Contents

Preface ........................................................................................................................................... xi
Audience ................................................................................................................................... xi
Documentation Accessibility ................................................................................................. xi

1 XML API Introduction

XML API Sample Client Overview ......................................................................................... 1-1
HTTP Communication Overview .......................................................................................... 1-2
  HTTP Support ...................................................................................................................... 1-2
  HTTP Session Management ............................................................................................... 1-2
  HTTP Layer Client ............................................................................................................ 1-2
    Login ................................................................................................................................. 1-2
    Sending a Request ......................................................................................................... 1-3
    Logout ............................................................................................................................... 1-4
  Connection Layer Client .................................................................................................... 1-4
    Establishing a Connection and Sending XML as Strings .............................................. 1-5
  XML Layer Client ............................................................................................................. 1-5
    Sending a Request ......................................................................................................... 1-5
    Using the XMLAPIFacade ............................................................................................. 1-6
  Model Layer Client .......................................................................................................... 1-7
    Creating an Order .......................................................................................................... 1-7
    Retrieving an Order at a Task ....................................................................................... 1-7
Cartridge Management ........................................................................................................ 1-8

2 XML API Functions

Backwards Compatibility for pre-OSM 2.5 Namespaces ...................................................... 2-2
Audience ................................................................................................................................. 2-2
Login/Logout .......................................................................................................................... 2-2
Message Formats ..................................................................................................................... 2-2
  Input XML Message Format .............................................................................................. 2-2
  Output XML Message Format ........................................................................................... 2-3
Authentication ......................................................................................................................... 2-3
Reserved Mnemonics ............................................................................................................. 2-4
XML API Functionality .......................................................................................................... 2-4
  AddOrderThread ............................................................................................................... 2-5
  Operation ............................................................................................................................ 2-5
ReceiveOrder ................................................................. 2-49
  Operation ................................................................. 2-49
  Parameters ............................................................... 2-50
  Request Example ..................................................... 2-50
  Response Example .................................................... 2-50
  Error Codes ............................................................. 2-50
ResumeOrder ................................................................. 2-50
  Operation ................................................................. 2-50
  Parameters ............................................................... 2-50
  Request Example ..................................................... 2-50
  Response Example .................................................... 2-51
  Error Codes ............................................................. 2-51
SuspendOrder ............................................................... 2-52
  Operation ................................................................. 2-52
  Parameters ............................................................... 2-52
  Request Example 1: Order ........................................ 2-52
  Request Example 2: Task ......................................... 2-53
  Response Example 1: Order ..................................... 2-53
  Response Example 2: Task ..................................... 2-53
  Error Codes ............................................................. 2-53
TaskDescription .......................................................... 2-53
  Operation ................................................................. 2-53
  Parameters ............................................................... 2-54
  Request Example 1 .................................................. 2-54
  Request Example 2 .................................................. 2-54
  Response Example .................................................. 2-54
  Error Codes ............................................................. 2-54
UpdateOrder ............................................................... 2-54
  Operation ................................................................. 2-54
  Parameters ............................................................... 2-54
  Request Example ..................................................... 2-55
Warning and Error Code Descriptions .............................................................................................. 2-60
Document Type Definitions (DTD) .................................................................................................. 2-62
AddOrderThread ................................................................................................................................. 2-62
  Request Example ............................................................................................................................... 2-62
  Response Example ............................................................................................................................. 2-63
AssignOrder ........................................................................................................................................... 2-63
  Request Example ............................................................................................................................... 2-63
  Response Example ............................................................................................................................. 2-63
CompleteOrder ........................................................................................................................................ 2-63
  Request Example ............................................................................................................................... 2-63
  Response Example ............................................................................................................................. 2-63
CopyOrder ............................................................................................................................................. 2-63
  Request Example ............................................................................................................................... 2-63
  Response Example ............................................................................................................................. 2-64
CreateOrder .......................................................................................................................................... 2-64
  Request Example ............................................................................................................................... 2-64
  Response Example ............................................................................................................................. 2-64
Error ......................................................................................................................................................... 2-65
GetOrder ................................................................................................................................................. 2-65
  Request Example ............................................................................................................................... 2-65
  Response Example ............................................................................................................................. 2-65
GetNextOrderAtTask ................................................................................................................................. 2-66
  Request Example ............................................................................................................................... 2-66
  Response Example ............................................................................................................................. 2-66
GetOrderDataHistory ................................................................................................................................. 2-66
  Request Example ............................................................................................................................... 2-66
  Response Example ............................................................................................................................. 2-67
GetOrderProcessHistory ............................................................................................................................ 2-67
  Request Example ............................................................................................................................... 2-67
  Response Example ............................................................................................................................. 2-67
GetOrderStateHistory ................................................................................................................................. 2-68
  Request Example ............................................................................................................................... 2-68
  Response Example ............................................................................................................................. 2-68
GetUserInfo ............................................................................................................................................ 2-68
  Request Example ............................................................................................................................... 2-68
  Response Example ............................................................................................................................. 2-68
ListExceptions ......................................................................................................................................... 2-69
  Request Example ............................................................................................................................... 2-69
  Response Example ............................................................................................................................. 2-69
Preface

This document provides information about the Oracle Communications Order and Service Management (OSM) XML Application Programming Interface (API).

**Audience**

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

- System interfaces
- XML
- Java development
- Java Messaging Service (JMS)

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

- Studio configuration
- Orders
- Order states
- Tasks
- Task states
- XML APIs

**Documentation Accessibility**


**Access to Oracle Support**

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.

The following sections detail the use of the Hypertext Transfer Protocol (HTTP) protocol, a built-in sample client library and a XML API sample client. The sample classes are not a complete client, but they provide a starting point for XML API projects.

For further information, see the Javadoc documentation provided with the OSM SDK. While use of the sample client is not required, it can help your development project by enabling you to focus on the solution requirements instead of the communication details.

**XML API Sample Client Overview**

The XML API sample client consists of multiple programming interface layers built on top of the HTTP, see Figure 1–1. Client software connects to the server and communicates with the XML API using any of the layers.

