm-db-installer-core/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 8-11 om_rule_def

Database Column	Value
rule_description	Null Rule
rule_comment	Null Rule

om_state

Table 8-12 describes the state description localization. This file is located in your *OSM_home*/**Database/osm-db-installer-core/install/OMS/main/model/process/_table/_data** directory. Localize the text that appears in the state_description column.

Table 8-12 om_state

Database Column	Value
state_description	Possible values are: • N/A • Received • Accepted • Assigned • Completed

om_state_category

Table 8-13 describes the state category description localization. This file is located in your *OSM_home*/**Database/osm-db-installer-core/install/OMS/main/model/process/_table/_data** directory. Localize the text that appears in the state_category_description column.

Table 8-13 om_state_category

Database Column	Value
state_category_description	Possible values are: • System • User defined

om_status_category

Table 8-14 describes the status category description localization. This file is located in your *OSM_home*/**Database/osm-db-installer-core/install/OMS/main/model/process/_table/_data** directory. Localize the text that appears in the status_category_description column.

Table 8-14 om_status_category

Database Column	Value
status_category_description	Possible values are: • System • User defined

om_task

Table 8-15 describes the task description localization. This file is located in your *OSM_home*/**Database/osm-db-installer-core/install/OMS/main/model/process/_table/_data** directory. Localize the text that appears in the task_description column.

Table 8-15 om_task

Database Column	Value
task_description	Possible values are: • Null Task • N/A • Recycle

om_view_order_node_label

Table 8-16 describes the view order node label localization. This file is located in your *OSM_home*/**Database/osm-db-installer-core/install/OMS/main/model/ordertemplate/_table/_data** directory. Localize the text that appears in the label column.

Table 8-16 om_view_order_node_label

Database Column	Value
label	Default

About Generic Preferences

This section lists the generic user preferences in the OSM database tables. The file is located in your *OSM_home*/**Database/osm-db-installer-core/install/OMS/main/model/user/preference/_table/_data** directory.

To edit these tables, open the file using an XML or text editor and replace the description text with the target language text.

> **Note:**
>
> You must only localize the description values and not the column names. Column names are not externally visible, and are necessary for the internal operation of OSM.

After you localize this file, make sure you provide an updated copy to Oracle Global Support, so they can assist you with any future issues that may arise.

om_generic_mnemonic

Table 8-17 describes the generic mnemonic localization. Localize the text that appears in the description column.

Table 8-17 om_generic_mnemonic

CLASS Column	Description
worklist_filter	Possible values are: • Accepted by current user • Assigned to current user • Received • Suspended • Completed
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 **oms.ear** file. See "Unpacking the oms.ear File."
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 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. See "Working with the oms.ear File."

Task Web Client Localization Resource Bundles

After you unpack the **oms.ear** file (see "Unpacking the oms.ear File"), the localizable resources for the Task web client are contained in the **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 the Process History Pages

The Process History pages in the Task web client user interface use the Oracle JavaScript Extension Toolkit (JET), which has built-in support for internationalization and localization.

When a user connects to this client using a web browser, the OSM server attempts to determine their preferred locale. If no customization can be found for the preferred locale, the default locale (English) is used. All modern web-browsers support an Accept-Language header which indicates locale preferences. If you specify multiple language preferences, OSM uses the top one listed to locate localization resources.

After you unpack the **oms.ear** file (see "Unpacking the oms.ear File"), the localizable resources for the Process History pages in the Task web client are contained in the **SDK/Customization/osm-ui-web/js/resources/nls/** directory.

This directory contains the **bundle.js** property file. This file contains all localizable strings for the Process History page. The **bundle.js** file is loaded by the server by default, if no customization is found. Several other properties files are included in the **nls** directory as samples, including:

- **zh/bundle.js** (localized for China)

- **cs/bundle.js** (localized for the Czech Republic)

To create a new localization property file:

1. Create a new directory with the name of the language code and copy the **bundle.js** file to the directory you created. For example, create a directory with the name **fr** (localized for France) and copy the **bundle.js** file to the **fr** directory.
2. For each parameter in the **fr/bundle.js** 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.
3. Add the new locale to the list of locales provided in the default **bundle.js** file.

   For example, if you want to localize the Process History pages for France, add the string **"fr":1** to **SDK/Customization/osm-ui-web/js/resources/nls/bundle.js** as shown below:

```
define({
   root: {
         appName: "Order and Service Management",
                          ...
                         ....
                 }
                  "cs": 1,
                  "zh": 1,
                   "fr" : 1
         });
```

4. Pack the **oms.ear** file and redeploy it to the WebLogic server.
5. Change the language settings in the browser to the language you want to use.

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 8-18 lists the localization data properties.

Table 8-18 Localizing Data Formats

Data Formats	Description
format.encoding	Character set encoding used in browser communication
format.currency	Mask used for currency display and validation
format.date.input	Mask used for date display and validation in input fields
format.date.display	Mask used for date display in tables

Table 8-18 (Cont.) Localizing Data Formats

Data Formats	Description
format.datetime.input	Mask used for datetime display and validation in input fields
format.datetime.display	Mask used for datetime display in tables
format.am	String for AM field in web calendar
format.pm	String for PM field in web calendar

You must edit the mask.date and mask.time properties together. You cannot edit one property and leave the other one empty.

Table 8-19 lists the available values for the mask.date and mask.time properties.

Table 8-19 Properties Values

Symbol	Description	Presentation	Examples
y	year: Use two or four characters. If a different number of y characters is used, a four-digit year will be displayed.	Number	05 2005
M	month in year: Use one or two M characters to display a numeric month value. Use three M characters to display a three-character text representation of the month. Use four M characters to display the full text representation of the month.	Text & Number	7 07 Jul July
d	day in month	Number	10
h	hour in am/pm (1~12)	Number	12
H	hour in day (0~23)	Number	0
m	minute in hour	Number	30
s	second in minute	Number	55
E	day in week	Text	Tue
D	day in year	Number	189
w	week in year	Number	27
a	am/pm marker	Text	PM
'	escape for text	Delimiter	N/A
''	single quote	Literal	'

Table 8-20 lists the available values for the mask.currency property.

Table 8-20 Mask Currency Values

Symbol	Meaning	Presentation
#	zeros show as absent	Number
0	zeros show as 0	Number

Table 8-20 (Cont.) Mask Currency Values

Symbol	Meaning	Presentation
,	the locale-specific grouping separator	Text
-	the locale-specific negative prefix	Text
;	separates positive number format from optional negative number	Text
'	escape for text	Delimiter
other	all other symbols appear as entered	Text

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. However, this value is set automatically based on the user's locale.

> **Note:**
>
> When displaying lists, the Administration area of the Order Management web client always displays Sunday as the first day of the week.

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:

```
# 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 **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 Oder Management web client:

1. Unpack the **oms.ear** file. See "Unpacking the oms.ear File."
2. Localize the files located in the **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:

      `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 *locale1* is the default locale you want to register and *locale2* is a different locale you want to register. See "Example 8-4".
   d. Save and close the file.
4. (Optional) Change the Order Management web client logo. See "Changing the Order Management Web Client Logo Image and Text."
5. Pack and redeploy the **oms.ear** file. See "Working with the oms.ear File."

Visit the following website to learn about the XLIFF standard:

`http://docs.oasis-open.org/xliff/xliff-core/xliff-core.html`

Example 8-1 shows a simple XLIFF example.

Example 8-1 XLIFF

```
<?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>
```

This file specifies a single localizable token (**MESSAGE**) as well as the text with which to replace the token (**Hello world!**) for the specified language (**en**, which is

English). The **note** element provides a description of the context in which the token is used. As with the **resources**_*locale*.**properties** file, an XLIFF file may contain parameters. However, unlike the numeric parameters in **resources**_*locale*.**properties**, the parameters in the Order Management web client XLIFF files are named. For example:

```
<source>Change Reference for Order {ORDER_ID}</source>
```

Building on Example 8-1, the XLIFF file might look like Example 8-2.

Example 8-2 XLIFF with <source>

```
<?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>
```

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 8-2, 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 8-3.

Example 8-3 XLIFF Language set to French

```
<?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 8-4 shows an example **faces.config.xml** with the language set to French and English.

Example 8-4 faces.config.xml Language set to French and English

```
<?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 **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 **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 version="1.0" encoding="windows-1252" ?>
   <xliff version="1.1" xmlns="urn:oasis:names:tc:xliff: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>
   </xliff>
   ```

   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.

Localizing the Order Lifecycle Management User Interface

You can localize the following elements in the Order Lifecycle Management user interface:

- Text Strings: Every non-database driven text message, including labels for fields and buttons
- Page titles
- Hyperlink text
- Error and informational messages

The Order Lifecycle Management user interface uses the Oracle JavaScript Extension Toolkit (JET), which has built-in support for internationalization and localization.

When a user connects to this client using a web browser, the OSM server attempts to determine their preferred locale. If no customization can be found for the preferred locale, the default locale (English) is used. All modern web-browsers support an Accept-Language header which indicates locale preferences. If you specify multiple language preferences, OSM uses the top one listed to locate localization resources.

To localize the Order Lifecycle Management user interface:

1. Unpack the **oms.ear** file.

   See "Unpacking the oms.ear File."

2. Create a directory in **oms.ear/osm-responsive-web/resources/nls** with the name of the language code. For example, **oms.ear/osm-responsive-web/resources/nls/es** for Spanish.

3. In the directory you created in the previous step, create a file named **bundle.js** and include any strings that need localization in the following format:

   ```
   define({
   'label1': 'Localized message 1',
   'label2': 'Localized message 2',
   ...
   'label_n': 'Localized message n'
   });
   ```

   The labels and label hierarchies should match those in the **oms.ear/osm-responsive-web/resources/nls/bundle.js** file.

   For more information about localizing Oracle JET interfaces, see the discussion of adding translation bundles to Oracle JET in *Oracle JET Developing Applications with Oracle JET*.

4. Edit the **oms.ear/osm-responsive-web/resources/nls/bundle.js** file and add the translation you have created after the root element, as described in the discussion of adding translation bundles to Oracle JET in *Oracle JET Developing Applications with Oracle JET*, for example:

   ```
   define({
     "root": {
     ...
     },
     "es": true
   });
   ```

5. Pack the **oms.ear** file and redeploy it to the WebLogic server. See "Working with the oms.ear File."

Working with the oms.ear File

The **oms.ear** file is an OSM WebLogic application component deployed to the WebLogic server. It contains OSM configuration directories and files, including localization files.

Unpacking the **oms.ear** file lets you access and change the OSM configuration files, and packing and redeploying the **oms.ear** file implements your changes. The scripts for unpacking and packing the **oms.ear** file are located in the **SDK/Customization** directory.

Undeploying and redeploying **oms.ear** allows the WebLogic server to pick up any changes you have made to **oms.ear**.

Unpacking the oms.ear File

Before you can work with a configuration file that is packed in the **oms.ear** file, you must unpack the **oms.ear** file.

To unpack the **oms.ear** file:

1. Copy the **oms.ear** file to your **SDK/Customization** directory:

   For a traditional OSM installation, you can access the ear from the *domain_home*/servers/*admin_server_name*/upload/oms/app/ directory

   where *admin_server_name* is the name of the WebLogic administration server.

   For OSM cloud native, the **oms.ear** file is available in the download pack.

   > **Note:**
   >
   > Oracle recommends that you make a backup copy of your original **oms.ear** file, in case you need to restore the default settings.

2. From the **SDK/Customization** directory, use a text editor to do one of the following:
   - On UNIX or Linux, open **unpackOMS.sh** and set the JAVA_HOME variable to the path to the JDK on your system.
   - On Windows, open **unpackOMS.bat** and set the JAVA_HOME variable to the path to the JDK on your system.

   > **Note:**
   >
   > If the path in JAVA_HOME contains a space, enclose the path in quotation marks. For example:
   >
   > ```
   > set JAVA_HOME="C:\oracle\middleware test\jdk170_67"
   > ```

3. Save and close the file.
4. Do one of the following:
   - On UNIX or Linux, run **unpackOMS.sh**.
   - On Windows, run **unpackOMS.bat**.

   The script unpacks the **oms.ear** file and creates the following new sub-directories in the **Customization** directory:
   - osm-ear
   - osm-ejb
   - osm-war
   - osmwebui
   - resources

Packing the oms.ear File

When you finish editing a configuration file, you must pack the **oms.ear** file and redeploy it to the OSM WebLogic server so that your edits take effect.

To pack the **oms.ear** file:

1. From the **SDK/Customization** directory, use a text editor to do one of the following:
   - On UNIX or Linux, open **packOMS.sh** and set the JAVA_HOME variable to the path to the JDK on your system.
   - On Windows, open **packOMS.bat** and set the JAVA_HOME variable to the path to the JDK on your system.

   > **Note:**
   >
   > If the path in JAVA_HOME contains a space, enclose the path in quotation marks. For example:
   >
   > ```
   > set JAVA_HOME="C:\oracle\middleware test\jdk170_51"
   > ```

2. Save and close the file.
3. Do one of the following:
   - On UNIX or Linux, run **packOMS.sh**.
   - On Windows, run **packOMS.bat**.

   The script creates a new **oms.ear** file.
4. For traditional OSM, copy the new **oms.ear** file to the *domain_home*/servers/*admin_server_name*/upload/oms/app/ directory:

   where *admin_server_name* is the name of the WebLogic administration server.
5. For traditional OSM, follow the steps in "Undeploying and Redeploying the oms.ear File". For OSM cloud native, see "Rebuilding OSM Container Image in OSM Cloud Native".

Rebuilding OSM Container Image in OSM Cloud Native

In OSM cloud native, applications are not deployed directly to WebLogic server. They are, instead, packaged inside the OSM container image used to create an OSM instance. After re-packing the **oms.ear** file, you must rebuild your OSM container image using the updated ear file. See "Chapter 3 Creating OSM Cloud Native Images" in *OSM Cloud Native Deployment Guide* for details. You can then specify the updated image in your project specification and either create a new instance or upgrade an existing one.

Undeploying and Redeploying the oms.ear File

Undeploying and redeploying the **oms.ear** file makes any changes you have made active on the OSM server.

To undeploy and redeploy the **oms.ear** file:

1. Log in to the WebLogic Server Administration console as a user with administrative privileges.
2. In the **Domain Structure** pane, click **Deployments**.
3. In the Deployments table, select the **oms** enterprise application deployment by clicking the check box for that row.
4. From the **Stop** menu in the Deployments table, select either **When work completes** or **Force stop now**.
5. When the server has stopped, select the **oms** deployment again and click **Delete**.
6. Click **Install**.

   The Install Application Assistant window is displayed.
7. Browse to the location of the new version of the **oms.ear** file and select **oms.ear**.
8. Click **Next**.
9. Select **Install this deployment as an application** and click **Next**.
10. Click **Finish**.

    You are returned to the Deployments window with the **oms** deployment added and active.

