ORACLE

Oracle CPQ – Oracle Order Management Integration Guide

Update 21B

APR 2021

Table of Contents

Revision History

This document will continue to evolve as existing sections change and new information is added.

All updates appear in the following table:

Date	What's Changed	Notes
APR 2021	21B Oracle CPQ – Oracle Order Management REST-Based Integration using OIC	

Introduction

This integration includes:

- REST-based Integration with Oracle Order Management using Oracle Integration Cloud (OIC)
- Support for pre-priced Order creation and cancellation
- Asynchronous update of quote order and line status from Oracle Order Management

Purpose

This installation guide describes how to implement the reference integration between Oracle CPQ and Oracle Order Management using order management REST APIs and OIC.

Audience

This installation guide is intended for administrators responsible for setting up and configuring the Oracle CPQ - Order Management solution. This guide assumes administrators have prior Oracle CPQ, Oracle Order Management, and OIC administration experience.

Prerequisites

Administrators must integrate the Oracle CPQ 21B Base Reference Application (RefApp) image with the following:

- Subscription Ordering Package, ABO RefApp Basic Package V3 - 19C
- Oracle Order Management 21B or later using OIC 20.3.3.0.0 or later middleware, which is used to establish an integration between Oracle CPQ and Oracle Order Management
- Customer Data Management (CDM) Integration which supports account integration
- Users can create products in the Oracle Fusion Product Model and synchronize the products into CPQ. Likewise, they can create parts in Oracle CPQ and synchronize them into the Oracle Fusion Product Model.

> **Notes**:
> - Administrators performing the integration installation must have administrator privileges on the Oracle CPQ, Order Management, and OIC sites.
> - For information about obtaining any of the above prerequisites, contact My Oracle Support.

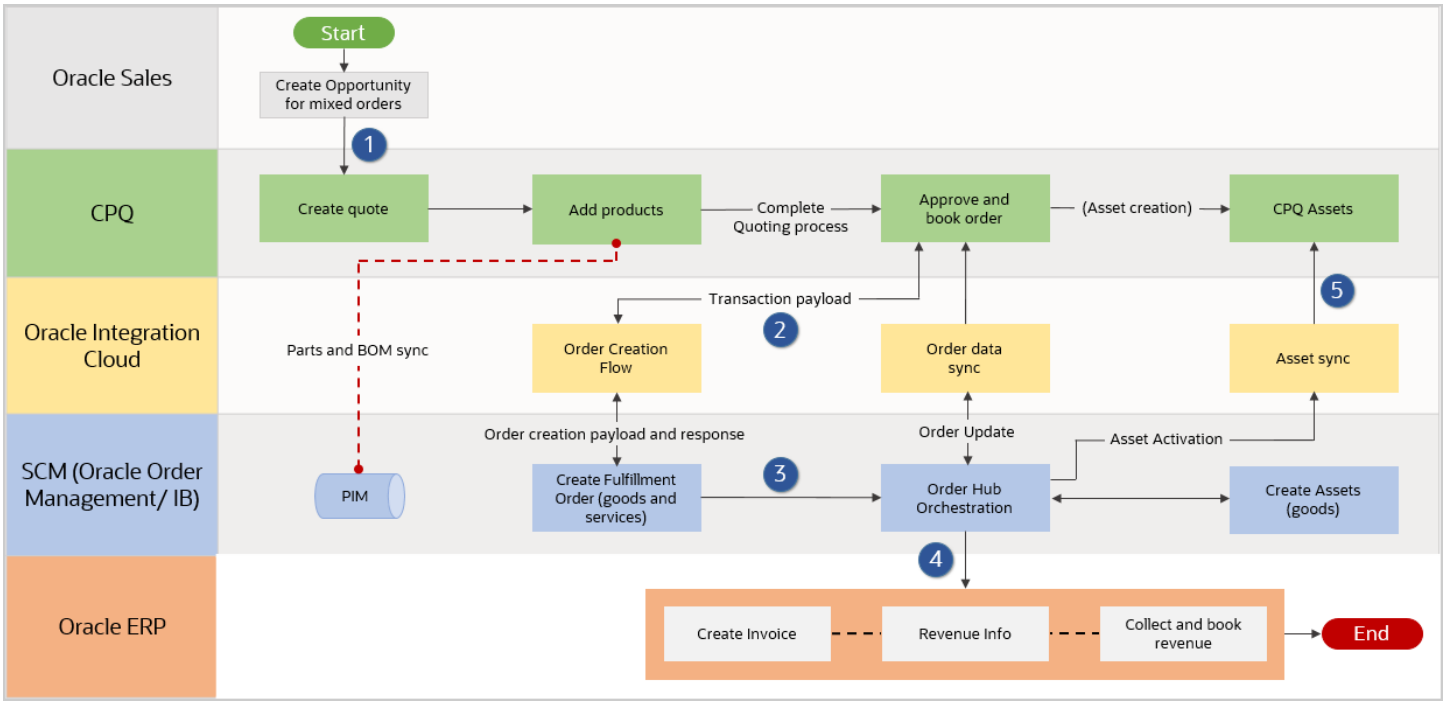
Acronym List

Definitions of the acronyms used within this document are provided in the following table. For additional information, refer to the Oracle CPQ Administration Online Help.

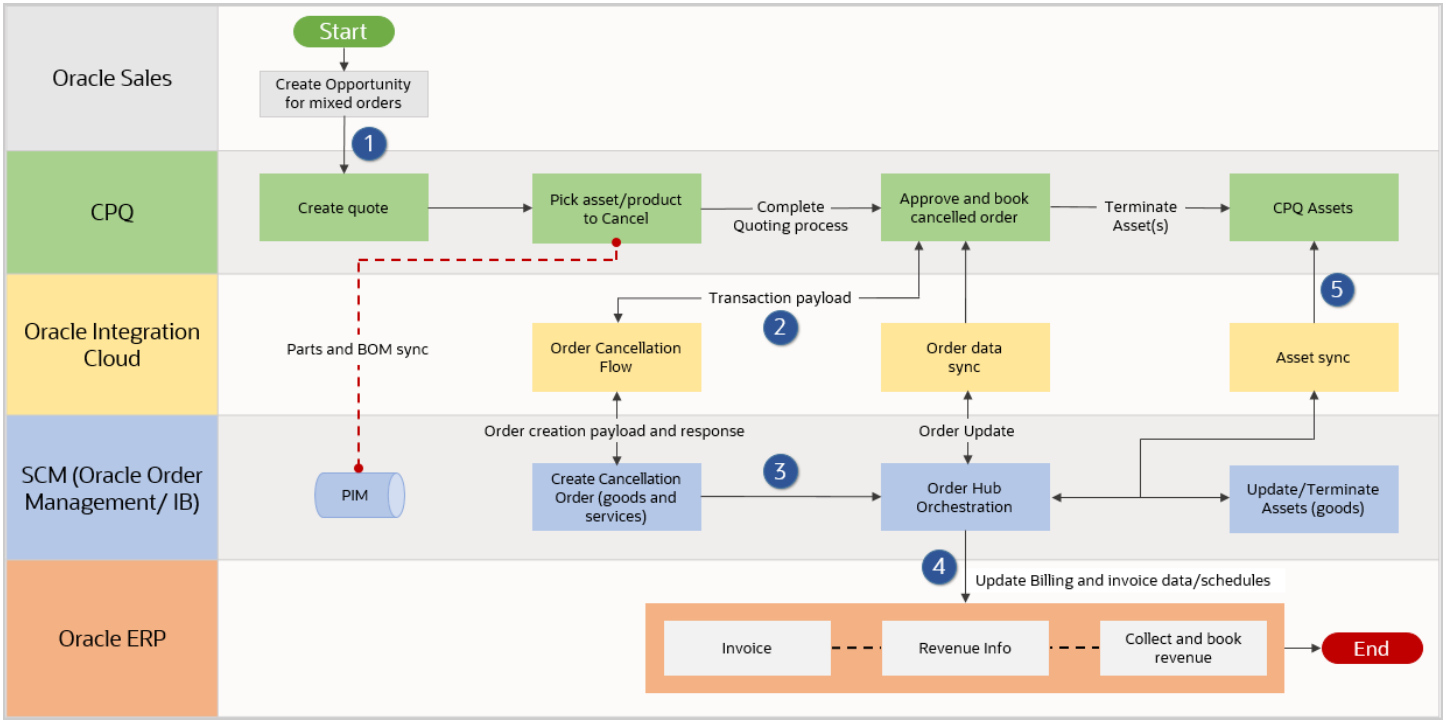
Acronym	Definition	Description
BML	Big Machines Extensible Language	A scripting tool used to capture a company's complex business logic within Oracle CPQ configuration and commerce.
BOM	Bill of Material	Fulfillment systems often maintain BOMs containing complex, multi-level part structures that differ from the configuration attributes used in Oracle CPQ when sales users configure products. BOM mapping provides a data-driven mechanism for mapping these differing product views. To use the Order Management solution to create an order from an Oracle CPQ transaction, the products must be modeled as a BOM.
CPQ	Configure Price and Quote	This Oracle solution enables companies to streamline their entire opportunity-to-quote-to-order process, including product selection, configuration, pricing, quoting, ordering, and approval workflows.
OM	Order Management	Oracle Order Management, previously known as Fusion Order management, is designed to improve order capture and fulfillment execution across the quote to cash process by providing a central order hub for multi-channel environments. The application provides the ability to capture, price and configure orders through direct order entry.
OIC	Oracle Integration Cloud	Oracle Integration Cloud Service is a cloud-based integration application designed to perform integrations between cloud-based applications – but also has capabilities that extend beyond that, to performing integrations with your on premises applications.

Order Flow Overview

Create Order Process Flow



Cancel Order Process Flow



OIC Integration Installation and Setup

This section contains information about importing the OIC Integration package into your OIC environment and creating web service connections between Oracle CPQ and Oracle Order Management.

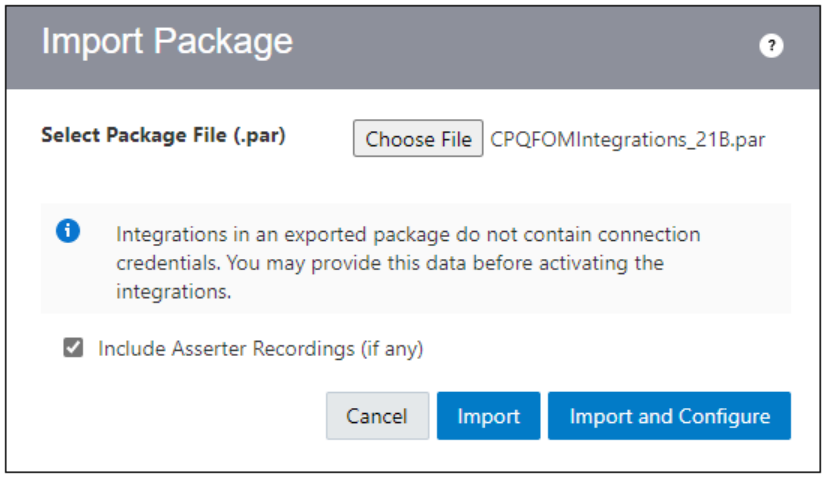
Prerequisites

- The SSL certificate of OIC environment is installed in Oracle Order Management for ERP event subscription.
- All required SSL certificates (CPQ and Order Management SSL certificates) are in OIC environment.

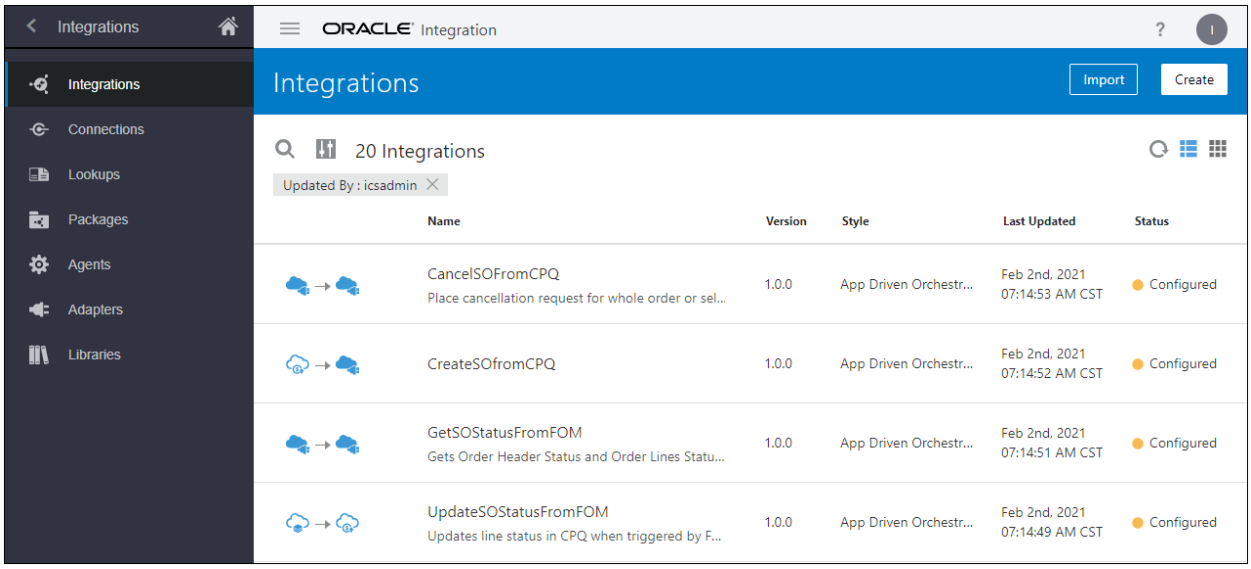
Import Integration Package

Perform the following steps to import the OIC integration package into OIC.

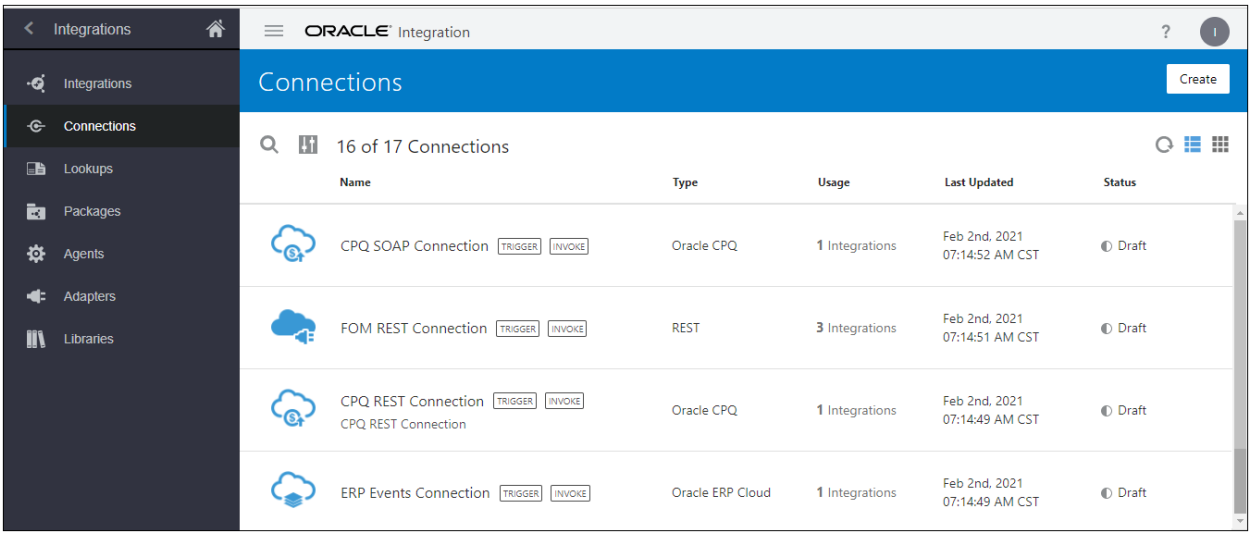
1. Log in to the OIC site as an administration user.
2. Select **Integrations** in the left side navigation panel, and then select **Packages**.
3. Click **Import**.
4. Click **Chose File**, and then select the CPQFOMIntegrations_21B.par package.



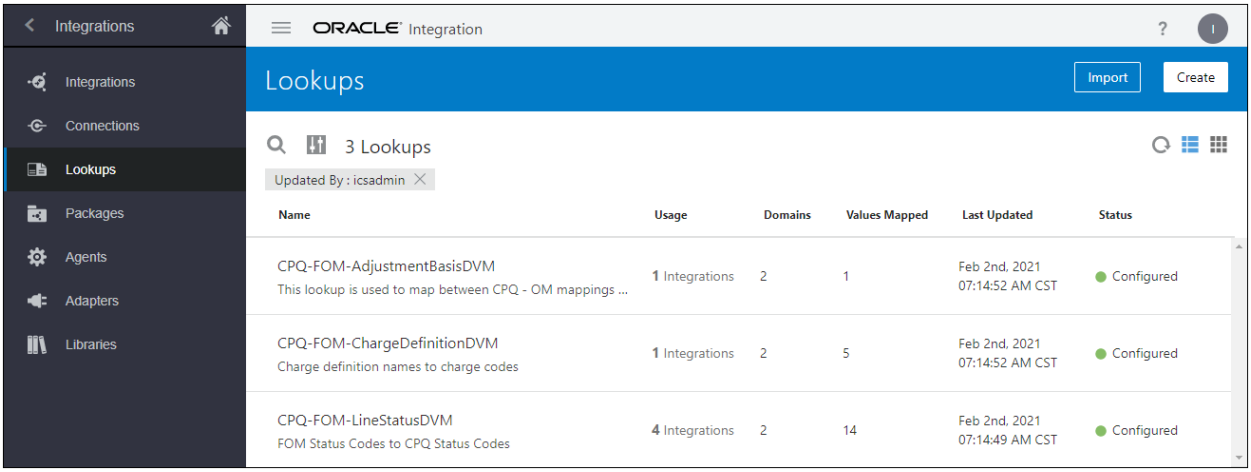
5. Click **Import and Configure**.
6. Select **Integrations** in the left side navigation panel, verify the following integrations have been imported:
   - CreateSOFromCPQ
   - UpdateSOStatusFromFOM
   - CancelSOFromCPQ
   - GetSOStatusFromFOM



7. Select **Connections** in the left side navigation panel, verify the following connections have been imported:
   - CPQ REST Connection
   - ERP Events Connection
   - FOM REST Connection
   - CPQ SOAP Connection



8. Select **Lookups** in the left side navigation panel, verify the following lookups have been imported:
   - CPQ-FOM-LineStatusDVM
   - CPQ-FOM-AdjustmentBasisDVM
   - CPQ-FOM-ChargeDefinitionDVM



9. Configure the CPQ REST Connection.
   a. Click the Edit icon for the **CPQ REST Connection**.
   b. Enter the connection URL in the following format:

      ```
      https://<CPQ host name>/rest/<latest rest version>/metadata-catalog
      ```

   c. Enter your CPQ administrator user name and password in the corresponding fields.
   d. Click **Test** to verify the connection.
   e. Click **Save**.
   f. Click the Back icon.

10. Configure the CPQ SOAP Connection
    a. Click on the Edit icon for the **CPQ SOAP Connection**.
    b. Select “wsdlUrl” for the **Connection Type**.
    c. Enter the connection URL in the following format:

```
https:// <CPQ host name>/v2_0/receiver/commerce/oraclecpqo?wsdl
```

    d. Enter your CPQ administrator user name and password in the corresponding fields.
    e. Click **Test** to verify the connection.
    f. Click **Save**.
    g. Click the Back icon.
11. Configure the Order Management REST Connection.
    a. Click the Edit icon for the **FOM REST Connection**.
    b. Select “restUrl” for the **Connection Type**.
    c. Select “TLSv1.2TLS” for the **Version**.
    d. Enter the connection URL in the following format

```
https://<Order Management host name>/fscmRestApi/resources/<latest rest version>/
```

    e. Select security policy as Basic Authentication.
    f. Enter your Order Management username and password in the corresponding fields.
    g. Click **Test** to verify the connection.
    h. Click **Save**.
    i. Click the Back icon.

> **Note**: The integration Order Management user should have appropriate roles to Create Orders using the Order Management REST APIs.

12. Configure the ERP Events Connection.
    a. Click the Edit icon for the **ERP Events Connection**.
    b. Enter ERP Cloud Host in the following format:

```
https://<Order Management host name>
```

    c. Enter your Fusion user name and password for Fusion in the corresponding fields.
    d. Click **Test** to verify the connection.
    e. Click **Save**.
    f. Click the Back icon.
13. On configuration editor page click on activate button for all four integrations one by one and activate all integration.

Notes:

- If an integration with the same name already exists in the OIC environment, deactivate the integration before replacing it.
- Oracle recommends that the **Enable Tracing** and **Include Payload** options are selected when activating an integration. Doing so will capture valuable troubleshooting information that may be useful in troubleshooting a failed integration run/instance.

OIC Mapping Details

This section contains the OIC mapping details of the Order Management payload and Oracle CPQ attribute

Header Attribute Mapping

Order Management Attribute	Mapped to CPQ Attribute	Comments
BuyingPartyContactFirstName	_customer_t_first_name	
BuyingPartyContactLastName	_customer_t_last_name	
BuyingPartyName	_invoiceTo_t_company_name OR customerCompanyName_t	
FreezePriceFlag	freezePriceFlag_t	
FreezeShippingChargeFlag	freezePriceFlag_t	
FreezeTaxFlag	freezePriceFlag_t	
OrderKey	orderKey_t	
CancelReasonCode	cancelReason_t	
StatusCode	status_t	
PackingInstructions	packingInstructions_t	
PartialShipAllowedFlag	oRCL_ERP_PartialShipAllowed_t	
PaymentTerms	paymentTerms_t	
RequestingBusinessUnitName	businessUnitName_t	
RequestingBusinessUnitId	businessUnitId_t	
SourceTransactionId	bs_id	
SourceTransactionNumber	transactionID_t	
SourceTransactionSystem	oRCL_ERP_CPQSourceSystemCode_t	
TransactionOn	orderDate_t	
TransactionTypeCode	transactionTypeCode_t	
TransactionalCurrencyCode	currency_t	
billToCustomer - AccountNumber	invoiceToPartyID_t OR partyId_t	If invoiceToPartyID_t is not null then invoiceToPartyID_t else partyId_t
shipToCustomer - PartyId	shipToPartyID_t	
shipToCustomer - PartyName	_shipTo_t_company_name	

Line Attribute Mapping

Order Management Attribute	Mapped to CPQ Attribute	Comments
EndCreditMethodCode	changeCode_l	When oRCL_ABO_ActionCode_l = 'TERMINATE' (For future use)
EndDate	contractEndDate_l	When oRCL_ABO_ActionCode_l = 'TERMINATE' (For future use)
EndReasonCode	changeReason_l	When oRCL_ABO_ActionCode_l = 'TERMINATE' (For future use)
associatedProductReferences – SourceLineId	associatedLineId_l	Applicable only for Subscription Lines with Associations (For future use)
associatedProductReferences - SourceOrderId	associatedOrderId_l	Applicable only for Subscription Lines with Associations (For future use)
associatedProductReferences - SourceOrderSystem	oRCL_ERP_CPQSourceSystemCode_t	Applicable only for Subscription Lines with Associations (For future use)
SubscriptionProfileId	subscriptionProfileId_t	Applicable only for Subscription Lines. ie., _part_custom_field9 = 'product' (For future use)
externalAssetReference - ExternalAssetKey	itemInstanceId_l	Applicable only for Subscription Lines. ie., _part_custom_field9 = 'product' (For future use)
ActionTypeCode	'ORA_END'	When oRCL_ABO_ActionCode_l = 'TERMINATE' (For future use)
CancelReasonCode	cancelReason_l	
ContractEndDate	contractEndDate_l	(For future use)
ContractStartDate	contractStartDate_l	(For future use)
OrderedQuantity	requestedQuantity_l	
OrderedUOMCode	oRCL_pRC_quantityUOM	
PackingInstructions	oRCL_ERP_PackingInstr_l	
ParentSourceTransactionLineId	parentDocNumber_l	
PartialShipAllowedFlag	oRCL_ERP_PartialShipAllowed_t	
ProductNumber	_part_number OR _model_name	
RequestedFulfillmentOrganizationCode	fulfillmentOrganizationCode_l	
RequestedShipDate	oRCL_ERP_RequestShipDate_l	
ShipSetName	oRCL_ERP_ShipSet_l	
ShippingInstructions	oRCL_ERP_ShippingInstr_l	
SourceScheduleNumber	_document_number	
SourceTransactionLineId	_document_number	
SourceTransactionLineNumber	_document_number	
SourceTransactionScheduleId	_document_number	
StatusCode	status_l	
TransactionCategoryCode	transactionCategoryCode_l	

Charge Attribute Mapping

Order Management Attribute	Mapped to CPQ Attribute	Comments
ApplyTo	oRCL_applyTo	
ChargeCurrencyCode	currency_t	
ChargeDefinitionCode	oRCL_chargeName	Uses OIC lookup CPQ-FOM-ChargeDefinitionDVM for domain value map. The value should match with the Order Management Pricing Charge Definition code.
ChargeTypeCode	oRCL_chargeType	
PricePeriodicityCode	oRCL_periodicity	Applicable only when oRCL_chargeType = ORA_RECURRING
PricedQuantity	requestedQuantity_l	
PricedQuantityUOMCode	oRCL_pRC_quantityUOM	
PrimaryFlag	oRCL_primaryCharge	
RollupFlag	rollupFlag_l	
SequenceNumber	oRCL_chargeSequenceNumber	
SourceChargeId	oRCL_chargeName	

Charge Attribute Mapping for List and Net Prices

Order Management Attribute	Mapped to CPQ Attribute	Comments
ChargeCurrencyCode	currency_t	
ChargeCurrencyDurationExtendedAmount	oRCL_listPrice	When a line is Subscription ie., _part_custom_field9 = 'product' (For future use)
ChargeCurrencyExtendedAmount	oRCL_listPrice	
ChargeCurrencyUnitPrice	oRCL_unitPrice	
HeaderCurrencyCode	currency_t	
HeaderCurrencyDurationExtendedAmount	oRCL_listPrice	
HeaderCurrencyExtendedAmount	oRCL_listPrice	
HeaderCurrencyUnitPrice	oRCL_unitPrice	
PriceElementCode	'QP_LIST_PRICE' OR 'QP_NET_PRICE'	Should match with the Price Element Code defined in Order Management
PriceElementUsageCode	'LIST_PRICE' OR 'NET_PRICE'	Should match with the Price Element Usage Code defined in Order Management
RollupFlag	False	
SequenceNumber	Charge component Sequence Number	
SourceChargeComponentId	oRCL_chargeName	Unique charge component Id generated using charge name.

Charge Attribute Mapping for Adjustments

Order Management Attribute	Mapped to CPQ Attribute	Comments
ChargeCurrencyCode	currency_t	
ChargeCurrencyDurationExtendedAmount	oRCL_totalAutoAdjustment OR oRCL_totalManualAdjustment	When a line is Subscription i.e., _part_custom_field9 = 'product' (For future use)
ChargeCurrencyExtendedAmount	oRCL_totalAutoAdjustment OR oRCL_totalManualAdjustment	
ChargeCurrencyUnitPrice	oRCL_autoAdjustment OR oRCL_manualAdjustment	
HeaderCurrencyCode	currency_t	
HeaderCurrencyDurationExtendedAmount	oRCL_totalAutoAdjustment OR oRCL_totalManualAdjustment	
HeaderCurrencyExtendedAmount	oRCL_totalAutoAdjustment OR oRCL_totalManualAdjustment	
HeaderCurrencyUnitPrice	oRCL_autoAdjustment OR oRCL_manualAdjustment	
PriceElementCode	'QP_CUSTOM_ADJUSTMENT'	
PriceElementUsageCode	'PRICE_ADJUSTMENT'	
RollupFlag	False	
SequenceNumber	Charge component Sequence Number	
SourceChargeComponentId	oRCL_chargeName	Unique charge component Id generated using charge name.
Explanation	'Auto adjustment' OR 'Manual adjustment'	
PricingSourceTypeCode	'MANUAL_ADJUSTMENT'	

Manual Price Adjustment Mapping

Order Management Attribute	Mapped to CPQ Attribute	Comments
AdjustmentAmount	oRCL_manual_adjustmentValue_l OR oRCL_auto_adjustmentValue_l	
AdjustmentElementBasis	oRCL_manual_adjustmentBasis_l OR oRCL_auto_adjustmentBasis_l	Uses OIC lookup for domain value map CPQ-FOM-AdjustmentBasisDVM
AdjustmentType	oRCL_manual_adjustmentType_l OR oRCL_auto_adjustmentType_l	
ChargeDefinitionCode	oRCL_manual_adjustmentChargeName_l OR oRCL_auto_adjustmentChargeName_l	Uses OIC lookup CPQ-FOM-ChargeDefinitionDVM for domain value map
ChargeRollupFlag	False	
Comments	oRCL_manual_adjustmentName_l OR oRCL_auto_adjustmentName_l	
EffectiveNumberOfPeriods	oRCL_manual_numberofPeriods_l OR oRCL_auto_numberOfPeriods_l	Applicable only when Product is subscription and oRCL_manual_adjustmentChargeType_l = ORA_RECURRING AND oRCL_manual_adjustmentEffectivity_l = ORA_ALL_TERM (For future use)
EffectivityTypeCode	oRCL_manual_adjustmentEffectivity_l OR oRCL_auto_adjustmentEffectivity_l	Only when Product is subscription and oRCL_manual_adjustmentChargeType_l = ORA_RECURRING (For future use)
Reason	oRCL_manual_adjustmentReason_l OR 'Price match'	'Price match' – For Auto Adjustment
SequenceNumber	oRCL_manual_adjustmentSequence_l OR oRCL_auto_adjustmentSequence_l	
SourceManualPriceAdjustmentId	oRCL_manual_adjustmentSequence_l OR oRCL_auto_adjustmentSequence_l	

Oracle CPQ Package Installation and Setup

Oracle creates implementation packages as a way to distribute elements needed by customers to implement new Oracle CPQ features. This section contains information about importing the CPQ -Order Management Package into your CPQ environment.

Package Name	Description
CPQ - Order Management Package **File Name** : (CPQ_FOM_Package_21B.zip)	Granular migration CPQ package containing CPQ artifacts in support of the Order Management Integration solution.

> **Note:** This package also contains some of the artifacts related to subscription products handling in CPQ and Order Management Integration for a future release.

CPQ Integration Center Setup

Before installing the pre-built Oracle CPQ – Order Management Package, administrators must create the following OIC integration and Generic Integrations in Oracle CPQ Integration Center.

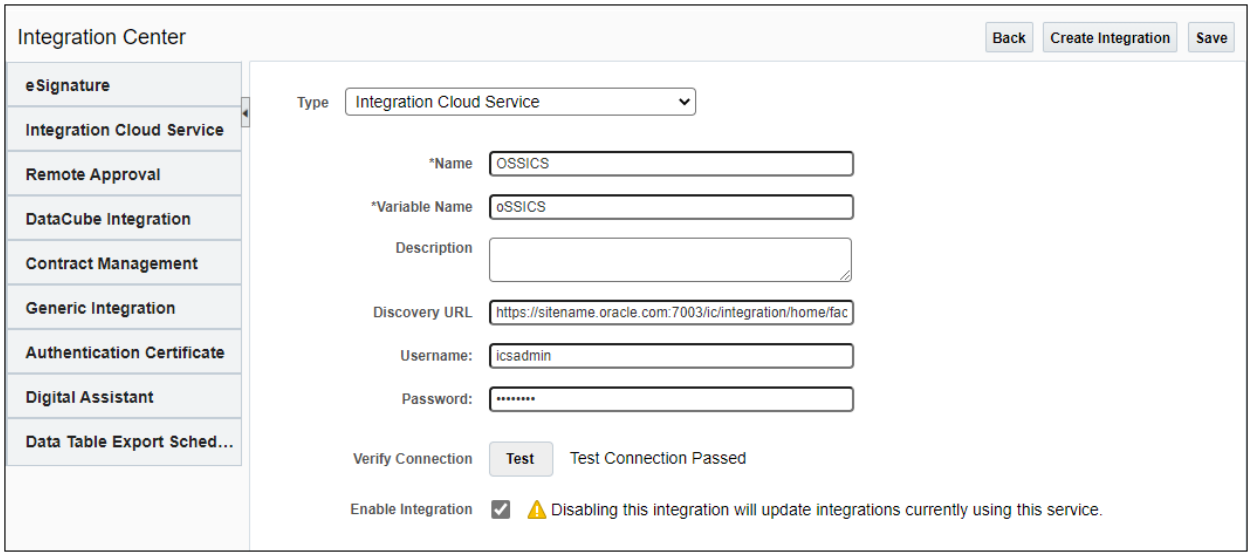
Create OIC Integration in CPQ Integration Center

Creating an OIC integration enables Oracle CPQ to connect to back office systems, on premise environments, and other Oracle products in a consistent, enhanced manner.