**Figure 1–1  Multiple Programming Interface Layers on Top of the HTTP Protocol**

This figure illustrates the following information about the interface layers:

- HTTP protocol layer uses Java’s built-in HTTP implementation and provides no XML API-specific features

---

XML API Introduction 1-1
Connection layer adds authentication support and session management
XML layer adds support for the OSM XML APIs specific XML messages
Model layer provides OSM domain objects

HTTP Communication Overview

HTTP is a content-neutral stateless synchronous communication protocol, the standard for transferring data across TCP/IP networks. The OSM XML API uses HTTP as the transport mechanism for requests from external clients.

HTTP Support

The widespread use of HTTP has led to implementations of the protocol being available for a wide variety of programming languages, including Java, C/C++, VB, Delphi, Perl, and Python. The choice of HTTP means that clients can be written in any major language. The examples in the following sections are written using Java.

HTTP Session Management

The HTTP protocol is stateless, therefore, there is no way to maintain a conversational state between the client and server. To address this need, a mechanism known as a Cookie was introduced. A Cookie consists of an extra HTTP header with the name 'Cookie'. The OSM server uses a cookie to identify authenticated users. Any external client using an HTTP library has to ensure that the library supports Cookies.

For Cookie maintenance, see the examples in the HTTP layer client section.

HTTP Layer Client

The Java SDK API provides an implementation of the HTTP protocol, java.net.HttpURLConnection. This class can be used to communicate with any HTTP server. The following samples demonstrate how it is used with the OSM XML API server.

Login

A login request consists of a HTTP POST request to:

http://server:port/oms/XMLAPI/login

There are also two parameters named **username** and **password**. If the login is unsuccessful, the HTTP 403: Forbidden response is returned, otherwise HTTP 200: OK is returned along with a SET-COOKIE header with the session cookie. The cookie must be stored for sending with subsequent requests.

**Example 1–1 Login Request**

```java
String user='oms'; String password='oms'; String cookie=null;
String str =
URLEncoder.encode ('username')+'='+URLEncoder.encode (user)+'&'
URLEncoder.encode ('password')+'='+URLEncoder.encode (password);
byte[] bytes = str.getBytes();
URL url = new URL ('http://localhost:7001/OrderManagement/XMLAPI/login');
```
HttpURLConnection connection =
(HttpURLConnection) url.openConnection();
connection.setAllowUserInteraction(false);
connection.setDoOutput(true);
connection.setDoInput(true);
connection.setRequestMethod("POST");
connection.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
connection.setRequestProperty("Content-Length", String.valueOf(bytes.length));
connection.connect();
OutputStream out = connection.getOutputStream();
out.write(bytes);
out.close();
int code = connection.getResponseCode();
String msg = connection.getResponseMessage();
if (code == HttpURLConnection.HTTP_OK) {
    String receivedcookie =
        connection.getHeaderField("Set-Cookie");
    if (receivedcookie == null) {
        throw new Exception(
            "Server did not return session cookie");
    }
    cookie = receivedcookie.substring(0,
        receivedcookie.indexOf(';'));
} else {
    throw new Exception("HTTP response code != 200 OK :"+code);
}
connection.disconnect();

Sending a Request
Sending XML requests is completed the same way as the login request, except the URL is:

http://server:port/oms/XMLAPI

The Content Type is "text/xml", and the session Cookie must be sent as a header parameter "Cookie".

Example 1–2 Request
String cookie=...;//initialized elsewhere
URL url = new URL("http://localhost:7001/OrderManagement/XMLAPI");

String request = "<Worklist.Request/>";
byte[] bytes = request.getBytes();

HttpURLConnection connection =
(HttpURLConnection) url.openConnection();
connection.setAllowUserInteraction(false);
connection.setDoOutput(true);
connection.setDoInput(true);
HTTP Communication Overview

connection.setRequestMethod("POST");
connection.setRequestProperty("Content-Type", "text/xml");
connection.setRequestProperty("Content-Length", String.valueOf(bytes.length));
connection.setRequestProperty("Cookie", cookie);

connection.connect();
OutputStream out = connection.getOutputStream();
out.write(bytes);
out.close();
connection.disconnect();

Logout
A logout request is sent to the following address as a POST request with the cookie set:
http://server:port/oms/XMLAPI/logout
No other parameters are required.

Example 1–3 Logout Request
URL url = new URL("http://localhost:7001/OrderManagement/XMLAPI/logout");

HttpURLConnection connection =
    (HttpURLConnection) url.openConnection();
connection.setAllowUserInteraction(false);
connection.setDoOutput(true);
connection.setDoInput(true);
connection.setRequestMethod("POST");
connection.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
connection.setRequestProperty("Content-Length", "0");

connection.connect();

Connection Layer Client
To simplify using the HTTP protocol to communicate with the XML API, there are sample classes that handle login/logout and XML string transmission. It is available in the com.mslv.oms.xmlclient.connection package, and contains the following classes:

- **Connection**: Maintains a connection with the XML API.
- **Driver**: Produces connections.
- **DriverManager**: Maintains a registry of drivers.
- **HTTPDriver**: Provides Connections for HTTP communication.
- **ConnectionException**: Indicates an abnormal situation on the connection.

For complete class documentation, see the Javadoc documentation provided in the OSM SDK.
Establishing a Connection and Sending XML as Strings
A connection can be established by using the DriverManager.getConnection() method. This method authenticates against the server and returns a connection for use in subsequent XML requests using Connection.send(). When the connection is no longer needed, close it with Connection.close().