9
Using XPath Functions

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:

  ```
  count(/journal/article[author/last="Jones"])
  ```

- Find all authors whose last name begins with Mc:

  ```
  /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.

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:journal)`

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:journal)`

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.

Example - `/journal/article[starts-with(title, 'Advanced')]`

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 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 Functions

The following describes the Number functions.

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 website (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 9-1 Abbreviated Syntax for Location Paths

Abbreviation	Syntax
(nothing)	child::
@	attribute::
//	/descendant-or-self::node()/
.	self::node()
..	parent::node()
/	Node tree root

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]

| [expr] / //

Booleans [XPath §3.4]

<=, <, >=, >=, != and or

Numbers [XPath §3.5]

-expr *, div, mod +, -

Node Types [XPath §5]

```
Root                Processing Instructions
Element         Comment
Attribute       Test
Namesspace
```

Object Types [§11.1, XPath §1]

Table 9-2 Object Types

Type	Values
boolean	True or False
number	Floating-point number
string	UCS characters
node-set	Set of nodes selected by a path

XPath Core Function Library

XPath core functions:

Node Set Functions [XPath §4.1]

```
number last()
number position()
number count(node-set)
node-set id(object)
string local-name(node-set?)
string namespace-uri(node-set?)
string name(node-set?)
```

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)
```

A

Automation and Compensation Examples

This appendix provides automation and compensation examples. You need to create automation plug-ins to use the Oracle Communications Order and Service Management (OSM) automation task and automated notification functionality. For information about the code required for the automation plug-ins, refer to the following topics:

- Predefined Automation Plug-ins
- Custom Java Automation Plug-ins
- Compensation XQuery Expressions
- Order Jeopardy Automation XQuery Plug-ins

Predefined Automation Plug-ins

The following topics provide automation plug-in examples for the predefined automation plug-in implementations that support XQuery and XSLT automations:

- Message Example
- Automation Plug-in XQuery Examples
- Automation Plug-in XSLT Examples
- Automation Plug-in Examples for Events, Jeopardies, and Notifications

Message Example

The predefined automation plug-in examples presuppose the following sample order:

```
<?xml version="1.0" encoding="UTF-8"?>
<ws:CreateOrder xmlns:ws="http://xmlns.oracle.com/communications/
ordermanagement">
  <ProcessSalesOrderFulfillmentEBM xmlns="http://xmlns.oracle.com/
EnterpriseObjects/Core/EBO/SalesOrder/V2" xmlns:sord="http://xmlns.oracle.com/
EnterpriseObjects/Core/EBO/SalesOrder/V2" xmlns:aia="http://www.oracle.com/XSL/
Transform/java/oracle.apps.aia.core.xpath.AIAFunctions" xmlns:xref="http://
www.oracle.com/XSL/Transform/java/oracle.tip.xref.xpath.XRefXPathFunctions">
     <corecom:EBMHeader xmlns:corecom="http://xmlns.oracle.com/EnterpriseObjects/
Core/Common/V2">
       <corecom:EBMID>2d32373630333234373636393035373 5</corecom:EBMID>
       <corecom:EBMName>{http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/
SalesOrder/V2} ProcessSalesOrderFulfillmentEBM</corecom:EBMName>
       <corecom:EBOName>{http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/
SalesOrder/V2} SalesOrderEBO</corecom:EBOName>
       <corecom:CreationDateTime>2009-03-09T18:46:36-07:00</
corecom:CreationDateTime>
       <corecom:VerbCode>process</corecom:VerbCode>
       <corecom:MessageProcessingInstruction>
         <corecom:EnvironmentCode>PRODUCTION</corecom:EnvironmentCode>
       </corecom:MessageProcessingInstruction>
```

```
      <corecom:Sender>
      <!-- Information about the sender - for example, a Siebel CRM -->
      </corecom:Sender>
      <corecom:BusinessScope></corecom:BusinessScope>
      <corecom:EBMTracking></corecom:EBMTracking>
    </corecom:EBMHeader>
    <DataArea>
      <corecom:Process xmlns:corecom="http://xmlns.oracle.com/EnterpriseObjects/
Core/Common/V2"/>
      <ProcessSalesOrderFulfillment>
          <corecom:Identification xmlns:corecom="http://xmlns.oracle.com/
EnterpriseObjects/Core/Common/V2">
               <corecom:BusinessComponentID schemeID="SALESORDER_ID"
schemeAgencyID="COMMON">34333939373132333239373135353138</
corecom:BusinessComponentID>
               <corecom:ID schemeID="SALESORDER_ID"
schemeAgencyID="SEBL_01">ScenarioA2</corecom:ID>
               <corecom:ApplicationObjectKey>
                    <corecom:ID schemeID="SALESORDER_ID"
schemeAgencyID="SEBL_01">88-2SGSG</corecom:ID>
               </corecom:ApplicationObjectKey>
               <corecom:Revision>
                   <corecom:Number>1</corecom:Number>
               </corecom:Revision>
          </corecom:Identification>
          <OrderDateTime>2009-03-09T18:40:21Z</OrderDateTime>
          <RequestedDeliveryDateTime>2009-03-10T00:00:00Z</
RequestedDeliveryDateTime>
          <TypeCode>SALES ORDER</TypeCode>
          <FulfillmentPriorityCode>9</FulfillmentPriorityCode>
          <FulfillmentSuccessCode>DEFAULT</FulfillmentSuccessCode>
          <FulfillmentModeCode>DELIVER</FulfillmentModeCode>
          <SalesChannelCode/>
          <ProcessingNumber/>
          <ProcessingTypeCode/>
          <corecom:Status xmlns:corecom="http://xmlns.oracle.com/
EnterpriseObjects/Core/Common/V2">
               <corecom:Code>OPEN</corecom:Code>
               <corecom:Description/>
          </corecom:Status>
          <corecom:BusinessUnitReference xmlns:corecom="http://
xmlns.oracle.com/EnterpriseObjects/Core/Common/V2">
               <corecom:BusinessUnitIdentification>
                    <corecom:ID schemeID="ORGANIZATION_ID"
schemeAgencyID="SEBL_01">0-R9NH</corecom:ID>
               </corecom:BusinessUnitIdentification>
          </corecom:BusinessUnitReference>
          <corecom:CustomerPartyReference xmlns:corecom="http://
xmlns.oracle.com/EnterpriseObjects/Core/Common/V2">
               <corecom:CustomerPartyAccountIdentification>
                    <corecom:BusinessComponentID
schemeID="CUSTOMERPARTY_ACCOUNTID"
schemeAgencyID="COMMON">2d353537333130353233303536343833</
corecom:BusinessComponentID>
                    <corecom:ID schemeID="CUSTOMERPARTY_ACCOUNTID"
schemeAgencyID="SEBL_01">88-2PB18</corecom:ID>
                    <corecom:ApplicationObjectKey>
                         <corecom:ID schemeID="CUSTOMERPARTY_ACCOUNTID"
schemeAgencyID="SEBL_01">88-2PB18</corecom:ID>
                    </corecom:ApplicationObjectKey>
               </corecom:CustomerPartyAccountIdentification>
```

```
                <corecom:CustomerPartyAccountName>Adam,10000</
corecom:CustomerPartyAccountName>
                <corecom:CustomerPartyAccountContactIdentification>
                      <corecom:BusinessComponentID
schemeID="CUSTOMERPARTY_CONTACTID"
schemeAgencyID="COMMON">2d353130393634353031313333353938</
corecom:BusinessComponentID>
                      <corecom:ApplicationObjectKey>
                           <corecom:ID schemeID="CUSTOMERPARTY_CONTACTID"
schemeAgencyID="SEBL_01">88-2MKA1</corecom:ID>
                      </corecom:ApplicationObjectKey>
                </corecom:CustomerPartyAccountContactIdentification>
                <corecom:CustomerPartyAccountContactAddressCommunication>
                      <corecom:AddressCommunication>
                           <corecom:Address>
                                <!-- Enter Address Nodes -->
                           </corecom:Address>
                      </corecom:AddressCommunication>
                </corecom:CustomerPartyAccountContactAddressCommunication>
                <corecom:CustomerPartyAccountTypeCode>RESIDENTIAL
</corecom:CustomerPartyAccountTypeCode>
           </corecom:CustomerPartyReference>
           <corecom:PriceListReference xmlns:corecom="http://
xmlns.oracle.com/EnterpriseObjects/Core/Common/V2">
                <corecom:PriceListIdentification>
                    <corecom:ID>88-2D1YC</corecom:ID>
                </corecom:PriceListIdentification>
           </corecom:PriceListReference>
           <corecom:ShipToPartyReference xmlns:corecom="http://
xmlns.oracle.com/EnterpriseObjects/Core/Common/V2">
               <corecom:LocationReference>
                    <corecom:Address>
                        <!-- Enter Address Nodes -->
                    </corecom:Address>
               </corecom:LocationReference>
               <corecom:CustomerPartyAccountIdentification>
                    <corecom:BusinessComponentID
schemeID="CUSTOMERPARTY_ACCOUNTID" schemeAgencyID="COMMON"/>
               </corecom:CustomerPartyAccountIdentification>
               <corecom:CustomerPartyAccountContactIdentification>
                    <corecom:BusinessComponentID
schemeID="CUSTOMERPARTY_CONTACTID"
schemeAgencyID="COMMON">2d353130393634353031313333353938</
corecom:BusinessComponentID>
                    <corecom:ApplicationObjectKey>
                         <corecom:ID schemeID="CUSTOMERPARTY_CONTACTID"
schemeAgencyID="SEBL_01">88-2MKA1</corecom:ID>
                    </corecom:ApplicationObjectKey>
               </corecom:CustomerPartyAccountContactIdentification>
           </corecom:ShipToPartyReference>
           <corecom:ParentSalesOrderReference xmlns:corecom="http://
xmlns.oracle.com/EnterpriseObjects/Core/Common/V2">
                <corecom:SalesOrderIdentification>
                    <corecom:BusinessComponentID schemeID="SALESORDER_ID"
schemeAgencyID="COMMON"/>
                </corecom:SalesOrderIdentification>
           </corecom:ParentSalesOrderReference>
           <corecom:ProjectReference xmlns:corecom="http://
xmlns.oracle.com/EnterpriseObjects/Core/Common/V2">
               <corecom:ProjectIdentfication>
                   <corecom:ID schemeID="PROJECT_ID" schemeAgencyID="SEBL_01"/>
```

```
                </corecom:ProjectIdentfication>
            </corecom:ProjectReference>
            <corecom:SalespersonPartyReference xmlns:corecom="http://
xmlns.oracle.com/EnterpriseObjects/Core/Common/V2">
                <corecom:PartyIdentification>
                     <corecom:ID schemeID="SALESPERSON_PARTYID"
schemeAgencyID="SEBL_01">0-1</corecom:ID>
                </corecom:PartyIdentification>
            </corecom:SalespersonPartyReference>
            <!-- Enter order line items here -->
       </ProcessSalesOrderFulfillment>
       </DataArea>
  </ProcessSalesOrderFulfillmentEBM>
</ws:CreateOrder>
```

Automation Plug-in XQuery Examples

The following topics provide XQuery automation plug-in examples for automation tasks:

- Internal XQuery Sender
- External XQuery Automator
- External XQuery Sender
- Internal XQuery Automator

Internal XQuery Sender

The Automated Task editor internal XQuery automator receives task data from OSM and sends data to an external system. You can send a message to an external system using whatever protocol that system requires, such as, Telnet, HTTP, CORBA, SOAP, or web services.

The XQuery has the following characteristics:

- XQuery context in prolog: The input document for any automated task automation plug-in is the order data defined in the Automation Task editor Task Data tab. You can access this data by declaring the TaskContext OSM Java class. Always declare this class along with the $context java binding. For example:

  ```
  declare namespace context = "java:com.mslv.oms.automation.TaskContext";
  ...
  declare variable $context external;
  ```

- Prolog: You must declare ScriptSenderContextInvocation in any internal XQuery automator which extends ScriptReceiverContextInvocation. Always declare this class along with the $automator java binding. For example:

  ```
  declare namespace automator =
  "java:oracle.communications.ordermanagement.automation.plugin.ScriptSenderCon
  textInvocation";
  ...
  declare variable $automator external;
  ```

  Oracle recommends that you use the standard Apache log class. Always declare this class along with the $log java binding.

```
declare namespace log = "java:org.apache.commons.logging.Log";
...
declare variable $log external;
```

You must use the TextMessage class for sending JMS based messages. Always declare this class along with the $outboundMessage Java binding. You can use JMS text based messages to send OSM Web Service messages to other OSM systems, such as a service order from an OSM COM system to an OSM SOM system.

```
declare namespace outboundMessage = "java:javax.jms.TextMessage";
...
declare variable $outboundMessage external;
```

> **Note:**
>
> If you need to support any other protocol for sending messages, you can implement a custom Java automation plug-in for the protocol or import a helper function implementation that supports the protocol.

- Body: The body for an internal XQuery sender can contain the following elements:
  - Use outboundMessage to set up the standard WebLogic JMS message properties for web services:

    ```
    outboundMessage:setStringProperty($outboundMessage,
    '_wls_mimehdrContent_Type', 'text/xml; charset=&quot;utf-8&quot;'),
    ```

  - Use outboundMessage to set up the OSM Web Service URI JMS message property:

    ```
    outboundMessage:setStringProperty($outboundMessage, 'URI', '/osm/wsapi'),
    ```

  - You can optionally use outboundMessage with the XML API to populate a JMS property value from order data. For example this code sets up an Ora_OSM_COM_OrderId parameter that is populated with the OSM order ID:

    ```
    outboundMessage:setStringProperty($outboundMessage,
    'Ora_OSM_COM_OrderId', /oms:GetOrder.Response/oms:OrderID),
    ```

  - You can optionally use outboundMessage to set the JMS Correlation ID for the automation task before sending the message. This allows OSM to route a return message with the same corresponding JMS property value to an external XQuery automator on the same automation task as the original sender automation plug-in. For example, the following code sets the JMS correlation ID using the original OSM COM order:

    ```
    outboundMessage:setJMSCorrelationID($outboundMessage, concat($order/
    oms:_root/oms:messageXmlData/ebo:ProcessSalesOrderFulfillmentEBM/
    ebo:DataArea/ebo:ProcessSalesOrderFulfillment/corecom:Identification/
    corecom:ID/text(),'-COM')),
    ```

    If this code were applied to "Message Example," the return value would be a concatenation of ScenarioA2 and -COM: `ScenarioA2-COM`.

> **Note:**
>
> Other correlation scenarios are possible. For example, you may send a message from an automation task without expecting any response to the same automation task. In this scenario, another automation task further down in the process may be dedicated to receiving the response message, in which case an automation plug-in would be required that would set the correlation ID expected from the return message for that automated task. See "Using Automation" for more information about asynchronous communication scenarios.

- Access to the task level order data (the task view) using the XML API `GetOrder.Response` function call. For example, the following code provides access to all order data passed into the task as a variable that is then used in other variables to access different parts of the data:

  ```
  let $order := /oms:GetOrder.Response
  let $othervariable := $order/oms:_root/oms:orderid
  ```

- Any XQuery logic your plug-in requires, such as if-then or if-then-else statements that evaluate based on one or more parameters within the response message. For example, there could be a choice of two or more messages that could be sent depending on the order data values, or you might log a message.

- A completeTaskOnExit method statement that completes the plug-in and transitions the task to the next task based on the status selected if the plug-in is intended to end the task. Typically, an automated task would contain an internal XQuery sender plug-in for sending a message and an external XQuery receiver plug-in for receiving a message, but you can also create an automation that only sends an order with another automation that receives the order. This can be useful if the response message takes a long time to return. If you are expecting the system to respond that you sent the message to, you must configure the internal XQuery sender with a reply to queue that listens for a message acknowledgement, whether the response is returned to an external automator on the same automation task or on another automation task.