> **Note:** OIC is also known as Integration Cloud Service (ICS). When creating the OIC integration in the Integration Center, select Integration Cloud Service as the integration type. Also use the same user name and password that was used to import the integration package in OIC.

Perform the following steps to create an OIC integration.

1. Open the Admin Home page.
2. Select **Integration Center** in the Integration Platform section.
3. Click **Create Integration**.
4. Select **Integration Cloud Service** from the **Type** drop-down.



5. Enter **OSSICS** in the Name field. The variable name should be **oSSICS** on both the source and the target site.
6. Enter the discovery URL in the following format: `https://<OIC hostname>/icsapis/v1/integrations`
   The host name is the OIC environment name.
7. Enter the username and password for the OIC environment.
8. Click **Test** to verify the connection. The status must return “Test Connection Passed” before proceeding.
9. Select the **Enable Integration** check box.
10. Click **Save**.

Generic Integrations

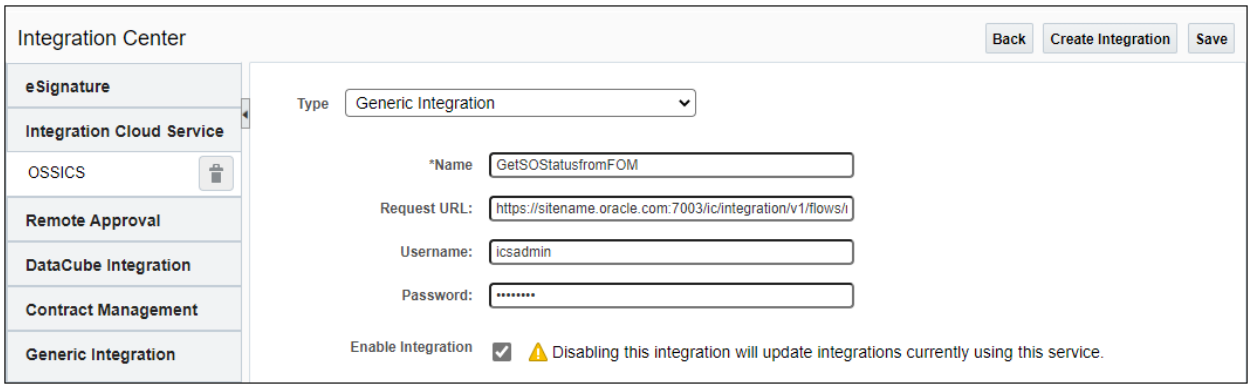
Generic integration is required to invoke OIC integrations for **Cancel Order** and **Get Order Status from FOM**

Generic Integration for Get Sales Order Status from FOM

This integration uses the OIC **GetSOStatusfromFOM** integration endpoint URL. Refer to Appendix K for instructions to retrieve the endpoint URL of an OIC integration.

Perform the following steps to create a generic integration to get sales order status from Order Management.

1. Open the Admin Home page.
2. Select **Integration Center** in the Integration Platform section.
3. Click **Create Integration**.
4. Select **Generic Integration** from the **Type** drop-down.



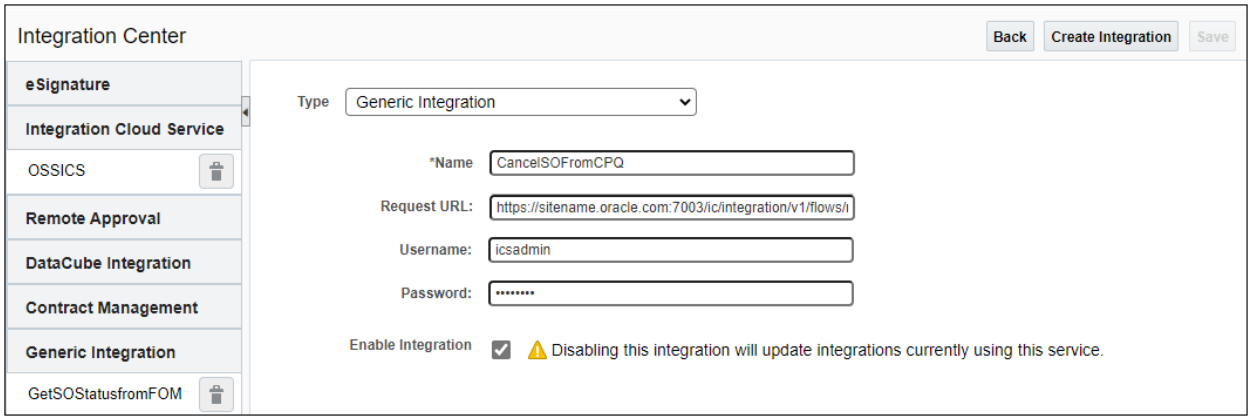
5. Enter **GetSOStatusfromFOM** in the Name field.
6. Enter the **Discovery URL** in the following format:
   `http://<OIC hostname>/ic/api/integration/v1/flows/rest/getsostatusfromfom/1.0/getsostatusfromfom/`
7. Enter the username and password for the OIC environment.
8. Select the **Enable Integration** check box.
9. Click **Save**.

Generic Integration for Cancel a Sales Order from CPQ

This integration uses the OIC **CancelSOfromCPQ** integration endpoint URL. Refer to Appendix K for instructions to retrieve the endpoint URL of an OIC integration.

Perform the following steps to create a generic integration to cancel a sales order from CPQ.

1. Open the Admin Home page.
2. Select **Integration Center** in the Integration Platform section.
3. Click **Create Integration**.
4. Select **Generic Integration** from the **Type** drop-down.



5. Enter **CancelSOFromCPQ** in the Name field.
6. Enter the discovery URL in the following format:

   `https://<OIC hostname>/ic/api/integration/v1/flows/rest/CANCELSOFROMCPQ/1.0/CancelSOfromCPQ`

   The hostname is the OIC environment name.
7. Enter the username and password for the OIC environment.
8. Select the **Enable Integration** check box.
9. Click **Save**.

Install CPQ-Oracle Order Management Package

The CPQ-Oracle Order Management package is a granular migration package containing new elements in support of the Order Management solution. As a granular migration package, administrators can add or remove specific elements from the package or remove specific elements when importing the package.

Perform the following steps to install the Order Management package.

1. Download the Oracle Order Management package (i.e. CPQ_FOM_Package_21B.zip) from My Oracle Support.
2. Open the Admin Home page.
3. Select **Migration** in the Utilities section.
4. Select **Import Package** from the **Select A Mode** drop-down. The Upload Package dialog opens.
5. Click **Browse** and navigate to the CPQ_FOM_Package_21B.zip package.
6. Select the target process from the Choose a target process for Cross Process Migration drop-down.
7. Click **Upload**.
8. Click **Migrate**.

When the migration completes, check the migration logs for errors.

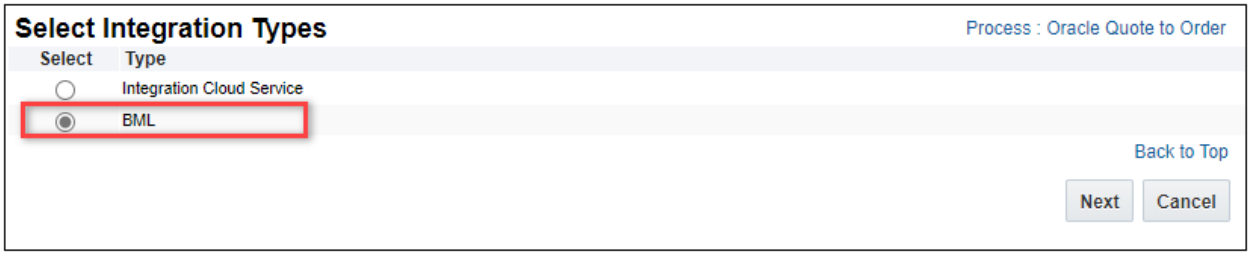
Commerce Integrations

Create the following commerce BML type integrations:

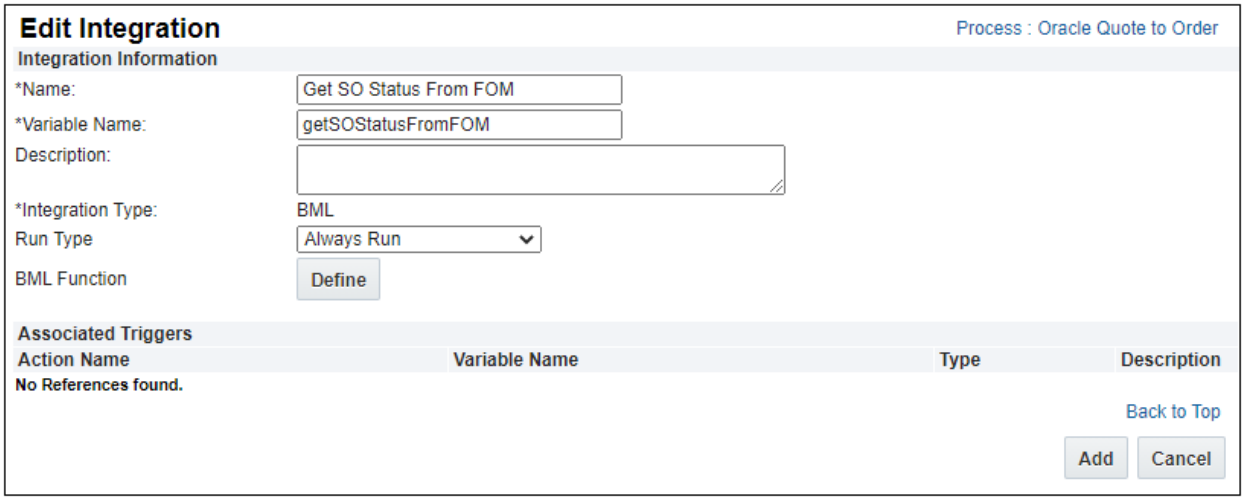
Commerce Integration for Get Sales Order Status from Oracle Order Management

Perform the following steps to create a commerce BML integration to get sales order status from Order Management.

1. Open the Admin Home page.
2. Select **Process Definition** in the Commerce and Documents section.
3. Select **Integrations** from the appropriate commerce process Navigation drop-down, and then and click **List**.
4. Click on **Add**.
5. Select the **BML** option, and then click **Next**.

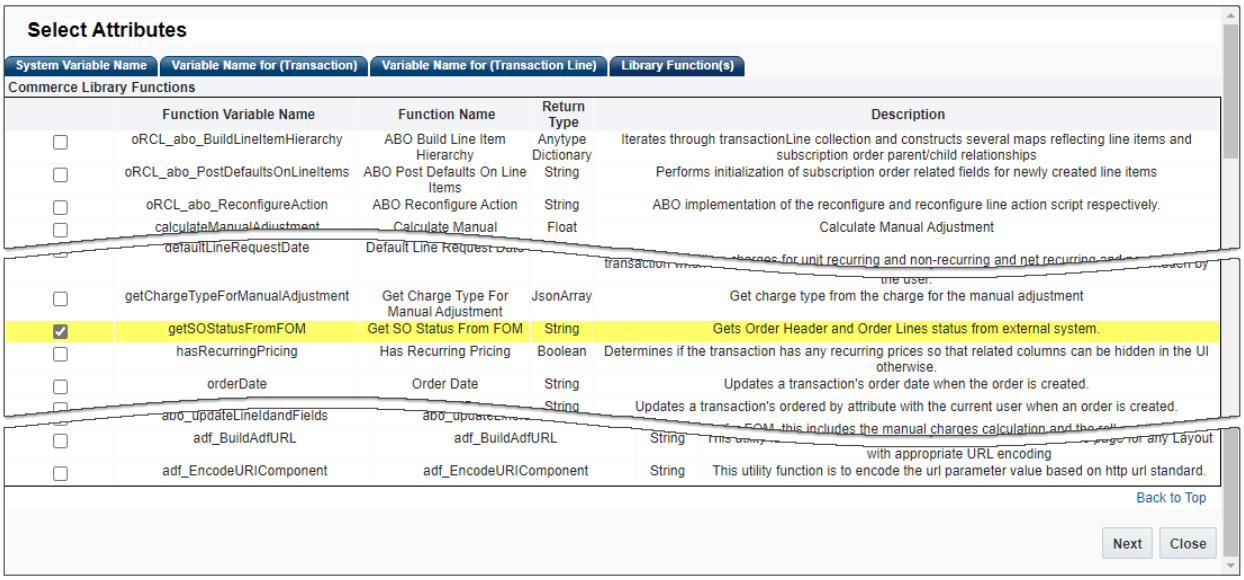


6. Enter the following details:
   - **Name**: Get SO Status From FOM
   - **Variable Name**: getSOStatusFromFOM
   - **Run Type**: Always Run



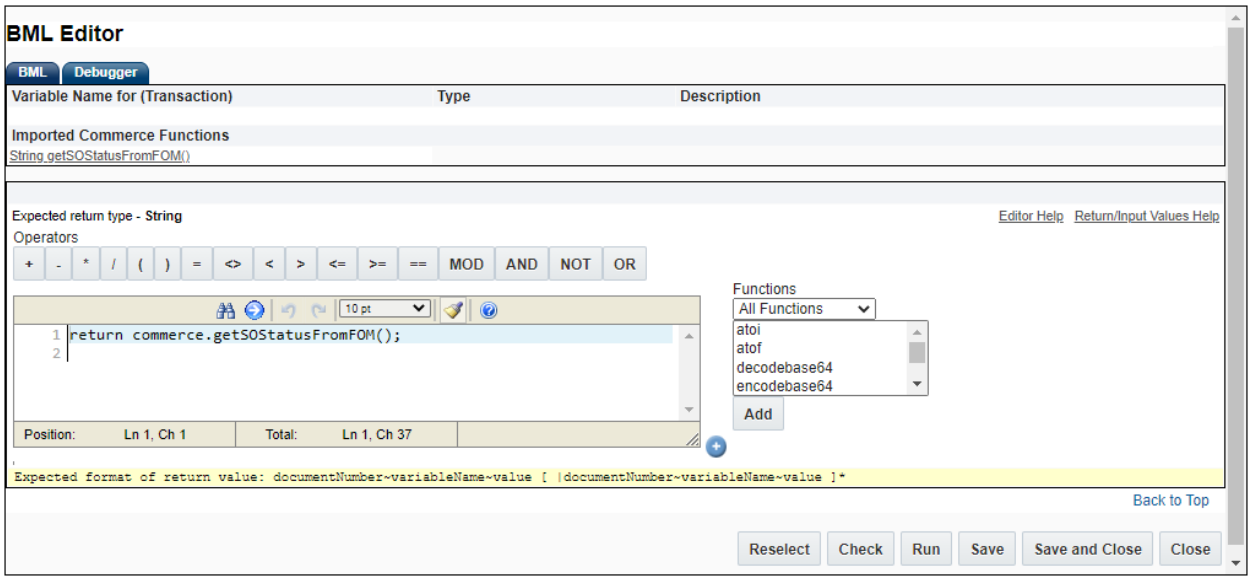
9. Click **Define** to add a BML Function. A new window pops up to Select Attributes.
10. Select the **Library Function(s)** tab.

11. Select the **getSOStatusFromFOM** function, in the Commerce Library Functions section, and then click **Next**.



12. Add the following script in the BML editor.

```
return commerce.getSOStatusFromFOM();
```

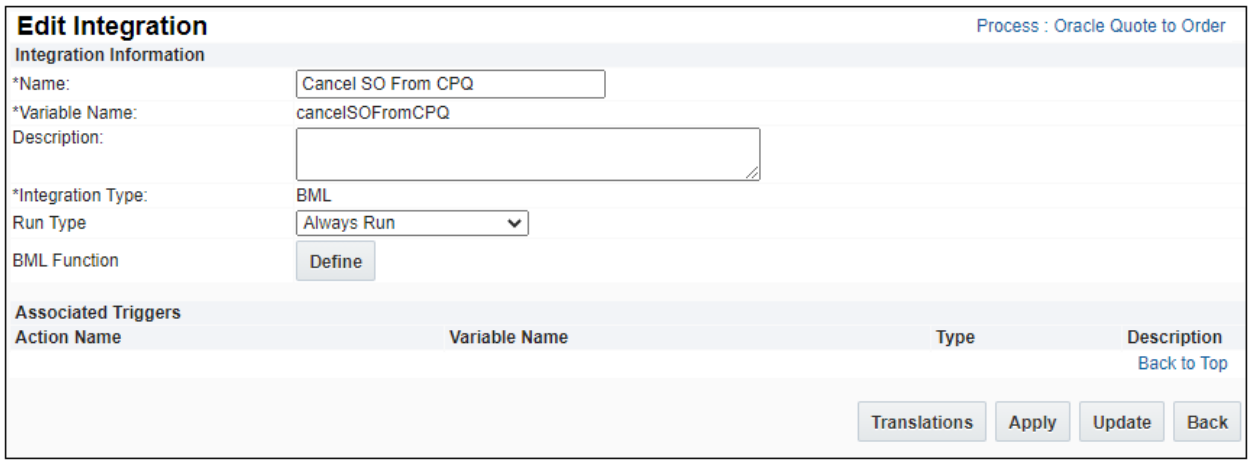


13. Click **Save and Close** to save the BML function.
14. Click **Add**.

Commerce Integration for Cancel Sales Order from CPQ

Perform the following steps to create a commerce BML integration to cancel a sales order status from CPQ.

1. Open the Admin Home page.
2. Select **Process Definition** in the Commerce and Documents section.
3. Select **Integrations** from the appropriate commerce process Navigation drop-down, and then and click **List**.
4. Click on **Add**.
5. Select the **BML** option, and then click **Next**.
6. Enter the following details:
   - **Name**: Cancel SO From CPQ
   - **Variable Name**: cancelSOFromCPQ
   - **Run Type**: Always Run



7. Click **Define** to add a BML Function. A new window pops up to Select Attributes.
8. Select the **Library Function(s)** tab.
9. Select the **cancelOrder** function, in the Commerce Library Functions section, and then click **Next**.
10. Enter the following script in the BML editor.

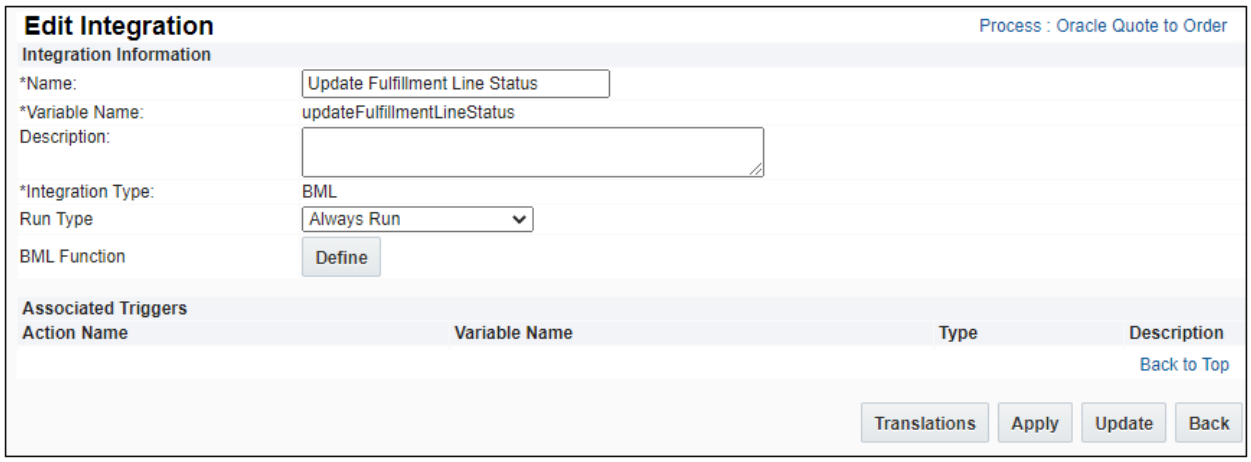
```
return commerce.cancelOrder();
```

11. Click **Save and Close** to save the BML function.
12. Click **Add**.

Commerce Integration for Update Fulfillment Line Status

Perform the following steps to create a commerce BML integration to update fulfillment line status.

1. Open the Admin Home page.
2. Select **Process Definition** in the Commerce and Documents section.
3. Select **Integrations** from the appropriate commerce process Navigation drop-down, and then and click **List**.
4. Click on **Add**.
5. Select the **BML** option, and then click **Next**.
6. Enter the following details:
   - **Name**: Update Fulfillment Line Status
   - **Variable Name**: updateFulfillmentLineStatus
   - **Run type**: Always Run



7. Click **Define** to add a BML Function. A new window pops up to Select Attributes.
8. Select the **Library Function(s)** tab.
9. Select the **updateFulfillmentLineStatus** function, from the Commerce Library Functions section, and then click **Next**.
10. Enter the following script in the BML editor.

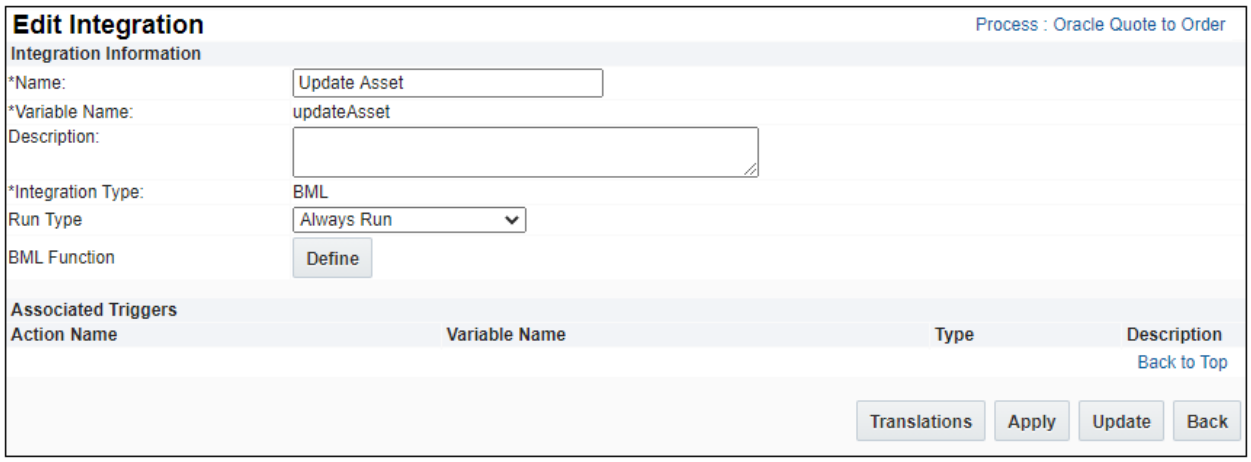
```
return commerce.updateFulfillmentLineStatus();
```

11. Click **Save and Close** to save the BML function.
12. Click **Add**.

Commerce Integration for Update Asset

Perform the following steps to create a commerce BML integration to update an asset.

1. Open the Admin Home page.
2. Select **Process Definition** in the Commerce and Documents section.
3. Select **Integrations** from the appropriate commerce process Navigation drop-down, and then and click **List**.
4. Click on **Add**.
5. Select the **BML** option, and then click **Next**.
6. Enter the following details:
    - **Name**: Update Asset
    - **Variable Name**: updateAsset
    - **Run type**: Always Run



13. Click **Define** to add a BML Function. A new window pops up to Select Attributes.
14. Select the **Library Function(s)** tab.
15. Select the **updateAsset** function, from the Commerce Library Functions section, and then click **Next**.
16. Enter the following script in the BML editor.

```
return commerce.updateAsset();
```

17. Click **Save and Close** to save the BML function.
18. Click **Add**.

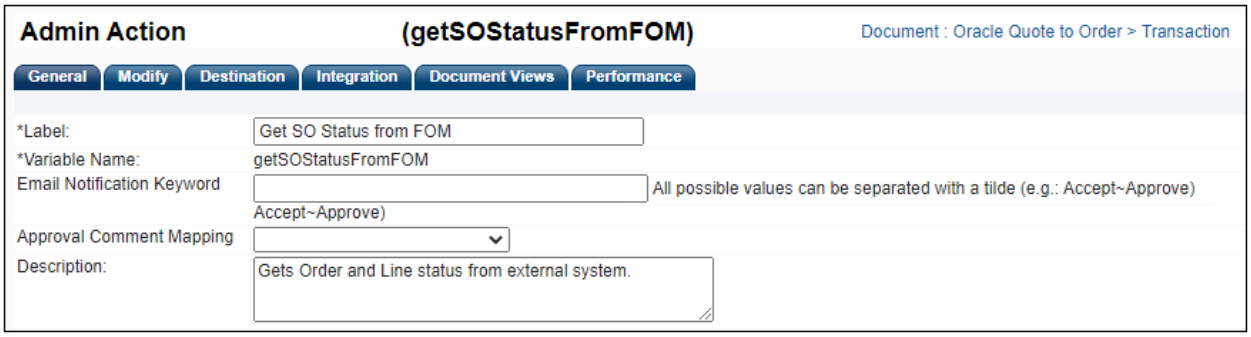
Commerce Actions

Update following Commerce actions.

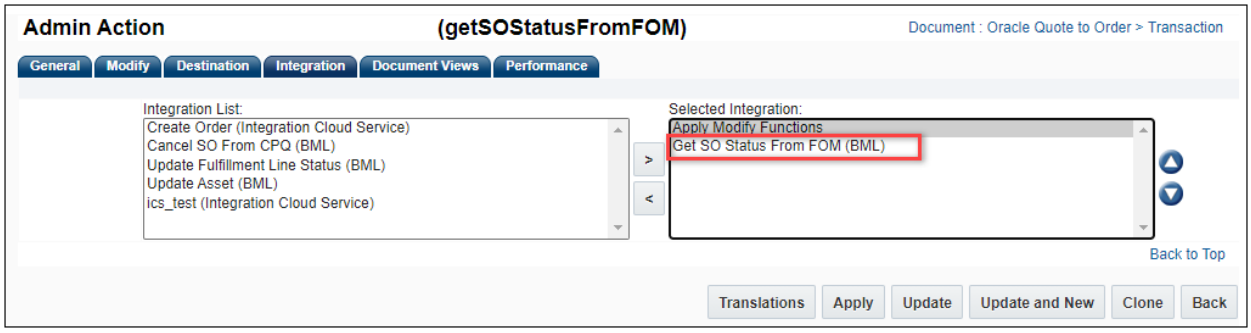
Get Sales Order Status from FOM Action

Perform the following steps to update the Get Sales Order Status from FOM action (getsostatusfromfom).

1. Open the Admin Home page.
2. Select **Process Definition** in the Commerce and Documents section.
3. Select **Documents** from the appropriate commerce process Navigation drop-down, and then and click **List**.
4. Select **Actions** from the main document Navigation drop-down, and then click **List**.
5. Click on the **Get SO Status from FOM** action.



6. Select the **Integration** tab.
7. Move the **Get SO Status from FOM (BML)** integration from Integration List to the Selection Integration window.

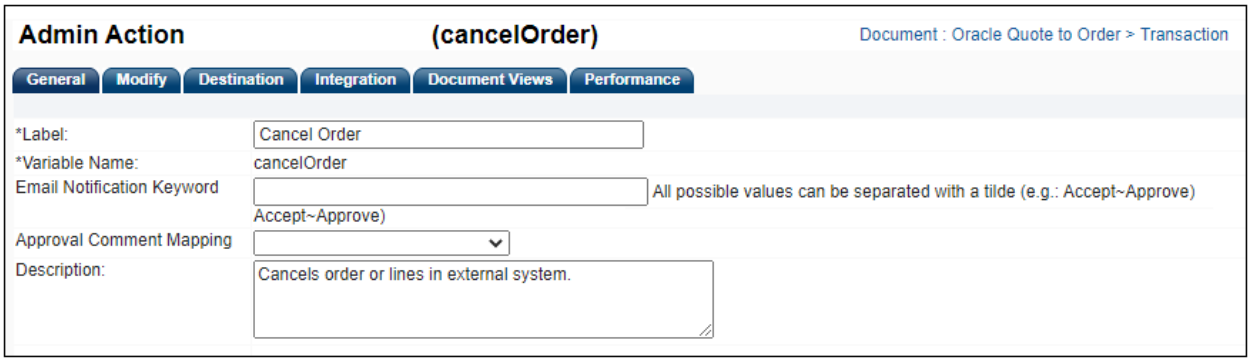


8. Click **Apply** or **Update**.

Cancel Order Action

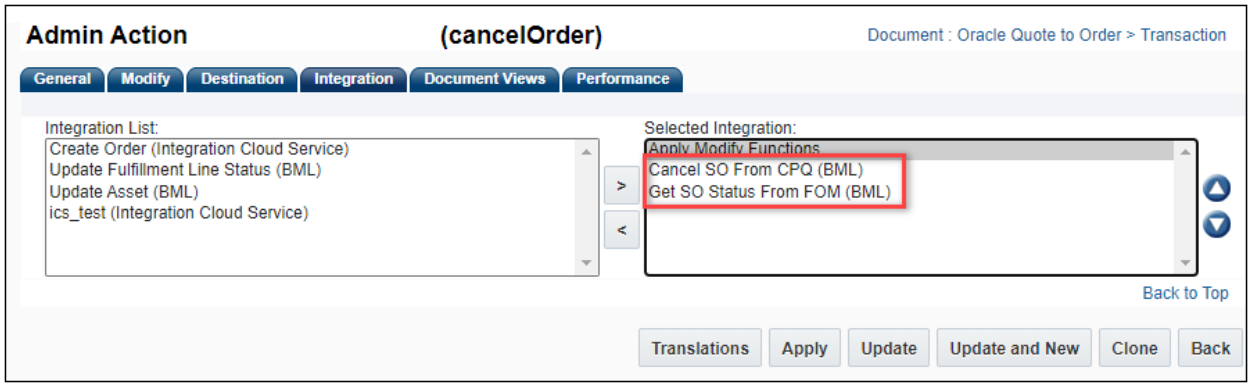
Perform the following steps to update the Cancel Order action (cancelorder).

1. Open the Admin Home page.
2. Select **Process Definition** in the Commerce and Documents section.
3. Select **Documents** from the appropriate commerce process Navigation drop-down, and then and click **List**.
4. Select **Actions** from the main document Navigation drop-down, and then click **List**.
5. Click on the **Cancel Order** action.



9. Select the **Integration** tab.
10. Move the **Cancel SO From CPQ (BML)** and **Get SO Status from FOM (BML)** integrations from Integration List to the Selection Integration window.

> **Note:** If on demand status update functionality is not added user can add only CancelSOFromCPQ.