Example 1–4 Establishing and Closing a Connection

```java
Connection conn = null;
try {
    conn = DriverManager.getConnection(
            "http://server:port/OrderManagement/XMLAPI", "user", "password" );
    String response = conn.send( "<Worklist.Request/>" );
    // ... perform additional processing and/or further calls ...
} 
catch ( ConnectionException ex ) {
    // handle the exception
}
finally {
    // Close the connection
    if ( conn != null ) {
        conn.close();
    }
}
```

XML Layer Client
While the connection layer simplifies handling the details of maintaining an HTTP session, it does not provide facilities for sending XML messages to a server or handling XML responses. The XML layer provides facilities to send and receive XML documents, report XML API error messages, and create OSM XML API requests. It is available in the com.mslv.oms.xmlclient.xml package, and contains the following primary classes:

- **XMLConnectionAdapter**: Adapts a connection to understand the OSM XML API.
- **XMLRequest**: A generic request to the XML API. This class is extended for specific requests.
- **XMLResponse**: A response from the XML API.
- **LAPIException**: Represents a server-side error response.
- **XMLAPIFacade**: A facade to simplify creation of XMLRequests.

Sending a Request
The XMLConnectionAdapter uses a connection and adds extra features that allow it to send and receive XML documents as well as translate OSM XML API error messages into client-side exceptions. Use XMLConnectionAdapter.sendXMLRequest() to send an XMLRequest, and XMLConnectionAdapter.sendDocument() to send a Document Object Model (DOM) document.

If an XML API error response is received, sendXMLRequest() throws an XMLAPIException. Retrieve the response error code with XMLAPIException.getCode().
Example 1–5 Sending a Request

Connection conn; // initialized elsewhere

XMLConnectionAdapter xmlConn = new XMLConnectionAdapter(conn);

try {

    // Setup a request to send
    XMLRequest request = new XMLGetUserInfoRequest();

    /* Send the request and get a response.
     * If an error occurred server-side, an
     * XMLAPIException is thrown.
     */
    XMLResponse response = xmlConn.sendXMLRequest(request);

    // Get the response org.w3c.dom.Document for further processing
    Document doc = response.getDocument();

} catch (ConnectionException ex) {

    // An error occurred in the underlying connection

} catch (XMLAPIException ex) {

    System.out.println("XML API error: "+ex.getCode());

} finally {

    if (xmlConn != null) {

        xmlConn.close();

    }
}

Using the XMLAPIFacade

To help manage the creation of XMLRequests, the XMLAPIFacade groups all the requests into a single class.

Example 1–6 XMLAPIFacade

Connection conn; // initialized elsewhere

XMLAPIFacade facade = new XMLAPIFacade(conn);

try {

    XMLResponse response = facade.getUserInfo();
    // Get the response org.w3c.dom.Document for further processing
    Document doc = response.getDocument();

} catch (ConnectionException ex) {

    // An error occurred in the underlying connection

} catch (XMLAPIException ex) {

    System.out.println("XML API error: "+ex.getCode());
Model Layer Client

The Model layer builds on the XML Layer to provide OSM-based representations of the XML documents. This lets developers work with the OSM concepts instead of manipulating XML documents. The package com.mslv.oms.xmlclient.model contains classes for representing an OSM order, its data description, and the data values.

Creating an Order

To create a new order using the Model layer, obtain a connection to the XML API and use the connection with the CreationOrder class.

Example 1–7  Creating an Order

```java
String url = "http://localhost:7001/OrderManagement/XMLAPI";
Connection conn = DriverManager.getConnection(url, "oms", "oms");

CreationOrder order =
    new CreationOrder(conn, "pots", "order_entry");
order.setReference("pots reference number");
OrderDataView rootView = order.getOrderDataViewRoot();
OrderData rootData = (OrderData)rootView.getOrderData().get(0);

    // Set the customer_order_number
OrderDataView customerOrderNumberView =
    rootView.getChild("customer_order_number");
OrderData customerOrderNumber =
    customerOrderNumberView.createOrderData(rootData);
customerOrderNumber.setValue("AAA-cust");

    ...//more Order Data is set
order.send();
order.complete("submit");
conn.close();
```

Retrieving an Order at a Task

The NextOrderAtTask class provides simplified access to order data using a task mnemonic and state.

Example 1–8  Retrieving an Order at a Task

```java
String url = "http://localhost:7001/OrderManagement/XMLAPI";
Connection conn = DriverManager.getConnection(url, "oms", "oms");

NextOrderAtTask nextOrder = new NextOrderAtTask(conn);
nextOrder.setTask("identify_ilec_co");
nextOrder.addState("received");
```
nextOrder.setAccept(true);

Order order = nextOrder.send();
if (order != null ) {
    System.out.println("Accepted order: *order.getOrderID()");
    OrderDataView rootView = order.getOrderDataViewRoot();
    OrderDataView serviceDetailsView =
        rootView.getChild("service_details");
    OrderData serviceDetails =
        (OrderData)serviceDetailsView.getOrderData().get(0);
    OrderDataView ilecView =
        serviceDetailsView.getChild("ilec_co_clli");
    if (ilecView.countOrderData() < 1) {
        ilecView.createOrderData(serviceDetails)
            .setValue("Big ILEC");
    
    order.send();
    }    
order.complete("true");}

Cartridge Management

Oracle Communications Design Studio provides the ability to create, build, and deploy service components into a single cartridge. For information on managing cartridges, see the discussion about Design Studio in OSM Developer’s Guide.
The Oracle Communications Order and Service Management (OSM) Extensible Markup Language (XML) Application Programming Interface (API) provides a single-access point located at an HTTP address. In your Web browser, enter:

http://server:port/OrderManagement/xmlapi/

where server is the specific server on which the application is deployed and port is the port on which the application listens.

All requests consist of an HTTP POST request and the responses consist of an HTTP response.

The OSM XML API provides the following functionality:

- Order creation, retrieval, and updating.
- Order transitioning: Provides functions to move an order through the various states and tasks of a process. This includes moving an order to a new process through exception processing.
- Next order at task: Search for the next available order at a given task.
- Copying 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.
- 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.
- Order data and process history: Retrieve the history of an order as it moves through the process.
- User information: Retrieve the user's name and description, as well as the name and description of each workgroup to which they belong.