The following example provides the code for an XQuery that sends a message from an OSM system in the COM role to an OSM system in the SOM role using the OSM Web Service interface and assumes JMS communication over T3S.

```
declare namespace automator =
"java:oracle.communications.ordermanagement.automation.plugin.ScriptSenderContext
Invocation";
declare namespace context = "java:com.mslv.oms.automation.TaskContext";
declare namespace log = "java:org.apache.commons.logging.Log";
declare namespace outboundMessage = "java:javax.jms.TextMessage";
declare namespace oms="urn:com:metasolv:oms:xmlapi:1";
declare namespace to="http://TechnicalOrder";
declare namespace provord="http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/
ProvisioningOrder/V1";
declare namespace corecom="http://xmlns.oracle.com/EnterpriseObjects/Core/Common/
V2";
declare namespace env="http://schemas.xmlsoap.org/soap/envelope/";
declare namespace cord="http://oracle.communications.c2a.model/internal/order";
declare namespace ebo="http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/
SalesOrder/V2";
```

```
declare variable $automator external;
declare variable $context external;
declare variable $log external;
declare variable $outboundMessage external;

let $order := /oms:GetOrder.Response
let $technicalActions := $order/oms:_root/oms:TechnicalActions
let $ebm := $order/oms:_root/oms:messageXmlData
let $bi := $order/oms:_root/oms:CaptureInteractionResponse

return(
outboundMessage:setStringProperty($outboundMessage, '_wls_mimehdrContent_Type',
'text/xml; charset=&amp;quot;utf-8&amp;quot;'),
outboundMessage:setStringProperty($outboundMessage, 'URI', '/osm/wsapi'),
outboundMessage:setStringProperty($outboundMessage, 'Ora_OSM_COM_OrderId', /
oms:GetOrder.Response/oms:OrderID),
outboundMessage:setJMSCorrelationID($outboundMessage, concat($order/oms:_root/
oms:messageXmlData/ebo:ProcessSalesOrderFulfillmentEBM/ebo:DataArea/
ebo:ProcessSalesOrderFulfillment/corecom:Identification/corecom:ID/text(),'-
COM')),
log:info($log,concat('Sending Service Order for COM order: ', $order/
oms:OrderID)),
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ord="http://xmlns.oracle.com/communications/ordermanagement">
        <soapenv:Header>
        <wsse:Security xmlns:wsse = "http://docs.oasis-open.org/wss/2004/01/
oasis-200401-wss-wssecurity-secext-1.0.xsd" soapenv:mustUnderstand="1">
        <wsse:UsernameToken xmlns:wsu = "http://docs.oasis-open.org/wss/2004/01/
oasis-200401-wss-wssecurity-utility-1.0.xsd" wsu:Id="UsernameToken-4799946">
        <wsse:Username>demo</wsse:Username>
        <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-username-token-profile-1.0#PasswordText">passw0rd</wsse:Password>
        </wsse:UsernameToken>
        </wsse:Security>
        </soapenv:Header>
   <soapenv:Body>
      <ord:CreateOrder>
               <ebo:ProcessProvisioningOrderEBM xmlns:ebo="http://
xmlns.oracle.com/EnterpriseObjects/Core/EBO/ProvisioningOrder/V1">
<ebo:DataArea>
                           <corecom:Process xmlns="http://xmlns.oracle.com/
EnterpriseObjects/Core/EBO/ProvisioningOrder/V1" xmlns:corecom="http://
xmlns.oracle.com/EnterpriseObjects/Core/Common/V2" xmlns:aia="http://
www.oracle.com/XSL/Transform/java/oracle.apps.aia.core.xpath.AIAFunctions"
xmlns:xref="http://www.oracle.com/XSL/Transform/java/
oracle.tip.xref.xpath.XRefXPathFunctions"
xmlns:oms="urn:com:metasolv:oms:xmlapi:1" xmlns:provord="http://xmlns.oracle.com/
EnterpriseObjects/Core/EBO/ProvisioningOrder/V1"/>
                           <provord:ProcessProvisioningOrder xmlns="http://
xmlns.oracle.com/EnterpriseObjects/Core/EBO/ProvisioningOrder/V1"
xmlns:aia="http://www.oracle.com/XSL/Transform/java/
oracle.apps.aia.core.xpath.AIAFunctions" xmlns:xref="http://www.oracle.com/XSL/
Transform/java/oracle.tip.xref.xpath.XRefXPathFunctions"
xmlns:oms="urn:com:metasolv:oms:xmlapi:1" xmlns:provord="http://xmlns.oracle.com/
EnterpriseObjects/Core/EBO/ProvisioningOrder/V1">
                           <corecom:SalesOrderReference>
                                <corecom:SalesOrderIdentification>
                                    {$order/oms:_root/oms:ServiceOrder/
cord:Order/cord:CustomerDetails/cord:OrderNumber/corecom:Identification/*}
                                </corecom:SalesOrderIdentification>
```

```
                                </corecom:SalesOrderReference>

<provord:RequestedDeliveryDateTime>2010-07-16T08:24:38Z </
provord:RequestedDeliveryDateTime>
                                <provord:TypeCode>SALES ORDER</provord:TypeCode>
                                <provord:FulfillmentPriorityCode>5</
provord:FulfillmentPriorityCode>
                                <provord:FulfillmentSuccessCode>DEFAULT </
provord:FulfillmentSuccessCode>
                                <provord:FulfillmentModeCode>DELIVER</
provord:FulfillmentModeCode>
                                <provord:ProcessingNumber/>
                                <provord:ProcessingTypeCode/>
                                <corecom:Status xmlns:corecom="http://
xmlns.oracle.com/EnterpriseObjects/Core/Common/V2">
                                    <corecom:Code>IN PROGRESS</corecom:Code>
                                </corecom:Status>
                                <corecom:BusinessUnitReference
xmlns:corecom="http://xmlns.oracle.com/EnterpriseObjects/Core/Common/V2">
                                    <corecom:BusinessUnitIdentification>
                                        <corecom:ID schemeID="ORGANIZATION_ID"
schemeAgencyID="SEBL_01">0-R9NH</corecom:ID>
                                    </corecom:BusinessUnitIdentification>
                                </corecom:BusinessUnitReference>
                                {$order/oms:_root/oms:ServiceOrder/cord:Order/
cord:CustomerDetails/cord:CustomerParty/corecom:CustomerPartyReference}
                                <corecom:ParentProvisioningOrderReference
xmlns:corecom="http://xmlns.oracle.com/EnterpriseObjects/Core/Common/V2">
                                    <corecom:ProvisioningOrderIdentification>
                                        <corecom:BusinessComponentID
schemeID="SALESORDER_ID" schemeAgencyID="COMMON"/>
                                    </corecom:ProvisioningOrderIdentification>
                                </corecom:ParentProvisioningOrderReference>
                                {
                                    for $x in $order/oms:_root/oms:ServiceOrder/
cord:Order/cord:ServiceOrderLine
                                    return
                                    <provord:ProvisioningOrderLine>
                                    <corecom:Identification xmlns:corecom="http://
xmlns.oracle.com/EnterpriseObjects/Core/Common/V2">
                                        <corecom:BusinessComponentID>{concat($x/
@id,'')} </corecom:BusinessComponentID>
                                        <corecom:ID schemeID="SALESORDER_LINEID"
schemeAgencyID="SEBL_01">{concat($x/@id,'')}</corecom:ID>
                                        <corecom:ApplicationObjectKey>
                                             <corecom:ID
schemeID="SALESORDER_LINEID" schemeAgencyID="SEBL_01">{concat($x/@id,'')}</
corecom:ID>
                                        </corecom:ApplicationObjectKey>
                                    </corecom:Identification>
                                    <provord:OrderQuantity>1</provord:OrderQuantity>
                                    <provord:ServiceActionCode>{$x/cord:Action/
text()} </provord:ServiceActionCode>
                                    <provord:ServicePointCode/>
                                    <corecom:Status xmlns:corecom="http://
xmlns.oracle.com/EnterpriseObjects/Core/Common/V2">
                                        <corecom:Code>IN PROGRESS</corecom:Code>
                                    </corecom:Status>
                                    <corecom:ServiceAddress xmlns:corecom="http://
xmlns.oracle.com/EnterpriseObjects/Core/Common/V2">
                                        <corecom:Identification>
```

```
                                        <corecom:BusinessComponentID
schemeAgencyID="COMMON"
schemeID="CUSTOMERPARTY_ADDRESSID">2d32373332323131353131383631333 1</
corecom:BusinessComponentID>
                                        <corecom:ApplicationObjectKey>
                                             <corecom:ID
schemeAgencyID="SEBL_01" schemeID="CUSTOMERPARTY_ADDRESSID">88-2KKNH</corecom:ID>
                                        </corecom:ApplicationObjectKey>
                                    </corecom:Identification>
                                    <corecom:LineOne>{$x/cord:Address/
cord:LineOne/text()} </corecom:LineOne>
                                    <corecom:CityName>{$x/cord:Address/
cord:CityName/text()} </corecom:CityName>
                                    <corecom:StateName>{$x/cord:Address/
cord:StateName/text()} </corecom:StateName>
                                    <corecom:ProvinceName>{$x/cord:Address/
cord:ProvinceName/ text()}</corecom:ProvinceName>
                                    <corecom:CountryCode>{$x/cord:Address/
cord:CountryCode /text()}</corecom:CountryCode>
                                    <corecom:PostalCode>{$x/cord:Address/
cord:PostalCode /text()}</corecom:PostalCode>
                                </corecom:ServiceAddress>
                                <corecom:ItemReference xmlns:corecom="http://
xmlns.oracle.com/EnterpriseObjects/Core/Common/V2">
                                <corecom:ItemIdentification>
                                    <corecom:BusinessComponentID
schemeAgencyID="COMMON" schemeID="ITEM_ITEMID"/>
                                    <corecom:ApplicationObjectKey>
                                        <corecom:ID schemeID="ITEM_ITEMID"
schemeAgencyID="SEBL_01">{concat($x/cord:InstanceID/text(),'')}</corecom:ID>
                                    </corecom:ApplicationObjectKey>
                                    <corecom:AlternateObjectKey>
                                        <corecom:ContextID/>
                                    </corecom:AlternateObjectKey>
                                    <corecom:SupplierItemID/>
                                </corecom:ItemIdentification>
                                <corecom:Name>{concat($x/@name,'')}</
corecom:Name>
                                    <corecom:ClassificationCode
listID="PermittedTypeCode"></corecom:ClassificationCode>
                                    <corecom:ClassificationCode
listID="BillingProductTypeCode"/>
                                    <corecom:ClassificationCode
listID="FulfillmentItemCode">{concat($x/@name,'')}</corecom:ClassificationCode>
                                    <corecom:ServiceIndicator>false</
corecom:ServiceIndicator>
                                    <corecom:TypeCode>SERVICE</corecom:TypeCode>
                                    <corecom:Description/>
                                    <corecom:SpecificationGroup>
                                        <corecom:Name>ExtensibleAttributes</
corecom:Name>
                                        {
                                            for $y in $x/cord:Attribute
                                            return
                                            <corecom:Specification>
                                                <corecom:ServiceActionCode> </
corecom:ServiceActionCode>
                                                <corecom:Name>{concat($y/
@name,'')} </corecom:Name>
                                                <corecom:DataTypeCode>Text</
corecom:DataTypeCode>
```

```
                                                    <corecom:Value>{$y/cord:Value/
cord:value/text()} </corecom:Value>
                                                </corecom:Specification>
                                            }
                                        </corecom:SpecificationGroup>

<corecom:PrimaryClassificationCode>{concat($x/@name,'')} </
corecom:PrimaryClassificationCode>
                                        <corecom:ServiceInstanceIndicator>true </
corecom:ServiceInstanceIndicator>
                                    </corecom:ItemReference>

<provord:ProvisioningOrderLineSpecificationGroup>
                                        <corecom:SpecificationGroup>
                                             <corecom:Name>ExtensibleAttributes</
corecom:Name>
                                             <corecom:Specification>

<corecom:Name>ParentSalesOrderLine</corecom:Name>
                                                  <corecom:Value>{$x/
cord:primaryMapping/text()} </corecom:Value>
                                             </corecom:Specification>
                                            {
                                                 for $z in $x/cord:secondaryMapping
                                                 return
                                                 <corecom:Specification>

<corecom:Name>ParentSalesOrderLine</corecom:Name>
                                                 <corecom:Value>{$z/text()}</
corecom:Value>
                                                 </corecom:Specification>
                                            }
                                        </corecom:SpecificationGroup>
                                    </
provord:ProvisioningOrderLineSpecificationGroup>
                                </provord:ProvisioningOrderLine>
                            }
                        </provord:ProcessProvisioningOrder>
                    </ebo:DataArea>
                </ebo:ProcessProvisioningOrderEBM>
      </ord:CreateOrder>
   </soapenv:Body>
</soapenv:Envelope>
```

External XQuery Automator

The Automated Task editor external XQuery automator receives task data from an external system and optionally updates OSM order data. The XQuery has the following characteristics:

- XQuery context in prolog: The input document for any automated task automation plug-in is the order data defined in the Automation Task editor Task Data tab. You can access this data by declaring the TaskContext OSM Java class. Always declare this class along with the $context java binding. For example:

  ```
  declare namespace context = "java:com.mslv.oms.automation.TaskContext";
  ...
  declare variable $context external;
  ```

- Prolog: You must declare ScriptReceiverContextInvocation in any external XQuery automator. Typically, you can use the getOrderAsDOM method to receive external

messages and the setUpdateOrder method to update the order data. Always declare this class along with the $automator java binding. For example:

```
declare namespace automator =
"java:oracle.communications.ordermanagement.automation.plugin.ScriptReceiverC
ontextInvocation";
...
declare variable $automator external;
```

Oracle recommends that you use the standard Apache log class. Always declare this class along with the $log java binding.

```
declare namespace log = "java:org.apache.commons.logging.Log";
...
declare variable $log external;
```

Another necessary declaration includes the xmlapi namespace, that you can use with the ScriptReceiverContextInvocation getOrderAsDom method to retrieve the order data for the task as a variable. This task data variable can be used in an OrderDataUpdate to update the order data with the data values received in the response message, if an update to the order data is required. For example:

```
declare namespace oms="urn:com:metasolv:oms:xmlapi:1";
let $taskData := fn:root(automator:getOrderAsDOM($automator))/
oms:GetOrder.Response
```

- Body: The body for an external XQuery automator can contain the following elements:
  - Any XQuery logic your plug-in requires, such as if-then or if-then-else statements that evaluate based on one or more parameters within the response message, or you might log a message.
  - A setUpdateOrder method statement that indicates whether there is an order data update. This method should be identical to what you selected in the Design Studio automation plug-in Properties View XQuery Tab Update Order check box.
  - A completeTaskOnExit method statement that completes the plug-in and transitions the task to the next task based on the status selected, if the plug-in is intended to end the task. Since there can be multiple plug-ins within a task, you would only need this method in the last plug-in listed. For example, the Failed status might transition to a fallout task, and the Succeed status may transition to the next task in the process.
  - An OrderDataUpdate statement that updates the order data based on the information returned in the response. 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:

```
declare namespace oms="urn:com:metasolv:oms:xmlapi:1";
declare namespace automator =
"java:oracle.communications.ordermanagement.automation.plugin.ScriptReceiverConte
xtInvocation";
declare namespace context = "java:com.mslv.oms.automation.TaskContext";
declare namespace log = "java:org.apache.commons.logging.Log";
declare namespace su="http://StatusUpdate";
declare namespace so="http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/
SalesOrder/V2";
declare namespace corecom="http://xmlns.oracle.com/EnterpriseObjects/Core/Common/
V2";

declare variable $automator external;
declare variable $context external;
declare variable $log external;

let $response := fn:root()/su:StatusUpdate (: fn:root(.) :)
let $items := fn:root()/su:StatusUpdate/su:OrderItem

let $taskData := fn:root(automator:getOrderAsDOM($automator))/
oms:GetOrder.Response
let $component := if (fn:exists($taskData/oms:_root/oms:ControlData/
oms:Functions/*/oms:componentKey)) then $taskData/oms:_root/oms:ControlData/
oms:Functions/*[fn:position()=1] else ()

return (
if($response/su:status/text()='SOM_Completed') then (
    log:info($log,concat('Received SOM Status Update: SOM_Completed;
', $response/su:status/text())),
    automator:setUpdateOrder($automator,"true"),
    context:completeTaskOnExit($context,"success"),
    (
        <OrderDataUpdate xmlns="http://www.metasolv.com/OMS/OrderDataUpdate/
2002/10/25">
            {
            for $item in $items
            for $parent in $item/su:ParentLineId
            for $orderComponentItem in $component/oms:orderItem[oms:orderItemRef/
oms:LineXmlData/so:SalesOrderLine/corecom:Identification/
corecom:ApplicationObjectKey/corecom:ID/text() = $parent/text()]
            return (
                    <Update path="{fn:concat("/ControlData/Functions/Provision/
orderItem[@index='",fn:data($orderComponentItem/@index),"']")}">
                       <ExternalFulfillmentState>{$item/su:Status/text()}</
ExternalFulfillmentState>
                    </Update>
              )

            }
        </OrderDataUpdate>
    )
) else if($response/su:status/text()='SOM_Failed') then (

    log:info($log,concat('Received SOM Status Update: SOM_Failed; ', $response/
su:status/text())),
    automator:setUpdateOrder($automator,"true"),
    context:completeTaskOnExit($context,"failure"),
```

```
    (
        <OrderDataUpdate xmlns="http://www.metasolv.com/OMS/OrderDataUpdate/
2002/10/25">
            {
            for $item in $items
            for $parent in $item/su:ParentLineId
            for $orderComponentItem in $component/oms:orderItem[oms:orderItemRef/
oms:LineXmlData/so:SalesOrderLine/corecom:Identification/
corecom:ApplicationObjectKey/corecom:ID/text() = $parent/text()]
            return (
                    <Update path="{fn:concat("/ControlData/Functions/Provision/
orderItem[@index='",fn:data($orderComponentItem/@index),"']")}">
                       <ExternalFulfillmentState>{$item/su:Status/text()}</
ExternalFulfillmentState>
                    </Update>
              )

            }
        </OrderDataUpdate>
    )
) else (
    log:info($log,concat('Received SOM Status Update: SOM_InProgress or
SOM_Canceled; ', $response/su:status/text())),
    automator:setUpdateOrder($automator,"true"),
    (
        <OrderDataUpdate xmlns="http://www.metasolv.com/OMS/OrderDataUpdate/
2002/10/25">
            {
            for $item in $items
            for $parent in $item/su:ParentLineId
            for $orderComponentItem in $component/oms:orderItem[oms:orderItemRef/
oms:LineXmlData/so:SalesOrderLine/corecom:Identification/
corecom:ApplicationObjectKey/corecom:ID/text() = $parent/text()]
            return (
                    <Update path="{fn:concat("/ControlData/Functions/Provision/
orderItem[@index='",fn:data($orderComponentItem/@index),"']")}">
                       <ExternalFulfillmentState>{$item/su:Status/text()}</
ExternalFulfillmentState>
                    </Update>
              )

            }
        </OrderDataUpdate>
    )
)
)
```