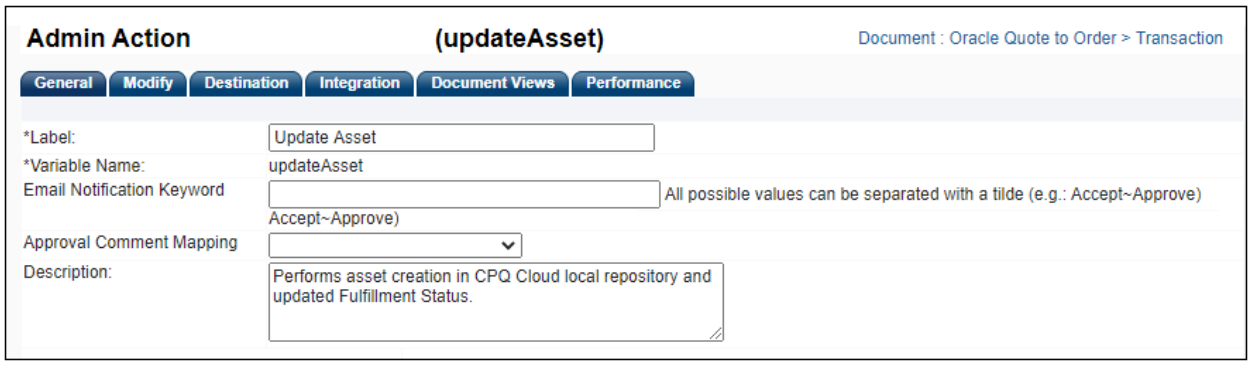


11. Click on **Apply** or **Update**.

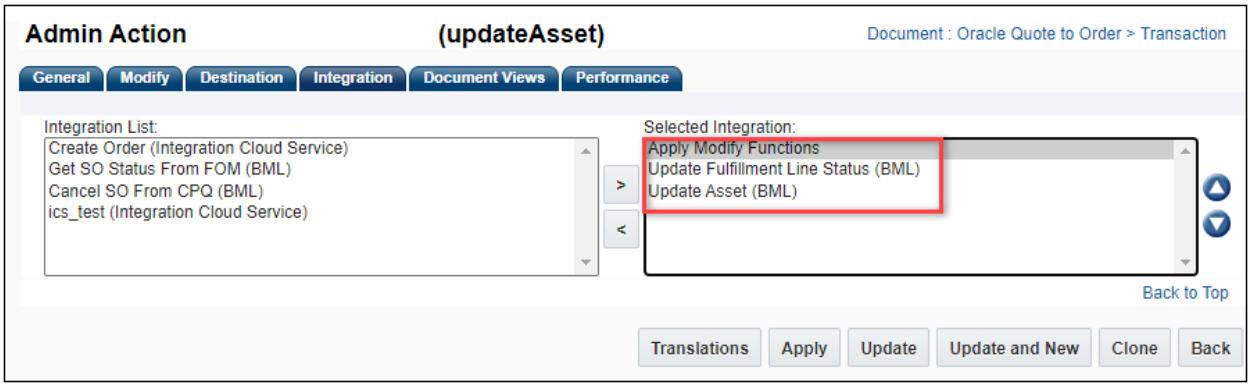
Update Asset Action

Perform the following steps to update the Update Asset action (updateasset).

1. Open the Admin Home page.
2. Select **Process Definition** in the Commerce and Documents section.
3. Select **Documents** from the appropriate commerce process Navigation drop-down, and then and click **List**.
4. Select **Actions** from the main document Navigation drop-down, and then click **List**.
5. Click on the **Update Asset** action.



6. Select the **Integration** tab.
7. Move the **Update Fulfillment Line Status (BML)** and **Update Assets (BML)** integrations from Integration List to the Selection Integration window.
8. Make sure the Selected Integrations are in the following order. If required, use the Up and Down buttons to reorder the integrations.
    1) Apply Modify Functions
    2) Update Fulfillment Line Status
    3) Update Assets

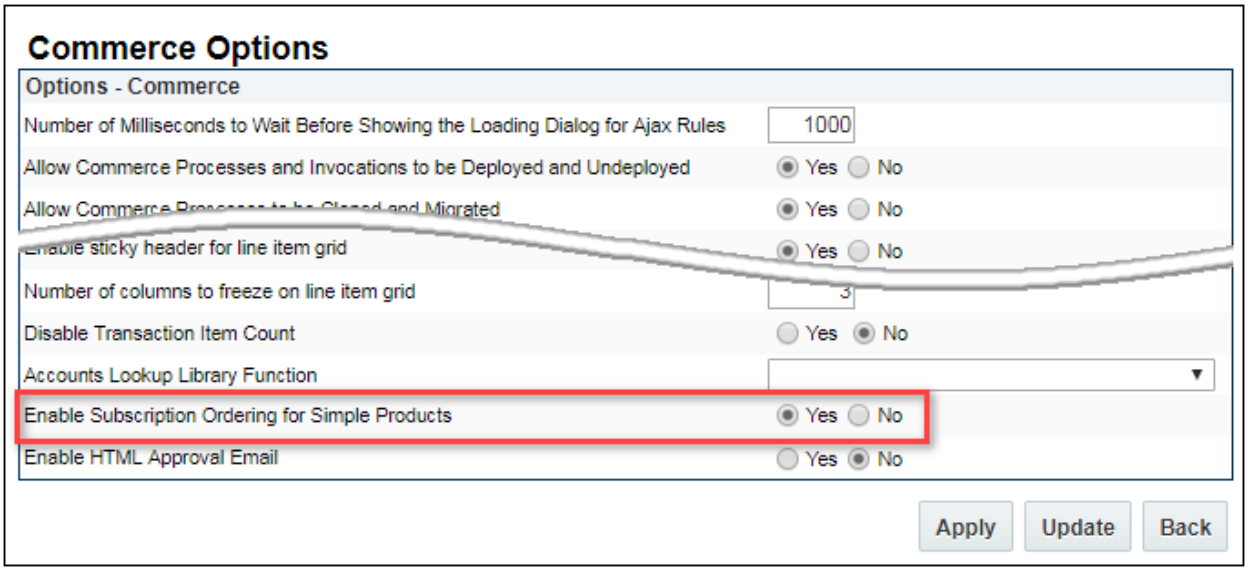


9. Click **Apply** or **Update**.
10. Deploy the commerce process from the **Deployment Center**.

Enable Subscription Ordering for Simple Products (Optional)

Perform the following steps to enable Subscription Ordering for Simple Products.

1. Open the Admin Home page.
2. Select **Commerce Settings** in the Commerce and Documents section.



3. Set the Enable Subscription Ordering for Simple Products option to **Yes**.
4. Click **Apply** or **Update**.

Pricing Setup

The Oracle Order Management package comes with a pricing configuration for the sample BOM for the ATO models. If you are adding new products, complete the following tasks for new products in BOM data:

- Define the price definition for products in Oracle Pricing Data Tables.
- Define tier information, if any, for products in the Pricing Data Tables.
- You can re-use the existing Pricing profile, enhance condition to run pricing for products in BOM, or add a new Pricing profile.

> **Note**: Refer to Pricing Engine Setup for detailed information.

Order Management Setup

Business events must be enabled in Order Management for different Fulfillment Statuses in order to synchronize the order header and line status and to create assets in the CPQ Asset repository. This section provides steps that are required in Order Management for integration with OIC using business events. For more details about the setup, refer to the Oracle Order Management Guide.

Enable Business Events

Manage Trigger Points for Business Events

OIC integration uses the Sales Order Notification business event raised by the Order Management system based on conditions defined as business event triggers.

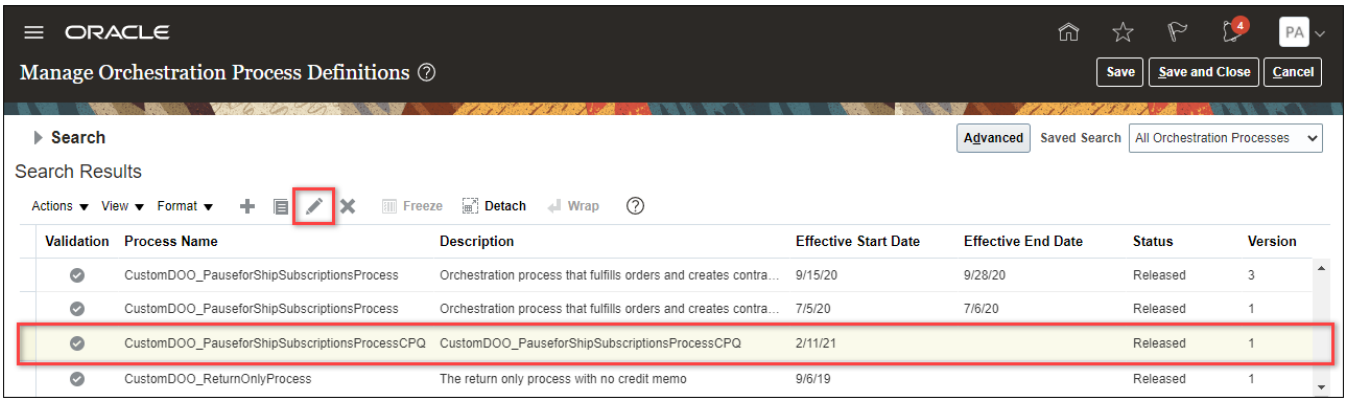
1. Login to your Order Management environment as admin user.
2. In the Navigator, click **Setup and Maintenance**.
3. Select **Order Management** from the Setup drop-down.
4. Search for the **Manage Business Event Trigger Points** task.
5. Click on the **Manage Business Event Trigger Points** task.
6. Select the **Active** option for the following triggers.
    - Fulfillment Line Status Update
    - Fulfillment Line Closed
    - Order Header Status Update

> **Note:** This will raise a business event and send a notification to each subscriber for the selected trigger points.

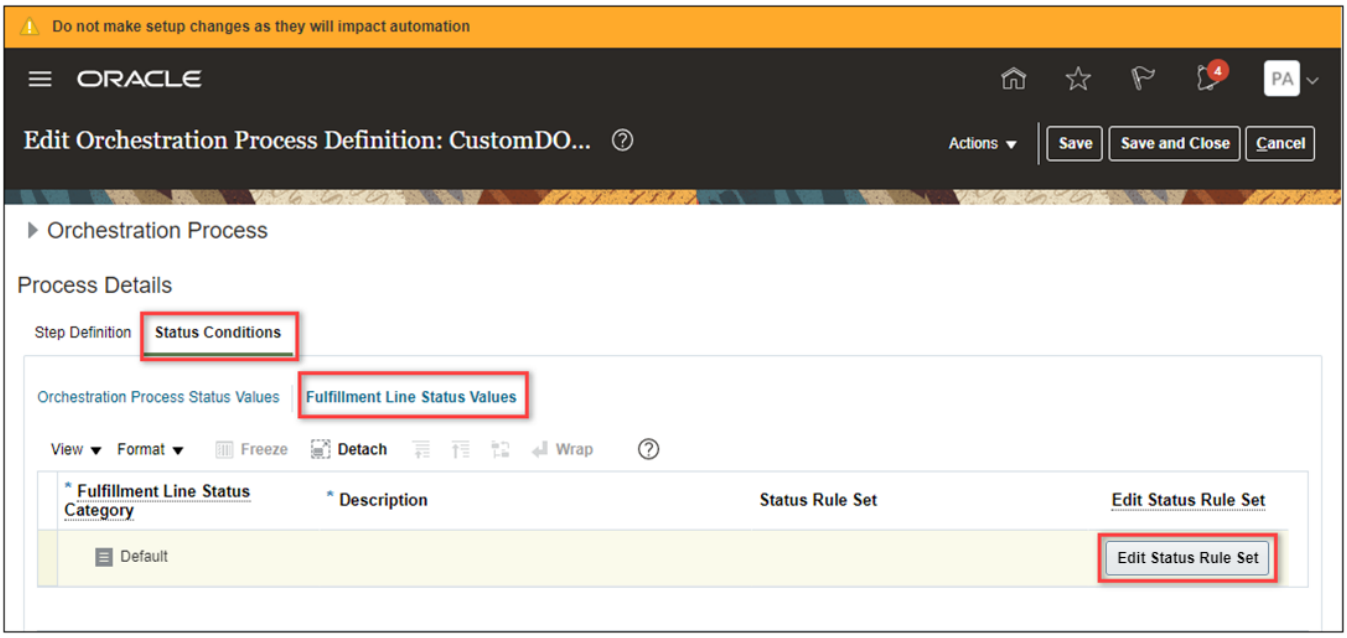
Send Status Updates for Fulfillment Lines

Perform the following steps to send an update to each subscriber when the status changes on a fulfillment line.

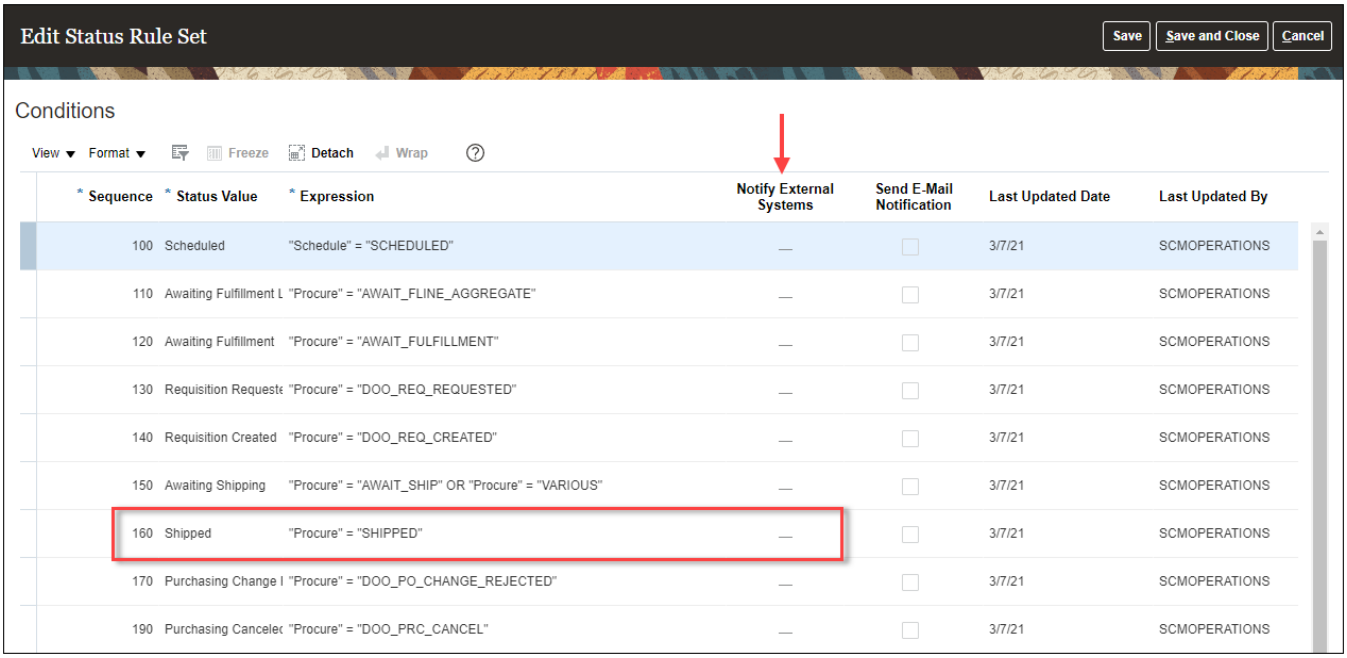
1. Login to your Order Management environment as admin user.
1. In the Navigator, click **Setup and Maintenance**.
2. Select **Order Management** from the Setup drop-down.
3. Search for the **Orchestration Process Definitions** task.
4. Click on the **Orchestration Process Definitions** task.
5. On the Manage Orchestration Process Definitions page, search for the orchestration process that your deployment uses. Each orchestration process controls the status value for each fulfillment line, so you must modify the orchestration process that controls the status value.
6. From search results, click on your process name and click on the pencil icon to edit the process. In the following screenshot we have selected one example process to demonstrate further steps.



7. In the Process Details area, click Status Conditions > Fulfillment Line Status Values > Edit Status Rule Set



8. Edit Status Rule Set page opens with all status conditions listed.
For any status that you want to send out notifications, select the checkbox for '**Notify External Systems**' column.



9. Click on **Save** or **Save and Close**.
10. Deploy
11. Repeat this procedure for each orchestration process in your deployment that updates status values.

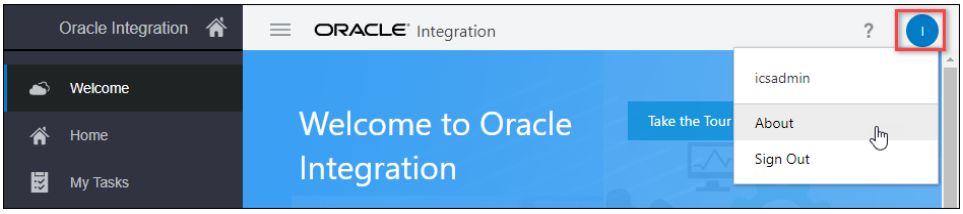
Setting Up Order Management Event Subscriptions

This step is required to subscribe to the Order Management business events using OIC.

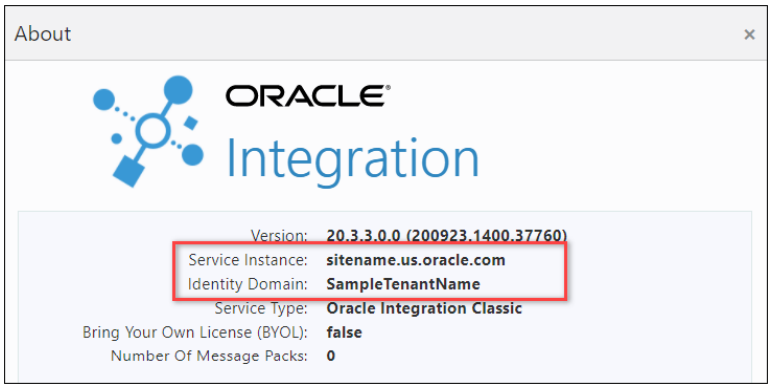
Get the CSF Key

Perform the following steps to acquire the CSF (Credential Store Framework) key.

1. Log in to the OIC site as an administration user.
2. Select **About** from the account drop-down.



3. Record the CSF Key which is a combination of **Identity Domain** and **Service Instance**: <Identity Domain><Service Instance>



Register the CSF Key in Order Management

Perform the following steps to register the CSF key in Order Management

1. Login to https://<OM host name>/soa/composer.
2. Click the **Manage Security** button, located in upper right corner.
3. Enter CSF key recorded from the “Get the CSF Key” procedure.
4. Enter your OIC admin credentials (OIC username and password).
5. Reenter the password for confirmation.
6. Click **Register**.
7. Verify the success message is printed on top.

Installed Oracle CPQ Elements

Installing the CPQ - Order Management package simplifies the implementation of Order Management solution by adding the following elements to the Oracle CPQ RefApp.

- Commerce Attributes
- Commerce Actions
- Library Functions
- Validation Rules
- Hiding Rules

Commerce Attributes

The installation of the Oracle CPQ – Order Management package adds several attributes to the Commerce process and modifies several existing attributes included with the Oracle CPQ Release 21B or later RefApp. For a complete list of the Commerce attributes used by the Order Management solution, refer to Appendix D: Commerce Attributes.

Commerce Actions

The installation of the Oracle CPQ-Order Management package adds or updates the following actions to the Commerce process.

- Create Order
- Cancel Order
- Customer Details
- Get SO Status from FOM
- Save
- Update Asset
- Save (Line)

> **Note:** The Oracle Quote to Order Commerce process is included with the Oracle CPQ RefApp. If customers have chosen to overwrite the Oracle Quote to Order Commerce process with an alternate Commerce process, the Oracle CPQ package adds the actions to the alternate process.

Create Order

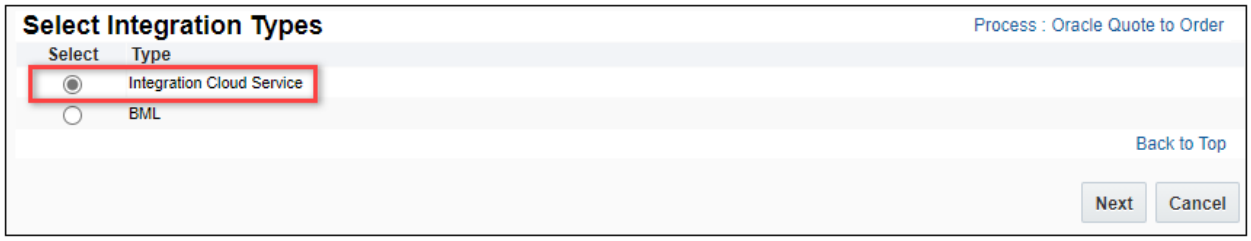
The Create Order action is associated with the Create Order button on the Transaction UI. The action is used to create an order in Order Management. This action is hidden if the transaction doesn't contain any line items.

Create Commerce Integration Cloud Service Integration

Administrators must create an Integration Cloud Service type integration that invokes OIC to initiate the Create Order workflow.

Perform the following steps to create the Integration Cloud Service integration.

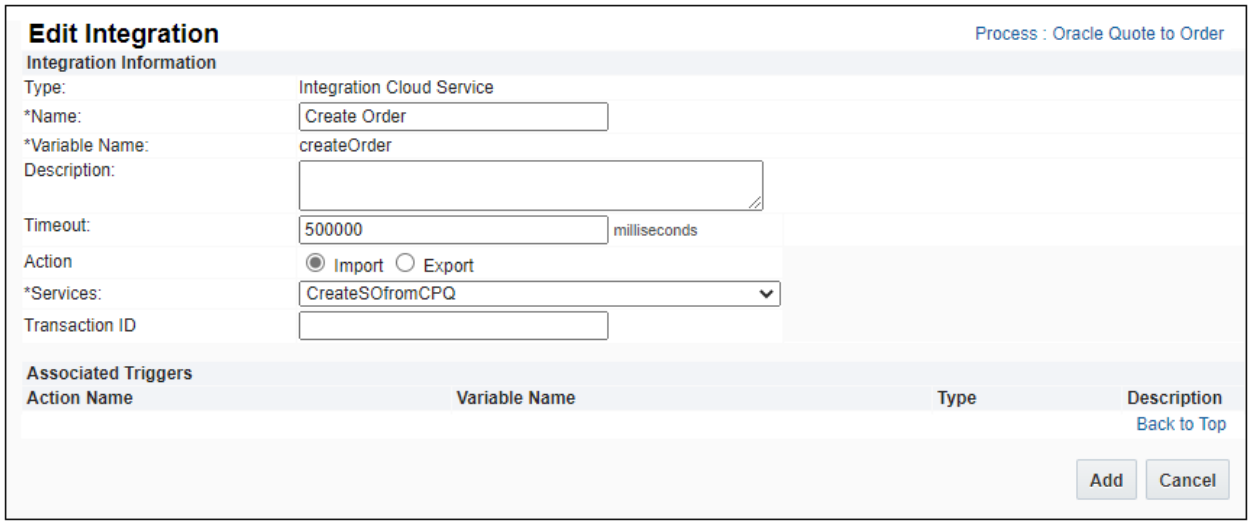
1. Open the Admin Home page.
2. Select **Process Definition** in the Commerce and Documents section.
3. Locate the Commerce process associated with the Order Management integration.
4. Select **Integrations** from the associated Navigation menu, and then click **List**.
5. Click **Add**.
6. Select the **Integration Cloud Service** option, and then click **Next**.



7. Populate the following fields:
   - **Name**: Create Order
   - **Variable Name**: createOrder
   - **Description**: Enter an optional description.
   - **Timeout**: Enter a value between 0 and 600,000 milliseconds.

     > **Note**: The Timeout value is used when calling the CreateSOfromCPQ services. If the service does not respond within the specified time, Oracle CPQ aborts the web service call and throws an error.

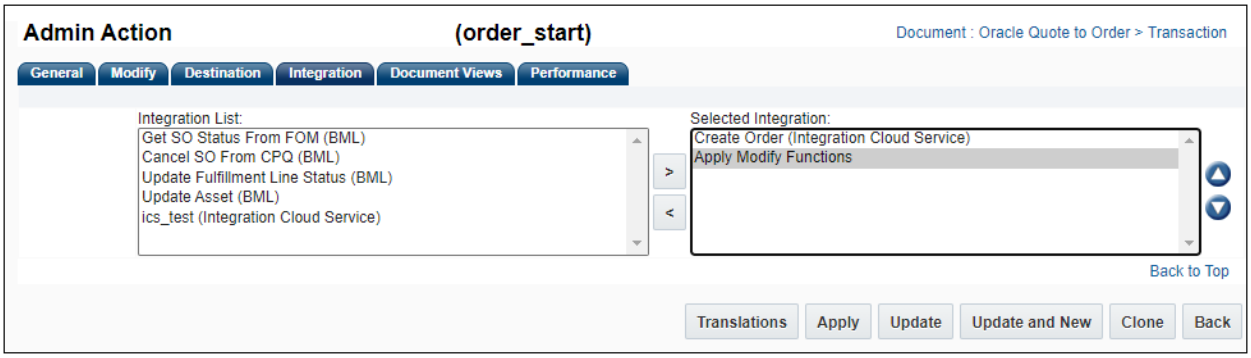
   - **Action**: Select the Import option
   - **Services**: Select - CreateSOfromCPQ
8. Click **Add**.



Add/Edit Create Order Action

After the Create Order integration is created, the integration is added to the Create Order action's integration tab.

1. Open the Admin Home page.
2. Select **Process Definition** in the Commerce and Documents section.
3. Select **Documents** from the appropriate commerce process Navigation drop-down, and then and click **List**.
4. Select **Actions** from the main document Navigation drop-down, and then click **List**.
   - If the Create Order action exists, select the Create Order action, and then proceed to step 8.
   - If the Create Order action does NOT exist, continue to the next step.
5. Click **Add**, and then populate the following fields:
   - **Label**: Create Order
   - **Variable Name**: order_start
   - **Action Type:** Modify
6. Click **Add**.
7. Enter the following **Description:** Creates a sales order for submission to an ERP system.
8. Select **Yes** for the **Show Loading Dialog** option.
9. Select the **Run Validation Before Modify** option.
10. Select the **Define Advanced Modify - After Formulas** option for Advanced Modify - After Formulas.
11. Click **Define Function** for Advanced Modify - After Formulas.
12. Select **Next**, and then enter BML script from Appendix E1: BML – Create Order.
13. Click **Save and Close**.
14. Select the **Integration** tab.
15. Move the **Create Order (Integration Cloud Service)** integration from Integration List to the Selection Integration window.
16. Make sure **Apply Modify Functions** is the following **Create Order (Integration Cloud Service)** in Select Integration. If required, use the Up and Down buttons to reorder the integrations.



17. Select the **Modify** tab, and then select the **Transaction** tab.
    a. Select the radio option for the **Order Date** attribute, and then click **Define**.
       i. Select the **orderDate** function, and then click **Next**.
       ii. Enter the following BML script in the BML editor.

```
return commerce.orderDate();
```

       iii. Click **Save and Close**.

b. Select the radio option for the **Ordered By** attribute, and then click **Define**.

   i. Click **Next**.

   ii. Enter the following BML script in the BML editor.

```
return commerce.orderDate();
```

   iii. Click **Save and Close**.

c. Select the radio option for the **Use Specified Value** for the **Win/Loss Status** attribute.

   i. Enter **WON** in the Use Specified Value**.**

d. Select the **Revert to Default** option for the following attributes:

   - **Last Updated By**
   - **Current Step**
   - **Integration Status**

18. Click **Apply** or **Update**.

Get Sales Order Staus from Oracle Order Management

The **Get SO Status from FOM** action is associated with the Get SO Status from FOM button. This action is used to get the order header and line status from FOM on demand.

Refer to the following setup and implementation:

- Generic Integration for Get Sales Order Status from FOM
- Commerce Integrations for Get Sales Order Status from Oracle Order Management
- Commerce Actions for Get Sales Order Status from FOM
- For the BML script, refer to Appendix E2: BML - Get Sales Order Status from FOM

Cancel Order

The **Cancel Order** action is associated with the Cancel Order button on the Transaction UI, and is used to cancel and entire order or the selected line items.

Refer to the following sections for setup and implementation:

- Generic Integration for Cancel a Sales Order from CPQ
- Commerce Integration for Cancel Sales Orders from CPQ
- Cancel Order Action
- For the BML script, refer to Appendix E3: BML - Cancel Order

Update Asset

The **Update Asset** action is used to create assets in the CPQ Asset Repository based on the order line status. The code associated with this action is executed via the REST API generated for this action. This REST API is invoked through the orchestration logic in UpdateSOStatusFromFOM OIC integration.

- Commerce Integration for Update Fulfillment Line Status
- Update Asset Action

Refer to the following sections for BML scripts:

- Appendix E4: BML - Update Fulfillment Line Status
- Appendix E5 : BML - Update Asset

For Update Asset REST API usage in integration, refer to Appendix J: OIC Integration - UpdateSOStatusFromFOM.

Save

The **Save** action is used to calculate the manual adjustments and the roll up charges for all lines.

- The Save action has a BML function associated with the Advanced Modify - After Formulas.
- For the BML script, refer to Appendix E6: BML – Save.

Save (Line)

The **Save (Line)** action used to calculate the manual adjustments and the roll up charges for one line.

- The Save (Line) action has a BML function associated with the Advanced Modify - After Formulas.
- For the BML script, refer to Appendix E7: BML – Save (Line).

Customer Details

The **Customer Details** action is used to support account integration. When users click the Customer Details tab on the Transaction page, a Customer Company Name field is available. By entering a customer company name and clicking Customer Details, the account fields (Party ID, Invoice to Party id, Bill To Site ID) are populated and mapped to the associated Order Management fields. These field mappings support the creation of a new order in Order Management based on the information provided in the Oracle CPQ Transaction.

- The BML is associated with the Define Advanced Modify - After Formulas function.
- Refer to Appendix E8: BML – Customer Details

Library Functions

The installation of the Oracle CPQ - Order Management package adds several library functions to the Commerce process. Refer to Appendix E: BML to view the BML scripts associated with these library functions.

Validation Rules

Validation Rules are used to validate attribute or field values. They are linked to an action and only run when a specific action is clicked by the user. When the Oracle CPQ - Order Management package is installed, the following Validation Rules are added to the Commerce process.

Main Document - Validate Line Status Values for Creation

Add following main document validation rule is used to validate that the 'Trigger Update Asset for Line Status' attribute is not empty when the Update Asset action is invoked.

Name	Validate Line Status Values For Asset Creation
Variable Name	validateLineStatusValuesForAssetCreation
Status	Active
Linked Actions	Save, Update Asset
Condition	
Condition Type	Always True
Components to Validate	
Action Type	Simple
Components	'Trigger Update Asset for Line Status' main doc attribute
Operator	Equals
Value	(Empty)
Message	This attribute should not be empty.
Message Location	Attribute

Hiding Rules

Hiding Rules tell Oracle CPQ to hide select attributes or actions, when a pre-defined condition is satisfied. They are made up of a condition and an action. The values of the attributes selected as the condition attributes determine the result of the condition, which when True trigger the hiding of the action attributes.

Main Document - Hide Create Order

Hide the **Create Order** action when the order is already created or if no lines are present in the transaction.

Name	Hide Create SO Action
Variable Name	hideCreateSOAction
Status	Active
Condition	
Condition Type	Advanced
Condition Script	Refer to Appendix : BML – Create Order
Components to Hide	
Action Type	Simple
Components	'Create Order' action

Main Document - Hide Cancel Order

Hide the **Cancel Order** action when the order is not created (i.e. if order Key is empty, hide the Cancel order action).

Name	Hide Cancel Order Button
Variable Name	hideCancelOrderButton
Status	Active
Condition	
Condition Type	Simple
Condition 1: Attribute	Order Key
Condition 1: Operator	Equals
Condition 1: Value	(Empty)
Row grouping	1
Components to Hide	
Action Type	Simple
Components	'CancelOrder' action 'Cancel Reason' main document attribute

Main Document - Hide SO Status Action

Hides the **Get SO Status From FOM** action when the order is not created (i.e. if order Key is empty, hide the Get SO Status From FOM action).

Name	Hide SO Status Action
Variable Name	hideSOStatusAction
Status	Active
Condition	
Condition Type	Simple
Attribute	Order Key
Operator	Equals
Value	(Empty)
Components to Hide	
Action Type	Simple
Components	'Get SO Status from FOM' action

Oracle CPQ Field Setup

Oracle CPQ administrators must setup fields in the CPQ environment and obtain the values for these fields from the Order Management administrator.