See the OSM_home/SDK/XMLSchema/oms-xmlapi.xsd schema for additional information about the OSM XML API.
Backwards Compatibility for pre-OSM 2.5 Namespaces

Versions of OSM previous to version 2.5 have "architel" and "nortelnetworks" xml-namespaces. There are limitations when using old XML API requests including the inability to use some of the functionality introduced with newer versions.

Audience

This chapter is designed for developers familiar with Extensible Markup Language (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.

Login/Logout

Before using any API messages, you must login by supplying a valid user ID and password. If the login is successful, a session key is supplied as an HTTP cookie which must be included in the HTTP header of subsequent requests. After using the OSM API, you must release the session key by requesting to logout. The login/logout request is only valid from OMS version 2.5 and is not part of the API message formats.

To login, an HTTP POST message must be sent to:


where server is the specific server on which the application is deployed, port is the port on which the application listens, and un and pw are an authorized userid and password.

If successful, an XML API cookie is sent to the client. If unsuccessful, an HTTP FORBIDDEN (403) response is sent.

To logout, an HTTP POST message must be sent to:

http://server:port/OrderManagement/XMLAPI/logout

where server is the specific server on which the application is deployed, and port is the port on which the application listens.

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 2–1  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 2–2.

Example 2–2  Output XML Message Format With Warnings

```xml
<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.

Example 2–3  Output XML Message Format With Errors

```xml
<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:

- & - &amp;
- < - &lt;
- > - &gt;
- ' - &apos;
- " - &quot;

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 2–1 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 2–1  Reserved Mnemonics

<table>
<thead>
<tr>
<th>Header</th>
<th>Mnemonic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order Sequence ID</td>
<td>_order_seq_id</td>
</tr>
<tr>
<td>Order History Sequence ID</td>
<td>_order_hist_seq_id</td>
</tr>
<tr>
<td>State</td>
<td>_order_state</td>
</tr>
<tr>
<td>Execution Mode</td>
<td>_execution_mode</td>
</tr>
<tr>
<td>Task Mnemonic</td>
<td>_task_id</td>
</tr>
<tr>
<td>Order Source Mnemonic</td>
<td>_order_source</td>
</tr>
<tr>
<td>Order Type Mnemonic</td>
<td>_order_type</td>
</tr>
<tr>
<td>Order State</td>
<td>_current_order_state</td>
</tr>
<tr>
<td>Target Order State</td>
<td>_target_order_state</td>
</tr>
<tr>
<td>Reference Number</td>
<td>_reference_number</td>
</tr>
<tr>
<td>Priority</td>
<td>_priority</td>
</tr>
<tr>
<td>User</td>
<td>_user</td>
</tr>
<tr>
<td>Process Description</td>
<td>_process_description</td>
</tr>
<tr>
<td>Order Status</td>
<td>_process_status</td>
</tr>
<tr>
<td>Date Created</td>
<td>_date_pos_created</td>
</tr>
<tr>
<td>Date Started</td>
<td>_date_pos_started</td>
</tr>
<tr>
<td>Root node</td>
<td>_root</td>
</tr>
<tr>
<td>Number of Remarks</td>
<td>_num_remarks</td>
</tr>
<tr>
<td>Expected Order Completion Date</td>
<td>_compl_date_expected</td>
</tr>
<tr>
<td>NotificationID</td>
<td>_notif_id</td>
</tr>
<tr>
<td>Notification Description</td>
<td>_notif_desc</td>
</tr>
<tr>
<td>Notification Type</td>
<td>_notif_type</td>
</tr>
<tr>
<td>Notification Priority</td>
<td>_notif_priority</td>
</tr>
<tr>
<td>Notification Timestamp</td>
<td>_notif_time</td>
</tr>
<tr>
<td>Namespace</td>
<td>Namespace_namespace</td>
</tr>
<tr>
<td>Version</td>
<td>Version_version</td>
</tr>
</tbody>
</table>

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.

**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**

```xml
<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>45432</zip>
    </address>
  </Add>
</AddOrderThread.Request>
```

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

**Response**

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

**Request Example with Multiple Pivot Nodes**

```xml
<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**

```xml
<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: If you receive an error code that is not listed here, refer to Table 2–2, "Error Code Descriptions".

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

```xml
<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**

```xml
<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:** If you receive an error code that is not listed here, refer to Table 2–2, "Error Code Descriptions".
**AcknowledgeNotification**

Adds an acknowledgment to a notification. A notification can be acknowledged any number of times until it 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**
```xml
<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**
```xml
<AcknowledgeNotification.Response
xmlns="urn:com:metasolv:oms:xmlapi:1" />
```

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

**Note**: If you receive an error code that is not listed here, refer to Table 2-2, "Error Code Descriptions".

---

**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

```xml
<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

```xml
<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: If you receive an error code that is not listed here, refer to Table 2–2, "Error Code Descriptions”.

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:
**Cancel Order**

**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**

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

**Request Example 2: Cancel and Undo**

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

or

```xml
<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

```xml
<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.

```xml
<CancelOrder.Response xmlns="urn:com:metasolv:oms:xmlapi:1">
    <OrderID>1234</OrderID>
    <OrderHistID/>
</CancelOrder.Response>
```
Example 2: Cancel and Undo

```xml
<CANCELORDER.getResponse xmlns="urn:com:metasolv:oms:xmlapi:1">
  <ORDERID>1234</ORDERID>
</CANCELORDER.getResponse>
```

Error Codes

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

---

### 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**

```xml
<COMPLETEORDER.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
  <ORDERID>1234</ORDERID>
  <ORDERHISTID>33251</ORDERHISTID>
  <STATUS>submit</STATUS>
</COMPLETEORDER.Request>
```

**Response Example**

```xml
<COMPLETEORDER.Response xmlns="urn:com:metasolv:oms:xmlapi:1">
  <ORDERID>1234</ORDERID>
  <ORDERHISTID/>
</COMPLETEORDER.Response>
```

**Response Example for Amendment**

```xml
<COMPLETEORDER.Response xmlns="urn:com:metasolv:oms:xmlapi:1">
  <ORDERID>7</ORDERID>
  <ORDERHISTID/>
  <AMENDMENT xmlns="urn: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

---

**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**

```xml
<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**

```xml
<CopyOrder.Response xmlns="urn:com:metasolv:oms:xmlapi:1">
  <OrderID>1236</OrderID>
</CopyOrder.Response>
```
<OrderHistID>23334</OrderHistID>
<OrderSource>source1</OrderSource>
<OrderType>phone_transfer</OrderType>
<OrderState>open.not_running.notStarted</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: If you receive an error code that is not listed here, refer to Table 2–2, "Error Code Descriptions".

---

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 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

```xml
<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>
```

XML API Functionality

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).