External XQuery Sender

The Automated Task editor external XQuery sender receives task data from an external system, then sends the data (after possibly transforming the data) to another external system or even returns the data back to the original external system. This XQuery combines characteristics of external XQuery automators and internal XQuery senders. 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:

  ```
  xmlns:context="java:com.mslv.oms.automation.TaskContext"
  ...
  <xsl:param name="context"/>
  ```

- Initial namespace declarations: You must declare ScriptSenderContextInvocation in any internal XSLT automator which extends ScriptReceiverContextInvocation. Always declare this class along with the automator java variable. For example:

```
xmlns:automator="java:oracle.communications.ordermanagement.automation.plugin
.ScriptSenderContextInvocation"
...
<xsl:param name="automator"/>
```

Oracle recommends that you use the standard Apache log class. Always declare this class along with the log java variable.

```
xmlns:log="java:org.apache.commons.logging.Log"
...
<xsl:param name="log"/>
```

You must use the TextMessage class for sending JMS based messages. Always declare this class along with the outboundMessage Java variable. You can use JMS text based messages to send OSM Web Service messages to other OSM systems, such as a service order from an OSM COM system to an OSM SOM system.

```
xmlns:outboundMessage="java:javax.jms.TextMessage"
...
<xsl:param name="outboundMessage"/>
```

> **Note:**
>
> If you need to support any other protocol for sending messages, you can implement a custom Java automation plug-in for the protocol or import a helper function implementation that supports the protocol.

- Body: The body for an internal XSLT sender can contain the following elements:
  - Use outboundMessage to set up the standard WebLogic JMS message properties for web services:

    ```
    <xsl:variable name="outboundMessage"
    select="java:setStringProperty($outboundMessage,
    '_wls_mimehdrContent_Type', 'text/xml; charset=&quot;utf-8&quot;')"/>
    ```

  - Use outboundMessage to set up the OSM Web Service URI JMS message property:

    ```
    <xsl:variable name="outboundMessage"
    select="java:setStringProperty($outboundMessage, 'URI', '/osm/wsapi')"/>
    ```

  - You can optionally use outboundMessage with the XML API to populate a JMS property value from order data. For example this code sets up an Ora_OSM_COM_OrderId parameter that is populated with the OSM order ID:

    ```
    <xsl:variable name="outboundMessage"
    select="java:setStringProperty($outboundMessage, 'Ora_OSM_COM_OrderId', /
    oms:GetOrder.Response/oms:OrderID)"/>
    ```

  - You can optionally use outboundMessage to set the JMS Correlation ID for the automation task before sending the message. This allows OSM to route a return message with the same corresponding JMS property value to an external XQuery automator on the same automation task as the original sender automation plug-in. For example, the following code sets the JMS correlation ID using the original OSM COM order:

    ```
    <xsl:variable name="void"
    select="java:setJMSCorrelationID($outboundMessage, concat($order/
    ```

```
oms:_root/oms:messageXmlData/ebo:ProcessSalesOrderFulfillmentEBM/
ebo:DataArea/ebo:ProcessSalesOrderFulfillment/corecom:Identification/
corecom:ID/text(),'-COM'))"/>
```

If this code were applied to "Message Example," the return value would be a concatenation of ScenarioA2 and -COM: `ScenarioA2-COM`.

> **Note:**
>
> Other correlation scenarios are possible. For example, you may send a message from automation task without expecting any response to the same automation task. In this scenario, another automation task further down in the process may be dedicated to receiving the response message, in which case an automation plug-in would be required that would set the correlation ID expected from the return message for that automated task. See "Using Automation" for more information about asynchronous communication scenarios.

- Access to the task level order data (the task view) using the XML API `GetOrder.Response` function call. For example, the following code provides access to all order data passed into the task as a variable that is then used in other variables to access different parts of the data:

```
    <xsl:template match="/">
        <xsl:variable name="order" select="oms:GetOrder.Response"/>
        <xsl:variable name="othervariable" select="$order/oms:_root/
oms:orderid"/>
```

- Any XSLT logic your plug-in requires, such as if-then or if-then-else statements that evaluate based on one or more parameters within the response message. For example, there could be a choice of two or more messages that could be sent depending on the order data values, or you might log a message.
- A completeTaskOnExit method statement that completes the plug-in and transitions the task to the next task based on the status selected if the plug-in is intended to end the task. Typically, an automated task would contain an internal XSLT sender plug-in for sending a message and an external XSLT receiver plug-in for receiving a message, but you can also create an automation that only sends an order with another automation that receives the order. This can be useful if the response message takes a long time to return. If you are expecting the system to respond that you sent the message to, you must configure the internal XSLT sender with a reply to queue that listens for a message acknowledgement, whether the response is returned to an external automator on the same automation task or on another automation task.

The following example provides the code for an XSLT that sends a message from an OSM system in the COM role to an OSM system in the SOM role using the OSM Web Service interface and assumes JMS communication over T3S.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" xmlns="http://www.metasolv.com/OMS/OrderDataUpdate"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:java="http://xml.apache.org/xslt/java"
    xmlns:xalan="http://xml.apache.org/xslt"
    xmlns:oms="urn:com:metasolv:oms:xmlapi:1"
xmlns:automator="java:oracle.communications.ordermanagement.automation.plugin.Scr
iptSenderContextInvocation"
```

```
    xmlns:context="java:com.mslv.oms.automation.TaskContext"
    xmlns:log="java:org.apache.commons.logging.Log"
    xmlns:outboundMessage="java:javax.jms.TextMessage"
    xmlns:to="http://TechnicalOrder"
    xmlns:provord="http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/
ProvisioningOrder/V1"
    xmlns:corecom="http://xmlns.oracle.com/EnterpriseObjects/Core/Common/V2"
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:ebo="http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/SalesOrder/V2"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    exclude-result-prefixes="xsl java xalan oms com ser soapenv xsi"
    xmlns:fn="http://www.w3.org/2005/02/xpath-functions">

    <!-- * -->
    <xsl:param name="automator"/>
    <xsl:param name="log"/>
    <xsl:param name="context"/>
    <xsl:param name="outboundMessage"/>

    <!-- * -->

    <xsl:output method="xml" indent="yes" omit-xml-declaration="no" xalan:indent-
amount="5"/>
    <xsl:template match="/">
        <xsl:variable name="order" select="oms:GetOrder.Response"/>
        <xsl:variable name="technicalActions" select="$order/oms:_root/
oms:TechnicalActions"/>
        <xsl:variable name="ebm" select="$order/oms:_root/oms:messageXmlData"/>
        <xsl:variable name="bi" select="$order/oms:_root/
oms:CaptureInteractionResponse"/>
        <xsl:variable name="outboundMessage"
select="java:setStringProperty($outboundMessage, '_wls_mimehdrContent_Type',
'text/xml; charset=&amp;quot;utf-8&amp;quot;')"/>
        <xsl:variable name="outboundMessage"
select="java:setStringProperty($outboundMessage, 'URI', '/osm/wsapi')"/>
        <xsl:variable name="outboundMessage"
select="java:setStringProperty($outboundMessage, 'Ora_OSM_COM_OrderId', /
oms:GetOrder.Response/oms:OrderID)"/>
        <xsl:variable name="void"
select="java:setJMSCorrelationID($outboundMessage, concat($order/oms:_root/
oms:messageXmlData/ebo:ProcessSalesOrderFulfillmentEBM/ebo:DataArea/
ebo:ProcessSalesOrderFulfillment/corecom:Identification/corecom:ID/text(),'-
COM'))"/>
        <xsl:variable name="log" select=java:info($log,concat('Sending Service
Order for COM order: ', $order/oms:OrderID))"/>
        <xsl:call-template name="sendSomOrder"/>
    </xsl:template>
    <!-- ================================
         Create the SOAP message for the sendSomOrder call
         ================================== -->
    <xsl:template name="sendSomOrder">

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ord="http://xmlns.oracle.com/communications/ordermanagement">
        <soapenv:Header>
        <wsse:Security xmlns:wsse = "http://docs.oasis-open.org/wss/2004/01/
oasis-200401-wss-wssecurity-secext-1.0.xsd" soapenv:mustUnderstand="1">
        <wsse:UsernameToken xmlns:wsu = "http://docs.oasis-open.org/wss/2004/01/
oasis-200401-wss-wssecurity-utility-1.0.xsd" wsu:Id="UsernameToken-4799946">
        <wsse:Username>demo</wsse:Username>
        <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
```

```
wss-username-token-profile-1.0#PasswordText">passw0rd</wsse:Password>
        </wsse:UsernameToken>
        </wsse:Security>
        </soapenv:Header>
   <soapenv:Body>
      <ord:CreateOrder>
                <ebo:ProcessProvisioningOrderEBM xmlns:ebo="http://
xmlns.oracle.com/EnterpriseObjects/Core/EBO/ProvisioningOrder/V1">
<ebo:DataArea>
                              <corecom:Process xmlns="http://xmlns.oracle.com/
EnterpriseObjects/Core/EBO/ProvisioningOrder/V1" xmlns:corecom="http://
xmlns.oracle.com/EnterpriseObjects/Core/Common/V2" xmlns:aia="http://
www.oracle.com/XSL/Transform/java/oracle.apps.aia.core.xpath.AIAFunctions"
xmlns:xref="http://www.oracle.com/XSL/Transform/java/
oracle.tip.xref.xpath.XRefXPathFunctions"
xmlns:oms="urn:com:metasolv:oms:xmlapi:1" xmlns:provord="http://xmlns.oracle.com/
EnterpriseObjects/Core/EBO/ProvisioningOrder/V1"/>
                              <provord:ProcessProvisioningOrder xmlns="http://
xmlns.oracle.com/EnterpriseObjects/Core/EBO/ProvisioningOrder/V1"
xmlns:aia="http://www.oracle.com/XSL/Transform/java/
oracle.apps.aia.core.xpath.AIAFunctions" xmlns:xref="http://www.oracle.com/XSL/
Transform/java/oracle.tip.xref.xpath.XRefXPathFunctions"
xmlns:oms="urn:com:metasolv:oms:xmlapi:1" xmlns:provord="http://xmlns.oracle.com/
EnterpriseObjects/Core/EBO/ProvisioningOrder/V1">
                              <corecom:SalesOrderReference>
                                   <corecom:SalesOrderIdentification>
                                       {$order/oms:_root/oms:ServiceOrder/
cord:Order/cord:CustomerDetails/cord:OrderNumber/corecom:Identification/*}
                                   </corecom:SalesOrderIdentification>
                              </corecom:SalesOrderReference>

<provord:RequestedDeliveryDateTime>2010-07-16T08:24:38Z </
provord:RequestedDeliveryDateTime>
                              <provord:TypeCode>SALES ORDER</provord:TypeCode>
                              <provord:FulfillmentPriorityCode>5</
provord:FulfillmentPriorityCode>
                              <provord:FulfillmentSuccessCode>DEFAULT </
provord:FulfillmentSuccessCode>
                              <provord:FulfillmentModeCode>DELIVER</
provord:FulfillmentModeCode>
                              <provord:ProcessingNumber/>
                              <provord:ProcessingTypeCode/>
                              <corecom:Status xmlns:corecom="http://
xmlns.oracle.com/EnterpriseObjects/Core/Common/V2">
                                  <corecom:Code>IN PROGRESS</corecom:Code>
                              </corecom:Status>
                              <corecom:BusinessUnitReference
xmlns:corecom="http://xmlns.oracle.com/EnterpriseObjects/Core/Common/V2">
                                  <corecom:BusinessUnitIdentification>
                                      <corecom:ID schemeID="ORGANIZATION_ID"
schemeAgencyID="SEBL_01">0-R9NH</corecom:ID>
                                  </corecom:BusinessUnitIdentification>
                              </corecom:BusinessUnitReference>
                              {$order/oms:_root/oms:ServiceOrder/cord:Order/
cord:CustomerDetails/cord:CustomerParty/corecom:CustomerPartyReference}
                              <corecom:ParentProvisioningOrderReference
xmlns:corecom="http://xmlns.oracle.com/EnterpriseObjects/Core/Common/V2">
                                  <corecom:ProvisioningOrderIdentification>
                                      <corecom:BusinessComponentID
schemeID="SALESORDER_ID" schemeAgencyID="COMMON"/>
                                  </corecom:ProvisioningOrderIdentification>
```

```
                                </corecom:ParentProvisioningOrderReference>
                                {
                                    for $x in $order/oms:_root/oms:ServiceOrder/
cord:Order/cord:ServiceOrderLine
                                    return
                                    <provord:ProvisioningOrderLine>
                                    <corecom:Identification xmlns:corecom="http://
xmlns.oracle.com/EnterpriseObjects/Core/Common/V2">
                                        <corecom:BusinessComponentID>{concat($x/
@id,'')} </corecom:BusinessComponentID>
                                        <corecom:ID schemeID="SALESORDER_LINEID"
schemeAgencyID="SEBL_01">{concat($x/@id,'')}</corecom:ID>
                                        <corecom:ApplicationObjectKey>
                                             <corecom:ID
schemeID="SALESORDER_LINEID" schemeAgencyID="SEBL_01">{concat($x/@id,'')}</
corecom:ID>
                                        </corecom:ApplicationObjectKey>
                                    </corecom:Identification>
                                    <provord:OrderQuantity>1</provord:OrderQuantity>
                                    <provord:ServiceActionCode>{$x/cord:Action/
text()} </provord:ServiceActionCode>
                                    <provord:ServicePointCode/>
                                    <corecom:Status xmlns:corecom="http://
xmlns.oracle.com/EnterpriseObjects/Core/Common/V2">
                                        <corecom:Code>IN PROGRESS</corecom:Code>
                                    </corecom:Status>
                                    <corecom:ServiceAddress xmlns:corecom="http://
xmlns.oracle.com/EnterpriseObjects/Core/Common/V2">
                                        <corecom:Identification>
                                            <corecom:BusinessComponentID
schemeAgencyID="COMMON"
schemeID="CUSTOMERPARTY_ADDRESSID">2d323733323231313531313836313331</
corecom:BusinessComponentID>
                                            <corecom:ApplicationObjectKey>
                                                 <corecom:ID
schemeAgencyID="SEBL_01" schemeID="CUSTOMERPARTY_ADDRESSID">88-2KKNH</corecom:ID>
                                            </corecom:ApplicationObjectKey>
                                        </corecom:Identification>
                                        <corecom:LineOne>{$x/cord:Address/
cord:LineOne/text()} </corecom:LineOne>
                                        <corecom:CityName>{$x/cord:Address/
cord:CityName/text()} </corecom:CityName>
                                        <corecom:StateName>{$x/cord:Address/
cord:StateName/text()} </corecom:StateName>
                                        <corecom:ProvinceName>{$x/cord:Address/
cord:ProvinceName/ text()}</corecom:ProvinceName>
                                        <corecom:CountryCode>{$x/cord:Address/
cord:CountryCode /text()}</corecom:CountryCode>
                                        <corecom:PostalCode>{$x/cord:Address/
cord:PostalCode /text()}</corecom:PostalCode>
                                    </corecom:ServiceAddress>
                                    <corecom:ItemReference xmlns:corecom="http://
xmlns.oracle.com/EnterpriseObjects/Core/Common/V2">
                                    <corecom:ItemIdentification>
                                        <corecom:BusinessComponentID
schemeAgencyID="COMMON" schemeID="ITEM_ITEMID"/>
                                        <corecom:ApplicationObjectKey>
                                            <corecom:ID schemeID="ITEM_ITEMID"
schemeAgencyID="SEBL_01">{concat($x/cord:InstanceID/text(),'')}</corecom:ID>
                                        </corecom:ApplicationObjectKey>
                                        <corecom:AlternateObjectKey>
```