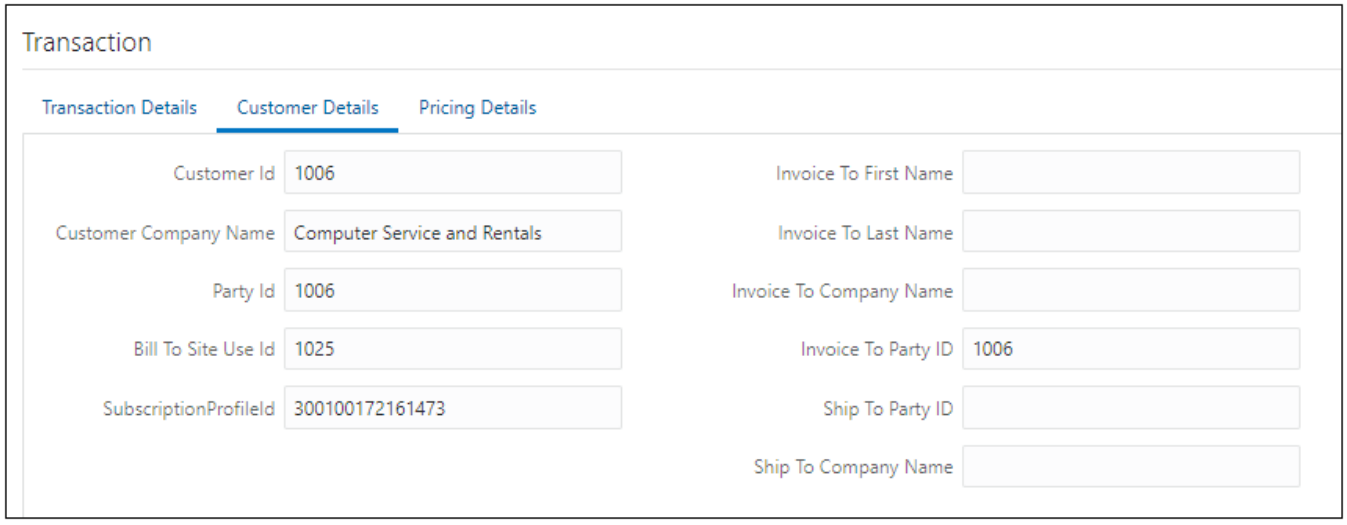
Business Unit ID Field

The Business Unit Id, or organization Id, uniquely identifies a Fusion site. Administrators must setup this field in Oracle CPQ on a per-site basis. For new sites, configure the field in Fusion. The Commerce process has a “Buld” attribute at the Transaction level. Administrators must obtain the value for this field from the Order Management administrator and populate the attribute in the Transaction accordingly.

Account Fields

The account information provided during order creation is used for order billing purposes. A sales user can obtain the account information for a customer by entering a customer company name and clicking **Customer Details**. The fields listed the following are populated and are mapped to the associated Order Management fields. These field mappings support the creation of a new order in Order Management based on the information provided in the CPQ Oracle Transaction. The fields should be added to the Customer Details tab of the Transaction layout.

- **Invoice To Party ID** OR **Party ID**: BillToCustomer-AccountNumber in Order Management
- **Invoice To Company Name** OR **Customer Company Name**: BuyingPartyName in Order Management
- **Customer ID**: Associated with the _customer_id attribute in CPQ for ABO use.



Part Custom Fields

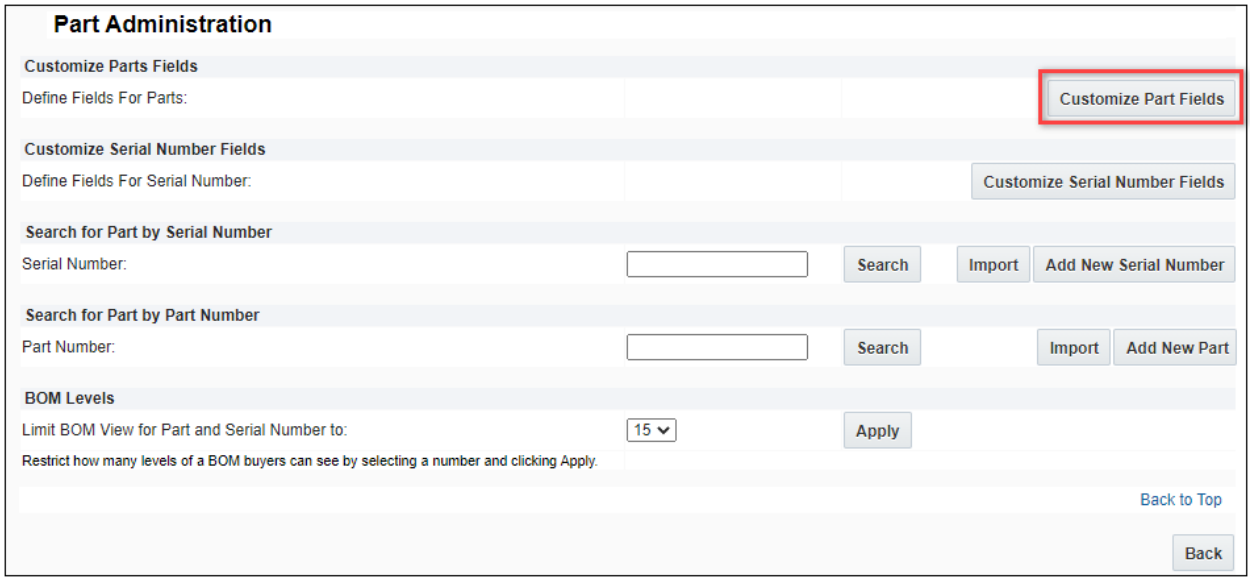
`part_custom_field9` will be used in the future to identify a subscription product. Subscription products will have value "product" for this field.

Part Field Setup

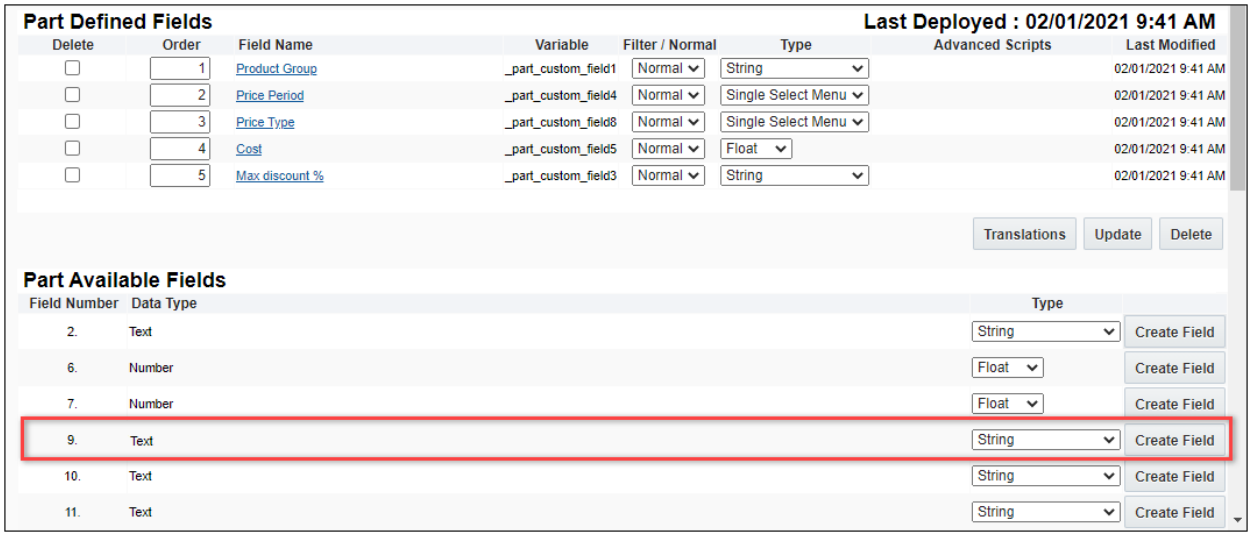
To help identify charge line and product, set up "`part_custom_field9`" as a subscription type configured as shown below. The field is empty for models and standard products. For Subscription type products, enter the `part_custom_field9` value as 'product'.

Perform the following steps to set up this field:

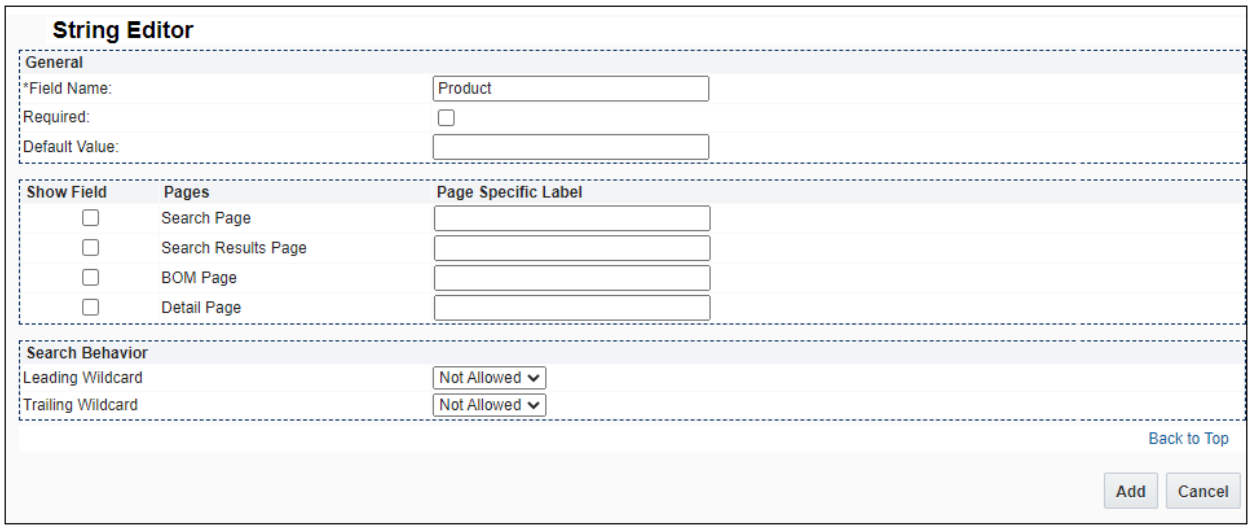
1. Go to Admin Home page.
2. Select **Parts** in the Products section.
3. Click **Customize Parts Fields**.



4. Click **Create Field** for Field Number 9.



5. In **Field Name**, add “Product” as shown in the following image.



6. Click **Add**.
7. You should see the newly added field in red. If so, click **Deploy**.

Layout Fields

This section contains the attributes and actions that need to be added to the Transaction layout.

Refer to Appendix D: Commerce Attributes for attribute information.

The following attributes are required for troubleshooting purposes only.

- Trigger Update Asset for Line Status
- Create Order Rest Response

The following main document attributes are required for Order details.

- Order Number
- Status
- Cancel Reason
- Payment terms
- Customer Company Name
- Business Unit Name
- Customer Id (_customer_id)
- Invoice to Party Id
- Party Id
- Bill to Site Use Id

The following main document actions are required.

- Create Order
- Cancel Order
- Customer Details
- Get SO Status from FOM
- Save
- Save (Line)

The following sub-document attributes are required in Line Item Grid.

- Document Number
- Status
- Fulfillment Status
- Fulfillment Line Id
- Document Number
- UnitPrice_NonRecurring
- UnitPrice_Recurring
- NetPrice_NonRecurring
- NetPrice_Recurring
- Action Code
- Cancelation Reason
- InstanceId
- RootAssetKey
- Calculation Information

Add the following attributes to the sub-document layout.

- Charge Array
- Auto Adjustment Array
- Manual Adjustment Array
- Quantity UOM
- Apply To

Demo Product Setup

Oracle provides an ATO & PTO demo product, which is a sample Configuration in CPQ. Use the demo product to understand the functionality available in Oracle Order management.

All the files related to Demo Product Setup are available in CPQ_FOM_Demo_Setup_Files.zip.

Name	File Name	Description
CPQ_FOM_Demo_Package_21B	CPQ_FOM_Demo_Package_21B_1.zip	Migration package to setup demo Configuration.
Data Table	• datatable_itemDef.zip • datatable_itemMap.zip • datatable_AttrDef.zip • datatable_AttrMap.zip	The BOM data used for Data Tables.
Parts	Part.zip	The following parts as standard item: • AS54888 • AS92888 The following parts ate added ATO demo product (configurable items): • OM_SI_ATO_MODEL • OM_SI_ATO_MODEL1_OC1 • OM_SI_ATO_MODEL1_OC1_OI1 The following parts ate added PTO demo product (configurable items): • OM_SI_PTO_MODEL1 • OM_SI_PTO_MODEL1_OC1 • OM_SI_PTO_MODEL1_OC1_OI1 The following part is One-time subscription item: • OAL_SUBSCRIPTION_OPEN (For future use) The following part is recurring subscription item: • OM_SI_SUBS_1Y_FIXED (For future use)

Note: The packages shown in the above table are optional packages. Install the packages to make the ATO and PTO demo product visible in Oracle CPQ.

Install the BOM Data Table Packages

The BOM Data Table packages add the demo product data into Oracle BOM tables, such as the Oracle_BomItemDef, Oracle_BomItemMap and Oracle_BomAttrMap BOM data tables. When the BOM Data Tables packages are installed, the BOM Configuration for ATO/PTO model is added to the sample Configuration.

Perform the following steps to Install BOM Data Tables package.

1. Download the BOM Data Table packages (i.e. datatable_ItemDef.zip, datatable_ItemMap.zip, datatable_AttrMap.zip, datatable_AttrDef.zip) from My Oracle Support.
2. Open the Admin Home page.
3. Select **Data Tables** under **Developer Tools**. The Data Tables page opens.
4. Create Folder with name **BOM Tables.**
5. Click **Menu** and select **Import** from the drop-down options.
6. Click **Browse**.
7. Select the downloaded zip file datatable_ItemDef.zip.
8. Select **BOM Tables** as the **Destination Folder**.
9. Click **Import**.
10. Repeat steps 5 through 9 for each of the BOM data table files (i.e. datatable_ItemMap.zip and datatable_AttrMap.zip).
11. Select the Oracle_BomItemDef, Oracle_BomItemMap, and Oracle_BomAttrMap data tables, and then select **Deploy** from the Navigation menu.
12. Click **Refresh** to check the status of the uploaded file.
13. Check the log corresponding to the uploaded files for errors.

Install the Parts Package

The Parts package adds parts for the ATO and PTO demo product. When the Parts package is installed, ATO and PTO models are added to the sample Configuration. To view these services in the CPQ sample Configuration, refer to the following examples:

- Appendix A1: Create Order – Standard Item Workflow
- Appendix A2: Create Order – Configurable Item Workflow

> **Note**: Monthly Fee, Consumption Fee, and Activation Fee products appear in the calculatedInfo field in the Line Item Grid.

Perform the following steps to install the Parts package.

1. Download the Parts package (i.e. Part.zip) from My Oracle Support.
2. Open the Admin Home page.
3. Select **Upload** under **Utilities**. The Upload Files List page opens.
4. Click **Browse**.
5. Select the downloaded zip file.
6. Click **Add**.
7. Click **Upload**.
8. Click **Refresh** to check the status of the uploaded file.
9. Check the log corresponding to the uploaded file for errors.

Install the ATO and PTO Demo Product BOM Package

The BOM Package contains the BOM data used for the ATO and PTO demo product.

Perform the following steps to install the BOM Package.

1. Download the BOM package (i.e. Demo_ATO_PTO_BOM_WithOutDataTable_1.zip) from My Oracle Support.
2. Open the Admin Home page.
3. Select **Migration** under **Utilities**.
4. Select **Import Package** from the **Select A Mode** drop-down. The Upload Package dialog opens.
5. Click **Browse** and navigate to the BOM package.
6. Click **Upload**.
7. Click **Migrate**. When the migration completes, check the migration logs for errors.
8. Navigate to the Admin Home page.
9. Select **Catalog Definition** under **Products**. The Supported Products page opens.
10. Click **List**. The Supported Product Families page opens.
11. Click **Add**.
12. Select **ATO** and **PTO-F**.
13. Click **Add**.
14. Select **Deployment Center** from the **ATO** Navigation drop-down, and then click **List**.
15. Click **Deploy.**
16. Click **Back**.
17. Select **Deployment Center** from the **PTO-F** Navigation drop-down, and then click **List**
18. Click **Deploy**.

Verify the Addition of All BOM Parts

Perform the following steps to verify that all of the BOM parts from the BOM package were added to the Oracle CPQ site.

1. Open the Admin Home page.
2. Select **BOM** under **Products**. The BOM Administration Platform opens.
3. Select **BOM Root Item List** under **BOM Products**. The BOM Root Items Administration List page opens.
4. Click on the variable names to verify that parts were added. The BOM Item Tree Administration page opens. Missing parts are shown in red on this page.

> **Notes:**
> - If any parts are missing, add the missing parts to your Oracle CPQ site.
> - Make sure to activate BOM tables from Admin-BOM-BOM Tables and Associate/Map them with respective tables.

Deploy the Home Page

Deploy the Oracle CPQ Home page to make the CPQ FOM Demo Package available on the Oracle CPQ Home page.

Perform the following steps to deploy the Home page.

1. Open the Admin Home page.
2. Select **Home Page** under **Styles and Templates**. The Home Page Setup page opens.
3. Verify that the Product Family definition is accurate in the Home page:
    a. Expand **ATO**.
    b. Select the **Model Punch-in** icon next to **ATO-L**. The Model Punch-ins List opens.
    c. Make sure the model exists in the Model Punch-ins List. If it does not exist, add it.
    d. Repeat steps 3a – 3c for **PTO-F**.
4. Click **Deployment Center** from the Home Page Setup page. The Deployment Center opens.
5. Click **Deploy**.
6. Click **Refresh** to verify the successful deployment of the Home page.

Order Management Pricing Integration

Standard item, configurable item, and subscription item pricing are derived using the pricing engine in CPQ.

Enable Pricing

Perform the following steps in Oracle CPQ to enable Pricing.

1. Open the Admin Home page.
2. Select **General Site Options** under **General**. The Options – General page opens.
3. Set the **Apply only the first matching Pricing profile** option to **No**.
4. Click **Update**. The Admin Home page opens.
5. Select **Commerce Settings** under **Commerce and Documents**. The Commerce Options page opens.
6. Set the **Commerce Pricing Behavior** option to the latest version (e.g. Version 3).
7. Click **Update**

Charges

The following two charges are supported by the Order Management integration solution:

- One Time Fee (e.g. Activation Fee)
- Recurring Fee (e.g. Monthly Fee)

> **Notes:**
> - Pricing for One Time charges and Recurring changes is calculated as unit price x quantity.
> - User BOM rules to model the charge structure for products in the Order Management integration solution.
> - For the Order Management integration solution to work, product items must be added as parts under the Model.

Adjustments

There can be multiple auto adjustments and multiple manual adjustments on One-Time and Recurring charges.

- Auto Adjustments: These are defined in pricing data tables by CPQ Admins. Sales users cannot change these adjustments.
- Manual Adjustments: Filled by sales users at the time of quoting process.

Pricing Engine Setup

The following Order Management Pricing-related configuration is deployed by default with the Oracle Order Management package. Pricing Definitions are stored in the following data tables.

ORCL_PRC_BASE_CHGS

This data table stores the charge-related information. The price definition of each products needs to be added to this table.

- **Charge_Id**: Unique Id for the charge
- **Product**: Product Name of the charge
- **Charge**: Name of the charge
- **AsOFdate**: The date the information provided is valid from.
- **Block_Size**: Block size for the charge
- **Price**: Price for the charge
- **Allowance**: Allowance for the charge
- **Charge_Type**: Type of the charge (ORA_ONE_TIME, ORA_RECURRING)
- **Periodicity**: Periodicity determines the frequency at which billing invoice is available for each charge type.
- **Tiered**: Recurring Usage Charge can be Tiered or Non Tiered (Y or N)
- **TierType**: If the Recurring Usage Charge is designated as Tiered, it can be set as ORA_ALL_TIERS or ORA_HIGHEST_TIER. By default the pricing profile is set to ORA_ALL_TIERS. The default can be modified to ORA_HIGHEST_TIER pricing profile.
- **ApplyTo**: The charge to applied on which type of pricing (price, shipping)
- **Primary_Charge**: Charge is primary or not (true or false).

ORCL_PRC_ADJUSTMENTS

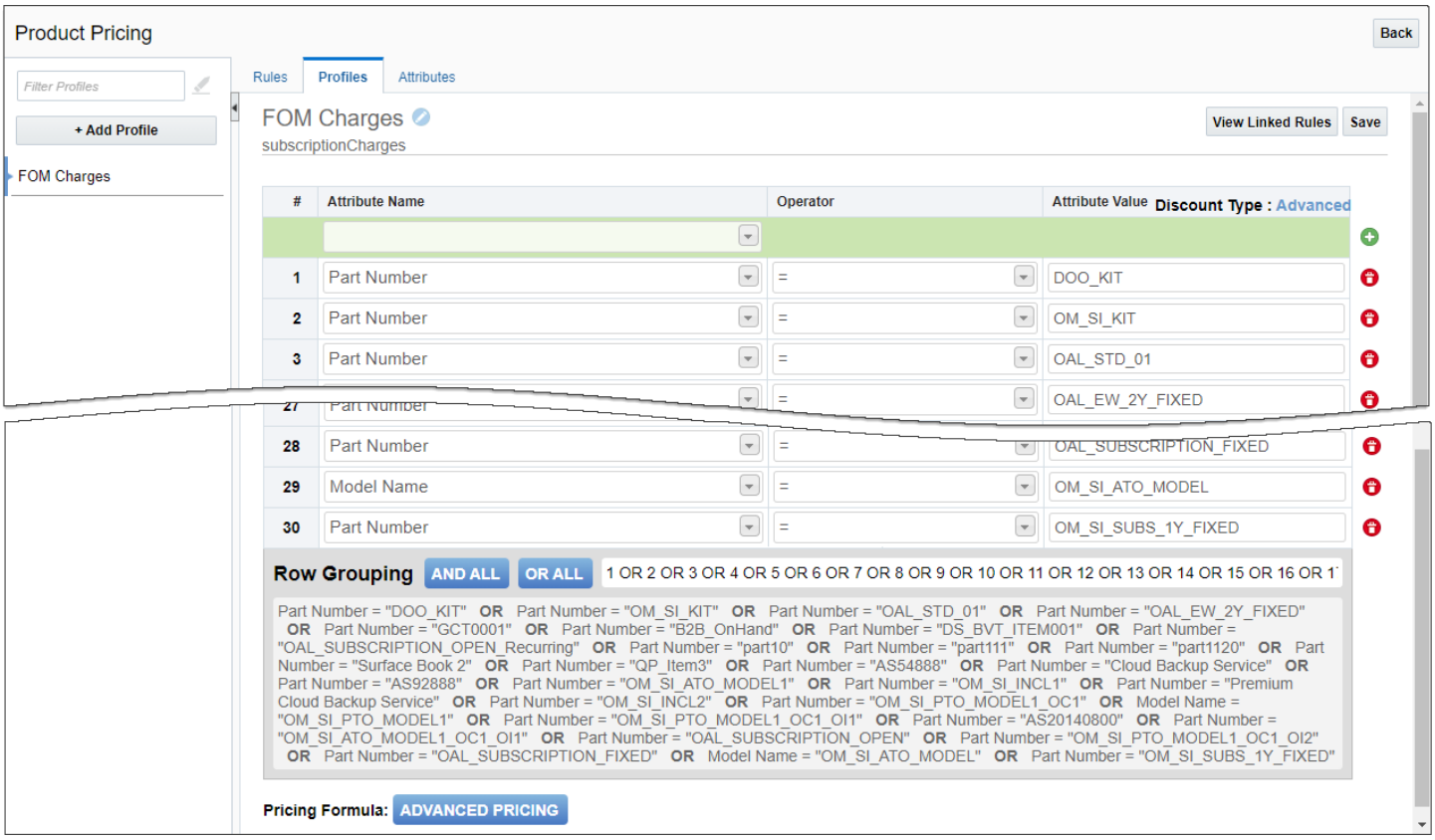
This data table stores the automatic adjustment data for each part. Automatic adjustments that are defined as part of the price definition are added to this data table.

- **Product**: Product Name of the charge
- **Charge_Name**: Name of the charge to apply the adjustment
- **Adjustment_Name**: Name of the adjustment
- **Adjustment_Type**: Type of the adjustment. Supported values are: ORA_DISCOUNT_PERCENT and ORA_DISCOUNT_AMOUNT.
- **Adjustment_Amount**: Actual value to be adjusted.
- **Adjust_Effectivity**: Effectivity of Adjustment. Supported values are: ORA_ALL_TERM, ORA_PERIODS_FROM_START_DT and ORA_PERIODS_BEFORE_END_DT.
  This is not required for one time charges.
- **Charge_Periods**: Used if Effectivity is ORA_PERIODS_FROM_START_DT or ORA_PERIODS_BEFORE_END_DT
- **Adjustment_Basis**: Basis for the adjustment. Supported values are: ORA_LIST_PRICE
- **Charge_Type**: The charge type of the charge on which the adjustment is applied
- **Adjustment_Reason**: Reason of the adjustment. Supported values are: Price match

Pricing Profile Configuration

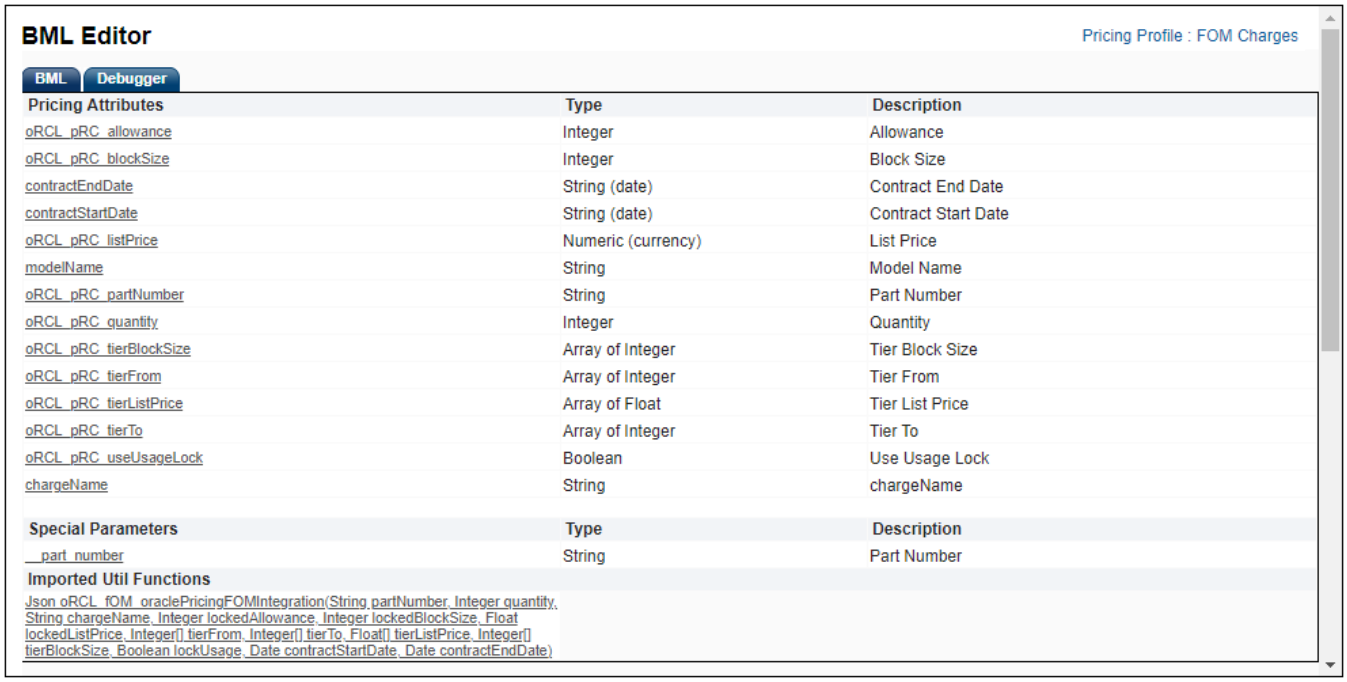
Profile Name: *FOM Charges*. The Order Management Charges pricing profile filters the list of products by pricing profile action.

Add the parts for which you want to see the charges



Profile BML: *Profile Action is associated to Advanced BML*. This BML is associated to the profile action and contains the logic for price calculation. Administrators can customize the BML logic to change the pricing logic.

Profile BML invokes the *oRCL_fOM_oraclePricingFOMIntegration* utility BML. This returns the JSON calculation information that provides all the charge information.



Script:

```
startDate = strtojavadate(contractStartDate , "MM/dd/YYYY HH:mm:ss");
endDate = strtojavadate(contractEndDate , "MM/dd/YYYY HH:mm:ss");
if(oRCL_pRC_partNumber == "" OR isnull(oRCL_pRC_partNumber)) {
  return util.oRCL_fOM_oraclePricingFOMIntegration(modelName,oRCL_pRC_quantity, chargeName,
oRCL_pRC_allowance, oRCL_pRC_blockSize, oRCL_pRC_listPrice, oRCL_pRC_tierFrom, oRCL_pRC_tierTo,
oRCL_pRC_tierListPrice, oRCL_pRC_tierBlockSize, oRCL_pRC_useUsageLock, startDate, endDate);
}
else {
  return util.oRCL_fOM_oraclePricingFOMIntegration(__part_number,oRCL_pRC_quantity, chargeName,
oRCL_pRC_allowance, oRCL_pRC_blockSize, oRCL_pRC_listPrice, oRCL_pRC_tierFrom, oRCL_pRC_tierTo,
oRCL_pRC_tierListPrice, oRCL_pRC_tierBlockSize, oRCL_pRC_useUsageLock, startDate, endDate);
}
```

Pricing Related Utility BML

The following Utility BMLs are used by FOM Pricing. The BML scripts are available in Appendix F: Pricing Related Utility BMLs.

Array Utility BML

The following BML populates calculated pricing information for charges array and the auto adjustment array.
This BML is invoked from the Transaction Line Advanced Default - After Formulas.

- Populate Charges: Populates the charge information for charge arrays

Charges Array
Charges array set is present in line level it is used to hold the charges information of an item. For an item there can be any number of charges. The following attributes are added in the array set.

Attribute Name	Variable Name	Description	Expected Value
Charge Name	`oRCL_chargeName`	Name of the charges	Example Values • Sale Price • Recurring Sale Price • Activation Fee • Maintenance Fee • Freight
Allowance	`oRCL_allowance`	Allowance for the charges	
Unit Price	`oRCL_unitPrice`	Unit Price of the item. For recurring charge, it represents price per period.	
List Price	`oRCL_listPrice`	List Price of the item • For One time charge: listPrice = unitPrice * quantity • For Recurring charges: listPrice = unitPrice * periods * quantity	
Unit List Price	`oRCL_unitListPrice`	Unit List Price of the item • For One time charge: listPrice = unitPrice • For Recurring charges: listPrice = unitPrice * periods	
Charge Type	`oRCL_chargeType`	Charge Type of the item	• ORA_ONE_TIME • ORA_RECURRING
Periodicity	`oRCL_periodicity`	Periodicity if the charge type is recurring	• DAY [DY] • WEEK [0zH] • MONTH[0zG] • QUARTER[0zF] • YEAR[0zE]
BlockSize	`oRCL_blockSize`	Block size for the charge	
Tiered	`oRCL_tiered`	Recurring Usage Charge can be Tiered (Y) or Non Tiered (N) **Note**: Recurring Usage is not supported in Oracle CPQ 21B	

Attribute Name	Variable Name	Description	Expected Value
Tier Type	oRCL_tierType	If the Recurring Usage Charge is designated as Tiered, it can be set as ORA_ALL_TIERS or ORA_HIGHEST_TIER. • By default the pricing profile is set to ORA_ALL_TIERS. • The default can be modified to ORA_HIGHEST_TIER in the pricing profile. **Note**: Recurring Usage is not supported in Oracle CPQ 21B	
Net Price	oRCL_netPrice	Net Price of the item • For One time charges: NetPrice = (unitPrice – adjustments) * quantity • For Recurring charges: NetPrice = (unitPrice – adjustments) * periods * quantity	
Sequence Number	oRCL_chargeSequenceNumber	Sequence number of the charges	
Unit Net Price	oRCL_unitNetPrice	Unit Net Price of the item • For One time charges: NetPrice = (unitPrice – adjustments) • For Recurring charges: NetPrice = (unitPrice – adjustments) * periods	
Apply To	oRCL_applyTo	On which type of price it is applied	Price Shipping
Unit Auto Adjustment	oRCL_autoAdjustment	Calculates the unit auto adjustment unitAutoAdjustment = Sum of auto adjustments	
Unit Manual Adjustment	oRCL_manualAdjustment	Calculates the unit manual adjustment unitAutoAdjustment = Sum of manual adjustments	
Periods	oRCL_periods	Periods is considered only for recurring charges. Period is derived based the contract start date and contract end date and the periodicity	
Total Manual Adjustment	oRCL_totalManualAdjustment	Total Manual Adjustment of the item • For One time charges: total Manual Adjustment = sum of the manual adjustment * quantity • For Recurring charges: total Manual Adjustment = sum of the manual adjustment * quantity * periods	
Total Auto Adjustment	oRCL_totalAutoAdjustment	Total Auto Adjustment of the item • For One time charges: total Auto Adjustment = sum of the auto adjustment * quantity • For Recurring charges: total auto Adjustment = sum of the auto adjustment * quantity * periods	
Primary Charge	oRCL_primaryCharge	Primary charge is used to determine whether the charge is primary or not.	true / false

Auto Adjustment Array

Auto Adjustment array is used to hold the auto adjustments provided for a particular charge type. The following attributes are added in the array set.