```xml
<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: If you receive an error code that is not listed here, refer to Table 2–2, “Error Code Descriptions”.

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

```xml
<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.

```xml
<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:** If you receive an error code that is not listed here, refer to Table 2–2, "Error Code Descriptions".

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**

```xml
<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**

```xml
<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>
  <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_
    <client_info>
      <name>John Doe</name>
      <address index="76579">1211 Lakeview Dr.</address>
      <city>New York</city>
      <state>NY</state>
      <country>USA</country>
      <zip>12345</zip>
    </address>
  </_root_>
</GetNextOrderAtTask.Response>
```
<address index="80132">
<street1>20 Biz drv.</street1>
<city>New York</city>
<state>NY</state>
<country>USA</country>
<zip>12345</zip>
</address>
... more order data ...
</root>
... amendment order data ...
</root>
<color>green</color>
<shape>circle</shape>
<customer_ref>Cust01</customer_ref>
<pattern>checkerboard</pattern>
</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>
</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:** If you receive an error code that is not listed here, refer to Table 2–2, "Error Code Descriptions".

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 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.

**ViewID**: The particular view 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.

**Request Example**

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

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. If children of the _root element have multiple instances, their index attribute in the order must be used to uniquely identify an instance in any subsequent order updates.

In the following response example, <address> is a multi-instance node.

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 (yyyy-MM-ddTHH:mm:ss time zone).

**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**

```xml
<GetOrder.Response xmlns="urn:metasolv:oms:xmlapi_1">
  <OrderID>1234</OrderID>
  <OrderHistID>34433</OrderHistID>
  <Task>SampleTask1</Task>
  <OrderSource>source1</OrderSource>
  <OrderType>phone_transfer</OrderType>
  <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>
  <client_info>
    <name>John Doe</name>
    <address index="76579">
      <street1>1211 Lakeview Dr.</street1>
      <city>New York</city>
      <state>NY</state>
      <country>USA</country>
      <zip>12345</zip>
    </address>
    <address index="80132">
      <street1>20 Biz drv.</street1>
      <city>New York</city>
      <state>NY</state>
      <country>USA</country>
      <zip>12345</zip>
    </address>
  </client_info>
  ... more order data ...
</GetOrder.Response>
```

```xml
<Remarks>
  <Remark>
    <RemarkID>13444</RemarkID>
    <Date>2000-10-30T14:44:33 EST</Date>
  </Remark>
</Remarks>
```
XML API Functionality

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.

<Response xmlns="urn:com:metasolv:oms:xmlapi:1">
  <OrderID>1234</OrderID>
  <OrderHistID>34433</OrderHistID>
  <Task>stask_a</Task>
  <State>received</State>
  <OrderChangeID>565</OrderChangeID>
  <OrderSource>OCMPerspectivesTestOrder</OrderSource>
  <OrderType>OCMPerspectivesTestOrder</OrderType>
  <OrderState>open.running.compensating.amending</OrderState>
  <ExecutionMode>redo</ExecutionMode>
  <Reference/>
  <Priority>5</Priority>
  <Namespace>OCMPerspectivesTest</Namespace>
  <Version>1.0.0</Version>
</Response>

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.

<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>
  <OrderType>OCMPerspectivesTestOrder</OrderType>
  <OrderState>open.running.compensating.amending</OrderState>
  <ExecutionMode>redo</ExecutionMode>
  <Reference/>
  <Priority>5</Priority>
  <Namespace>OCMPerspectivesTest</Namespace>
  <Version>1.0.0</Version>
</Response>
<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>
<TaskExecutionHistory>
<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>

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

Note: If you receive an error code that is not listed here, refer to Table 2–2, "Error Code Descriptions".

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

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.

Request Example

<GetOrderAtTask.Request xmlns="urn:com:metasolv:oms:xmapi:1">
<OrderID>1234</OrderID>
<Task>SampleTask1</Task>
</GetOrderAtTask.Request>

Response Example

<GetOrderAtTask.Response xmlns="urn:com:metasolv:oms:xmapi:1">
<OrderID>204</OrderID>
<OrderHistID>443</OrderHistID>
<Task>SampleTask1</Task>
</GetOrderAtTask.Response>
<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>
    <ProcessStatus>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>200.0</currency>
        </nested_group>
    </order_origination>
</_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 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 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:

```xml
<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.

```xml
<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.

```xml
<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.

```xml
<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

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

or

```xml
<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 (yyyy-MM-ddTHH:mm:ss time zone).

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

```xml
<GetOrderDataHistory.Response xmlns="urn:com:metasolv:oms:xmapi: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"/>
  <Change action="create" user="jdoe" time="2000-01-28T14:33:22 EST" time="2000-01-28T14:35:23 EST">4169999999</Change>
  <Change action="update" user="jdoe" time="2000-01-28T14:35:23 EST">4168888888</Change>
</Field>
<Field path="/client_info/phone" namespace="DSL_Highspeedline " version ="1.1" index="12552" parentIndex="1222">
  <Change action="create" user="oms" time="2000-01-28T14:33:22 EST"/>
  <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: If you receive an error code that is not listed here, refer to Table 2–2, "Error Code Descriptions".
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**

```xml
<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 (yyyy-MM-ddTHH:mm:ss time zone).
- **ActualDuration**: The sum of the duration of all transitions of the order in seconds.
- **StartDate**: The date the order was started in the process (yyyy-MM-ddTHH:mm:ss time zone).
- **CompleteDate**: The date the order was completed (yyyy-MM-ddTHH:mm:ss time zone).