```
                                        <corecom:ContextID/>
                                    </corecom:AlternateObjectKey>
                                    <corecom:SupplierItemID/>
                                </corecom:ItemIdentification>
                                <corecom:Name>{concat($x/@name,'')}</
corecom:Name>
                                    <corecom:ClassificationCode
listID="PermittedTypeCode"></corecom:ClassificationCode>
                                    <corecom:ClassificationCode
listID="BillingProductTypeCode"/>
                                    <corecom:ClassificationCode
listID="FulfillmentItemCode">{concat($x/@name,'')}</corecom:ClassificationCode>
                                    <corecom:ServiceIndicator>false</
corecom:ServiceIndicator>
                                    <corecom:TypeCode>SERVICE</corecom:TypeCode>
                                    <corecom:Description/>
                                    <corecom:SpecificationGroup>
                                        <corecom:Name>ExtensibleAttributes</
corecom:Name>
                                        {
                                            for $y in $x/cord:Attribute
                                            return
                                            <corecom:Specification>
                                                <corecom:ServiceActionCode> </
corecom:ServiceActionCode>
                                                <corecom:Name>{concat($y/
@name,'')} </corecom:Name>
                                                <corecom:DataTypeCode>Text</
corecom:DataTypeCode>
                                                <corecom:Value>{$y/cord:Value/
cord:value/text()} </corecom:Value>
                                            </corecom:Specification>
                                        }
                                    </corecom:SpecificationGroup>

<corecom:PrimaryClassificationCode>{concat($x/@name,'')} </
corecom:PrimaryClassificationCode>
                                    <corecom:ServiceInstanceIndicator>true </
corecom:ServiceInstanceIndicator>
                                </corecom:ItemReference>

<provord:ProvisioningOrderLineSpecificationGroup>
                                    <corecom:SpecificationGroup>
                                        <corecom:Name>ExtensibleAttributes</
corecom:Name>
                                        <corecom:Specification>

<corecom:Name>ParentSalesOrderLine</corecom:Name>
                                            <corecom:Value>{$x/
cord:primaryMapping/text()} </corecom:Value>
                                        </corecom:Specification>
                                        {
                                            for $z in $x/cord:secondaryMapping
                                            return
                                            <corecom:Specification>

<corecom:Name>ParentSalesOrderLine</corecom:Name>
                                            <corecom:Value>{$z/text()}</
corecom:Value>
                                            </corecom:Specification>
                                        }
```

```
                                    </corecom:SpecificationGroup>
                                </
provord:ProvisioningOrderLineSpecificationGroup>
                                    </provord:ProvisioningOrderLine>
                                }
                            </provord:ProcessProvisioningOrder>
                        </ebo:DataArea>
                    </ebo:ProcessProvisioningOrderEBM>
      </ord:CreateOrder>
   </soapenv:Body>
</soapenv:Envelope>
</xsl:template>
    <!-- * -->
    <xsl:template match="* | @* | text()">
        <!-- do nothing -->
        <xsl:apply-templates/>
    </xsl:template>
</xsl:stylesheet>
```

External XSLT Automator

The Automated Task editor external XSLT automator receives task data from an external system and optionally updates OSM order data. The XSLT has the following characteristics:

- XSLT context in prolog: The input document for any automated task automation plug-in is the order data defined in the Automation Task editor Task Data tab. You can access this data by declaring the TaskContext OSM Java class. Always declare this class along with the context java binding. For example:

  ```
  xmlns:context="java:com.mslv.oms.automation.TaskContext"
  ...
  <xsl:param name="context"/>
  ```

- Prolog: You must declare ScriptReceiverContextInvocation in any external XQuery automator. Typically, you can use the getOrderAsDOM method to receive external messages and the setUpdateOrder method to update the order data. Always declare this class along with the automator java binding. For example:

  ```
  xmlns:automator="java:oracle.communications.ordermanagement.automation.plugin
  .ScriptReceiverContextInvocation"
  ...
  <xsl:param name="automator"/>
  ```

  Oracle recommends that you use the standard Apache log class. Always declare this class along with the $log java binding.

  ```
  xmlns:log="java:org.apache.commons.logging.Log"
  ...
  <xsl:param name="log"/>
  ```

  Another necessary declaration includes the xmlapi namespace, that you can use with the ScriptReceiverContextInvocation getOrderAsDom method to retrieve the order data for the task as a variable. This task data variable can be used in an OrderDataUpdate to update the order data with the data values received in the response message, if an update to the order data is required. For example:

  ```
  xmlns:oms="urn:com:metasolv:oms:xmlapi:1"
  <xsl:variable name="taskData"
  select="fn:root(java:getOrderAsDOM($automator))/oms:GetOrder.Response"/>
  ```

- Body: The body for an external XSLT automator can contain the following elements:
  - Any XSLT logic your plug-in requires, such as if-then or if-then-else statements that evaluate based on one or more parameters within the response message, or you might log a message.
  - A setUpdateOrder method statement that indicates whether there is an order data update. This method should be identical to what you selected in the Design Studio automation plug-in Properties View XSLT Tab Update Order check box.
  - A completeTaskOnExit method statement that completes the plug-in and transitions the task to the next task based on the status selected, if the plug-in is intended to end the task. Since there can be multiple plug-ins within a task, you would only need this method in the last plug-in listed. For example, the Failed status might transition to a fallout task, and the Succeed status may transition to the next task in the process.
  - An OrderDataUpdate statement that updates the order data based on the information returned in the response. 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 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.Scr
iptReceiverContextInvocation"
    xmlns:context="java:com.mslv.oms.automation.TaskContext"
    xmlns:log="java:org.apache.commons.logging.Log"
    xmlns:su="http://StatusUpdate"
    xmlns:so="http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/SalesOrder/V2"
    xmlns:corecom="http://xmlns.oracle.com/EnterpriseObjects/Core/Common/V2"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    exclude-result-prefixes="xsl java xalan oms soapenv xsi">

    <!-- * -->
    <xsl:param name="automator"/>
    <xsl:param name="log"/>
    <xsl:param name="context"/>
```

```
    <!-- * -->
    <xsl:output method="xml" indent="yes" omit-xml-declaration="no" xalan:indent-
amount="5"/>

    <xsl:template match="/">
     <xsl:variable name="taskData"
select="fn:root(java:getOrderAsDOM($automator))/oms:GetOrder.Response"/>
     <xsl:variable name="response" select="fn:root()/su:StatusUpdate (:
fn:root(.) :)"/>
     <xsl:variable name="items" select="fn:root()/su:StatusUpdate/su:OrderItem"/>
    <xsl:variable name="component" select="if (fn:exists($taskData/oms:_root/
oms:ControlData/oms:Functions/*/oms:componentKey)) then $taskData/oms:_root/
oms:ControlData/oms:Functions/*[fn:position()=1] else ()"/>
        <xsl:apply-templates/>
</xsl:template>

<!-- Match the status SOM_Complete -->
    <xsl:template match="$response[su:status/text()='SOM_Completed']">
     <xsl:variable name="log" select="java:info($log,concat('Received SOM Status
Update: SOM_Completed; ', $response/su:status/text()))"/>
     <xsl:variable name="automator" select="java:setUpdateOrder($automator,
true())"/>
     <xsl:variable name="context" select="java:completeTaskOnExit($context,
success())"/>
     <OrderDataUpdate xmlns="http://www.metasolv.com/OMS/OrderDataUpdate/
2002/10/25">
        <xsl:for-each select="su:ParentLineId">
            <xsl:variable name="parent" select="."/>
            <xsl:for-each select="$component/oms:orderItem[oms:orderItemRef/
oms:LineXmlData/so:SalesOrderLine/corecom:Identification/
corecom:ApplicationObjectKey/corecom:ID/text() = $parent/text()]">
                <xsl:variable name="index" select="@index"/>
                    <Update path="{fn:concat("/ControlData/Functions/Provision/
orderItem[@index='",fn:data($orderComponentItem/@index),"']")}>
                     <ExternalFulfillmentState>{$item/su:Status/text()}</
ExternalFulfillmentState>
                    </Update>
            </xsl:for-each>
        </xsl:for-each>
     </OrderDataUpdate>
    </xsl:template>

<!-- Match the status SOM_Failed -->
    <xsl:template match="$response[su:status/text()='SOM_Failed']">
     <xsl:variable name="log" select="java:info($log,concat('Received SOM Status
Update: SOM_Failed; ', $response/su:status/text()))"/>
     <xsl:variable name="automator" select="java:setUpdateOrder($automator,
true())"/>
     <xsl:variable name="context" select="java:completeTaskOnExit($context,
success())"/>
     <OrderDataUpdate xmlns="http://www.metasolv.com/OMS/OrderDataUpdate/
2002/10/25">
        <xsl:for-each select="su:ParentLineId">
            <xsl:variable name="parent" select="."/>
            <xsl:for-each select="$component/oms:orderItem[oms:orderItemRef/
oms:LineXmlData/so:SalesOrderLine/corecom:Identification/
corecom:ApplicationObjectKey/corecom:ID/text() = $parent/text()]">
                <xsl:variable name="index" select="@index"/>
                    <Update path="{fn:concat("/ControlData/Functions/Provision/
orderItem[@index='",fn:data($orderComponentItem/@index),"']")}>
                     <ExternalFulfillmentState>{$item/su:Status/text()}</
```

```
ExternalFulfillmentState>
                    </Update>
            </xsl:for-each>
        </xsl:for-each>
     /OrderDataUpdate>
    </xsl:template>

    <xsl:template match="$response[su:status/text()='']">
     <xsl:variable name="log" select="java:info($log,concat('Received SOM Status
Update: SOM_InProgress or SOM_Canceled; ', $response/su:status/text()))"/>
     <xsl:variable name="automator" select="java:setUpdateOrder($automator,
false())"/>
     <xsl:variable name="context" select="java:completeTaskOnExit($context,
success())"/>
     <OrderDataUpdate xmlns="http://www.metasolv.com/OMS/OrderDataUpdate/
2002/10/25">
        <xsl:for-each select="su:ParentLineId">
            <xsl:variable name="parent" select="."/>
            <xsl:for-each select="$component/oms:orderItem[oms:orderItemRef/
oms:LineXmlData/so:SalesOrderLine/corecom:Identification/
corecom:ApplicationObjectKey/corecom:ID/text() = $parent/text()]">
                <xsl:variable name="index" select="@index"/>
                    <Update path="{fn:concat("/ControlData/Functions/Provision/
orderItem[@index='",fn:data($orderComponentItem/@index),"']")}>
                     <ExternalFulfillmentState>{$item/su:Status/text()}</
ExternalFulfillmentState>
                    </Update>
            </xsl:for-each>
        </xsl:for-each>
     </OrderDataUpdate>
    </xsl:template>

<!-- * -->
    <xsl:template match="* | @* | text()">
        <!-- do nothing -->
        <xsl:apply-templates/>
    </xsl:template>
</xsl:stylesheet>
```

External XSLT Sender

The Automated Task editor external XSLT sender receives task data from an external system, then sends the data (after possibly transforming the data) to another external system or even returns the data back to the original external system. This XSLT combines characteristics of external XSLT automators and internal XSLT senders. 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."

> **Note:**
>
> For an event automation plug-in you must declare the OrderNotificationContext instead of TaskContext. For example:
>
> ```
> declare namespace context =
> "java:com.mslv.oms.automation.OrderNotificationContext";
> ```

The following example is an internal sender automation plug-in that uses methods available to the OrderNotificationContext class to get milestone data from the order and sends an notification message to an external system. Because this sender does not expect a response message (a fire-and-forget message), you must use the OrderNotificationContext class ackNotificationOnExit method to clear the JMS correlation ID for the notification. Also, events do not transition tasks, so you must not specify completeTaskOnExit in a notification.

```
declare namespace saxon="http://saxon.sf.net/";
declare namespace xsl="http://www.w3.org/1999/XSL/Transform";
```

```
declare namespace log = "java:org.apache.commons.logging.Log";
declare namespace outboundMessage = "java:javax.jms.TextMessage";
declare namespace oms="urn:com:metasolv:oms:xmlapi:1";
declare namespace osm="http://xmlns.oracle.com/communications/ordermanagement/
model";
declare namespace context =
"java:com.mslv.oms.automation.OrderNotificationContext";

declare variable $context external;
declare variable $log external;
declare variable $outboundMessage external;

let $taskData := fn:root(.)/oms:GetOrder.Response
let $correlationId := $taskData/oms:_root/oms:Id/text()
let $controlDataArea := if (fn:exists($taskData/oms:_root/oms:ControlData))
                            then $taskData/oms:_root/oms:ControlData
                            else ()

return
(
log:info($log, fn:concat('COMCartridge: Invoking orderCompletionNotification
for order[',$taskData/oms:OrderID/text(),'] with correlation [',
$correlationId,']')),
context:ackNotificationOnExit($context),
outboundMessage:setStringProperty($outboundMessage, "COMCorrelationID",
$correlationId),
outboundMessage:setStringProperty($outboundMessage, "SUB_FOLDER_NAME", $taskData/
oms:_root/oms:OrderNumber/text()),
outboundMessage:setStringProperty($outboundMessage, "COMMilestone",
"COMOrderCompleteEvent"),
<orderNotification xmlns="http://xmlns.oracle.com/communications/sce/dictionary/
CommonResourcesCartridge/Notifications"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <OSMOrderId>{$taskData/oms:OrderID/text()}</OSMOrderId>
    <Id>{$correlationId}</Id>
    <OrderNumber>{$taskData/oms:_root/oms:OrderNumber/text()}</OrderNumber>
    {
    for $serviceInstance in $controlDataArea/oms:OrderItem
    return
        <Instance>
            <InstanceID>{$serviceInstance/oms:instanceID/text()}</InstanceID>
            <OrderLineId>{$serviceInstance/oms:orderLineId/text()}</OrderLineId>
            <Status>{$serviceInstance/oms:status/text()}</Status>
        </Instance>
    }
</orderNotification>
)
```

Jeopardy Automators

An order jeopardy automation plug-in can be triggered when a particular condition is met, such as when a task exceeds the expected duration configured for the task or when the process that the task is a part of exceeds its excepted process duration. The automation can be any internal XQuery, XSLT, or custom automation since the jeopardy, by definition, can only be triggered by events happening within the task or the process. For more information about the characteristics for these automations, see "Automation Plug-in XQuery Examples," "Automation Plug-in XSLT Examples," and "Custom Java Automation Plug-ins."

> **Note:**
>
> For an order level jeopardy automation plug-in you must declare the OrderNotificationContext instead of TaskContext. For example:
>
> ```
> declare namespace context =
> "java:com.mslv.oms.automation.OrderNotificationContext";
> ```
>
> For a task level jeopardy automation plug-in, if the task level jeopardy condition **Multiple events per Task instance** is set indicating that the task is a multi-instance task and the event should be triggered for each instance, then you must declare `TaskNotificationContext` so that the task data is passed to each instance of the event. If the task is not a multi-instance task, then `OrderNotificationContext` should be declared.