ATTRIBUTE NAME	VARIABLE NAME	DESCRIPTION	EXPECTED VALUE
Auto Adjustment Sequence Number	`oRCL_auto_adjustmentSequence_l`	Sequence number of the auto adjustment	
Auto Adjustment Type	`oRCL_auto_adjustmentType_l`	Type of adjustment	• Discount Percent [ORA_DISCOUNT_PERCENT] • Discount Amount [ORA_DISCOUNT_AMOUNT]
Auto Adjustment Name	`oRCL_auto_adjustmentName_l`	Name of the Auto Adjustment	
Auto Adjustment Charge Name	`oRCL_auto_adjustmentChargeName_l`	Name of the charge on which the auto adjustment is applied	Example Values • Sale Price • Recurring Sale Price • Activation Fee • Maintenance Fee • Freight
Auto Adjustment Effectivity	`oRCL_auto_adjustmentEffectivity_l`	Auto adjustment effectivity is required only for recurring items	• All Term [ORA_ALL_TERM] • Period From Start Date [ORA_PERIODS_FROM_START_DT] • Period From End Date [ORA_PERIODS_FROM_END_DT]
Auto Adjustment Period	`oRCL_auto_numberOfPeriods_l`	Auto Adjustment period is required to be filled for effectivity with Period From Start Date and Period From Ends Date. This determines for how much period the adjustment is applied	
Auto Adjustment Value	`oRCL_auto_adjustmentValue_l`	Adjustment Value	
Auto Adjustment Basis	`oRCL_auto_adjustmentBasis_l`	On which basis the adjustment is applied	List Price [ORA_LIST_PRICE]
Auto Adjustment Reason	`oRCL_autoAdjustmentReason`	Adjustment Reason	‘Price match’
Auto Adjustment Charge Type	`oRCL_autoAdjustmentChargeType_l`	On which charge type adjustment is applied	• One Time [ORA_ONE_TIME] • Recurring [ORA_RECURRING]

Manual Adjustment Array

Manual Adjustment array is used to hold the manual adjustments provided for a particular charge type. The following attributes are added in the array set.

ATTRIBUTE NAME	VARIABLE NAME	DESCRIPTION	EXPECTED VALUE
Manual Adjustment Sequence	`oRCL_manual_adjustmentSequence_l`	Sequence number of the manual adjustment array	
Manual Adjustment Name	`oRCL_manual_adjustmentName_l`	Manual adjustment name	
Manual Adjustment Type	`oRCL_manual_adjustmentType_l`	Type of adjustment	• Discount Percent ORA_DISCOUNT_PERCENT] • Discount Amount [ORA_DISCOUNT_AMOUNT]
Manual Adjustment Basis	`oRCL_manual_adjustmentBasis_l`	On which basis the adjustment is applied	List Price [ORA_LIST_PRICE]
Manual Adjustment Effectivity	`oRCL_manual_adjustmentEffectivity_l`	Auto adjustment effectivity is required only for recurring items	• All Term [ORA_ALL_TERM] • Period From Start Date [ORA_PERIODS_FROM_START_DT] • Period From End Date [ORA_PERIODS_FROM_END_DT]
Manual Number Of Period	`oRCL_manual_numberofPeriods_l`	Auto Adjustment period is required to be filled for effectivity with Period From Start Date and Period From Ends Date. This determine for how much period the adjustment is applied	
Manual Adjustment Value	`oRCL_manual_adjustmentValue_l`	Adjustment Value	
Manual Adjustment Charge Name	`oRCL_manual_adjustmentChargeName_l`	Name of the charge on which the auto adjustment is applied	Example Values • Sale Price • Recurring Sale Price • Activation Fee • Maintenance Fee • Freight
Manual Adjustment Reason	`oRCL_manual_adjustmentReason_l`	Adjustment Reason	'Price match'
Manual Adjustment Charge Type	`oRCL_manual_adjustmentChargeType_l`	On which charge type adjustment is applied	• One Time [ORA_ONE_TIME] • Recurring [ORA_RECURRING]

Notes:

- Though there are certain attributes with Tier pricing in Charges but these are not supported in this release.
- Recurring Usage Charge type is not supported.

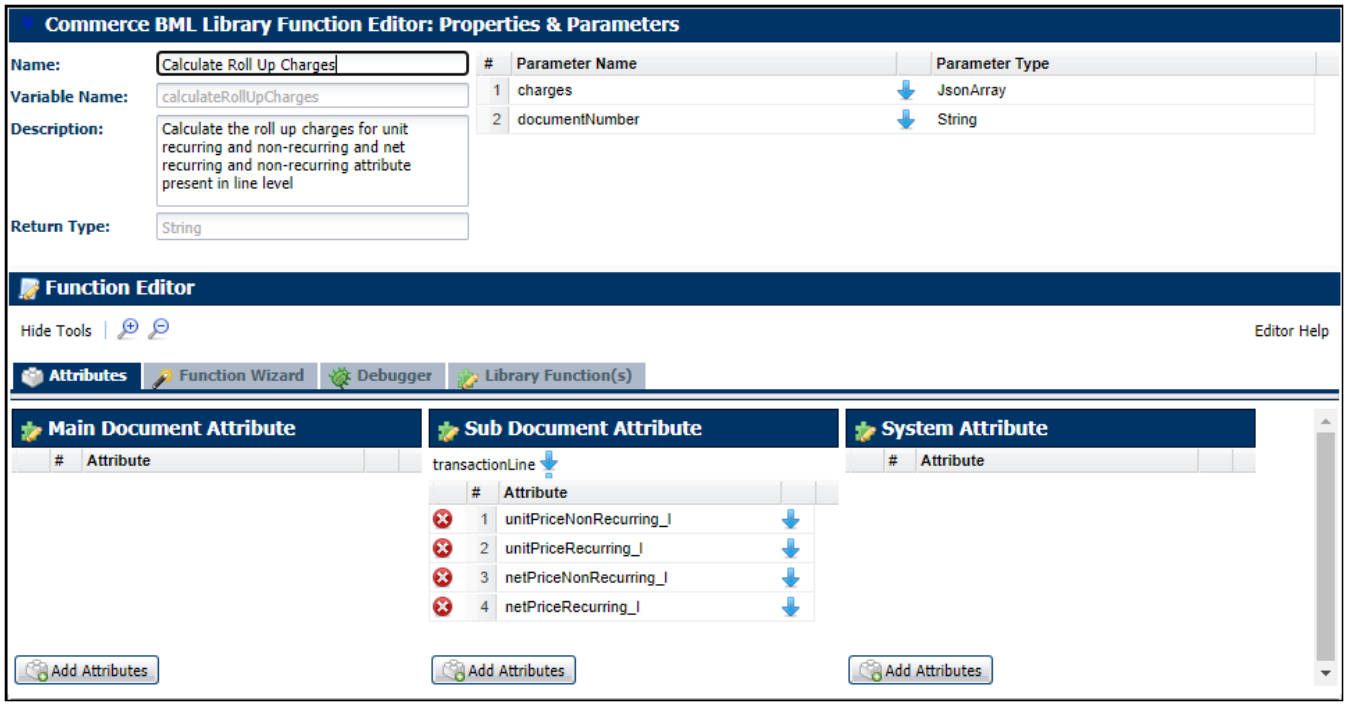
However customer can extend the functionality by making use of these attributes.

Roll Up Charges

Roll Up charges are sum of the unit recurring and non-recurring and net recurring and non-recurring charges. The total is displayed in the line level attribute.
The attributes used for calculating the roll up charges.

Attribute Name	Variable Name	Description
UnitPrice_NonRecurring	`unitPriceNonRecurring_l`	Holds the sum of unit price of non-recurring charges
UnitPrice_Recurring	`unitPriceRecurring_l`	Holds the sum of unit price of recurring charges
NetPrice_NonRecurring	`netPriceNonRecurring_l`	Holds the sum of net price of non-recurring charges
NetPrice_Recurring	`netPriceRecurring_l`	Holds the sum of net price of recurring charges

Roll Up charges are calculated when the transaction is loaded and when Save action is performed. The following BML is used to calculate the Roll Up charges.



Script:

```
returnString = "";
document_number = documentNumber;
unitPriceNonRecurring = 0.0;
netPriceNonRecurring = 0.0;
unitPriceRecurring = 0.0;
netPriceRecurring = 0.0;
chargesSize = string[jsonarraysize(charges)];
count = 0;
for ch in chargesSize {
  charge = jsonarrayget(charges,count,"json");
  chargeType = jsonget(charge,"chargeType");
  unitPrice = jsonget(charge,"unitPrice", "float");
  netPrice = jsonget(charge,"netPrice", "float");
  if(chargeType == "ORA_ONE_TIME") {
    unitPriceNonRecurring = unitPriceNonRecurring + unitPrice;
    netPriceNonRecurring = netPriceNonRecurring + netPrice;
  } else {
    unitPriceRecurring = unitPriceRecurring + unitPrice;
    netPriceRecurring = netPriceRecurring + netPrice;
  }
  count = count + 1;
}
returnString = returnString +"|"+ document_number+"~unitPriceNonRecurring_l~" +
string(round(unitPriceNonRecurring,2));
returnString = returnString +"|"+ document_number+"~netPriceNonRecurring_l~" +
string(round(netPriceNonRecurring,2));
returnString = returnString +"|"+ document_number+"~unitPriceRecurring_l~" +
string(round(unitPriceRecurring,2));
returnString = returnString +"|"+ document_number+"~netPriceRecurring_l~" +
string(round(netPriceRecurring,2));
return returnString;
```

Oracle CPQ Account Integration

This section identifies the library functions that support account integration and the manual changes that administrators must make to the INT_SYSTEM_DETAILS Data Table and the INT_SYSTEM_TEMPLATES Data Table to support account integration.

> **Note:** For more details about Account Integration with Oracle Customer Data Management (CDM), refer to CPQ-CDM Integration Whitepaper on My Oracle Support under CPQ to Fusion Financials Integration (Doc ID 2012010.1).

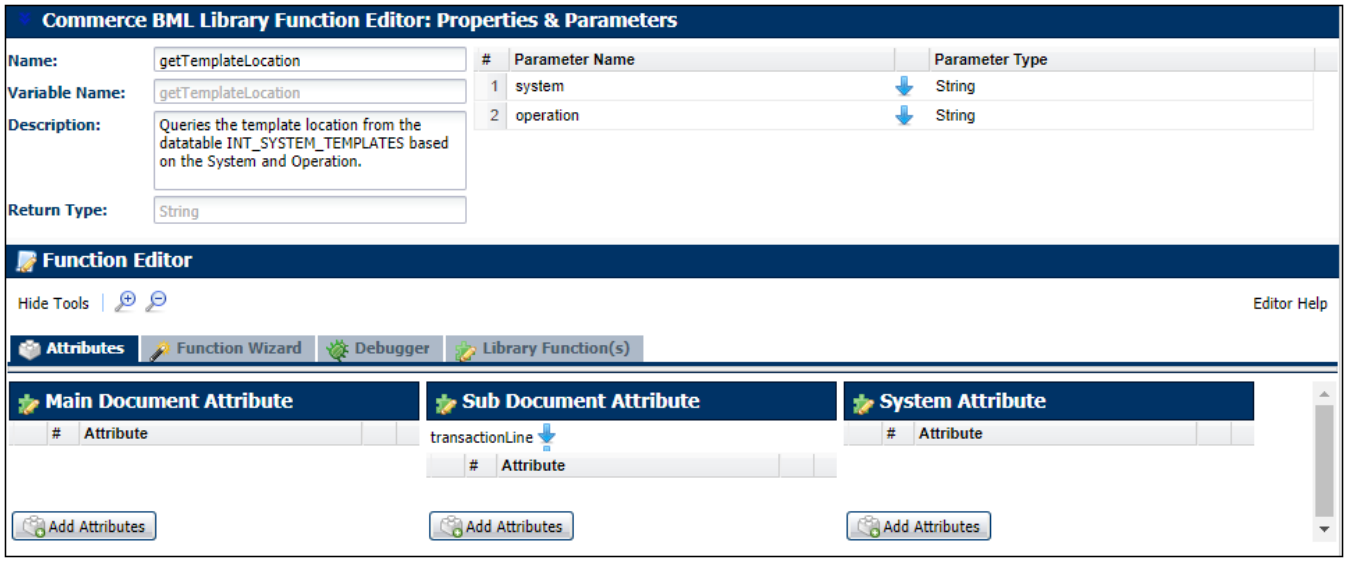
Library Functions

The library functions within this section support the CDM integration by retrieving account details.

String getTemplateLocation(String system, String operation)

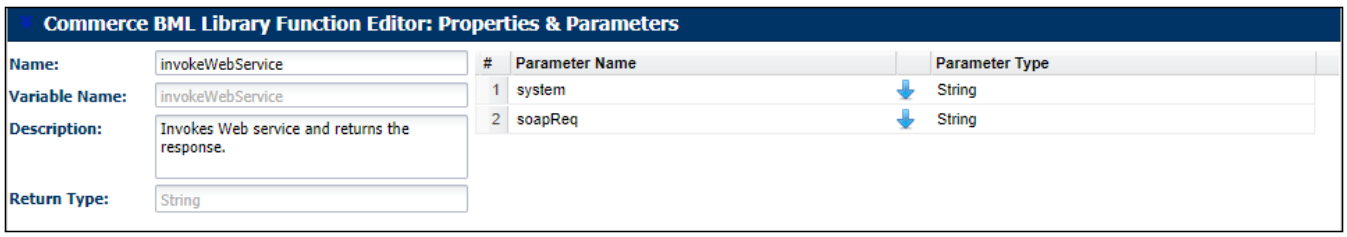
Queries the template location from the INT_SYSTEM_TEMPLATES Data Table based on the system and operation.
This is a Commerce library function.

The Return Type, input information, and attributes used by this library function are shown in the following image.



String invokeWebService(String system, String soapReq)

This is a Commerce library function that invokes Web Services and returns the response.
The Return Type, input information, and attributes used by this library function are shown in the following image.



Refer to Appendix H: Miscellaneous Commerce Library Functions for BML scripts.

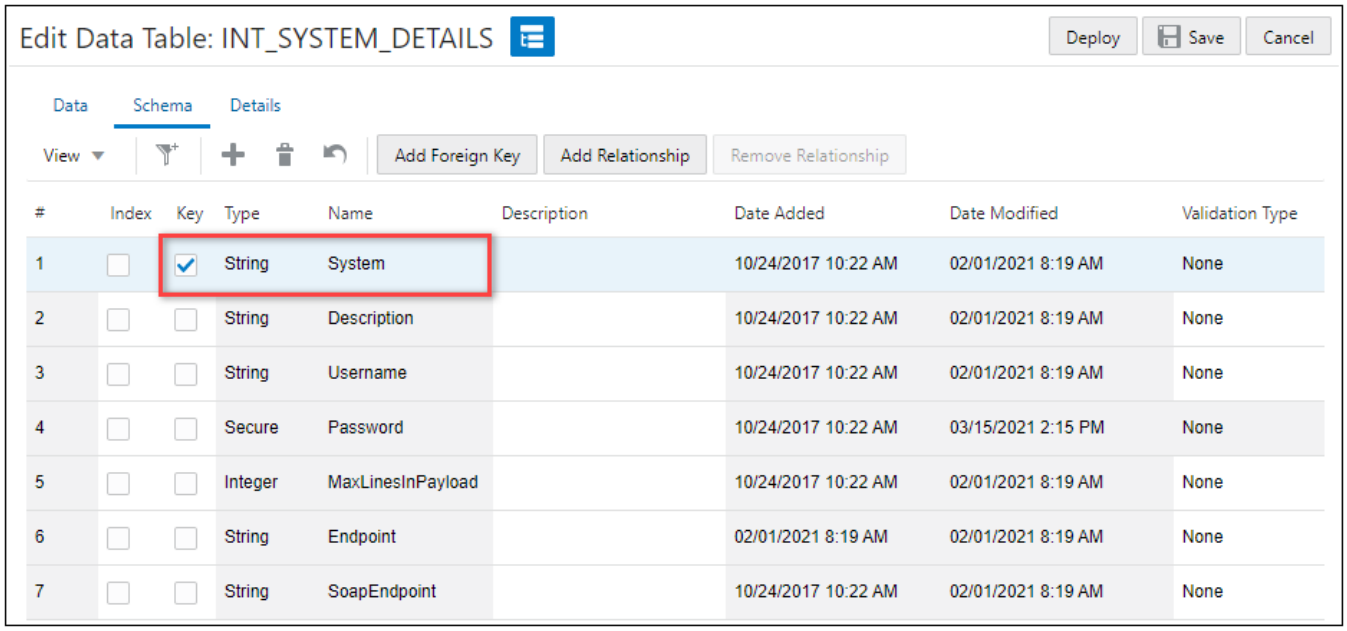
Manual Data Table Changes

The INT_SYSTEM_DETAILS and INT_SYSTEM_TEMPLATES data tables are added to the CPQ site for account integration.

INT_SYSTEM_DETAILS

System (Key)	Username	Endpoint	Description
TCA-OrgService	<Enter the username here to call the web service endpoint>	<Enter the web service endpoint to call the service related to TCA>	TCA Find Organization details
TCA-AccService	<Enter the username here to call the web service endpoint>	<Enter the web service endpoint to call the service related to TCA>	TCA customer Account

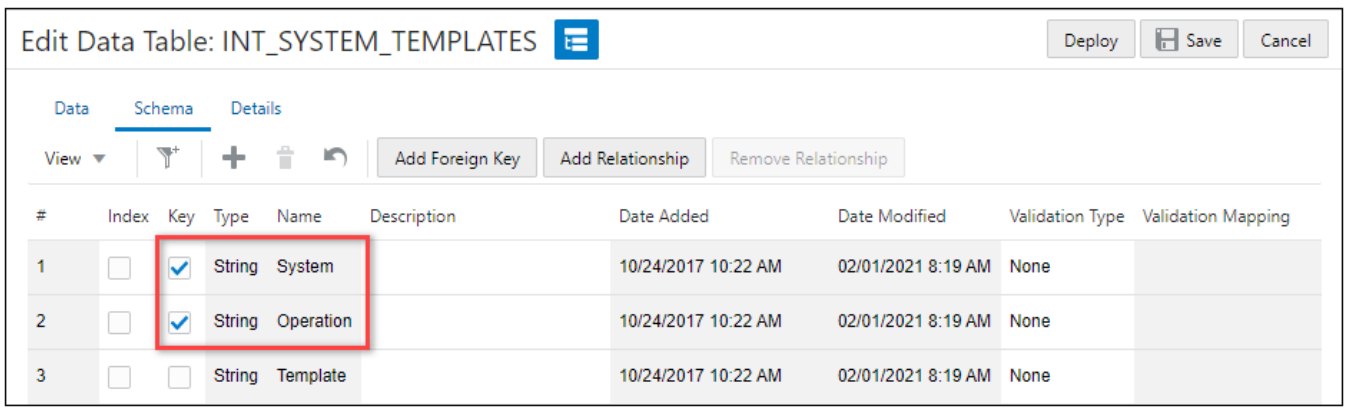
As shown in the following image, administrators must manually select the **Key** option for the **System** column.



INT_SYSTEM_TEMPLATES

System (Key)	Operation (Key)	Templates
TCA-OrgService	FindOrg	<Enter the template URL path that is uploaded in File Manager>
TCA-AccService	FindAcc	<Enter the template URL path that is uploaded in File Manager>

As shown in the following image, administrators must manually select the **Key** option for the **System and Operation** columns.



Add Template Dependencies to File Manager

Oracle CPQ administrators must add the Find Organization payload template file and the Customer Account payload template file to File Manager. These template files support account integration.

Perform the following steps to add the template dependencies to File Manager.

1. Download the **findOrganizationPayload.txt** and **customerAccountPayload.txt** payload template files from My Oracle Support.
2. Open the Admin Home page.
3. Navigate to **File Manager** under **Utilities** category and create a new folder named **TCA**.
4. Click **Browse** under **Add Files**. The Choose File to upload dialog opens.
5. Navigate to the findOrganizationPayload.txt file and click **Open**.
6. Click **Add File**. The findOrganizationPayload.txt file displays in File Manager.
7. Complete steps 2-6 for the customerAccountPayload.txt file.

> **Note:** To view the BML included in the payload template files, refer to Appendix G: Payload Template Files.

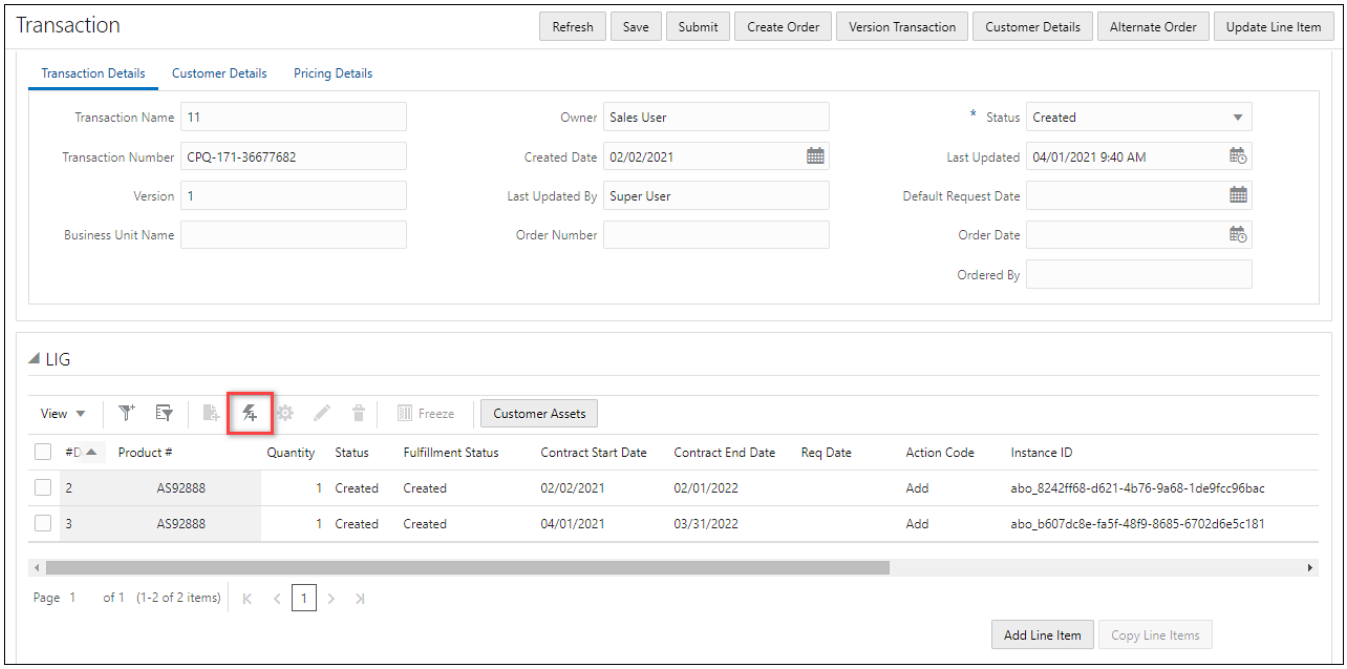
Appendix A: Create Order

Appendix A1: Create Order – Standard Item Workflow

OIC is the middleware used to establish an integration between Oracle CPQ and Oracle Order Management. Once this integration is established, sales users can use Oracle CPQ to create a Transaction and invoke OIC integration to create an order in Oracle Order Management.

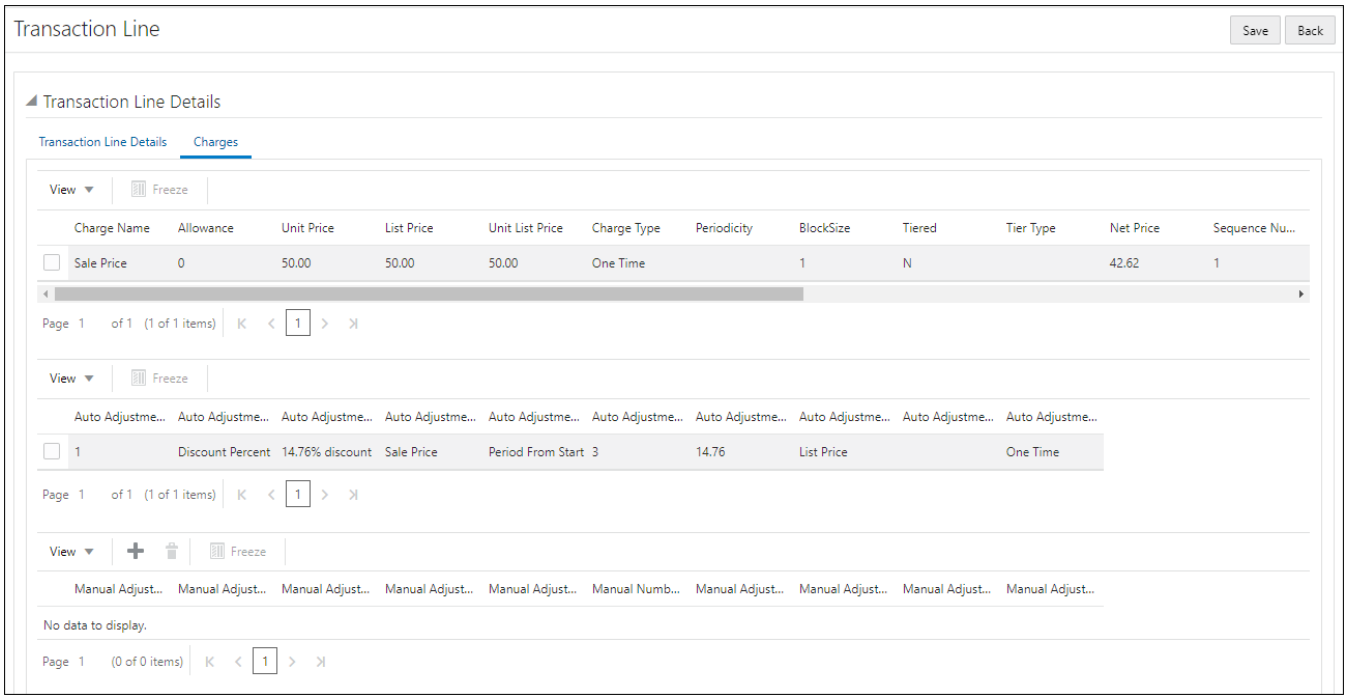
Perform the following steps in Oracle CPQ to create a standard order.

1. Navigate to the Transaction Manager.
2. Click **New Transaction**. The Transaction page opens.
3. Add Standard products to the line item using Quick Add.

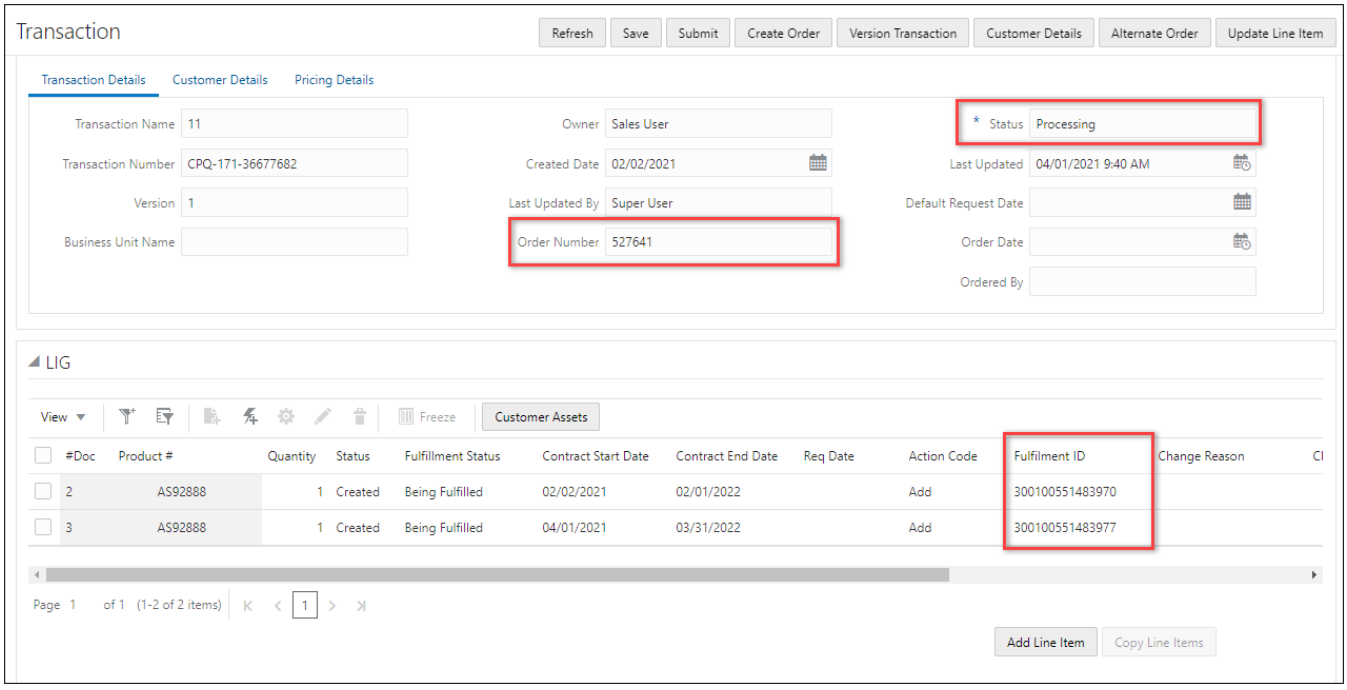


4. Open the Transaction Line Details page for a transaction line item, and then edit the line details.

5. Navigate to Transaction Line Details > Charges, then verify and add Adjustments as required.



6. Edit the Transaction and Transaction Line details as required.
7. Click **Save**.
8. Click **Create Order**.
9. Order Number, Order Status, and Fulfilment Id for each line are returned from Order Management on successful creation of order.