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 (yyyy-MM-ddTHH:mm:ss time zone).
- **CompleteDate**: The date and time the task and state was completed (yyyy-MM-ddTHH:mm:ss time zone).
- **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**

```xml
  <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
**GetOrderStateHistory**

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

**Operation**

GetOrderStateHistory

**Parameter**

OrderID: The order ID for the order.

**Request Example**

```xml
<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 (yyyy-MM-ddTHH:mm:ss time zone).
- **TransitionCompletedDate**: The date/time the order state was completed (yyyy-MM-ddTHH:mm:ss time zone).
- **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**

```xml
<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>
    </OrderState>
  </OrderStates>
</GetOrderStateHistory.Response>
```

**Note:** If you receive an error code that is not listed here, refer to Table 2–2, "Error Code Descriptions."
<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: If you receive an error code that is not listed here, refer to Table 2–2, "Error Code Descriptions".

GetTaskStatuses

Provides a list of all statuses for a given task.

Operation

GetTaskStatuses

Parameter

Task - A task mnemonic.

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:** If you receive an error code that is not listed here, refer to Table 2-2, "Error Code Descriptions".

---

**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**

```xml
<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

```xml
<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: If you receive an error code that is not listed here, refer to Table 2–2, "Error Code Descriptions".

---

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

```xml
<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

```xml
<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: If you receive an error code that is not listed here, refer to Table 2–2, "Error Code Descriptions".

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</OrderHistID>
</ListStatesNStatuses.Response>
<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: If you receive an error code that is not listed here, refer to Table 2–2, "Error Code Descriptions".

ListViews
Lists the view 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.

Request Example

```xml
<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

```xml
<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:** If you receive an error code that is not listed here, refer to Table 2–2, "Error Code Descriptions".

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 filenames 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

```xml
<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 filename 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. This must be used when transmitting the file to create a filename consisting of the attachment ID followed by the string '.srv'.
- **FileName**: The filename specified for the attachment.

Response Example

```xml
<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:** If you receive an error code that is not listed here, refer to Table 2–2, "Error Code Descriptions.”

---

**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.
**XML API Functionality**

**XML API Functions**

>_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.

In addition, for each flexible header assigned to the user:

- An element with the mnemonic path of the flexible header as name, and value based on the flexible header's data type:
  - Does not exist in the order: the element is empty.
  - NM (with or without a mask): the numeric value
  - TX (with or without a mask): the string value
  - DT with a date mask: a string representation of the date in the form: `yyyy-MM-dd` (for example `2001-10-30`)
  - DT with a date/time mask: a string representation of the date in the form: `yyyy-MM-ddThh:mm:ss time zone` (for example `2001-10-30T14:33:22 EST`).
  - PH: a ten-digit integer (for example `4161234567`)
  - YN: the string 'Yes' or 'No'
  - CY: the numeric value
  - LK: the lookup code
  - _namespace
  - _version

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**

```xml
<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**

```xml
<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. ">#"/>
  </Header>
</Notifications.Response>
```
<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>
</Notification>
XML API Functionality

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 xmlns="urn:com:metasolv:oms:xmlapi:1">
<TypeNSource namespace="DSL_Highspeedline" version="1.1">
<Type mnemonic="phone_activation" category="Customer Service" desc="Phone Activation Order"/>
</TypeNSource>
<TypeNSource namespace="DSL_Highspeedline" version="1.1">
<Type mnemonic="phone_transfer" category="Customer Service" desc="Phone Number Transfer"/>
</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: If you receive an error code that is not listed here, refer to Table 2–2, "Error Code Descriptions".
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 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 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

```xml
<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

```xml
<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="1"/>
```
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**: If you receive an error code that is not listed here, refer to Table 2–2, “Error Code Descriptions”.

---

**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**

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

**For an Order Not In a Process**

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

or

```xml
<OrderViewTemplate.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
  <OrderSource>source1</OrderSource>
</OrderViewTemplate.Request>
```
XML API Functionality

<OrderType>phone_transfer</OrderType>
<View>phone_transfer</View>
<Namespace>DSL_Highspeedline</Namespace>
-Version>1.1</Version>
</OrderViewTemplate.Request>

Error Codes

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

---

Note: If you receive an error code that is not listed here, refer to Table 2–2, "Error Code Descriptions".

---

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 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>

Error Codes

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

---

Note: If you receive an error code that is not listed here, refer to Table 2–2, "Error Code Descriptions".

---

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.

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.
TaskID: A 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, all available flexible headers are returned. 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.

**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:

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

If querying for a range of values:

```xml
<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. Use the namespace and version attributes to specify the namespace and version mnemonic of order type/source.

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**

```xml
<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

```xml
<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

```xml
<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

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

**Response Example**

```xml
<Query.Response xmlns="urn:com:metasolv:oms:xmlapi:1">
</Query.Response>
```
<Header>
  <_order_seq_id desc="Order ID" />
  <_order_state desc="State" />
  <_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" />
  <_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" />
  <_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" />
  <_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>1234</_order_seq_id>
  <_order_state>accepted</_order_state>
  <_execution_mode>do</_execution_mode>
  <_task_id>order_entry</_task_id>
  <_order_source>source1</_order_source>
  <_order_type>phone_transfer</_order_type>
  <_current_order_state>open.not_running.notStarted</_current_order_state>
  <_target_order_state />
  <_reference_number>23abcd3f</_reference_number>
  <_priority>5</_priority>
  <_user>oms</_user>
  <_process_status />
  <_date_pos_created>2000-03-23T08:15:23 EST</_date_pos_created>
  <_requested_delivery_date>2000-03-28T08:15:23 EST</_requested_delivery_date>
  <_expected_start_date>2000-03-23T08:15:23 EST</_expected_start_date>
  <_expected_duration>P5D</_expected_duration>
  <_compl_date_expected>2000-05-28T08:15:23 EST</_compl_date_expected>
  <_ord_completion_date />
  <_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>1235</_order_seq_id>
  <_order_state>accepted</_order_state>
  <_execution_mode>do</_execution_mode>
  <_task_id>order_entry</_task_id>
  <_order_source>source1</_order_source>
  <_order_type>phone_transfer</_order_type>
  <_current_order_state>open.not_running.notStarted</_current_order_state>
</Orderdata>
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. 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:** If you receive an error code that is not listed here, refer to Table 2–2, "Error Code Descriptions".