The following example is an internal automator plug-in that uses methods available to the OrderNotificationContext class to get notification details from the task in combination with the XML API Notification.Request that logs the jeopardy notification details. Other jeopardy examples could also send an email or trigger a pager.

```
declare namespace oms="urn:com:metasolv:oms:xmlapi:1";
declare namespace automator =
"java:oracle.communications.ordermanagement.automation.plugin.ScriptReceiverConte
xtInvocation";
declare namespace context =
"java:com.mslv.oms.automation.OrderNotificationContext";
declare namespace log = "java:org.apache.commons.logging.Log";

declare option saxon:output "method=xml";
declare option saxon:output "saxon:indent-spaces=2";

declare variable $automator external;
declare variable $context external;
declare variable $log external;

declare variable $exitStatus := "success";

let $thisOrderId := context:getOrderId($context)
(: let $taskMnemonic := context:getTaskMnemonic($context) :)
let $notificationName := context:getNotificationName($context)
let $notificationType := context:getNotificationType($context)
let $orderId := fn:root(.)/oms:GetOrder.Response/oms:_root/oms:orderId
let $xmlRequest := '<Notifications.Request
xmlns="urn:com:metasolv:oms:xmlapi:1" />'
let $notifications := context:processXMLRequest($context, $xmlRequest)
return (
  log:info($log, fn:concat("XQuery jeopardy: order[", $thisOrderId,
    "], notificationContext [", context:getClass($context),
    "], notificationName[", $notificationName,
    "], notificationType[", $notificationType,
    "], notifications[", $notifications,
    "] entered order ID [", $orderId/text(), "]")),
  <placeholder/>
)
```

Order Notification Automation Plug-ins

An order notification automation plug-in can be triggered when specified data changes in the order. For example, you can monitor order status changes using the orchestration data element **ControlData/OrderFulfillmentState** or individual order item status changes using **ControlData/OrderItem/OrderItemFulfillmentState** so OSM triggers an internal XQuery sender automation plug-in that sends these status changes to another system, such as from a SOM OSM system to a COM OSM system, or from a COM OSM system to a CRM.

The automation can be any internal XQuery, XSLT, or custom automation since the notification, by definition, can only be triggered by a change in the internal order data. For more information about the characteristics for these automations, see "Automation Plug-in XQuery Examples," "Automation Plug-in XSLT Examples," and "Custom Java Automation Plug-ins."

> **Note:**
>
> For an order notification automation plug-in you must declare the OrderDataChangeNotificationContext instead of TaskContext. For example:
>
> ```
> declare namespace context =
> "java:com.mslv.oms.automation.OrderDataChangeNotificationContext";
> ```

The following example is an internal XQuery sender that sends any order and order item fulfillment state changes to another OSM system. It also provides stubs for transforming the fulfillment states to external system message formats.

```
declare namespace osm="urn:com:metasolv:oms:xmlapi:1";
declare namespace log = "java:org.apache.commons.logging.Log";
declare namespace to="http://TechnicalOrder";
declare namespace automator =
"java:oracle.communications.ordermanagement.automation.plugin.ScriptSenderContext
Invocation";
declare namespace su="http://StatusUpdate";
declare namespace context =
"java:com.mslv.oms.automation.OrderDataChangeNotificationContext";
declare namespace outboundMessage = "java:javax.jms.TextMessage";

declare variable $log external;
declare variable $outboundMessage external;

(:
   This function is for indication purposes only.
   OSM Fulfillment State can be mapped according the expectation of Upstream
:)
declare function local:getUpstreamFulfillmentState($fulfillmentState as
xs:string) as xs:string {
    (: fn:concat('Order_Upstream_' , $fulfillmentState) :)
    fn:concat('' , $fulfillmentState)
};

(:
   This function is for indication purposes only.
   OSM Fulfillment State can be mapped according the expectation of Upstream
```

```
:)
declare function local:getUpstreamOrderItemFulfillmentState($fulfillmentState as
xs:string) as xs:string {
    (: fn:concat('OrderItem_Upstream_' , $fulfillmentState) :)
    fn:concat('' , $fulfillmentState)
};

let $order := ..//osm:GetOrder.Response
let $orderFulfillmentState := $order/osm:_root/osm:ControlData/
osm:OrderFulfillmentState
let $mappedUpstreamFulfillmentState := if(exists($orderFulfillmentState)) then
local:getUpstreamFulfillmentState($orderFulfillmentState/text()) else ()

return
(
log:info($log,'Sending Upstream Fulfillment State'),
outboundMessage:setStringProperty($outboundMessage, "SOMTOMCorrelationHeader",
concat($order/osm:_root/osm:messageXmlData/to:TechnicalOrder/to:SOMOrderId/
text(),'-SOM')),
if (fn:count($order/osm:_root/osm:ControlData/osm:OrderItem)=0) then (
<StatusUpdate xmlns="http://StatusUpdate">
<numSalesOrder>{$order/osm:Reference/text()}</numSalesOrder>
<numOrder>{$order/osm:OrderID/text()}</numOrder>
<typeOrder>{$order//osm:OrderHeader/osm:typeOrder/text()}</typeOrder>
<errorCode>0</errorCode>
<status>cancelled</status>
</StatusUpdate>
) else (
<StatusUpdate xmlns="http://StatusUpdate">
<numSalesOrder>{$order/osm:Reference/text()}</numSalesOrder>
<numOrder>{$order/osm:OrderID/text()}</numOrder>
<typeOrder>{$order//osm:OrderHeader/osm:typeOrder/text()}</typeOrder>
<errorCode>0</errorCode>
<status>{$mappedUpstreamFulfillmentState}</status>
{
    for $orderItem in $order/osm:_root/osm:ControlData/osm:OrderItem
    where exists($orderItem/osm:OrderItemFulfillmentState)
    return
        <OrderItem>
            <LineName>{$orderItem/osm:LineName/text()}</LineName>
            <LineId>{$orderItem/osm:LineId/text()}</LineId>
            <ParentLineId>{$orderItem/osm:ParentLineId/text()}</ParentLineId>
            <SpecificationName>{$orderItem/osm:TypeCode/text()}</
SpecificationName>
            <Status>{local:getUpstreamOrderItemFulfillmentState($orderItem/
osm:OrderItemFulfillmentState/text())}</Status>
        </OrderItem>
    }
</StatusUpdate>
)
)
```

Custom Java Automation Plug-ins

This topic provides common usage examples for custom Java automation plug-ins.

- Internal Custom Java Automator
- Internal Custom Java Sender
- External Custom Java Automator that Changes the OSM Task Status

Internal Custom Java Automator

A basic internal custom Java automator has the following characteristics:

- The name of the custom automation package. For example:

```
package com.mslv.oms.sample.atm_frame;
```

- Import statements required for this custom automation plug-in. For example:

```
import com.mslv.oms.automation.plugin.*;
import com.mslv.oms.automation.*;
import java.rmi.*;
```

- An arbitrary class name that extends AbstractAutomator. For the automation framework to call an internal custom Java automator, the plug-in must extend the AbstractAutomator class. This class resides in the com.mslv.automation.plugin package. For example:

```
public class MyPlugin extends AbstractAutomator {
```

- The required run method, as dictated by the parent class, AbstractAutomator

```
protected void run(String inputXML, AutomationContext context)
                throws com.mslv.oms.automation.AutomationException {
```

- Cast the AutomationContext object to the TaskContext object. This example assumes that the custom automation plug-in is triggered by an automated task, so the code is expecting the context input an argument to be an instance of the TaskContext object.

```
TaskContext taskContext = (TaskContext)context;
```

> **Note:**
>
> You can use the TaskContext object to do many things, such as complete the task, suspend it, and so on. For more information about this class, see the OSM Javadocs.

- Call a method on the TaskContext object to retrieve the task name.

```
String taskName = taskContext.getTaskMnemonic();
```

- Add any require business logic.

```
this.performAutomation(taskname);
```

The following example shows the minimal amount of code required for a custom automation plug-in to run. This example assumes that it is triggered by an automated task.

```
package com.mslv.oms.sample.atm_frame;

  import com.mslv.oms.automation.plugin.*;
```

```
import com.mslv.oms.automation.*;
import java.rmi.*;

public class MyPlugin extends AbstractAutomator {
 protected void run(String inputXML, AutomationContext context)
                throws com.mslv.oms.automation.AutomationException {
    try {
      TaskContext taskContext = (TaskContext)context;
      String taskName = taskContext.getTaskMnemonic();
      this.performAutomation(taskname);
    catch(RemoteException ex) {
      throw new AutomationException(ex); }
    catch(AutomationException x) {
      throw x; }
  }
}
```

Internal Custom Java Sender

A basic internal custom Java sender has the following characteristics:

- The name of the custom automation package. For example:

  ```
  package com.mslv.oms.sample.atm_frame;
  ```

- Import statements required for this custom automation plug-in. For example:

  ```
  import com.mslv.oms.automation.plugin.*;
  import com.mslv.oms.automation.*;
  import java.rmi.*;
  ```

- An arbitrary class name that extends AbstractSendAutomator. For the automation framework to call an internal custom Java sender, the plug-in must extend the AbstractSendAutomator class. This class resides in the com.mslv.automation.plugin package. For example:

  ```
  public class MyPlugin extends AbstractSendAutomator {
  ```

- The required run method, as dictated by the parent class, AbstractSendAutomator

  ```
  protected void run(String inputXML, AutomationContext context)
                  throws com.mslv.oms.automation.AutomationException {
  ```

- Cast the AutomationContext object to the TaskContext object. This example assumes that the custom automation plug-in is triggered by an automated task, so the code is expecting the context input an argument to be an instance of the TaskContext object.

  ```
  TaskContext taskContext = (TaskContext)context;
  ```

  > **Note:**
  >
  > You can use the TaskContext object to do many things, such as complete the task, suspend it, and so on. For more information about this class, see the OSM Javadocs.

- Call a method on the TaskContext object to retrieve the task name.

  ```
  String taskName = taskContext.getTaskMnemonic();
  ```

- Sets the text for the outbound message, which is sent to the external message queue defined by the automation definition. The custom code does not establish a connection to an external system or send the message; the automation framework handles the connection and sends the message upon completion of the makeRequest method.

```
        outboundMessage.setText("Received task event for task = " +
taskName);}
```

> **Note:**
>
> OSM provides outboundMessage in the OSM automation framework as a JMS message with text content. If you require other message formats or protocols, do not use outboundMessage. You must implement an internal custom java automator or helper class with the required code.

The following example shows the minimal amount of code required for a custom automation plug-in that sends data to run. This example assumes that it is triggered by an automated task.

```
  package com.mslv.oms.sample.atm_frame;

  import com.mslv.oms.automation.plugin.*;
  import com.mslv.oms.automation.*;
  import javax.jms.TextMessage;
  import java.rmi.*;

  public class MyPlugin extends AbstractSendAutomator {
    protected void makeRequest(String inputXML, AutomationContext context,
                  TextMessage outboundMessage)
                  throws com.mslv.oms.automation.AutomationException {
      try {
        TaskContext taskContext = (TaskContext)context;
        String taskName = taskContext.getTaskMnemonic();

        // optional - You can use this code if you want to define your own
correlation ID rather than an autogenerated correlation ID.
        Correlator correlator = getCorrelator(context);
        correlator.add(createCustomCorrelationId(taskContext));

        outboundMessage.setText("Received task event for task = " + taskName);}
      catch(javax.jms.JMSException ex) {
        throw new AutomationException(ex); }
      catch(RemoteException x) {
        throw new AutomationException(x); }
    }

    private String createCustomCorrelationId(TaskContext taskContext) {
     // Create a custom correlation ID using task name and unique order history
ID
     // Actual correlation calculation depends on solution logic
        String corrId = taskContext.getTaskMnemonic()
                        + "-"
                        + String.valueOf(taskContext.getOrderHistoryId());
        return corrId;
    }

  }
```

External Custom Java Automator that Changes the OSM Task Status

A basic external custom Java automator that changes the OSM task status has the following characteristics:

- The name of the custom automation package. For example:

```
package com.mslv.oms.sample.atm_frame;
```

- Import statements required for this custom automation plug-in. For example:

```
import com.mslv.oms.automation.plugin.*;
import com.mslv.oms.automation.*;
import java.rmi.*;
```

- An arbitrary class name that extends AbstractAutomator. For the automation framework to call an external custom Java sender, the plug-in must extend the AbstractAutomator class. This class resides in the com.mslv.automation.plugin package. The name reflects that this example is an external event receiver, receiving information from ASAP. For example:

```
public class AsapResponseHandler extends AbstractAutomator {
```

- The required run method, as dictated by the parent class, AbstractAutomator.

```
public void run(String inputXML, AutomationContext task)
            throws AutomationException {
```

- Cast the AutomationContext object to the TaskContext object. This example assumes that the custom automation plug-in is triggered by an automated task, so the code is expecting the context input an argument to be an instance of the TaskContext object.

```
TaskContext taskContext = (TaskContext)context;
```

> **Note:**
>
> You can use the TaskContext object to do many things, such as complete the task, suspend it, and so on. For more information about this class, see the OSM Javadocs.

- Call a method on the TaskContext object to retrieve the task name.

```
String taskName = taskContext.getTaskMnemonic();
```

- Logs the information regarding the response that the plug-in is handling. AtmFrameCatalogLogger is available to this example plug-in based on the package in which the plug-in resides. You must replace this with your own solution logic.

```
AtmFrameCatalogLogger.logTaskEventResponse
  (taskName,tctx.getOrderId(),tctx.getOrderHistoryId(),inputXML);
```

> **Note:**
>
> The automation framework keeps track of the order ID and the order history ID of the task that triggered the automation. There are two ways you can get the Order History ID:
>
> – By parsing the inputXML
>
> – By calling the TaskContext.getOrderHistoryId method as shown in this example.
>
> In most cases, these return the same order history ID. However, if you use automation to handle task events, the order history ID obtained from:
>
> – Parsing the inputXML returns the order history ID as it was when the task was generated
>
> – Calling the TaskContext.getOrderHistoryID method returns the order history ID as it is now (current)

- Update the task status by calling a method on the TaskContext object.

```
tctx.completeTaskOnExit("activation_successful"); }
```

The following example shows an external custom automator that updates the OSM task status. This example assumes that the automation definition is an external event receiver that is receiving a message from ASAP, and that it is triggered by an automated task.

```
package com.mslv.oms.sample.atm_frame;

import com.mslv.oms.automation.plugin.*;
import com.mslv.oms.automation.*;
import java.rmi.*;

public class AsapResponseHandler extends AbstractAutomator {
  public void run(String inputXML, AutomationContext task)
              throws AutomationException {
    try {
      TaskContext tctx = (TaskContext)task;
      String taskName = tctx.getTaskMnemonic();
      AtmFrameCatalogLogger.logTaskEventResponse
        (taskName,tctx.getOrderId(),tctx.getOrderHistoryId(),inputXML);
      tctx.completeTaskOnExit("activation_successful"); }
    catch(RemoteException ex) {
      throw new AutomationException(ex); }
    catch(AutomationException x) {
      throw x; }
  }
}
```

External Custom Java Automator that Updates Order Data

If an automated task sends data to an external system and the external system sends a response back, you may need to update OSM with the data received from the external system.