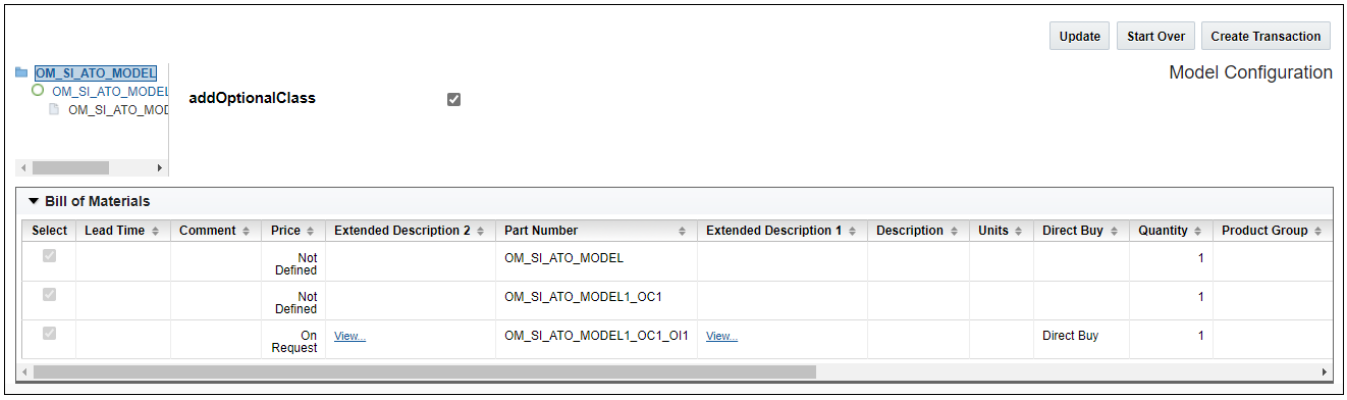


Note: Use Oracle Order Management to check the order creation using returned Order Number.

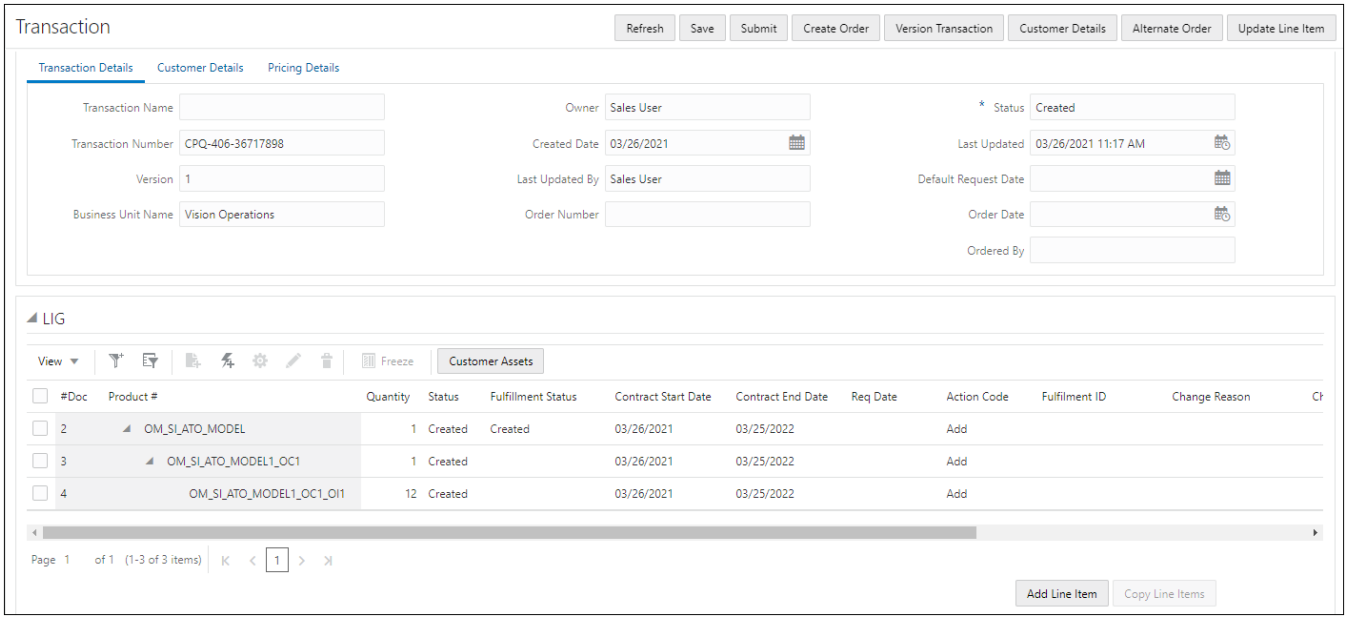
Appendix A2: Create Order – Configurable Item Workflow

Perform the following steps in Oracle CPQ to create a configurable item order.

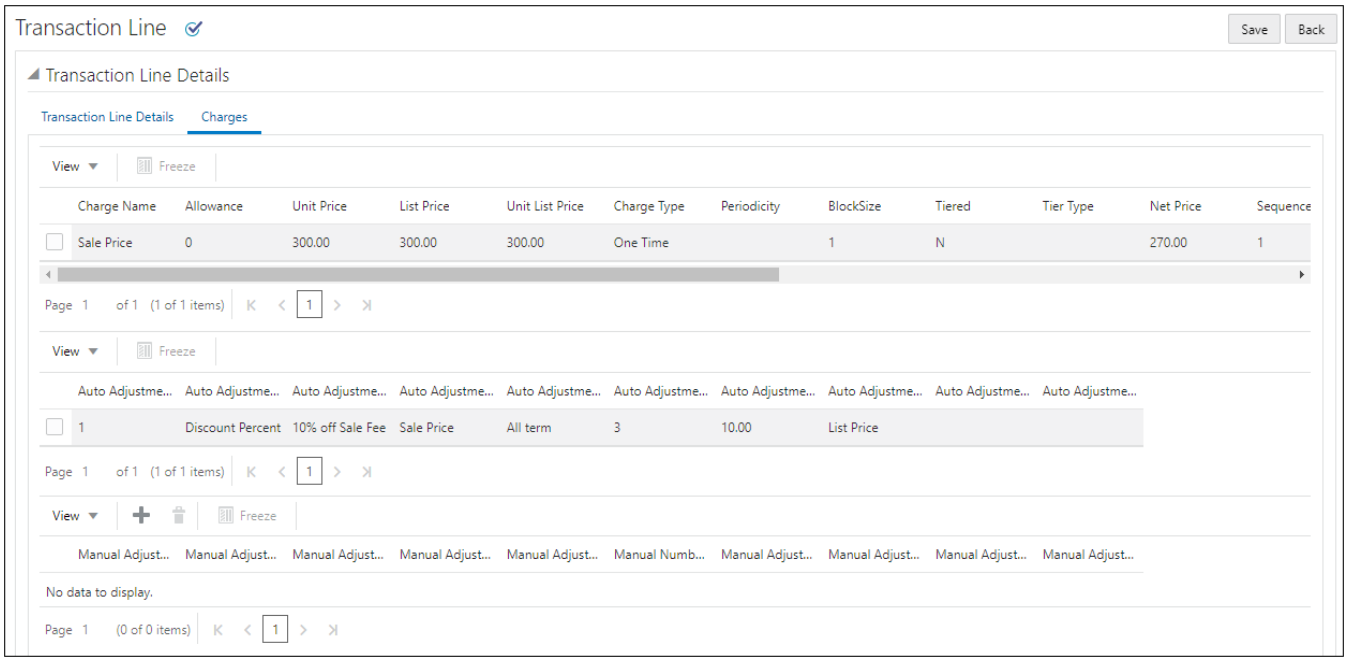
1. Open Model Configuration.
2. Select the root model.
3. Select the **addoptionalclass** check box.
4. Click **Update**. This will display optional class available for the root model.



5. Click **Create Transaction**. This will create new transaction with the configurable items.



6. Edit the line details under Transaction Line Details > Charges. Net Price will be displayed based on the Adjustments applied. Discounts are based on the adjustments mentioned in the data tables.



7. Manual Adjustments can be added in the charges, and then click Save. This will update the Net Price.
8. Edit the Transaction and Transaction Line details as required.
9. Click **Save**.
10. Click **Create Order**.
11. The Order Number, Order Status, and Fulfilment Id for each line are returned from Order Management on successful creation of order.

Notes:

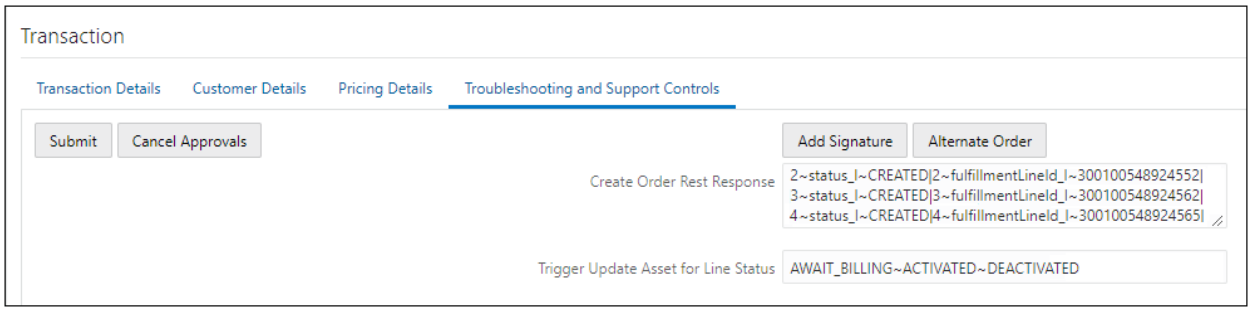
- Use Order Management to check the order creation using the returned Order Number.
- BOM setup is required for the configuration.

Appendix B: Auto Sync Status Workflow

Oracle Order Management allows customers to configure Business Events to be raised during various stages of fulfilment. We subscribe to these Business events through OIC integrations and invoke CPQ REST APIs to update vital information.

This functionality synchronizes the Order and Order Line status from Order Management to CPQ.

1. Create a transaction in CPQ.
2. Record the **Trigger Update Asset for Line Status** value.



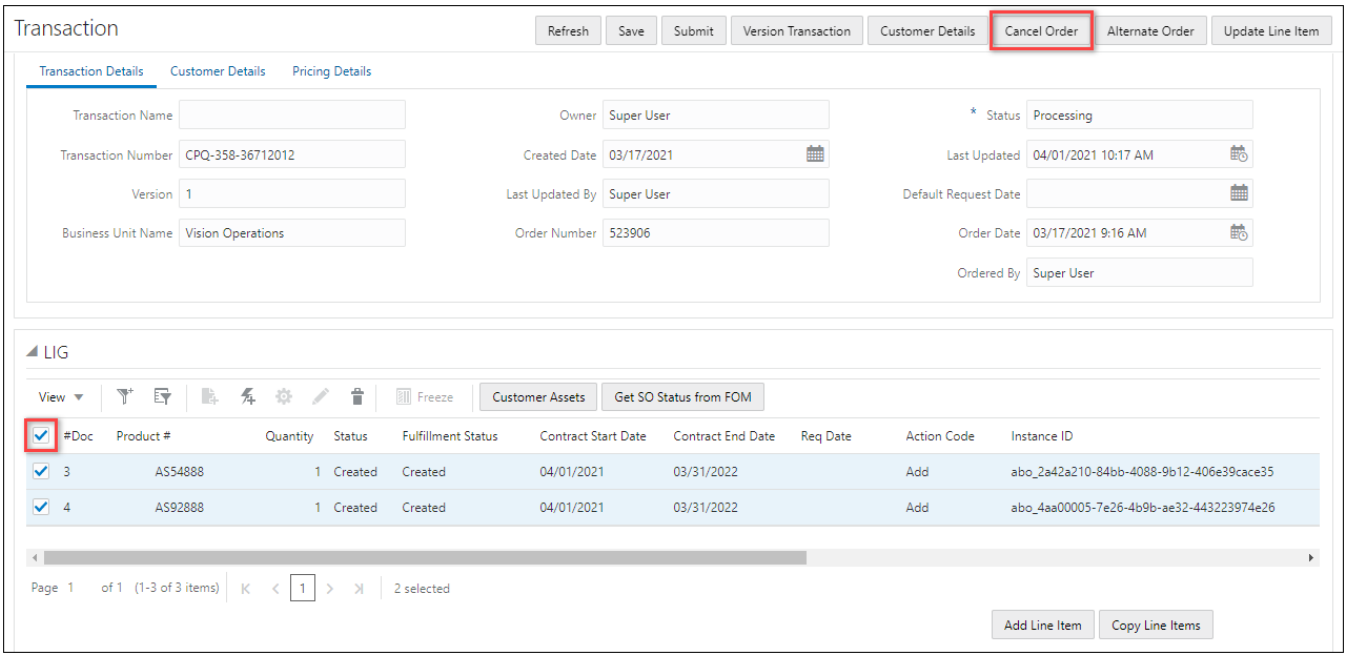
3. Click **Create Order** to create an order in Order Management. Refer to Appendix A1: Create Order.
4. Use Order Management to check the order creation using the Order Number returned on order creation.
5. When the Order/ Order Line status changes on Order Management, it should reflect the updated Status attribute in CPQ based on Order Status Mapping in OIC. Refer to Appendix I: CPQ-OM Status Mapping.
6. If the line status in CPQ is one of the values in 'Trigger Update Asset for Line Status', the Update Asset functionality will also be triggered and Fulfilment Status will be updated to 'Fulfilled'.
7. Open the Subscription Workbench page and observe the asset is created for the Fulfilled lines.

Appendix C: Cancel Order Workflow

This work flow support the cancellation of full order (all lines) and partial order (selected line items).

Appendix C1: Full Order Cancellation

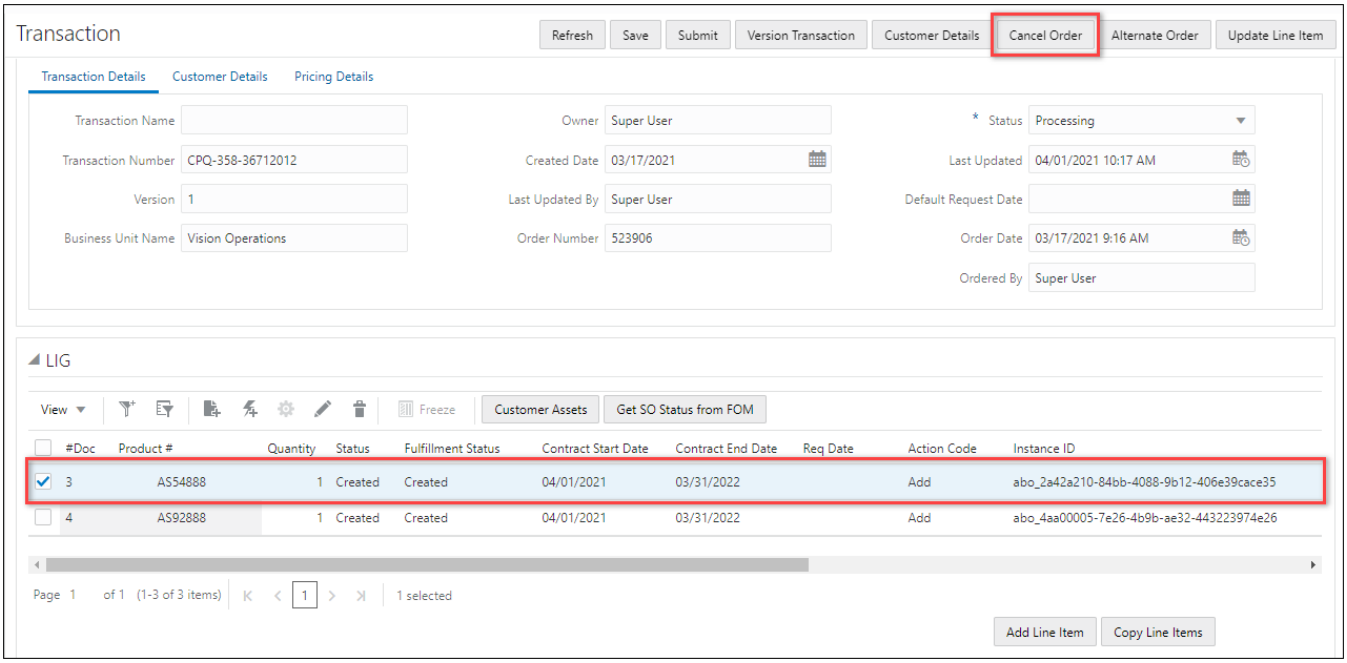
1. Navigate to the commerce and open the transaction in which order was created.
2. Validate there is an order number associated with the transaction.
3. Validate the order is in processing status.
4. Click the **Select All** option in the Line Item Grid.
5. (Optional) Select the cancellation reason in the transaction details.
6. Click **Cancel Order**.



7. After the cancellation action is complete, verify the line item status is updated to Cancellation Requested or Cancelled.

Appendix C2: Partial Order Cancellation

1. Navigate to the commerce and open the transaction in which order was created.
2. Validate there is an order number associated with the transaction.
3. Validate the order is in processing status.
4. Select the line items to cancel in the Line Item Grid.
5. (Optional) Select a cancellation reason for each of the lines to be cancelled in the Line Item Grid.
6. Click **Cancel Order**.



7. After the cancellation action is complete, verify the status is updated to 'Cancellation Requested' or 'Canceled' for the selected lines.

Notes:

- Cancellation is not allowed for line with status such as 'BILLED', "SHIPPED', 'ACTIVATED', 'FULFILLED', 'AWAIT_BILLING', 'DELIVERED', 'AWAIT_ASSET_CREATION', 'SHIPPED' etc.
- An order which is in DRAFT status cannot be cancelled.
- For configurable Items only root items can be cancelled.

Appendix D: Commerce Attributes

Appendix D1: Main Document (e.g. Transaction) Attributes

Attribute Name	Variable Name	Type	Description
Customer Id	_customer_id	Customer Id	This is the special Customer ID (with attribute type = Customer ID) used in CPQ-platform-supplied CRM integrations. This attribute is associated with the automatically-created Autofill, Browse, Select Alternate Address, and View Actions.
Account Number	accountNumber_t	Text	Customer Account number
Bill To Site Use Id	billToSiteUseId_t	Text	Customer bill-to Site use identifier.
Transaction Internal ID	bs_id	Text	Current Unique Buy-side ID. This is the internal identifier for a transaction within a CPQ site.
Buld	businessUnitId_t	Text	Customer business unit identifier.
Business Unit Name	businessUnitName_t	Text	Customer business unit name.
Cancel Reason	cancelReason_t	Menu	The reason expressed by the customer for canceling this order. The reason is selected by the sales user from a list of standard values for the site.
Created Date	createdDate_t	Date	The date when the transaction was created.
Created By	createdBy_t	Text	The person that created the transaction.
Create Order REST Response	createOrderRestResponse	Text Area	This attribute holds the rest response of create order.
Currency	currency_t	Menu	The currency used to price and invoice this transaction.
Customer Company Name	customerCompanyName_t	Text	When the Customer Company Name field is populated, Customer Details, partyId, accountId, and BillToSiteUseId are populated for the entered company.
Customer ID	customerID_t	Text	The Customer ID to be used in package and custom integrations other than the CPQ-platform-supplied CRM integrations. In BASE Commerce Process this attribute defaults to the value for the CPQ-platform-supplied integration attribute _customer_id.
Customer Party ID	customerPartyID_t	Text	The Party ID for the CRM Customer. This attribute is used when a generalized Party object is used to identify customer parties in addition to the Customer ID.
Customer Party Site ID	customerPartySiteID_t	Text	The ID for the Party Site or Address of the CRM Customer.
Freeze Price Flag	freezePriceFlag_t	Boolean	Contains one of the following values: true or false. If true, then price the sales order. If false, then do not price. The default value is true.

Attribute Name	Variable Name	Type	Description
Invoice To Party ID	invoiceToPartyID_t	Text	The Party ID for the Invoice To Customer. This attribute is used when a generalized Party object is used to identify customer parties. The Invoice To Party may differ from the CRM Customer and Sold To Parties.
Invoice To Party Site ID	invoiceToPartySiteID_t	Text	The ID for the Party Site or Address of the Invoice To party.
CPQ Source System Code	oRCL_ERP_CPQSourceSystemCode_t	Text	The code used in the integrated ERP system to identify CPQ as the source for order transactions.
Order Number	oRCL_ERP_OrderNumber_t	Text	The order number assigned by the external Order Management and ERP system.
Partial Ship Allowed	oRCL_ERP_PartialShipAllowed_t	Boolean	Indicates whether partial shipment of the requested quantity is acceptable to the customer.
Order Key	orderKey_t	Text	Value that uniquely identifies the sales order. This value is derived by concatenating the value of SourceOrderSystem, a colon, and the value of SourceOrderId.
Packing Instructions	packingInstructions_t	Text	Text that describes how to pack the item.
Party Id	partyId_t	Text	CDM integration will provide value to this field based on Customer company name.
Payment Terms	paymentTerms_t	Menu	The payment terms the customer will receive on their invoice.
Status	Status_t	Menu	The most advanced milestone reached by lines of this transaction.
Subscription Profile Id	subscriptionProfileId_t	Text	Customer subscription profile identifier to be used in subscription management to specify subscription contract preferences. (For future use)
Transaction Number	transactionID_t	Text	The number or public unique identifier of the transaction.
Transaction Name	transactionName_t	Text	The number or public unique identifier of the transaction.
Transaction Type Code	transactionTypeCode_t	Menu	Abbreviation that identifies the type of transaction. A list of accepted values is defined in the Order Management lookup type ORA_DOO_ORDER_TYPES.
Trigger Update Asset for Line Status	triggerUpdateAssetForLineStatus_t	Text	List of line.status_l values which indicate that line has finished processing on Order Management and is ready to move to asset in CPQ (pending other lines' status within same root).
Customer Attributes		Additional Address Set	An Additional Address Set consists of address fields that are created and deleted together.
Invoice To Attributes		Additional Address Set	An Additional Address Set consists of address fields that are created and deleted together.

Attribute Name	Variable Name	Type	Description
Ship To Attributes		Additional Address Set	An Additional Address Set consists of address fields that are created and deleted together.
Customer Last Name	_customer_t_last_name	Text	Array Attribute in Additional Address Set. Last Name of the contact person.
Customer First Name	_customer_t_first_name	Text	Array Attribute in Additional Address Set. First Name of the contact person.
Customer Company Name	_customer_t_company_name	Text	Company Name of the buying organization.
Invoice To Company Name	_invoiceTo_t_company_name	Text	Company Name of the organization being billed for this transaction.
Ship To Address	_shipTo_t_address	Text	First Line of the Address of the organization where this transaction will be shipped.
Ship To City	_shipTo_t_city	Text	City of the organization where this transaction will be shipped.
Ship To Country	_shipTo_t_country	Text	Country of the organization where this transaction will be shipped.
Ship To Company Name	_shipTo_t_company_name	Text	Company Name of the organization where this transaction will be shipped.
Ship To Zip	_shipTo_t_zip	Text	Zip Code of the organization where this transaction will be shipped.
Ship To State	_shipTo_t_state	Text	State or Province of the organization where this transaction will be shipped.

Appendix D1: Sub-Document (e.g. Transaction Line) Attributes

Attribute Name	Variable Name	Type	Description
Configuration ID	`_configuration_id`	Text	Configuration id for integrations.
Associated Line Id	`associatedLineId_l`	Text	Document number to be associated with the subscription line. (For future use)
Associated Order Id	`associatedOrderId_l`	Text	Source order Id associated with the subscription line (For future use)
Cancel Reason	`cancelReason_l`	Menu	The value must be same as defined in the Order Management lookup type DOO_RETURN_REASON.
Change Code	`changeCode_l`	Menu	A single select menu to handle all the possible codes to close a subscription. (For future use) Possible values (variable names in parentheses): • Full (`ORA_FULL`) • Prorate Without Credit (`ORA_PRORATE_WITHOUT_CREDIT`) • Prorate With Credit (`ORA_PRORATE_WITH_CREDIT`)
Change Reason	`changeReason_l`	Menu	A single select menu to handle all the possible reasons for closing a subscription. (For future use) Possible values (variable names in parentheses): • Breach (`ORA_BREACH`) • Non-Compliant (`ORA_NON_COMPLIANCE`) • Quantity Change (`ORA_CHANGE_QUANTITY`) • Downgrade (`ORA_DOWNGRADE`) • Term Change (`ORA_TERM_CHANGE`) • Upgrade (`ORA_UPGRADE`)
Charge Hierarchy	`chargeHeirarchy`	Read-only text or HTML	This field is read-only and displays the charge hierarchy associated to the subscription line.
Contract Start Date	`contractStartDate_l`	Date	Date that the customer starts to receive the service.
Contract End Date	`contractEndDate_l`	Date	Date that the customer stops to receive the service.
Document Number	`documentNumber_l`	Text	Document number of the line
Fulfillment Line Id	`fulfillmentLineId_l`	Text	Value that uniquely identifies the fulfillment line.
Fulfillment Organization Code	`fulfillmentOrganizationCode_l`	Text	Abbreviation that identifies the organization that ships the shipment.
Fulfillment Status	`fulfillmentStatus_l`	Menu	The status for this line supplied by the fulfillment system after the transaction is submitted as a sales order.

Attribute Name	Variable Name	Type	Description
Instance ID	itemInstanceId_l	Text	The unique, invariant, internal identifier of this instance of a product. It is used to link sales order lines to assets they change in an asset based order and as the key stored in a sales order line association. This ID is assigned when the product instance is created in either an external federated asset repository or in CPQ when a sales order line is created.
Net Price Non-Recurring	netPriceNonRecurring_l	Float	Roll Up Net Price for all onetime items
Net Price Recurring	netPriceRecurring_l	Float	Roll Up Net Price for all recurring items
Unit Price Non-Recurring	unitPriceNonRecurring_l	Float	Roll Up unit Price for all onetime items.
Unit Price Recurring	unitPriceRecurring_l	Float	Roll Up unit Price for all recurring items.
Action Code	oRCL_ABO_ActionCode_l	Menu	Indicates the changes required by the transaction line.
Packing Instructions	oRCL_ERP_PackingInstr_l	Text	Packing Instructions for the order line.
Shipping Instructions	oRCL_ERP_ShippingInstr_l	Text	Shipping Instructions for the order line.
Ship Set	oRCL_ERP_ShipSet_l	Text	The shipping set in which this order line will be included.
Parent Document Number	parentDocNumber_l	Text	Document number of the parent line item.
Payment Terms	paymentTerms_l	Menu	Payment terms to apply to collect payment. Review and update the value for this attribute using the Setup and Maintenance work area, and the Manage Receivables Payment Terms task. To collect data for this attribute, use the Collect Order Reference Data task for Fusion Source.
Quantity	requestedQuantity_l	Integer	Quantity of associated line which is being requested. Can be user entered or set logically
Rollup Flag	rollupFlag_l	Boolean	Specifies order management to roll up a charge component or not, as a rollup value or aggregate value for the element code of the charge component price.
RootAssetKey	rootAssetKey_l	Text	Asset key of the root line item.
Status	status_l	Menu	The status of this line.
Transaction Category Code	transactionCategoryCode_l	Menu	Brief text that identifies the transaction category. • ORDER. Process a new source order. • RETURN. Process a return of an existing sales order.
Block Allowance	oRCL_pRC_blockAllowance	Integer	Pricing attribute to store the block allowance. (For future use)

Attribute Name	Variable Name	Type	Description
Block Size	oRCL_pRC_blockSize	Integer	Pricing attribute to store the block size. Charge line's block size. Usually a static value defined as a Line attribute in BOM attribute table. (For future use)
Duration	oRCL_pRC_duration	Text	Pricing attribute to store the periodicity.
Duration UOM	oRCL_pRC_durationUOM	Text	Unit of measures for duration.
External Parent Key	oRCL_pRC_externalParentKey	Text	__itemInstanceId_t of Product
Quantity UOM	oRCL_pRC_quantityUOM	Text	Unit of measures for the priced quantity.
Tiered	oRCL_pRC_tierd_l	Menu	Pricing attribute to store tiered info. Recurring Usage Charge can be Tiered or Non Tiered (Y or N) (For future use)
TierType	oRCL_pRC_tiertype_l	Menu	Pricing attribute to store the Tier type. If the Recurring Usage Charge is designated as Tiered, it can be set as ORA_ALL_TIERS or ORA_HIGHEST_TIER. By default ORA_ALL_TIERS is supported for the pricing profile. The default can be modified to ORA_HIGHEST_TIER pricing profile. (For future use)
Unit List Price	oRCL_pRC_unitListPrice	Currency	Pricing attribute to store the unit list price.
Unit Net Price	oRCL_pRC_unitNetPrice	Currency	Pricing attribute to store the unit net price.
Use Usage Lock	oRCL_pRC_useUsageLock	Boolean	This field indicates whether there is a Price Override. If there is a Price Override, then the price in the Create Subscription payload is required for pricing the subscription. (For future use)
Tier Information	oRCL_pRC_tierInfo	Array Set	Array set which with contain attributes to hold the tier information. (For future use)
Auto Adjustments	oRCL_autoAdjustments	Array Set	Displays the list of adjustments which are already applied to the items. Refer to Pricing Engine Setup for more details about the Auto Adjustment Array set and the associated attributes.
Charges	oRCL_charges	Array Set	Displays the list of charges associated to the items. Refer to Pricing Engine Setup for more details about the Charges Array set and the associated attributes
Manual Adjustments	oRCL_manualAdjustments	Array Set	Displays the list of manual adjustments to be applied on the items. Refer to Pricing Engine Setup for more details about the Manual Adjustment Array set and the associated attributes.

Appendix E: BML

Appendix E1: BML - Create Order

The following BML is associated with the Create Order action Advanced Modify - After Formulas.

```
retStr = "";
if (createOrderRestResponse <  > "") {
  for line in transactionLine {
    if (line._line_bom_parent_id == "") {
      retStr = retStr + line._document_number + "~fulfillmentStatus_l~BEING_FULFILLED|";
    }
  }
  retStr = retStr + createOrderRestResponse;
}
return retStr;
```

The following BML is associated with Hide Create SO Action hiding rule.

```
count = 0;
for transaction in transactionLine{
  count = count + 1;
  break;
}
if (count == 0 OR orderKey_t <  > "") {
  return true;
}
return false;
```

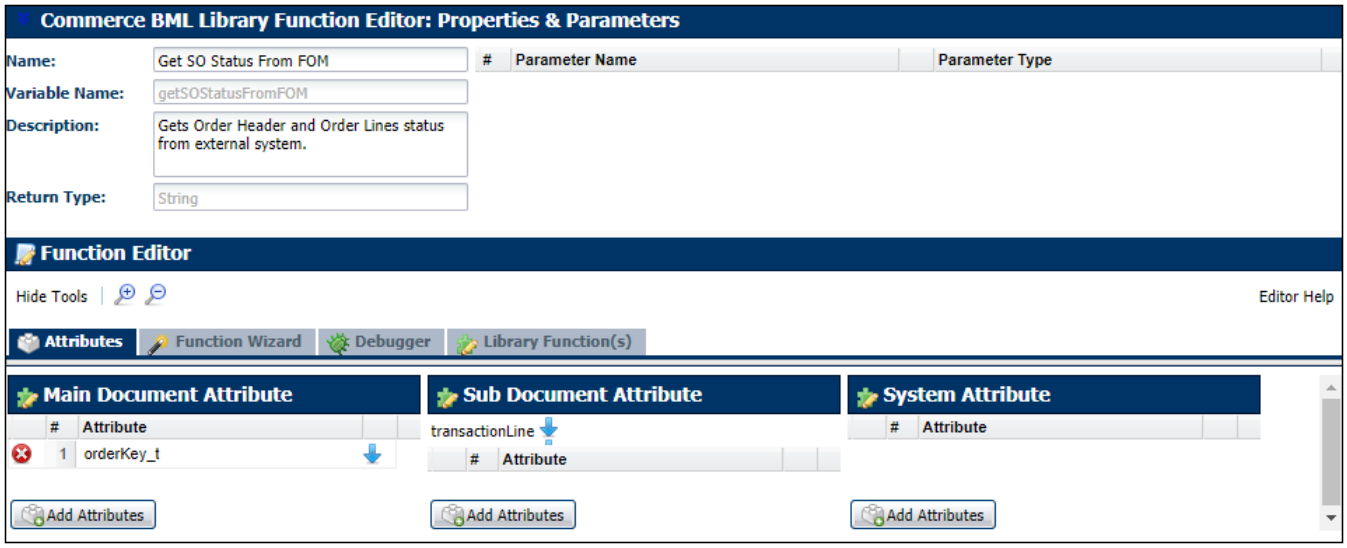
Appendix E2: BML - Get Sales Order Status from FOM

This commerce library function is associated with Get Sales Order Status from FOM BML Integration which is associated with the Get SO Status From FOM action.