---

**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: If you receive an error code that is not listed here, refer to Table 2–2, “Error Code Descriptions”.

ResumeOrder
Releases a suspended order back into the system and returns it to the state from which it transitioned. 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

```xml
<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: If you receive an error code that is not listed here, refer to Table 2-2, "Error Code Descriptions".

---

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

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

Response Example

```xml
<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
SuspendOrder

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**

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

or

```xml
<SuspendOrder.Request xmlns="urn:com:metasolv:oms:xmlapi:1">
  <OrderID>1234</OrderID>
</SuspendOrder.Request>
```
<GracePeriodExpiryDate>2006-10-11T10:10 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: If you receive an error code that is not listed here, refer to Table 2-2, "Error Code Descriptions".

TaskDescription

The TaskDescription operation retrieves the list of all available tasks.

Operation

TaskDescription
XML API Functionality

Parameters

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

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

Request Example 1

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

Request Example 2

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

Response Example

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

Error Codes

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

**Note:** If you receive an error code that is not listed here, refer to Table 2-2, "Error Code Descriptions".

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 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.

**NewReference**: An optional new reference number for the order.

**AddMandatory**: This parameter is true if the mandatory fields defined in the order view 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 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 filename for a new attachment. If both the Text and Attachments elements are empty, a remark is not created.

**Request Example**

```xml
<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>
    </address>
  </Add>
</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 OMS.
- **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 OMS. This must be used when transmitting the file to create a filename consisting of the attachment ID followed by the string '.srv'.
  - **FileName**: The filename specified for the attachment.

**Response Example**

```xml
<UpdateOrder.Request>
  <Add>
    <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>
  </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 OMS.
- **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 OMS. This must be used when transmitting the file to create a filename consisting of the attachment ID followed by the string '.srv'.
  - **FileName**: The filename specified for the attachment.

**Response Example**

```xml
<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>

**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:** If you receive an error code that is not listed here, refer to Table 2–2, "Error Code Descriptions".

---

**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 want to 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

```xml
<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

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

or

```xml
<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

```xml
<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" />
    </Header>
</Worklist.Response>
```
<_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" />  
<_numRemarks 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>  
<_numRemarks>0</_numRemarks>  
<_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>  
</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>
Error Codes

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

Note: If you receive an error code that is not listed here, refer to Table 2–2, "Error Code Descriptions".

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 2–2 lists the error codes and their descriptions.

Table 2–2  Error Code Descriptions

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>100: Order type/source not found</td>
<td>The order type/source does not exist, or is not available to the user.</td>
</tr>
<tr>
<td>110: Order not found</td>
<td>The order does not exist, or is not available to the user.</td>
</tr>
<tr>
<td>120: Order template not found</td>
<td>The order template does not exist, or is not available to the user.</td>
</tr>
<tr>
<td>160: Remark not found</td>
<td>The remark could not be found.</td>
</tr>
<tr>
<td>150: Namespace/version not found.</td>
<td>The namespace or version does not exist or is not available to the user.</td>
</tr>
<tr>
<td>152: Invalid namespace mnemonic.</td>
<td>The order cannot be completed with the given namespace mnemonic.</td>
</tr>
</tbody>
</table>
### Table 2–2 (Cont.) Error Code Descriptions

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>153: No legacy data found. Namespace and Version need to be supplied.</td>
<td>Because the legacy data could not be found you must specify a valid Namespace and Version.</td>
</tr>
<tr>
<td>170: Header for mnemonic path not found</td>
<td>The header for a given mnemonic path does not exist, or is not available to the user.</td>
</tr>
<tr>
<td>190: Notification not found</td>
<td>If the notification ID does not exist, is no longer active, or is not assigned to the user ID or user ID's workgroup.</td>
</tr>
<tr>
<td>200: Order data invalid</td>
<td>The format of the order data is not correct. The message details the error location. For example, a message detail, such as &quot;com.mslv.oms.handler.OrderDataInvalidException: Could not convert input value: value_for_x for field data_field_x&quot;, may indicate that value_for_x exceeds the field's acceptable length set in OSM Administrator.</td>
</tr>
<tr>
<td>230: Order not accepted by user</td>
<td>An attempt to update an order was made without first retrieving the order with an Accept parameter of &quot;true&quot;.</td>
</tr>
<tr>
<td>232: Order update failed</td>
<td>The order could not be updated due to a data format error. The message details the reason for failure.</td>
</tr>
<tr>
<td>250: Mandatory check failed</td>
<td>A mandatory field was not given a value when attempting to create, assign, complete, or suspend an order.</td>
</tr>
<tr>
<td>251: Transition invalid</td>
<td>The order cannot be transitioned to that state. Use ListStatesNStatuses.Request to get a list of valid states.</td>
</tr>
<tr>
<td>252: Unable to accept order</td>
<td>When retrieving an order for update, the order cannot be accepted by the current user.</td>
</tr>
<tr>
<td>253: User not found</td>
<td>The order cannot be assigned to the given user ID.</td>
</tr>
<tr>
<td>254: Invalid state mnemonic</td>
<td>The order cannot be suspended with the given state mnemonic. <strong>Note:</strong> Only user-defined states are valid. If you want to complete or assign an order, you must use the appropriate request.</td>
</tr>
<tr>
<td>255: Invalid status mnemonic</td>
<td>The order cannot be completed with the given status mnemonic.</td>
</tr>
<tr>
<td>256: Invalid exception status mnemonic</td>
<td>You used an invalid exception status mnemonic when raising the process exception.</td>
</tr>
<tr>
<td>257: Invalid Task Mnemonic</td>
<td>The supplied task mnemonic could not be found.</td>
</tr>
<tr>
<td>260: Remark cannot be modified</td>
<td>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.</td>
</tr>
<tr>
<td>270: Transaction not allowed</td>
<td>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.</td>
</tr>
<tr>
<td>300: Request unknown</td>
<td>The request type could not be identified from the root element of the XML document of the message.</td>
</tr>
<tr>
<td>302: Request parameter error</td>
<td>A parameter for the request is missing or invalid. The message details the parameter in question.</td>
</tr>
</tbody>
</table>
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.

#### Request Example