The following example shows how to update data in OSM. The code is an example of updating OSM with data received from Oracle Communications Unified Inventory Management (UIM) when calling the server extension `FRDemo.AssignFacilities`.

```
package com.mslv.oms.sample.atm_frame;

import com.mslv.oms.automation.plugin.*;
import com.mslv.oms.automation.*;
import java.rmi.*;
import java.util.*;
import java.io.*;
import java.net.*;
import org.xml.sax.*;
import org.w3c.dom.*;
import javax.xml.parsers.*;

public class UIMResponseHandler extends AbstractAutomator {

  public void run( String inputXML, AutomationContext task)
              throws AutomationException {
    try {
      TaskContext tctx = (TaskContext)task;
      String taskName = tctx.getTaskMnemonic();
      AtmFrameCatalogLogger.logTaskEventResponse
        (taskName,tctx.getOrderId(),tctx.getOrderHistoryId(),inputXML);

      // Using the data returned from UIM, update the OSM order data
      String updateXml = generateOMSUpdateString(inputXML);
      tctx.updateOrderData(updateXml);

      // Complete the OSM task with the correct status
      tctx.completeTaskOnExit( "success" ); }

    catch(OrderUpdateException ex) {
      throw new AutomationException( ex ); }
    catch(RemoteException ex) {
      throw new AutomationException( ex ); }
   catch(AutomationException x ) {
     throw x; }
  }

  static private String generateOMSUpdateString(String inputXML) {
    StringBuffer osmUpdate = new StringBuffer("");
    try {
      osmUpdate = new StringBuffer
       ("<OrderDataUpdate xmlns=\"http://www.w3.org/2001/XMLSchema\""+
        " xmlns:xs=\"http://www.w3.org/2001/XMLSchema\"" +
        " xmlns:odu=\"http://www.oracle.com/OMS/OrderDataUpdate\"" +
        " targetNameSpace=\"http://www.oracle.com/OMS/OrderDataUpdate\">");

      // Use updates from UIM to update OSM
      osmUpdate.append("<AddMandatory>true</AddMandatory>");
      DocumentBuilderFactory docBuilderFactory =
        DocumentBuilderFactory.newInstance();
      DocumentBuilder parser = docBuilderFactory.newDocumentBuilder();
      Document doc = parser.parse(new StringBufferInputStream(inputXML));
      Element root = doc.getDocumentElement();
      root.normalize();
      NodeList a_site_list = root.getElementsByTagName("a_site information");
      NodeList a_site_data = a_site_list.item(0).getChildNodes();
```

```
        for(int i=0;i<a_site_data.getLength();i++) {
          Element e = (Element)a_site_data.item(i);
          osmUpdate.append("<Add path=\"/a_site_information/");
          osmUpdate.append(e.getTagName());
          osmUpdate.append("\">");
          osmUpdate.append(e.getFirstChild().getNodeValue());
          osmUpdate.append("</Add>");
        }

        NodeList z_site_list = root.getElementsByTagName("z_site_information");
        NodeList z_site_data = z_site_list.item(0).getChildNodes();

        for(int i=0;i<a_site_data.getLength();i++) {
          Element e = (Element)a_site_data.item(i);
          osmUpdate.append("<Add path=\"/z_site_information/");
          osmUpdate.append(e.getTagName());
          osmUpdate.append("\">");
          osmUpdate.append(e.getFirstChild().getNodeValue());
          osmUpdate.append("</Add>");
        }

        osmUpdate.append("</OrderDataUpdate>");

        System.out.println(omsUpdate.toString()); }

    catch(Exception e) {
      System.out.println(e.getMessage()); }

    return omsUpdate.toString();
    }
  }
```

The following code snippets from this example show:

- How to display where OSM data is updated, using XML input to describe which data nodes to update.

  ```
  tctx.updateOrderData(updateXml);
  ```

- How to build the OrderDataUpdate XML string to update the data in OSM using data garnered by parsing the UIM XML. See "Using OrderDataUpdate Elements to Pass Order Modification Data" for more information. This differs for every order template and every external system. This code represents the translation step where you convert the data from the format of an external system to the format that OSM expects.

  ```
  static private String generateOMSUpdateString(String inputXML) {
    StringBuffer osmUpdate = new StringBuffer("");
    try {
      osmUpdate = new StringBuffer
       ("<OrderDataUpdate xmlns=\"http://www.w3.org/2001/XMLSchema\""+
        " xmlns:xs=\"http://www.w3.org/2001/XMLSchema\"" +
        " xmlns:odu=\"http://www.oracle.com/OMS/OrderDataUpdate\"" +
        " targetNameSpace=\"http://www.oracle.com/OMS/OrderDataUpdate\">");

      // Use updates from UIM to update OSM
      osmUpdate.append("<AddMandatory>true</AddMandatory>");
      DocumentBuilderFactory docBuilderFactory =
        DocumentBuilderFactory.newInstance();
      DocumentBuilder parser = docBuilderFactory.newDocumentBuilder();
      Document doc = parser.parse(new StringBufferInputStream(inputXML));
      Element root = doc.getDocumentElement();
  ```

```
        root.normalize();
        NodeList a_site_list = root.getElementsByTagName("a_site
information");
        NodeList a_site_data = a_site_list.item(0).getChildNodes();

        for(int i=0;i<a_site_data.getLength();i++) {
          Element e = (Element)a_site_data.item(i);
          osmUpdate.append("<Add path=\"/a_site_information/");
          osmUpdate.append(e.getTagName());
          osmUpdate.append("\">");
          osmUpdate.append(e.getFirstChild().getNodeValue());
          osmUpdate.append("</Add>");
        }

        NodeList z_site_list =
root.getElementsByTagName("z_site_information");
        NodeList z_site_data = z_site_list.item(0).getChildNodes();

        for(int i=0;i<a_site_data.getLength();i++) {
          Element e = (Element)a_site_data.item(i);
          osmUpdate.append("<Add path=\"/z_site_information/");
          osmUpdate.append(e.getTagName());
          osmUpdate.append("\">");
          osmUpdate.append(e.getFirstChild().getNodeValue());
          osmUpdate.append("</Add>");
        }

        osmUpdate.append("</OrderDataUpdate>");

        System.out.println(omsUpdate.toString()); }

    catch(Exception e) {
      System.out.println(e.getMessage()); }

    return omsUpdate.toString();
    }
```

The structure of the XML document to update OSM data is as follows:

```
<OrderDataUpdate xmlns=\"http://www.w3.org/2001/XMLSchema\"
xmlns:xs=\"http://www.w3.org/2001/XMLSchema\"
xmlns:odu=\"http://www.oracle.com/OMS/OrderDataUpdate\"
targetNameSpace=\"http://www.oracle.com/OMS/OrderDataUpdate\">
<AddMandatory>true</AddMandatory>
<Add path=\"/service_details/new_number\">98765</Add>
<Update path=\"/customer_details/service_address/street\">55 Updated St</
Update>
<Delete path=\"/service_details/current_account_number\"></Delete>
</OrderDataUpdate>
```

This example illustrates adding a data node (Add path), updating a data node (Update path), and deleting a data node (Delete path).

- How to specify a mandatory parameter. If set to true, the following rules apply:

  ```
  osmUpdate.append("<AddMandatory>true</AddMandatory>");
  ```

  – If you delete a mandatory node, AddMandatory replaces the node and populates it with the default value.

  – If the update is missing a mandatory node, AddMandatory adds the missing node and populates it with the default value.

> **Note:**
>
> If you add a mandatory field, but do not include a value, AddMandatory will not add a default value and the request will generate an error-error code 200.

Using OrderDataUpdate Elements to Pass Order Modification Data

You use OrderDataUpdate XML elements to pass data add, modify and delete data nodes in an order.

OrderDataUpdate elements can be passed as a parameter to updateOrderData(). XSL translations whose results are passed to setUpdateOrder() must be in OrderDataUpdate format. See the OSM Javadocs for details on both methods. You can also pass OrderDataUpdate format elements to the DataChange Web Service (see the SDK schema OrderManagementWS.xsd) and UpdateOrder.request XML API call (see the SDK schema oms-xmlapi.xsd).

For update and delete operations on multi-instance nodes, you must specify the order node index as it exists in the input XML. Specify the order node index as "[@index='index_value']" where index_value is the order node index.

The following example shows how to specify the addition of an order node with OrderDataUpdate. The path attribute identifies the parent node under which to add the element:

```
<OrderDataUpdate>
  <Add path="/">
    <ProvisioningOrderResponse>
      <OrderInformation>
        <OrderNumber>1238723</OrderNumber>
      </OrderInformation>
    </ProvisioningOrderResponse>
  </Add>
</OrderDataUpdate>
```

The following example shows a combined update and delete operation on a multi-instance node using OrderDataUpdate. In Delete attributes, the path attribute identifies the data to delete. In Update attributes, the path attribute identifies the data to update. Indexes are required on Update and Delete attributes when modifying multi-instance nodes. Note how the order node index values are specified in the Update and Delete attributes.

```
<OrderDataUpdate>
  <Delete path="/client_info/address[@index='80132']/city" />
  <Update path="/client_info/address[@index='76579']/city">Newark</Update>
  <Update path="/customer_details/service_address/street">55 Updated St</Update>"
  <Delete path="/service_details/current_account_number"></Delete>
</OrderDataUpdate>
```

See "External Custom Java Automator that Updates Order Data" for an example in which OrderDataUpdate XML data is created dynamically within Java code and passed to UpdateOrderData().

The schema for OrderDataUpdate is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://
www.metasolv.com/OMS/OrderDataUpdate" xmlns:odu="http://www.metasolv.com/
OMS/OrderDataUpdate" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">

  <element name="OrderDataUpdate">
  <complexType>
    <choice maxOccurs="unbounded">
      <element ref="odu:Add"/>
      <element ref="odu:Delete"/>
      <element ref="odu:Update"/>
    </choice>
  </complexType>
  </element>

  <element name="Add">
    <annotation>
      <documentation>It contains a node to be added. The path attribute
identifies the parent node under which to add the element.</documentation>
    </annotation>
    <complexType>
      <sequence>
        <any/>
      </sequence>
      <attribute name="path" type="string" use="required"/>
    </complexType>
  </element>

  <element name="Delete">
    <annotation>
      <documentation>It contains a node to be deleted. The path attribute
identifies the node to delete.</documentation>
    </annotation>
    <complexType>
      <attribute name="path" type="string" use="required"/>
    </complexType>
  </element>

  <element name="Update">
    <annotation>
      <documentation>It contains a node to update. The path attribute identifies
the node to update.</documentation>
    </annotation>
    <complexType>
      <simpleContent>
        <extension base="string">
          <attribute name="path" type="string" use="required"/>
        </extension>
      </simpleContent>
    </complexType>
  </element>
</schema>
```

Examples of Sending Messages to External Systems

Automation simplifies the process of sending messages to external systems. The automation framework does the following:

- Assumes the protocol is JMS. The products (Siebel, OSM, UIM, ASAP, IP Service Activator) all have JMS APIs.
- Takes care of establishing and maintaining the various JMS connections.
- Constructs the JMS messages, setting the required message properties.
- Guarantees delivery of the message and handles any errors or exceptions. It retries until the message is delivered.
- Automatic message correlation.
- Poison message handling.

An OSM event that is sent to an external system follows this process flow:

1. OSM runs an automation that triggers an automation plug-in.
2. Internally, the automation framework maps the plug-in, using the **automationMap.xml** configuration, onto custom business logic and calls the makeRequest method on the custom automator class.
3. The makeRequest method performs some business logic and sets the content of the outbound message.
4. The automation framework adds properties to the outbound message to aid in correlating external system responses to requests.
5. The automation framework uses information from the **automationMap.xml** to send the JMS message to the JMS queue representing the external system.

The following example shows a custom automation plug-in that sends data to an external system.

```
 package com.mslv.oms.sample.atm_frame;

 import com.mslv.oms.automation.plugin.*;
 import com.mslv.oms.automation.*;
 import javax.jms.TextMessage;
 import java.rmi.*;

 public class ObjectelPlugin extends AbstractSendAutomator {

 protected void makeRequest(String inputXML, AutomationContext context,
TextMessage outboundMessage) throws com.mslv.oms.automation.AutomationException
{

 try {
 TaskContext taskContext = (TaskContext)context;
 String taskName = taskContext.getTaskMnemonic();
 AtmFrameCatalogLogger.logTaskEvent(taskName, taskContext.getOrderId(),
taskContext.getOrderHistoryId(), inputXML);

 //
 // Set the outgoing message
 //
 String xmlRequest =
"<Message type=\"ni\"><iLibPlus:findFunctionalPortOnLocation.Request
xmlns:iLibPlus=\"http://www.oracle.com/
objectel\"><location><DS><AG2ObjectID>189438</
AG2ObjectID><AG2ParentID>189428</AG2ParentID><CLLIX>XML.CO.1</
CLLIX><SiteName>XML.CO.1</SiteName></DS></location><feType>PP</
feType><portType>$FEP</portType><selectionMethod>LOAD_BALANCE</
selectionMethod><portSelectionAttribName><string>AG2ObjectID</
```

```
string><string>AG2ParentID</string><string>AG2PortLabel</string></
portSelectionAttribName><portSelectionAttribValue><string>189508</
string><string>189478</string><string>F-31-OC-48</string></
portSelectionAttribValue><portUpdateAttribName/><portUpdateAttribValue/></
iLibPlus:findFunctionalPortOnLocation.Request></Message>";
 outboundMessage.setText( xmlRequest );

 } catch( javax.jms.JMSException x ) {
 throw new AutomationException( x );
 } catch(RemoteException ex){
 throw new AutomationException( ex );
 }
 }
 }
```

The following code snippets from this example show:

- how to generate an output XML string. In this example it is hard coded. In a business case you would use business logic to transform OSM data into what the external system expects

  ```
   String xmlRequest =
  "<Message type=\"ni\"><iLibPlus:findFunctionalPortOnLocation.Request
  xmlns:iLibPlus=\"http://www.oracle.com/
  objectel\"><location><DS><AG2ObjectID>189438</
  AG2ObjectID><AG2ParentID>189428</AG2ParentID><CLLIX>XML.CO.1</
  CLLIX><SiteName>XML.CO.1</SiteName></DS></location><feType>PP</
  feType><portType>$FEP</portType><selectionMethod>LOAD_BALANCE</
  selectionMethod><portSelectionAttribName><string>AG2ObjectID</
  string><string>AG2ParentID</string><string>AG2PortLabel</string></
  portSelectionAttribName><portSelectionAttribValue><string>189508</
  string><string>189478</string><string>F-31-OC-48</string></
  portSelectionAttribValue><portUpdateAttribName/><portUpdateAttribValue/></
  iLibPlus:findFunctionalPortOnLocation.Request></Message>";
  ```

- how to set the output data:

  ```
   outboundMessage.setText( xmlRequest );
  ```

- How this code does not establish a connection to an external system or send a message. After the data is set in the code, the message is automatically sent upon exit of the makeRequest method.

Examples of Handling Responses from External Systems

In Message Property Correlation, the following steps describe how responses from external systems are handled.

1. The plug-in populates the message content.
2. The plug-in sets a property on the outbound JMS message, with name of the value set for correlationproperty in the **automationMap.xml** file, and a value decided by the business logic. For example, you could use this to correlate on a reference number.
3. If the value of the correlationproperty in the **automationMap.xml** file is set to the value JMSCorrelationID, the plug-in is not required to set the property on the outbound message (as described in Step 2). The automation framework does this automatically.

4. The automation framework saves the message properties set for each message with the event information.
5. The automation framework sets the replyTo property on the JMS message.
6. The external system copies the properties on the request message to the response message.
7. The external system sends the message to the reply queue specified in the **automationMap.xml** file.
8. The automation framework uses the configuration in the **automationMap.xml** file to map messages from external systems to plug-ins. The plug-ins are automators written by system integrators. Configuration of an automator for receiving messages from an external system are defined within Design Studio and saved to the **automationMap.xml** file.
9. The automation framework uses the message properties of the response, plus the correlation information saved in step four above, to reload a Context for the response message.
10. The run method of the external system automator is called and is passed the Context created in step 9.
11. The automator performs business logic, such as completing the task.

The following example shows a custom automation plug-in that handles and processes response messages from an external system.

```
package com.mslv.oms.sample.atm_frame;

import com.mslv.oms.automation.plugin.*;
import com.mslv.oms.automation.*;
import java.rmi.*;

public class UIMResponseHandler extends AbstractAutomator {

public void run( String inputXML, AutomationContext task)
throws AutomationException {

try {
TaskContext tctx = (TaskContext)task;

tctx.completeTaskOnExit( "success" );

} catch(RemoteException ex){
throw new AutomationException( ex );
} catch(AutomationException x ) {
throw x;
}
}
}
```

This automation plug-in does not need to send JMS messages to any system, so it extends AbstractAutomator and is intended to process Task automation responses, so it casts the Context to a TaskContext then completes the task.

The following example shows what the external system is expected to do for the message property correlation to work.

```
public void sendMessage(Message originalMessage) {
try {
//
```

```
 // Set up the JMS connections
 //
 QueueConnectionFactory connectionFactory =
(QueueConnectionFactory)jndiCtx.lookup(connectionFactoryName);
 QueueConnection queueConnection = connectionFactory.createQueueConnection();
 QueueSession queueSession = queueConnection.createQueueSession(false,
Session.AUTO_ACKNOWLEDGE);
 Queue replyQueue = (Queue)originalMessage.getJMSReplyTo();
 QueueSender queueSender = queueSession.createSender(replyQueue);

 //
 // Create the message
 //
 TextMessage textMessage =
queueSession.createTextMessage(((TextMessage)originalMessage).getText());
 textMessage.setStringProperty("MESSAGE_NAME","ActivationResponse");
 textMessage.setJMSCorrelationID(originalMessage.getJMSCorrelationID());

 //
 // Send the message
 //
 queueSender.send(textMessage, javax.jms.DeliveryMode.PERSISTENT,
javax.jms.Message.DEFAULT_PRIORITY, 1800000);

 } catch(javax.jms.JMSException ex){
 ex.printStackTrace();
 } catch(javax.naming.NamingException ex){
 ex.printStackTrace();
 }
 }
```

The following code snippets from this example show:

- how the external system chooses which JMS destination to send the reply to.