Name	Get SO Status From FOM
Variable Name	`getSOStatusFromFOM`
Description	Retrieves Status for Order Header and Order Lines. Calls ICS REST API to retrieve the status values.
Return Type	String

Import following attributes:

Main Document Attribute	`orderKey_t`



Script:

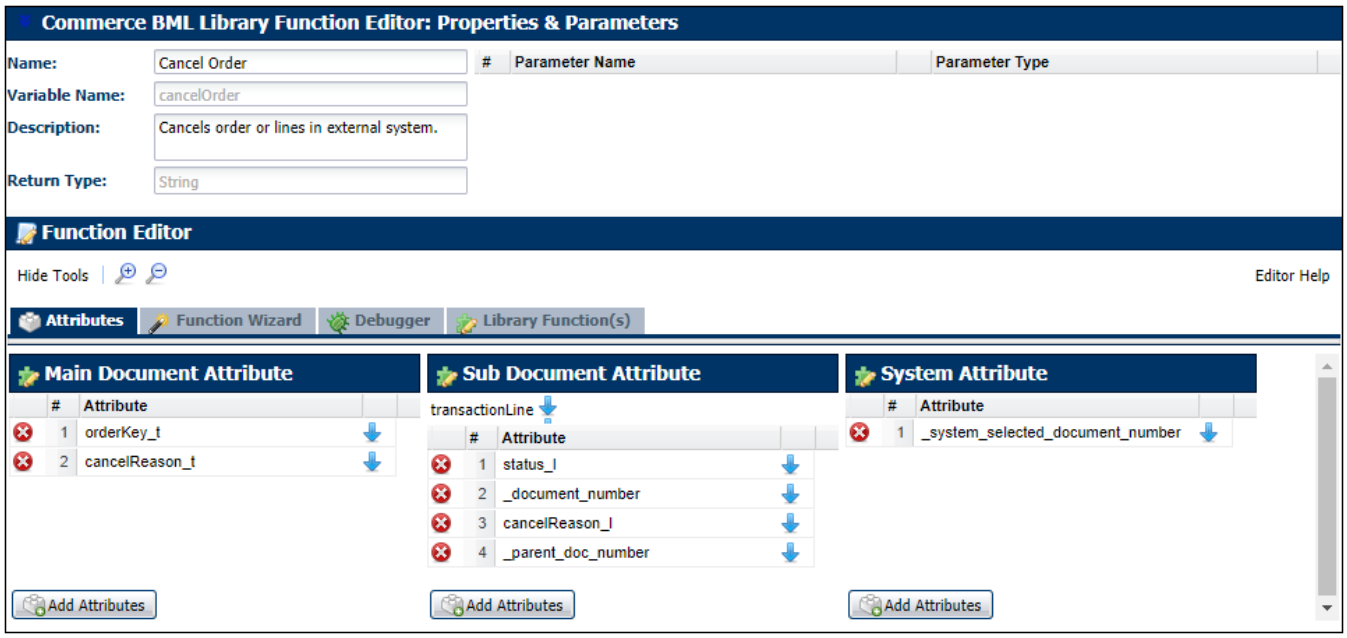
```
//getSOStatusFromFOM
////////////////////////////////////////
//Function Definition
//
//  Imported MainDoc Attributes:
//      "orderKey_t"
//  Returns string - concatenation of docNum~AttributeVarName~AttributeValue strings
//                   in the format compatible with BML integration

// Get connection details stored in Generic Integration - GetSOStatusfromFOM

connectionInfo = bmql("select username,password,requestUrl from integration.genericIntegration where name
= 'GetSOStatusfromFOM'");
icsUrl = "";
username = "";
password = "";
for info in connectionInfo {
  username = get(info, "username");
  password = get(info, "password");
  icsUrl = get(info, "requestUrl");
}
auth = encodebase64(username + ":welcome2");
// create request headers
headers = dict("string");
put(headers, "Authorization", "Basic " + auth);
put(headers, "Accept", "application/json");
// create request url - <ICS url>/<orderKey>
url = icsUrl + orderKey_t;
// Call ICS REST API
restResponse = urldata(url, "GET", headers, "", 100000);
// Get the status code
statusCode = get(restResponse, "Status-Code");
if (statusCode <  > "200") {
// The request errored out
  errorMsg = get(restResponse, "Error-Message");
  message = "";
  if (statusCode == "404") {
    message = "Error: Order with Order Key " + orderKey_t + " not found.";
    throwerror(message);
  } else {
    message = "Error: " + errorMsg;
    throwerror(errorMsg);
  }
}
// Request was successful, hence get the formatted concatenated string from response and return it.
resp = get(restResponse, "Message-Body");
respJson = json(resp);
cpqresp = jsonget(respJson, "statusResponse");
return cpqresp;
```

Appendix E3: BML - Cancel Order

Cancel Order

This commerce library function is associated with Cancel Order BML Integration which is associated with Cancel Order action.

Name	Cancel Order
Variable Name	`cancelOrder`
Description	This function cancels an order or order lines in Order Management. This function is being called from cancelSOFromCPQ commerce library function.
Return Type	String
Main Document Attributes	• `orderKey_t` • `cancelReason_t`
Sub Document Attributes	• `status_l` • `_document_number`
System Attribute	`_system_selected_document_number`



Script:

```
//Validation if the user has not selected any line item ,Throw an error
if(_system_selected_document_number=="")
{
throwerror("Select at least one line item(s) to cancel.");
}
//Spited the selected line item system attribute to find the selected line items by user
selectedLineArray=split(_system_selected_document_number, "~");
//Create an array to add all the status that can not be canceled
statusCanNotBeCanceled= String [8];
statusCanNotBeCanceled[0] ="AWAIT_BILLING";
statusCanNotBeCanceled[1] ="BILLED";
statusCanNotBeCanceled[2] ="DELIVERED";
statusCanNotBeCanceled[3] ="ACTIVATED";
statusCanNotBeCanceled[4] ="FULFILLED";
statusCanNotBeCanceled[5] ="DELETED";
statusCanNotBeCanceled[6] ="AWAIT_ASSET_CREATION";
statusCanNotBeCanceled[7] ="SHIPPED";
//Take string to add all the line number that can not be canceled
invalidLineItemSelected="";
index=0;
//Boolean variable to check if current product's root item is selected
isRootItemSelected=false;
//String to collect child line which root item are not selected
LonelyChildLineNumber="";
//Selected Cancelled line
selectedCancelledLine="";
// create the JSON array to add each line
lineJsonline = jsonarray();
for line in transactionLine{
     //Increase the index for each line item
      index=index+1;
      //check if the user has selected the line
      if(findinarray(selectedLineArray,line._document_number)>=0){
  if (line._parent_doc_number == "") {
     //current root item is selected
     isRootItemSelected=true;
     }elif(NOT(isRootItemSelected))
     { if(LonelyChildLineNumber==""){
          LonelyChildLineNumber=LonelyChildLineNumber+string(index);
          }else{
          LonelyChildLineNumber=LonelyChildLineNumber+","+string(index);
         }
     }

  /*Check if the line can be canceled based on line item status,if it can not be canceled add it
  to invalid string or else add the document number to JSON array*/

   if(findinarray(statusCanNotBeCanceled, line.status_l)>-1)
   {
     if(invalidLineItemSelected=="")
      {
      invalidLineItemSelected=""+string(index);
      }
     else{
      invalidLineItemSelected=invalidLineItemSelected+","+string(index);
      }
   } //Validate if the selected line item is already canceled
```

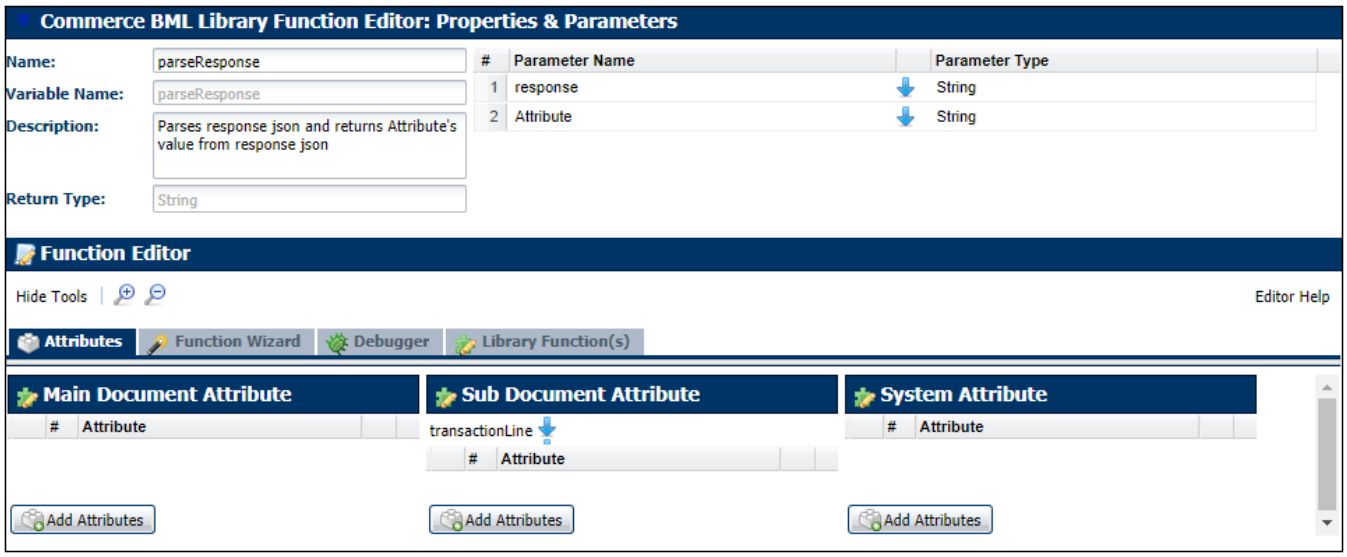
```
   elif( line.status_l=="CANCELED" OR line.status_l=="CANCEL_REQUESTED")
   {
     if(selectedCancelledLine=="")
      {
      selectedCancelledLine=""+string(index);
      }
     else{
      selectedCancelledLine=selectedCancelledLine+","+string(index);
      }
   }
  else{
   jsonLineObject=json();
   jsonput(jsonLineObject,"SourceTransactionLineId",line._document_number);
   jsonput(jsonLineObject,"SourceTransactionScheduleId" ,line._document_number);
   jsonput(jsonLineObject,"CanceledFlag" ,true);
   jsonput(jsonLineObject,"CancelReasonCode" ,line.cancelReason_l);
   jsonarrayappend(lineJsonline ,jsonLineObject);
   }
 }//Current root item is not selected
 elif (line._parent_doc_number == "") {
 isRootItemSelected=false;
 }
}
if(invalidLineItemSelected<>"" AND LonelyChildLineNumber<>""){
  throwerror("Below selected line item(s) can not be cancelled as one of more products are already
shipped: \n Line Number:"+invalidLineItemSelected+"\n The selected line item(s) can not be cancelled. Only
the root line item(s) can be cancelled. \n Line Number:"+LonelyChildLineNumber);
}
//If there is at least one line item selected that can not be canceled than throw error
if(invalidLineItemSelected<>"")
{
  throwerror("Below selected line item(s) can not be cancelled as one of more products are already
shipped: \n Line Number:"+invalidLineItemSelected);
}
if(LonelyChildLineNumber<>"")
{
  throwerror("The selected line item(s) can not be cancelled. Only the root line item(s) can be cancelled.
\n Line Number:"+LonelyChildLineNumber);
}
//Create JSON payload and include the order key ,cancellation reason code
jsonMainObject=json();
jsonput(jsonMainObject,"OrderKey",orderKey_t);
jsonput(jsonMainObject,"CancelReasonCode",cancelReason_t);
//Check if the user has selected all line item than send canceled Flag true at header level to cancel
whole order
if(index==sizeofarray(selectedLineArray))
{
 jsonput(jsonMainObject,"CanceledFlag",true);
}else{
if(selectedCancelledLine<> "")
{
//Throw error message for canceled lines in case of partial cancellation
 throwerror("The selected line item(s) are already cancelled/Cancellation Requested. \n Line
Number:"+selectedCancelledLine);
}
//Else add the line json array to the json payload
  jsonput(jsonMainObject,"lines",lineJsonline);
}
//convert the JSON payload to String
```

```
payload=jsontostr(jsonMainObject);
print payload;
restResponse = dict("string");
headers = dict("string");
//Get connection details stored in Generic Integration - CancelSOFromCPQ
connectionInfo = bmql("select username,password,requestUrl from integration.genericIntegration where name
= 'CancelSOFromCPQ'");
icsUrl = "";
username = "";
password = "";
for info in connectionInfo{
  username = get(info, "username");
  password = get(info, "password");
  icsUrl = get(info, "requestUrl");
  }
auth = encodebase64(username+":welcome2");
//create request headers
headers = dict("string");
put (headers , "Authorization", "Basic "+auth);
put (headers , "Content-Type", "application/json");
put (headers , "Accept", "application/json");
// Call ICS Rest API
restResponse = urldata(icsUrl, "POST",headers,payload, 120000);
statusCode = get(restResponse,"Status-Code");
if(statusCode <> "200"){
if(statusCode =="-1"){
throwerror("The request is taking longer than usual. Cancellation has been requested for selected
line(s).");
}else{
throwerror("Some error occured");
}
}
resp = get(restResponse, "Message-Body");
ErrorString = commerce.parseResponse(resp, "error");
//If there is an error in response throw error
if( ErrorString <>"" )
{
  throwerror(ErrorString );
}
// Request was successful, hence get the formatted concatenated string from response and return it.
return commerce.parseResponse(resp,"response");
```

Parse Response

The following BML is used in cancel order commerce library function to parse response.

Name	Parse Response
Variable Name	`parseResponse`
Description	This function is to parse the JSON response from Fusion .This function is being called from cancelSOFromCPQ commerce library function.
Return Type	String
Input type	response (String) Attribute (String)



Script:

```
jobj=json(response);
cpqresp = jsonget(jobj,Attribute);
return cpqresp;
```

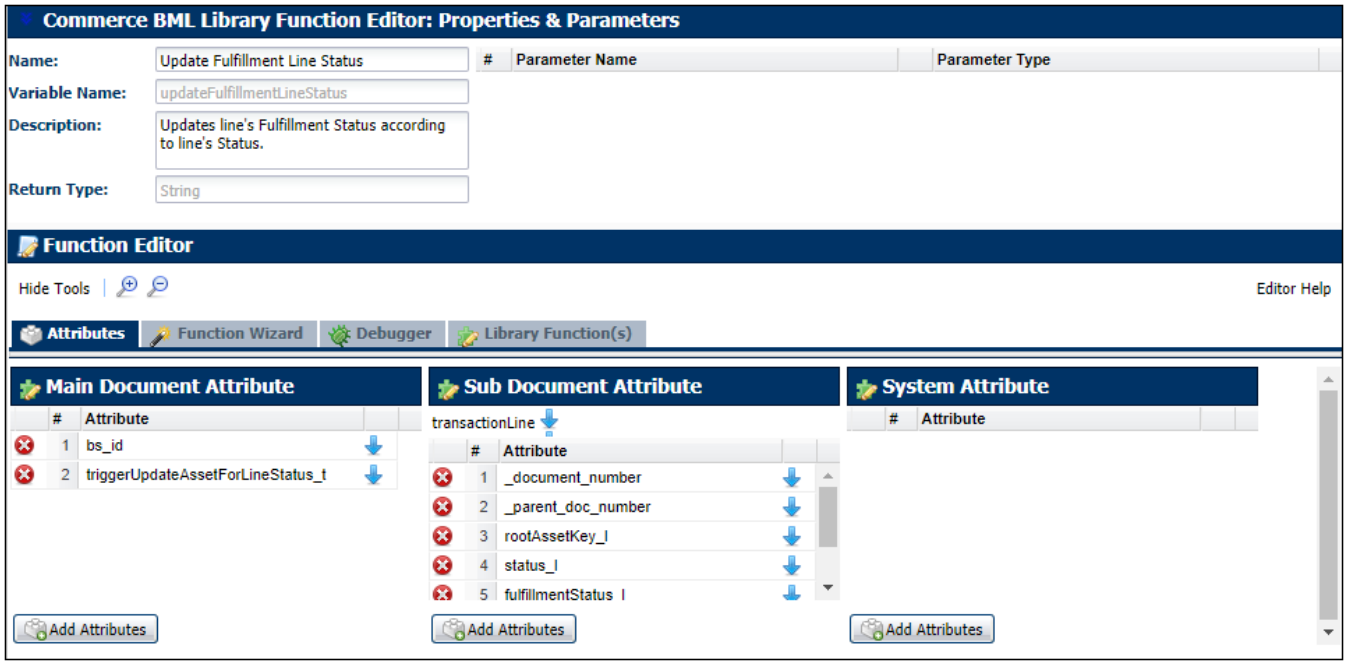
Appendix E4: BML - Update Fulfillment Line Status

This commerce library function is associated with Update Fulfillment Line Status BML Integration. This BML loops through the quote lines and change the fulfilment status (fulfillmentStatus_l) of Configuration Item (root only) or standard items to 'AWAIT_ASSET_CREATION' based on 'Trigger Update Asset for Line Status' (triggerUpdateAssetForLineStatus_t) attribute values.

Name	Update Fulfillment Line Status
Variable Name	`updateFulfillmentLineStatus`
Description	Updates line's Fulfillment Status according to line's Status.
Return Type	String

Import following attributes:

Main Document Attributes	• `Bs_id` • `triggerUpdateAssetForLineStatus_t`
Sub Document Attributes	• `_document_number` • `_parent_doc_number` • `_part_number` • `_part_custom_field9` • `rootAssetkey_l` • `status_l` • `fulfillmentStatus_l` • `itemInstanceId_l`



Script:

```
/*MainDoc UpdateAsset Script
Purpose: Helper to convert to asset, all or selected lines in the transaction
System Variables:
  _system_buyside_id
Main doc fields:
  _transaction_customer_id(_transaction prefix is from maindco varname)
    (process specific):
    currency_t,
    paymentTerms_t
Line fields:
  status_l
  _document_number,
  requestDate_l,
  itemInstanceId_l,
  oRCL_ABO_ActionCode_l,
  _line_bom_parent_id,
  fulfillmentStatus_l
Library Functions:
  abo_initializeContext,
  abo_updateAsset,
  convertLibraryFunctionDateStringToDate
*/
print "Initiate Update Asset: ";
commerceProcess = "oraclecpqo";
abocontext = util._ORCL_ABO.abo_initializeContext(commerceProcess);
bDiagnosisOff = jsonget(abocontext, "AboDiagnosticDisabled", "boolean", true);

//step1. generic input parsing which can be copied to any actions
inputJson = json();

//step2 common transaction heading initialization
customer_id = _transaction_customer_id;
currency = currency_t;
paymentTerm = paymentTerms_t;

//abo main logic
FULFILLED = "FULFILLED";
AWAIT_ASSET_CREATION = "AWAIT_ASSET_CREATION";
DEACTIVATED = "DEACTIVATED";
DOCUMENT_NUMBER = "documentNumber";
CUSTOMER = "customer";
TRANSACTION_ID = "transactionId";
CURRENCY_CODE = "currency";
REQUEST_DATE = "requestDate";
ACTION_CODE = "actionCode";
ITEM_INSTANCE_ID = "itemInstanceId";
PAYMENT_TERM = "paymentTerms";

//we won’t allow asset creation until you select a customer.
if (customer_id == ""
  OR isnull(customer_id)) {
  throwError("Pleaes select an customer.");
}
//  since some transaction attribute value are needed to update asset
//  we will also collect some transaction info and for internal,
//  we will place them into 2 level hierarchy json
txn_json = json();
jsonput(txn_json, CUSTOMER, customer_id);
jsonput(txn_json, CURRENCY_CODE, currency);
```

```
jsonput(txn_json, PAYMENT_TERM, paymentTerm);
jsonput(txn_json, TRANSACTION_ID, _system_buyside_id);

//step3a formal processing
//processing for internal case.
//now we collect the list of lines need to update-asset
successString = "";
lineJsonArray = jsonArray();
for line in transactionLine {
  if (NOT(line.status_l == DEACTIVATED OR line.fulfillmentStatus_l == AWAIT_ASSET_CREATION)) {
    continue; //skip lines that are NOT awaiting asset creation or NOT Deactivated
  }
  if (line._line_bom_parent_id <  > "") { //skip non-root line
    continue;
  }
  if (line.itemInstanceId_l == "") { //skip non abo & non model line
    continue;
  }
  lineJson = json();
  jsonput(lineJson, DOCUMENT_NUMBER, line._document_number);

  //for transaction date we will use db format within abo script,
  // also for empty date we treat as today as of processing ime
  transactionDate = line.requestDate_l;
  if (transactionDate == ""
    OR isnull(transactionDate)) {
      transactionDate = datetostr(getDate(false), "yyyy-MM-dd HH:mm:ss");
    } else {
      tranDate = commerce.convertLibraryFunctionDateStringToDate(transactionDate);
      transactionDate = datetostr(tranDate, "yyyy-MM-dd HH:mm:ss");
  }
  jsonput(lineJson, REQUEST_DATE, transactionDate);
  jsonput(lineJson, ACTION_CODE, line.oRCL_ABO_ActionCode_l);
  jsonput(lineJson, ITEM_INSTANCE_ID, line.itemInstanceId_l);
  jsonarrayappend(linejsonArray, lineJson);

  //also prepare the return string to update root lines when the updateAsset is successful
  if (successString<>"") {
     successString = successString + "|";
  }
  successString = successString + line._document_number + "~fulfillmentStatus_l~" + FULFILLED;
}

//now inovke utility to load line detail, transfer to bom, and aggregate open order,
//and generate delta action and invoke asset syc
//if updateAsset fail, expect the abo_updateAsset to throwerror from inside

response = util._ORCL_ABO.abo_updateAsset(txn_json, lineJsonArray);
if (successString<>"") {
  successString = successString + "|";
}

//print successString;
//print "Update Asset completed";

return successString;
```

Appendix E5: BML - Update Asset

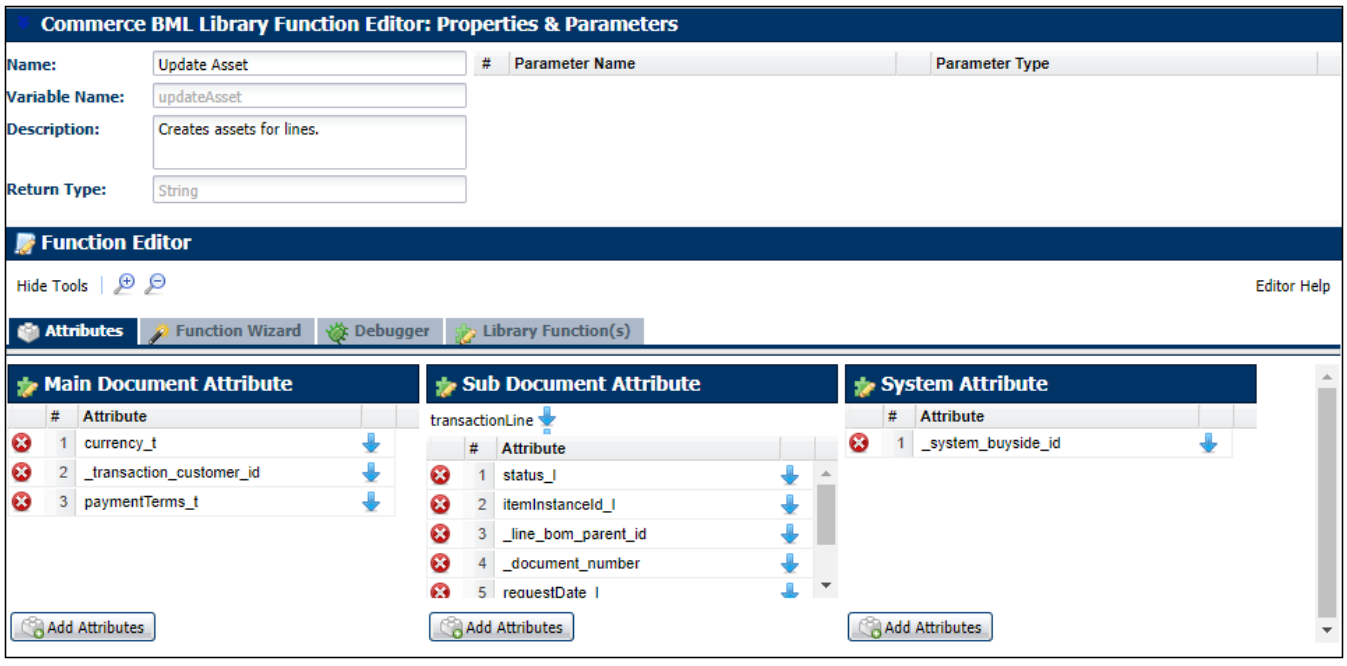
This commerce library function is associated with Update Asset BML Integration. The BML loops through the all the quote lines with fulfilment status as 'AWAIT_ASSET_CREATION' and create/update the asset in CPQ Asset Repository and changes the fulfilment status (fulfillmentStatus_l) to ' FULFILLED.

Update Asset

Name	Update Asset
Variable Name	`updateAsset`
Description	Creates assets for lines.
Return Type	String

Import the following attributes:

System Attribute	`_system_buyside_id`
Main Document Attribute	• `currency_t` • `paymentTerms_t` • `_transaction_customer_id`
Sub Document Attribute	• `transactionLine (Collection)` • `_document_number` • `itemInstanceId_l` • `status_l` • `fulfillmentStatus_l` • `requestDate_l` • `_line_bom_parent_id` • `oRCL_ABO_ActionCode_l`
Library Function(s)	• `abo_initializeContext` • `abo_updateAsset`



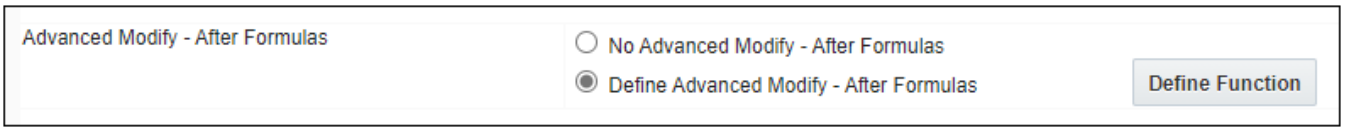
Script:

```
/*MainDoc UpdateAsset Script
Purpose: Helper to convert to asset , all or selected lines in the transaction
System Variables :
  _system_buyside_id
Main doc fields:
  _transaction_customer_id(_transaction prefix is from maindco varname)
    (process specific):
    currency_t,
    paymentTerms_t
Line fields:
  status_l
  _document_number,
  requestDate_l,
  itemInstanceId_l,
  oRCL_ABO_ActionCode_l,
  _line_bom_parent_id,
  fulfillmentStatus_l
Library Functions:
  abo_initializeContext,
  abo_updateAsset,
  convertLibraryFunctionDateStringToDate
*/
print "Initiate Update Asset: ";
commerceProcess = "oraclecpqo";
abocontext = util._ORCL_ABO.abo_initializeContext(commerceProcess);
bDiagnosisOff = jsonget(abocontext, "AboDiagnosticDisabled", "boolean", true);

//step1. generic input parsing which can be copied to any actions
inputJson = json();

//step2 common transaction heading initialization
customer_id = _transaction_customer_id;
currency = currency_t;
paymentTerm = paymentTerms_t;

//abo main logic
FULFILLED = "FULFILLED";
AWAIT_ASSET_CREATION = "AWAIT_ASSET_CREATION";
DEACTIVATED = "DEACTIVATED";
CLOSED = "CLOSED";
DOCUMENT_NUMBER = "documentNumber";
CUSTOMER = "customer";
TRANSACTION_ID = "transactionId";
CURRENCY_CODE = "currency";
REQUEST_DATE = "requestDate";
ACTION_CODE = "actionCode";
ITEM_INSTANCE_ID = "itemInstanceId";
PAYMENT_TERM = "paymentTerms";

//we won’t allow asset creation until you select a customer.
if (customer_id == ""
  OR isnull(customer_id)) {
  throwError("Pleaes select an customer.");
}
//  since some transaction attribute value are needed to update asset
//  we will also collect some transaction info and for internal,
//  we will place them into 2 level hierarchy json
txn_json = json();
jsonput(txn_json, CUSTOMER, customer_id);
jsonput(txn_json, CURRENCY_CODE, currency);
```

```
jsonput(txn_json, PAYMENT_TERM, paymentTerm);
jsonput(txn_json, TRANSACTION_ID, _system_buyside_id);

//step3a formal  processing
//processing for internal case.
//now we collect the list of lines need to update-asset
successString = "";
lineJsonArray = jsonArray();
for line in transactionLine {
  if (NOT(line.status_l == DEACTIVATED OR line.fulfillmentStatus_l == AWAIT_ASSET_CREATION)) {
    continue; //skip lines that are NOT awaiting asset creation or NOT Deactivated
  }
  if (line._line_bom_parent_id<>"") { //skip non-root line
    continue;
  }
  if (line.itemInstanceId_l == "") { //skip non abo & non model line
    continue;
  }
  lineJson = json();
  jsonput(lineJson, DOCUMENT_NUMBER, line._document_number);

  // for transaction date we will use db format within abo script,
  // also for empty date we treat as today as of processing ime

  transactionDate = line.requestDate_l;
  if (transactionDate == ""
    OR isnull(transactionDate)) {
      transactionDate = datetostr(getDate(false), "yyyy-MM-dd HH:mm:ss");
    } else {
      tranDate = commerce.convertLibraryFunctionDateStringToDate(transactionDate);
      transactionDate = datetostr(tranDate, "yyyy-MM-dd HH:mm:ss");
    }
    jsonput(lineJson, REQUEST_DATE, transactionDate);
    jsonput(lineJson, ACTION_CODE, line.oRCL_ABO_ActionCode_l);
    jsonput(lineJson, ITEM_INSTANCE_ID, line.itemInstanceId_l);
    jsonarrayappend(linejsonArray, lineJson);

    //also prepare the return string to update root lines when the updateAsset is successful
    if (successString<>"") {
      successString = successString + "|";
    }
    if( line.oRCL_ABO_ActionCode_l == "TERMINATE") {
      successString = successString + line._document_number + "~fulfillmentStatus_l~" + CLOSED;
    } else {
      successString = successString + line._document_number + "~fulfillmentStatus_l~" + FULFILLED;
  }
}
// now inovke utility to load line detail, transfer to bom, and aggregate open order,
// and generate delta action and invoke aset syc
// if updateAsset fail, expect the abo_updateAsset to throwerror from inside
response = util._ORCL_ABO.abo_updateAsset(txn_json, lineJsonArray);
if (successString<>"") {
  successString = successString + "|";
}
print successString;
print "Update Asset completed";
return successString;
```

Appendix E6: BML – Save

This BML is associated with Advanced Modify - After Formulas for Save (cleanSave_t) action on transaction level.