```xml
<!-- 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
```

---

Table 2–2 (Cont.) Error Code Descriptions

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>350: Pivot node data is not provided</td>
<td>The order cannot proceed because no pivot node is indicated.</td>
</tr>
<tr>
<td>351: Process position supplied is not a sub process task.</td>
<td>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.</td>
</tr>
<tr>
<td>352: No sub process task is currently pending.</td>
<td>There are no sub-process tasks to which you can add a thread.</td>
</tr>
<tr>
<td>354: Process position not found.</td>
<td>The indicated process position does not exist or is inaccessible by the order.</td>
</tr>
<tr>
<td>355: Pivot node not found.</td>
<td>The indicated pivot node does not exist or is inaccessible by the order.</td>
</tr>
<tr>
<td>356: Cannot spawn threads for sub-process tasks that support sequential sub-processing.</td>
<td>You cannot add a sub-process thread to a sub-process that supports sequential sub-process.</td>
</tr>
<tr>
<td>400: Not authorized</td>
<td>The user is not authorized to make the request.</td>
</tr>
<tr>
<td>401: Database Connection Failed.</td>
<td>The XML API cannot connect to OSM.</td>
</tr>
<tr>
<td>419: The process exception is restricted.</td>
<td>A process exception can not be raised, because it is restricted.</td>
</tr>
<tr>
<td>420: Not authorized to modify order priority.</td>
<td>The user does not have the necessary privilege to modify order priority.</td>
</tr>
<tr>
<td>500: Internal error.</td>
<td>An internal application error has occurred. The message details further information.</td>
</tr>
<tr>
<td>601: Deprecated parameter</td>
<td>The parameter identified in the warning details has been deprecated. The details specify an applicable replacement parameter.</td>
</tr>
</tbody>
</table>
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,
Response Example

<!-- copy_order_response.dtd -->
<!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)>
<!ELEMENT _root(children*)>
<!ELEMENT children(#PCDATA | children*)>
<!ELEMENT Namespace(#PCDATA)>
<!ELEMENT Version(#PCDATA)>

Response Example

<!-- create_order_response.dtd -->
<!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)>

CreateOrder

The CreateOrder XML API is used to create an order.
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 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)>
GetNextOrderAtTask

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

Request Example

```xml
<!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

```xml
<!ELEMENT GetNextOrderAtTask.Response((OrderID, OrderHistID, OrderSource, OrderType, OrderState, State, ExecutionMode, Reference, Priority, _root, Warnings?, Namespace?, Version?) | EMPTY)>
<!ELEMENT OrderID(#PCDATA)>
<!ELEMENT OrderHistID(#PCDATA)>
<!ELEMENT ViewID(#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

```xml
<!ELEMENT GetOrderDataHistory.Request(OrderID, (OrderHistID | ViewID), Field*)>
<!ELEMENT OrderID(#PCDATA)>
<!ELEMENT OrderHistID(#PCDATA)>
<!ELEMENT ViewID(#PCDATA)>
```
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 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)>
GetOrderStateHistory

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

Request Example

```xml
<!ELEMENT GetOrderStateHistory.Request(OrderID)>
<!ELEMENT OrderID(#PCDATA)>
```

Response Example

```xml
<!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

```xml
<!ELEMENT GetUserInfo.RequestEMPTY>```

Response Example

```xml
<!ENTITY % warning SYSTEM "warning.dtd">
%warning;
<!ELEMENT GetUserInfo.Response(User, Workgroup*FlexibleHeaders)>
<!ELEMENT User(#PCDATA)>
<!ATTLIST User desc CDATA #IMPLIED>
<!ELEMENT Workgroup(#PCDATA)>
<!ATTLIST Workgroup desc CDATA #IMPLIED>
<!ELEMENT FlexibleHeaders((FlexibleHeader*))>```
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)>
<!ELEMENT Suspend(userstates+)>
<!ELEMENT Completed(userstatuses+)>
Let userstates := mnemonics for user defined states -->
<!ELEMENT? Suspend(userstates+)>
<!ELEMENT? 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

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

Response Example

```xml
<!-- 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

```xml
<!-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

```xml
<!-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)>
<!ELEMENT Namespace(#PCDATA)>
<!ELEMENT Version(#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">
Query

The Query XML API is used to query an order.

Request Example

```xml
<!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>
<!ELEMENT CompletedDateEMPTY>
<!ELEMENT Field(#PCDATA | From, To)>
<!ELEMENT FlexibleHeaders(FlexibleHeader*)>
<!ELEMENT FlexibleHeader(#PCDATA)>
<!ELEMENT Namespace(#PCDATA)>
<!ELEMENT Version(#PCDATA)>
```

Response Example

```xml
<!ELEMENT Query.Response(Header, Orderdata*, Warnings?, Namespace?, Version?)>
<!-- The contents of Header cannot be described in a dtd. -->
```
Document Type Definitions (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)>
SuspendOrder

The SuspendOrder XML API is used to suspend an order.

Request Example

```xml
<!-- 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

```xml
<!-- 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

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

Response Example

```xml
<!-- 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

```xml
<!-- update_order_request.dtd -->
<!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 Update element 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 headerchild EMPTY>
<!ATTLIST headerchild desc CDATA #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 orderedatachild := each of the children of Orderdata -->
<!ELEMENT orderedatachild (#PCDATA)>
<!ELEMENT Namespace (#PCDATA)>
<!ELEMENT Version (#PCDATA)>