  ```
  Queue replyQueue = (Queue)originalMessage.getJMSReplyTo();
  QueueSender queueSender = queueSession.createSender(replyQueue);
  ```

- the external system setting a property that identifies the nature of the JMS message. This implies that the automation was defined with a message property selector select statement that matches these parameters.

  ```
  textMessage.setStringProperty("MESSAGE_NAME","ActivationResponse");
  ```

- the external system echoing the correlation information onto the reply message. This implies that the automation was defined to correlate based on JMSCorrelationID.

  ```
  textMessage.setJMSCorrelationID(originalMessage.getJMSCorrelationID());
  ```

Compensation XQuery Expressions

The following topics provide information about automation and manual task compensation XQuery expressions.

- Task Re-Evaluation and Rollback XQuery Expressions
- In Progress Compensation Include XQuery Expressions
- In Progress Compensation Complete XQuery Expressions
- In Progress Compensation Grace Period XQuery Expressions

For general OSM XQuery information, see *OSM Modeling Guide*.

Task Re-Evaluation and Rollback XQuery Expressions

You can dynamically assign compensation strategies to tasks by creating XQuery expressions in the Design Studio **Task Editor Compensation** tab for re-evaluation compensation strategies or compensation strategies for when a task is no longer required.

> **Note:**
>
> If the XQuery expression is invalid OSM logs the error but does not rollback the transaction. Instead, OSM uses the static compensation strategy as the default.

This section refers to the Design Studio OSM **Automated Task** or **Manual Task** editor, **Compensation** tab Compensation Expression XQuery field for re-evaluation compensation strategies:

- Context: The context for this XQuery is the current order data. You can get the current order data using the XML API GetOrder.Response function.
- Prolog: You can declare the XML API namespace to use the GetOrder.Response function in the XQuery body to extract the order information. You must declare the java:oracle:communications.ordermanagement.compensation.ReevaluationContext OSM Java package that provides methods that access the contemporary and historical order perspectives and compares the two. You can use the results of this comparison to determine what compensation strategy is required for a task based on revision order data.

  For example:

  ```
  declare namespace osm = "urn:com:metasolv:oms:xmlapi:1";
  declare namespace context =
  "java:oracle.communications.ordermanagement.compensation.ReevaluationContext"
  ;
  declare namespace log =  "java:org.apache.commons.logging.Log";

  declare variable $log external;
  declare variable $context external;
  ```

  For more information about the classes in the OSM packages, install the OSM SDK and extract the OSM Javadocs from the **SDK/osm7.***w.x.y.z***-javadocs.zip** file (where *w.x.y.z* represents the specific version numbers for OSM). See *OSM Installation Guide* for more information about installing the OSM SDK.
- Body: The body must return a valid compensation option.

  For example, the following XQuery expression creates variables for the ReevaluationContext methods. The expression then checks that a specific value exists in the $value variable and that the value in the $significantValue variable both exists and is significant. If the value exists and is significant, then the expression sets the compensation strategy for the task to **Undo then Do** (undoThenDo in the ReevaluationContext Java class). If not, then the expression sets the compensation strategy to **Redo** (redo in the ReevaluationContext Java class).

```
    let $inputDoc := self::node()
    let $hopDoc := context:getHistoricalOrderDataAsDOM($context)
    let $ropDoc := context:getCurrentOrderDataAsDOM($context)
    let $diffDoc := context:getDataChangesAsDOM($context)
    let $value := $inputDoc/GetOrder.Response/_root/service[name='BB']//
    orderItemRef/specificationGroup//specification[value='100']
    let $significantValue := $diffDoc/Changes/Add[@significant='true']/
    specification[value='100']
    let $currentValue := $ropDoc/ GetOrder.Response/_root/service[name='BB']//
    orderItemRef/specificationGroup//specification[value='100']

    return if (fn:exists($value) and fn:exists($significantValue))
    then
        context:undoThenDo($context)
    else
        context:redo($context)
```

This section refers to the Design Studio OSM **Automated Task** or **Manual Task** editor, **Compensation** tab **Compensation Expression** XQuery field for when a task is no longer required. The context, prolog, and body are similar to the XQuery expression for the re-evaluation strategy, except that the XQuery expression implements the java:oracle:communications.ordermanagement.compensation.RollbackContext package.

For example:

```
declare namespace osm = "urn:com:metasolv:oms:xmlapi:1";
declare namespace context =
"java:oracle.communications.ordermanagement.compensation.RollbackContext";
declare namespace log =  "java:org.apache.commons.logging.Log";

declare variable $log external;
declare variable $context external;

let $inputDoc := self::node()
let $hopDoc := context:getHistoricalOrderDataAsDOM($context)
let $ropDoc := context:getCurrentOrderDataAsDOM($context)
let $diffDoc := context:getDataChangesAsDOM($context)

let $value := $inputDoc/GetOrder.Response/_root/service[name='BB']//orderItemRef/
specificationGroup//specification[value='100']
return if (fn:exists($value))
then
    context:undo($context)
else
    context:doNothing($context)
```

In Progress Compensation Include XQuery Expressions

You can determine if an in progress task should be compensated by writing an XQuery expression in the Design Studio **Task Editor Compensation** tab.

> **Note:**
>
> If the XQuery expression is invalid OSM logs the error and includes the in progress task in the compensation plan as it defaults the expression to true.

This section refers to the Design Studio OSM **Automated Task** or **Manual Task** editor, **Compensation** tab, **In Progress Compensation Include Expression** XQuery field for dynamically defining when in progress tasks should be included in compensation. This XQuery expression runs when OSM first analyzes the task for compensation:

- Context: The context for this XQuery is the current task order data. You can get the current task order data using the XML API GetOrder.Response function.
- Prolog: You can declare the XML API namespace to use the GetOrder.Response function in the XQuery body to extract the order information.

  For example:

```
declare namespace osm = "urn:com:metasolv:oms:xmlapi:1";
declare namespace log =  "java:org.apache.commons.logging.Log";

declare variable $log external;
declare variable $context external;
```

- Body: Based on task context data, the body must return **true** if the in progress task requires compensation or **false** if it does not.

For example:

```
declare namespace osm = "urn:com:metasolv:oms:xmlapi:1";
declare namespace log =  "java:org.apache.commons.logging.Log";
declare variable $log external;

let $inputDoc := self::node()
let $value := $inputDoc/GetOrder.Response/_root/data

return (
    if (fn:contains($value, "includeInCompensation")) then
        fn:true()
    else
        fn:false()
)
```

In Progress Compensation Complete XQuery Expressions

You can determine when the compensation for an in progress task is complete by writing an XQuery expression in the Design Studio **Task Editor Compensation** tab.

> **Note:**
>
> If the XQuery expression is invalid OSM logs the error and includes the in progress task in the compensation plan as it defaults the expression to true.

This section refers to the Design Studio OSM **Automated Task** or **Manual Task** editor, **Compensation** tab, **In Progress Compensation Complete Expression** XQuery field for dynamically defining when in progress tasks completes compensation activities. This XQuery expression runs whenever data changes on the compensating task:

- Context: The context for this XQuery is the current task order data. You can get the current task order data using the XML API GetOrder.Response function.
- Prolog: You can declare the XML API namespace to use the GetOrder.Response function in the XQuery body to extract the order information.

For example:

```
declare namespace osm = "urn:com:metasolv:oms:xmlapi:1";
declare namespace log =  "java:org.apache.commons.logging.Log";

declare variable $log external;
declare variable $context external;
```

- Body: Based on task context data, the body must return **true** if the in progress task has completed all compensation activities or **false** if it has not.

For example:

```
declare namespace osm = "urn:com:metasolv:oms:xmlapi:1";
declare namespace log =  "java:org.apache.commons.logging.Log";
declare variable $log external;
let $inputDoc := self::node()
let $value := $inputDoc/GetOrder.Response/_root/data
return (
    if (fn:contains($value, "compensationDone")) then
                fn:true()
    else
                fn:false()
```

In Progress Compensation Grace Period XQuery Expressions

You can determine whether a grace period should be observed before starting compensation for an in progress task by writing an XQuery expression in the Design Studio **Task Editor Compensation** tab.

> **Note:**
>
> If the XQuery expression is invalid OSM logs the error and includes the in progress task in the compensation plan as it defaults the expression to true.

This section refers to the Design Studio OSM **Automated Task** or **Manual Task** editor, **Compensation** tab, **When an amendment occurs if this task is in progress it will:** tab, **Dynamic Expression** XQuery field for dynamically defining the grace period for an in progress task based on task data. This XQuery expression runs after OSM has determined whether the in progress task needs to be compensated:

- Context: The context for this XQuery is the current task order data. You can get the current task order data using the XML API GetOrder.Response function.
- Prolog: You can declare the XML API namespace to use the GetOrder.Response function in the XQuery body to extract the order information. You can also declare the $gracePeriod variable in the XQuery prolog which contains the grace period specified on the order life-cycle policy.

  For example:

```
declare namespace osm = "urn:com:metasolv:oms:xmlapi:1";
declare namespace log =  "java:org.apache.commons.logging.Log";

declare variable $gracePeriod external;
declare variable $log external;
declare variable $context external;
```

- Body: The XQuery body returns a duration value based on the XQuery you enter:

  PyYmMdDThHmMsS

  where

  - **P** begins the expression.
  - *y***Y** specifies the year.
  - *m***M** specifies the month.
  - *d***D** specifies the day.
  - **T** separates the parts of the expression indicating the date from the parts of the expression indicating the time.
  - *h***H** specifies the hour.
  - *m***M** specifies the minutes.
  - *s***S** specifies the seconds.

For example, this XQuery uses order data to define the specific grace period duration for the task. The last statement calls the $gracePeriod variable which represents the grace period duration specified on the order life-cycle policy:

```
declare namespace osm = "urn:com:metasolv:oms:xmlapi:1";
declare namespace log =  "java:org.apache.commons.logging.Log";
declare variable $log external;
declare variable $gracePeriod external;

let $inputDoc := self::node()
let $value := $inputDoc/GetOrder.Response/_root/data

return (
    if (fn:contains($value, '-immediate-')) then
        xs:duration('PT0S')
    else if (fn:contains($value, '-override-')) then
        xs:duration('PT20S')
    else if (fn:contains($value, '-negative-')) then
        xs:duration('-PT10S')
    else if (fn:contains($value, '-invalidNumber-')) then
        fn:number(0)
    else if (fn:contains($value, '-invalidString-')) then
        xs:string('UNKNOWN')
    else
        xs:duration(fn:concat('PT', $gracePeriod, 'S'))
```

Order Jeopardy Automation XQuery Plug-ins

This topic provides information about order jeopardy XQuery expressions. These XQuery expressions apply to order jeopardies configured in the Order Jeopardy editor, not order jeopardies configured in the Order editor.

For general OSM XQuery information, see *OSM Modeling Guide*.

You can configure automations for order jeopardies in the Order Jeopardy editor, **Automation** tab. If you choose to use an XQuery automation type, create an XQuery file and reference it in the **Script** subtab **Script** field.

- Context: The context for this XQuery is the Order Jeopardy Notification context.

- Prolog: You should declare the XML namespace for the Order Jeopardy Notification context, and if you are using a date (rather than a duration) you can declare a namespace for the date format as well.

  For example:

  ```
  declare namespace context =
  "java:oracle.communications.ordermanagement.orderjeopardy.automation.OrderJeo
  pardyNotificationContext";
  declare namespace dateFormat = "java:java.text.DateFormat";
  ```

  You should then declare the $context variable to contain the actual context:

  ```
  declare variable $context external;
  ```

  Then if you want to use order data in your XQuery, you can get the order data into a variable. For example:

  ```
  let $orderData := fn:root(automator:getOrderAsDOM($automator))/
  oms:GetOrder.Response
  ```

  You can then access individual data elements on the order. For example:

  ```
  let $date := $orderData/oms:_root/oms:ojPostponeDate/text()
  ```

- Body: There are several calls you can use in the order jeopardy XQuery file in addition to the normal calls available for notification plug-ins. Following are brief descriptions of the available calls:

  – postponeTimerOnExit(interval): If this call receives a numeric parameter, it postpones the due date for the number of milliseconds contained in the parameter.

  – postponeTimerOnExit(dateTime): If this call receives a date/time parameter, it postpones the due date to the indicated date/time.

  – logAndParkNotificationOnExit(logMessage): This call acknowledges the notification with the passed-in message, but does not reset/deactivate the notification. It will still be available in the Order Management web client.

  – ackNotificationOnExit: This call acknowledges and resets/deactivates the notification.

  – getNotificationAckStatus: This call returns true if the notification has been acknowledged, and false if it has not.

  The following example postpones the jeopardy to a specified date:

  ```
  declare namespace context =
  "java:oracle.communications.ordermanagement.orderjeopardy.automation.OrderJeo
  pardyNotificationContext";
  declare namespace dateFormat = "java:java.text.DateFormat";

  declare variable $context external;

  let $dateFormat := dateFormat:getDateTimeInstance(3, 3)
  let $date := dateFormat:parse($dateFormat, "09/30/15 03:30 PM")
  return
      context:postponeTimerOnExit($context, $date)
  ```

B

AutomationMap.xml File

This appendix provides examples of the generated **automationMap.xml** file for Oracle Communications Order and Service Management (OSM).

> **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 **automationMap.xml** file. The file is placed in the *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 **automationMap.xml** file.

AutomationMap.xml Examples for Automated Tasks

This section provides various examples of generated **automationMap.xml** files. The examples include predefined and custom automations defined for automated tasks. In the XML, an automated task is defined by the `<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

Example B-1 XSLTSender Internal Event Receiver

```
<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>
  <receive xsi:type="am:InternalReceiver">
    <mnemonic>MyTask</mnemonic>
    <executionModes>do</executionModes>
  </receive>
```

```
  <implement xsi:type="am:XsltSender">
    <to>
      <jndiName>MyTask.MyXsltSenderIntAutomation.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 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 B-2 XSLTSender External Event Receiver

```
<taskAutomator>
  <pluginJndiName>MyTask.MyTask.MyXsltSenderExtAutomation</pluginJndiName>
  <ejbName>MyTask.MyTask.MyXsltSenderExtAutomation</ejbName>
  <className>com.mslv.oms.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 B-3 XSLTAutomator Internal Event Receiver

```
<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. (XSLTReceiver 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 B-4 XSLTAutomator External Event Receiver

```
<taskAutomator>
  <pluginJndiName>MyTask.MyTask.MyXsltAutomatorExtAutomation</pluginJndiName>
      <ejbName>MyTask.MyTask.MyXsltAutomatorExtAutomation</ejbName>
  <className>com.mslv.oms.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 B-5 Custom Automation Internal Event Receiver

```
<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 B-6 Custom Automation External Event Receiver

```
<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: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 `<notificaitonAutomator>` 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.

- The value of the <event> type is based on the type of notification. As a result, the value differs based on the type of notification, which in turn dictates the <event> subelements that are generated.

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
- Java class name: GenericNotif

Example B-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 B-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 B-9 State-Based Notification / Process Event Tab

```
<notificationAutomator>
      <ejbName>
    OsmProcess_MyProcessTaskStateRule.OsmProcess.
    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 an 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 B-10 Task Status-Based Notification

```
<notificationAutomator>
      <ejbName>
    OsmProcess_MyProcessTaskStatusRule.OsmProcess.
    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 B-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 B-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 B-13 Task Jeopardy Notification

```
<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.

C

Automation: Start to Finish

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.

> **Note:**
>
> This appendix assumes that you have read "Using Automation."

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 *cartridgeName*Order. 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 *cartridgeName*Order.
- Order Model Error: Default process is not defined for order *cartridgeName*Order.

- 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 *cartridgeName*Order has empty Order Template.
- Order Model Error: There is no Permission defined for order *cartridgeName*Order.

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.
14. On the Manual Task editor **Permissions** tab, set permissions for the task by assigning a role.
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 "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.