BML Editor

Action : Oracle Quote to Order > Transaction > Save

BML | Debugger

System Variable Name	Type	Description
_system_current_document_number	String	Current Document Number

Variable Name for (Transaction Line)	Type	Description
transactionLine	Collection of Sub Documents	
_document_number	String	Document Number
_part_number	String	Part Number
contractStartDate_I	String (date)	Contract Start Date
_model_name	String	Model Name
requestedQuantity_I	Integer	Quantity
contractEndDate_I	String (date)	Contract End Date
_price_calculation_info	String	Calculation Information
oRCL_pRC_useUsageLock	Boolean	Use Usage Lock
oRCL_pRC_tierd_I	String	Tiered
oRCL_manualAdjustments	Array Set	Manual Adjustments

Imported Util Functions

Json oRCL_fOM_oraclePricingFOMIntegration(String partNumber, Integer quantity, String chargeName, Integer lockedAllowance, Integer lockedBlockSize, Float lockedListPrice, Integer[] tierFrom, Integer[] tierTo, Float[] tierListPrice, Integer[] tierBlockSize, Boolean lockUsage, Date contractStartDate, Date contractEndDate)

Imported Commerce Functions

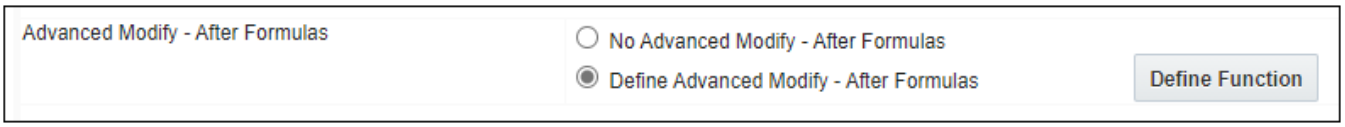
String oRCL_fOM_populatePrices(Json calcInfo, JsonArray manualAdjustment, Integer quantity, String document_number)

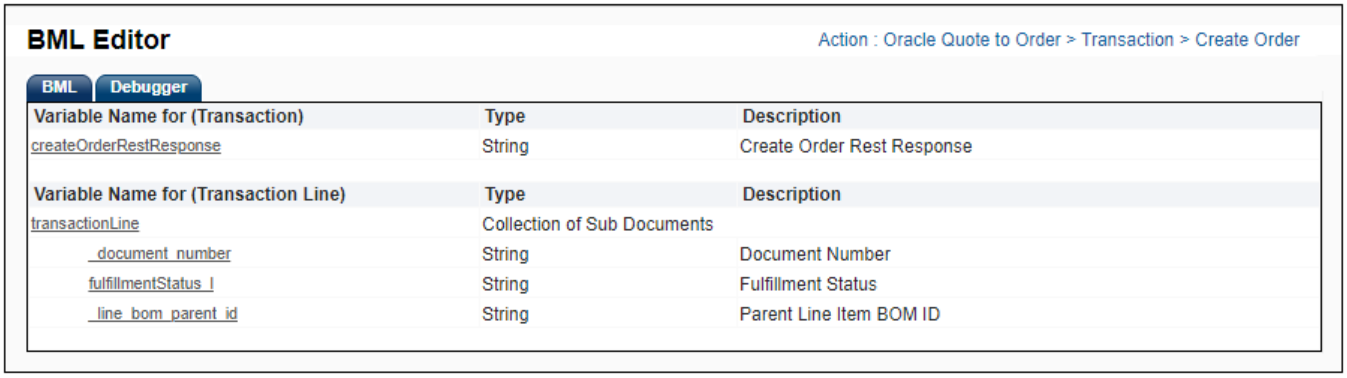
Script

```
//At Transaction level save
//System Variable Name    Type    Description
//_system_current_document_number String  Current Document Number

//Variable Name for (Transaction Line)    Type    Description
//transactionLine Collection of Sub Documents
//      _document_number  String  Document Number
//      _price_calculation_info   String  Calculation Information
//      oRCL_pRC_useUsageLock Boolean Use Usage Lock
//      oRCL_pRC_tierd_l  String  Tiered

//Imported Util Functions
//String _SM.oRCL_pRC_populateCharges(JsonArray charges, String documentNumber)

returnString = "";
for line in transactionLine {
  if(((line.oRCL_pRC_useUsageLock == false OR line.oRCL_pRC_tierd_l == "N") AND
line._price_calculation_info <> "")) {
    calcInfo = jsonarrayget(jsonarray(line._price_calculation_info),0,"json");
    returnString = commerce.oRCL_populateFOMPrices(calcInfo, line.oRCL_manualAdjustments,
line.requestedQuantity_l, line._document_number);
  } elif (line._part_number == "") {
    tierFrom = integer[1];
    tierListPrice = float[1];
    startDate = strtojavadate(line.contractStartDate_l , "MM/dd/YYYY HH:mm:ss");
    endDate = strtojavadate(line.contractEndDate_l , "MM/dd/YYYY HH:mm:ss");
        calculationInfo = util.oRCL_fOM_oraclePricingFOMIntegration(line._model_name,
line.requestedQuantity_l, "", 0, 0, 0.0, tierFrom, tierFrom, tierListPrice, tierFrom, false, startDate,
endDate);
    calcInfo =  jsonget(calculationInfo, "calculationInfo", "json");
    returnString = commerce.oRCL_populateFOMPrices(calcInfo, line.oRCL_manualAdjustments,
line.requestedQuantity_l, line._document_number);
  }
}
return returnString;
```

Appendix E7: BML – Save (Line)

Advanced BML is Advanced Modify - After Formulas





Script:

```
//At TrnsactionLine level save
//System Variable Name    Type    Description
//_system_current_document_number String  Current Document Number

//Variable Name for (Transaction Line)    Type    Description
//transactionLine Collection of Sub Documents
//      _document_number  String  Document Number
//      _price_calculation_info   String  Calculation Information
//      oRCL_pRC_useUsageLock Boolean Use Usage Lock
//      oRCL_pRC_tierd_l  String  Tiered

//Imported Util Functions
//String _SM.oRCL_pRC_populateCharges(JsonArray charges, String documentNumber)

returnString = "";
for line in transactionLine {
  if(line._document_number == _system_current_document_number){
    if((line.oRCL_pRC_useUsageLock == false OR line.oRCL_pRC_tierd_l == "N") AND
line._price_calculation_info <> "") {
      calcInfo = jsonarrayget(jsonarray(line._price_calculation_info),0,"json");
      returnString = commerce.oRCL_populateFOMPrices(calcInfo, line.oRCL_manualAdjustments,
line.requestedQuantity_l, line._document_number);
    } elif (line._part_number == "") {
      tierFrom = integer[1];
      tierListPrice = float[1];
      startDate = strtojavadate(line.contractStartDate_l , "MM/dd/YYYY HH:mm:ss");
      endDate = strtojavadate(line.contractEndDate_l , "MM/dd/YYYY HH:mm:ss");
          calculationInfo = util.oRCL_fOM_oraclePricingFOMIntegration(line._model_name,
line.requestedQuantity_l, "", 0, 0, 0.0, tierFrom, tierFrom, tierListPrice, tierFrom, false, startDate,
endDate);
      calcInfo =  jsonget(calculationInfo, "calculationInfo", "json");
      returnString = commerce.oRCL_populateFOMPrices(calcInfo, line.oRCL_manualAdjustments,
line.requestedQuantity_l, line._document_number);
    }
  }
}
return returnString;
```

Appendix E8: BML - Customer Details

The BML for the Customer Details action is used to support account integration. When sales users enter a customer company name and click Customer Details, the BML retrieves the PrimaryPartyId, BillToAccountId, and BillToSiteUseId fields from the Oracle EBS Customer Data Management application:

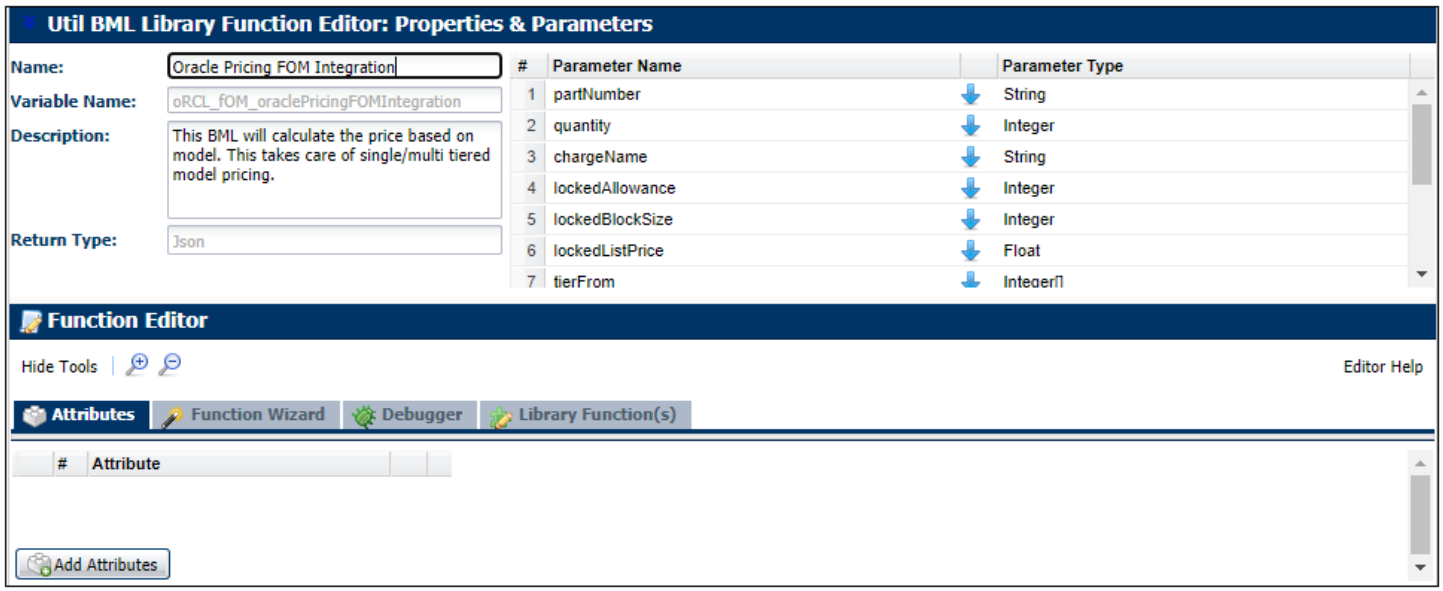
```
//1. Get Template Location
returnPartyId = "";
defaultErrorMessage = "";
errorString = "Error in TCA Service";
organizationSoapResponse ="";
if(isnull(_transaction_customer_id)  OR _transaction_customer_id  == "")
{
system="TCA-OrgService";
operation="FindOrg";
organizationSoapRequestLocation = commerce.getTemplateLocation(system, operation);

//payload = commerce.getUserAttributes(system);
organizationSoapRequest=applytemplate(organizationSoapRequestLocation ,dict("string"),
defaultErrorMessage);
print "request";
print organizationSoapRequest;
print "response";
organizationSoapResponse = commerce.invokeWebService(system, organizationSoapRequest );
print organizationSoapResponse;
xpaths = string[1];
xpaths[0] = "//ns2:PartyId";
output = readxmlsingle(organizationSoapResponse, xpaths);
  if (containskey(output,xpaths[0]))
       {  returnPartyId = get(output,xpaths[0]);   }
  else {
    returnPartyId = "Check if customer company name is valid & also TCA service is up.";
    returnSiteNumber = "Check if customer company name is valid & also TCA service is up" ;
    return "1~invoiceToPartyID_t~"+returnPartyId+"|1~partyId_t~"+returnPartyId+"|"+
"1~billToSiteUseId_t~"+returnSiteNumber+"|";
}
}else{
returnPartyId = _transaction_customer_id;
}

// Get Template Location
system="TCA-AccService";
operation="FindAcc";
customerAccountSoapRequestLocation  = commerce.getTemplateLocation(system, operation);

//payload1 = commerce.getUserAttributes(system,returnPartyId);
payload1 = dict("string");
put(payload1,"returnPartyId",returnPartyId);
customerAccountSoapRequest=applytemplate(customerAccountSoapRequestLocation,payload1,
defaultErrorMessage);
print customerAccountSoapRequest;
customerAccountsoapResponse = commerce.invokeWebService("TCA-AccService", customerAccountSoapRequest);
print customerAccountsoapResponse;
xpathsAcct = string[1];
xpathsAcct[0] = "//ns2:CustomerAccountId";
returnAccountNumber = "" ;
outputAcct = readxmlsingle(customerAccountsoapResponse, xpathsAcct);
  if (containskey(outputAcct,xpathsAcct[0]))
    {  returnAccountNumber = get(outputAcct,xpathsAcct[0]);   }
  else {
```

```
    returnAccountNumber = "Check if customer company name is valid & also TCA service is up";
    returnSiteNumber = "Check if customer company name is valid & also TCA service is up ";
    return "1~accountNumber_t~"+returnAccountNumber+"|"+"1~billToSiteUseId_t~
"+returnSiteNumber +"|";
}
returnBillToSiteUseId="";
xpathsbilltositeuseid = string[1];
xpathsbilltositeuseid[0] =
"//ns2:Value/ns2:CustomerAccountSite/ns2:CustomerAccountSiteUse[ns2:SiteUseCode=‘BILL_TO’ and
ns2:PrimaryFlag=‘true’]/ns2:SiteUseId";
outputbilltositeuseid = readxmlsingle(customerAccountsoapResponse, xpathsbilltositeuseid);
  if (containskey(outputbilltositeuseid ,xpathsbilltositeuseid[0])) {
    returnBillToSiteUseId= get(outputbilltositeuseid,xpathsbilltositeuseid[0]);   }
  else {
    returnBillToSiteUseId= "Check if customer company name is valid & also TCA service is up.";
}
xpathCustomerAccountSiteUse = string[1];

// get the XML Element called Customer AccountSiteUse where primary is true and use is BILL TO
xpathCustomerAccountSiteUse[0] =
"//ns2:Value/ns2:CustomerAccountSite/ns2:CustomerAccountSiteUse[ns2:SiteUseCode=‘BILL_TO’ and
ns2:PrimaryFlag=‘true’]";

outputCustomerAccountSiteUse = readxmlsingle(customerAccountsoapResponse, xpathCustomerAccountSiteUse);
 returnSiteId ="";
 if (containsKey(outputCustomerAccountSiteUse,xpathCustomerAccountSiteUse[0]))
 {
   CustomerAccountSiteUseXmlFragment = get(outputCustomerAccountSiteUse,xpathCustomerAccountSiteUse[0]);
   xpath1 = string[1];
   xpath1[0] = "//ns2:CustomerAccountSiteId";
   output1 = readxmlsingle(CustomerAccountSiteUseXmlFragment,xpath1);
   if (containsKey(output1,xpath1[0]))
   {
    returnSiteId = get(output1,xpath1[0]);
   }
 }
  if (returnSiteId == "") {
   returnSiteId = errorString;
   }
partySiteId = "";
xpathCustomerAccountSite = string[1];

// get the XML Element called Customer AccountSite where ID = returnSiteId
xpathCustomerAccountSite[0] = "//ns2:Value/ns2:CustomerAccountSite[ns2:CustomerAccountSiteId=" +
returnSiteId + "]";

outputCustomerAccountSite = readxmlsingle(customerAccountsoapResponse, xpathCustomerAccountSite);

if (containsKey(outputCustomerAccountSite,xpathCustomerAccountSite[0]))
 {
   CustomerAccountSiteXmlFragment = get(outputCustomerAccountSite,xpathCustomerAccountSite[0]);
   xpath2 = string[1];
   xpath2[0] = "//ns2:PartySiteId";
   output2 = readxmlsingle(CustomerAccountSiteXmlFragment,xpath2);
   if (containsKey(output2,xpath2[0]))
   {
    partySiteId = get(output2,xpath2[0]);
   }
   if (partySiteId == "") {
   partySiteId = errorString;
```

```
   }
 }
partySiteNumber = "";
xpathPartySiteNumber = string[1];

// get the XML Element called Customer AccountSite where ID = returnSiteId
xpathPartySiteNumber[0] = "//ns2:Value/ns2:PartySite[ns1:PartySiteId=" + partySiteId + "]";
outputPartySiteNumber = readxmlsingle(organizationSoapResponse, xpathPartySiteNumber);

 if (containsKey(outputPartySiteNumber,xpathPartySiteNumber[0]))
 {
   PartySiteNumberXmlFragment = get(outputPartySiteNumber,xpathPartySiteNumber[0]);
   xpath3 = string[1];
   xpath3[0] = "//ns1:PartySiteNumber";
   output3 = readxmlsingle(PartySiteNumberXmlFragment,xpath3);
   if (containsKey(output3,xpath3[0]))
   {
    partySiteNumber = get(output3,xpath3[0]);
   }
 }
return "1~invoiceToPartyID_t~"+returnPartyId+"|1~partyId_t~"+returnPartyId+"|"+"1~accountNumber_t~"
+returnAccountNumber+"|"+"1~billToSiteUseId_t~"+returnBillToSiteUseId+"|"+"1~_customer_id~
"+returnPartyId+"|";
```

Appendix F: Pricing Related Utility BMLs

Oracle Pricing FOM Integration (oRCL_fOM_oraclePricingFOMIntegration): Calculates the price based models.



Script:

```
// This utility BML will calculate the price based on pricing model (Single or Tiered).
// Name: Oracle Pricing Subscription Base Profile
// Variable Name: oRCL_fOM_oraclePricingFOMIntegration
// Input:
// partNumber(String) - This is used to look up rows from the ORCL_PRC_BASE_CHGS table
// quantity(Integer) - Used for calculating the price.
// chargeName(String)
// lockedAllowance(Integer)
// lockedBlockSize(Integer)
// lockedListPrice(Float)
// tierFrom(Integer[])
// tierTo (Integer[])
// tierListPrice(Float[])
// tierBlockSize(Integer[])
// lockUsage(Boolean)
// Output:
//  JSON - Includes UnitPrice and other JSON elements for Subscription Service.
// Dependency : Calculate List Price, Prepare Tier Info Json

lang = dict("string");
fields = dict("string");
where = "";

// If the chargeName is provided, query the data table for specific charges. otherwise queries all
charges.
if(isnull(chargeName) OR chargeName == ""){
  put(fields, "$field1", partNumber);
  where = "Product= $field1";
}else{
  put(fields, "$field1", partNumber);
```

```
  put(fields, "$field2", chargeName);
  where = "Product= $field1 AND Charge= $field2";
}

// Get the results from the ORCL_PRC_BASE_CHGS to calculate the price.
charges = bmql("select Charge_Id, Product, Charge, Price, Block_Size, Allowance, Charge_Type, Periodicity,
Tiered, TierType, ApplyTo, Primary_Charge from ORCL_PRC_BASE_CHGS where $where",lang,fields);
chargeList = jsonarray();
returnPayload = json();
errorInPricing = false;
errorMessagesJsonArr = jsonarray();
sequenceNumber = 0;

for charge in charges { // Iterating throgh charges queried from data table ORCL_PRC_BASE_CHGS.
                    listPrice = 0.0;
  unitPrice = 0.0;
  unitListPrice = 0.0;
  totalAutoAdjustment = 0.0;
  periods = 0.0;
  startDate = contractStartDate;
  endDate = contractEndDate;
  tierList = jsonarray();
  // Reading the each charge details
  tiered = get(charge, "Tiered");
  chargeId = getint(charge, "Charge_Id");
  product = get(charge, "Product");
  chargeType = get(charge, "Charge_Type");
  chargeNameValue = get(charge, "Charge");
  periodicity = get(charge, "Periodicity");
  tierType = get(charge,"TierType");
  allowance = getint(charge, "Allowance");
  applyTo = get(charge, "ApplyTo");
  primaryCharge = get(charge, "Primary_Charge");
  adjustedAmount = 0.0;

  chargeJson= json();
  if(tiered == "N"){ // Non tiered
    // Checking charge is of type lockable, if yes and locked then calculating the price based on locked
values
    if(chargeType == "ORA_RECURRING_USAGE" AND lockUsage) {
      price = lockedListPrice;
      blockSize = lockedBlockSize;
      allowance = lockedAllowance;
    }else{
      price = getfloat(charge, "Price");
      blockSize = getint(charge, "Block_Size");
    }

    // Price from data table
    unitListPrice  = price;

    // Calculating List Price
    listPrice = util.oRCL_fOM_calculateListPrice(quantity, allowance, blockSize, price);
```

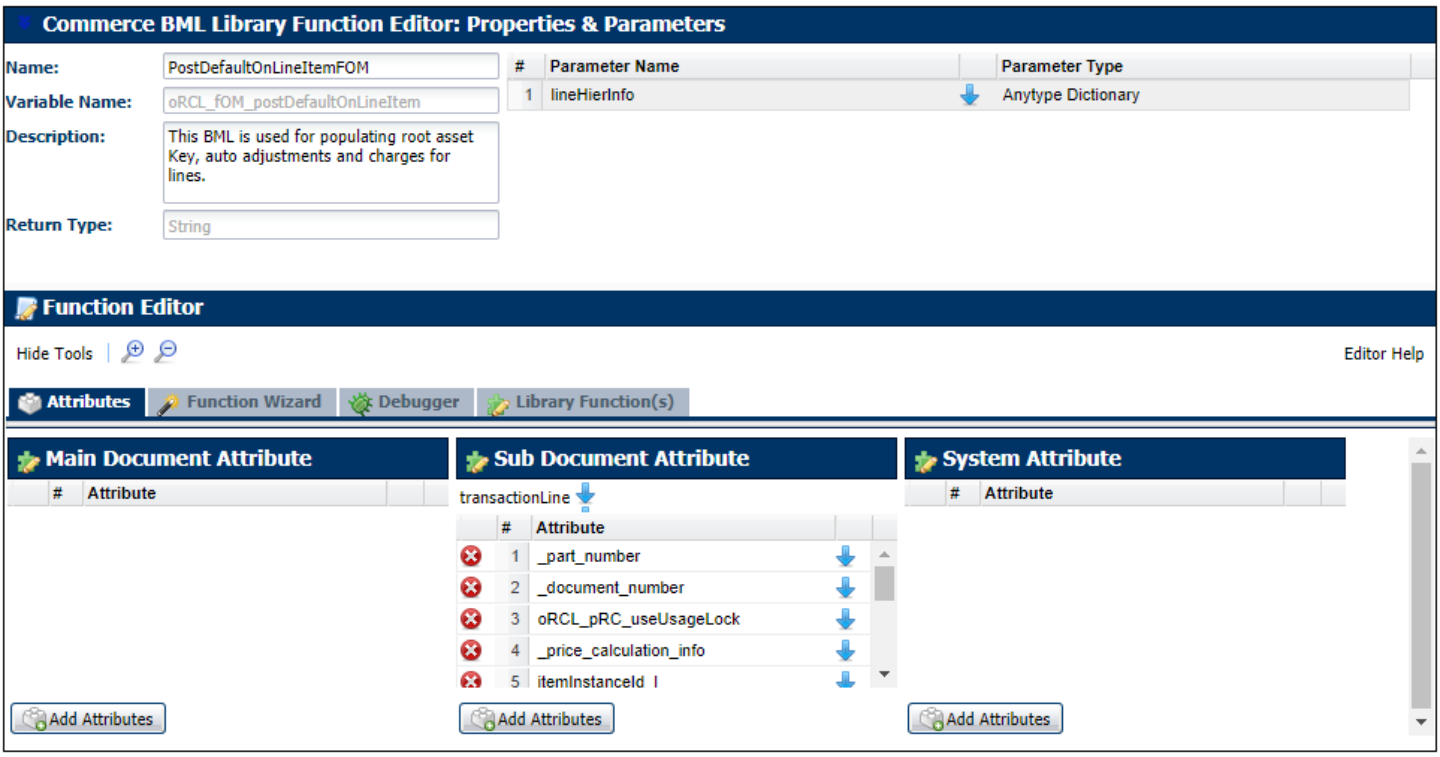
```
    // Calculating Unit Price
    if(quantity <> 0){
      unitPrice = listPrice / quantity;
    }

  }else{
    if(tierType == "ORA_ALL_TIERS"){
      prevMax = 0;
      tierSeq = 1;
      // Checking charge is of type lockable, if yes and locked then calculating the price based on locked
values
      if(chargeType == "ORA_RECURRING_USAGE" AND lockUsage) {
        tierSize= sizeofarray(tierFrom);
        tierLoopArr = range(tierSize);
        for tierNum in tierLoopArr{
          tierMin = tierFrom[tierNum];
          tierMax = tierTo[tierNum];
          blockSize = tierBlockSize[tierNum];
          tierPrice = tierListPrice[tierNum];
          if(tierMax == 0 OR (tierMax >= quantity)){ // True if we are at the last row of the tier.
            iterationQuantity = quantity - prevMax; // Get the quantity to charge for this tier.
            if(iterationQuantity < 0){
              iterationQuantity = 0;
            }
          }else{ // True if the tier still has more rows after first tier.
              iterationQuantity = tierMax - prevMax; // Get the quantity to charge for this row.
          }

          listPrice = listPrice + util.oRCL_fOM_calculateListPrice(iterationQuantity, allowance,
blockSize, tierPrice);

          if(quantity <> 0){
            unitPrice = listPrice / quantity;
          }

          // Preparing tier info
          tierInfoJson = util.oRCL_fOM_prepareTierInfoJson(tierSeq, tierMin, tierMax, tierPrice,
blockSize);

          //Adding tier Info to tier list
          jsonarrayappend(tierList, tierInfoJson);

          prevMax = tierMax;  // Set the previous max for the if statement above on the next loop.
          tierSeq = tierSeq + 1; // Increment the tier sequence number for the next iteration of the loop.
          if(tierMax <= 0){ // stop processing tiers if <= 0 since that is used as a max identifier
            break;
          }
        }
      }else{
        // If tiered pricing then querying ORCL_PRC_BASE_TIERS data table to get the tier info.
        tiers = bmql("select Tier_Min, Tier_Max, Block_Size, Price from ORCL_PRC_BASE_TIERS where
Charge_Id = $chargeId ORDER BY Tier_Min ASC");
```

```
        for tier in tiers { // Iterating through tiers
            // Reading the tier info
          tierMin = getint(tier, "Tier_Min");
          tierMax = getint(tier, "Tier_Max");
          blockSize = getint(tier, "Block_Size");
          tierPrice = getfloat(tier, "Price");
          if(tierMax == 0 OR (tierMax >= quantity)){ // True if we are at the last row of the tier.
            iterationQuantity = quantity - prevMax; // Get the quantity to charge for this tier.
            if(iterationQuantity < 0){
              iterationQuantity = 0;
            }
          }else{ // True if the tier still has more rows after first tier.
              iterationQuantity = tierMax - prevMax; // Get the quantity to charge for this row.
          }

          listPrice = listPrice + util.oRCL_fOM_calculateListPrice(iterationQuantity , allowance,
blockSize , tierPrice);

          if(quantity <> 0){
            unitPrice = listPrice / quantity;
          }
          // Preparing tier info
          tierInfoJson = util.oRCL_fOM_prepareTierInfoJson(tierSeq, tierMin, tierMax, tierPrice,
blockSize);

          //Adding tier Info to tier list
          jsonarrayappend(tierList, tierInfoJson);

          prevMax = tierMax;  // Set the previous max for the if statement above on the next loop.
tierSeq = tierSeq + 1; // Increment the tier sequence number for the next iteration of the loop.
          if(tierMax <= 0){ // stop processing tiers if <= 0 since that is used as a max identifier
            break;
          }
        }
      }
    }
    if(tierType == "ORA_HIGHEST_TIER"){
      tierSeq = 1;
      // Checking charge is of type lockable, if yes and locked then calculating the price based on locked
values
      if(chargeType == "ORA_RECURRING_USAGE" AND lockUsage) {
        tierSize= sizeofarray(tierFrom);
        tierLoopArr = range(tierSize);
        for tierNum in tierLoopArr{
          tierMin = tierFrom[tierNum];
          tierMax = tierTo[tierNum];
          blockSize = tierBlockSize[tierNum];
          tierPrice = tierListPrice[tierNum];

          if(tierMin <= quantity AND (tierMax >= quantity OR tierMax == 0)){ // True if we are at the last
row of the tier.
```

```
            listPrice = listPrice + util.oRCL_fOM_calculateListPrice(quantity, allowance, blockSize,
tierPrice);

            if(quantity <> 0){
              unitPrice = listPrice / quantity;
            }
          }
// Preparing tier info
          tierInfoJson = util.oRCL_fOM_prepareTierInfoJson(tierSeq, tierMin, tierMax, tierPrice,
blockSize);

          //Adding tier Info to tier list
          jsonarrayappend(tierList, tierInfoJson);
          tierSeq = tierSeq + 1; // Increment the tier sequence number for the next iteration of the loop.

        }
      }else{
        // If tiered pricing then querying ORCL_PRC_BASE_TIERS data table to get the tier info.
        tiers = bmql("select Tier_Min, Tier_Max, Block_Size, Price from ORCL_PRC_BASE_TIERS where
Charge_Id = $chargeId ORDER BY Tier_Min ASC");

        for tier in tiers { // Iterating through tiers
            // Reading the tier info
          tierMin = getint(tier, "Tier_Min");
          tierMax = getint(tier, "Tier_Max");
          blockSize = getint(tier, "Block_Size");
          tierPrice = getfloat(tier, "Price");
          if(tierMin <= quantity AND (tierMax >= quantity OR tierMax == 0)){ // True if we are at the last
row of the tier.
            listPrice = listPrice + util.oRCL_fOM_calculateListPrice(quantity, allowance, blockSize,
tierPrice);

            if(quantity <> 0){
              unitPrice = listPrice / quantity;
            }
          }
          // Preparing tier info
          tierInfoJson = util.oRCL_fOM_prepareTierInfoJson(tierSeq, tierMin, tierMax, tierPrice,
blockSize);

          //Adding tier Info to tier list
jsonarrayappend(tierList, tierInfoJson);

          tierSeq = tierSeq + 1; // Increment the tier sequence number for the next iteration of the loop.

        }
      }
    }
  }
  periods = util.oRCL_fOM_calculatePeriod(startDate, endDate, periodicity, chargeType);
  unitNetPrice = util.oRCL_fOM_calculateAutoAdjustments(product, chargeNameValue, unitListPrice,
chargeType, periodicity, periods);
  netPrice = unitNetPrice * quantity;
```

```
  sequenceNumber = sequenceNumber + 1;
  unitAutoAdjustment = util.oRCL_fOM_calculateUnitTotalAutoAdjustment(product, chargeNameValue,
unitListPrice);
  totalAutoAdjustment = util.oRCL_fOM_calculateTotalAutoAdjustment(product, chargeNameValue,
unitListPrice, periods, chargeType, quantity);
  if(chargeType == "ORA_RECURRING" OR chargeType == "ORA_RECURRING_USAGE") {
    listPrice = listPrice * periods;
    unitListPrice = unitListPrice * periods;
  }

  // Preparing the charge details
  jsonput(chargeJson, "chargeName", chargeNameValue);
  jsonput(chargeJson, "unitPrice", unitPrice);
  jsonput(chargeJson, "listPrice", listPrice);
  jsonput(chargeJson, "unitListPrice",unitListPrice);
  jsonput(chargeJson, "chargeType", chargeType);
  jsonput(chargeJson, "periodicity", periodicity);
  jsonput(chargeJson, "netPrice", netPrice);
  jsonput(chargeJson, "sequenceNumber", sequenceNumber);
  jsonput(chargeJson, "unitNetPrice", unitNetPrice);
  jsonput(chargeJson, "applyTo", applyTo);
  jsonput(chargeJson, "autoAdj", unitAutoAdjustment);
  jsonput(chargeJson, "periods", periods);
  jsonput(chargeJson, "totalAutoAdj", totalAutoAdjustment);
  jsonput(chargeJson, "primaryCharge", primaryCharge);

  if(tiered == "Y"){ // Include Tier information if tiered pricing.
    jsonput(chargeJson, "tiered", "Y");
    jsonput(chargeJson, "tierList", tierList);
    jsonput(chargeJson, "tierType", tierType);
  }else{
    jsonput(chargeJson, "Allowance", allowance );
    jsonput(chargeJson, "BlockSize", blockSize );
  }
  // Adding charge details to charge list
  jsonarrayappend(chargeList , chargeJ son);
}
calculationInfoPayload = json();

jsonput(returnPayload, "unitPrice", 0.0);
jsonput(calculationInfoPayload , "charges", chargeList);

jsonput(calculationInfoPayload, "hasErrors", errorInPricing); // Adding the hasErrors item
if(errorInPricing){ // Including Error information if there are errors
  jsonput(calculationInfoPayload, "errorInfo", errorMessagesJsonArr);
}
jsonput(returnPayload, "calculationInfo", calculationInfoPayload ); // Adding the calculationInfo child to
the payload.
return returnPayload;
```

Post Default on Line Item FOM (oRCL_fOM_postDefaultOnLineItem)



Script:

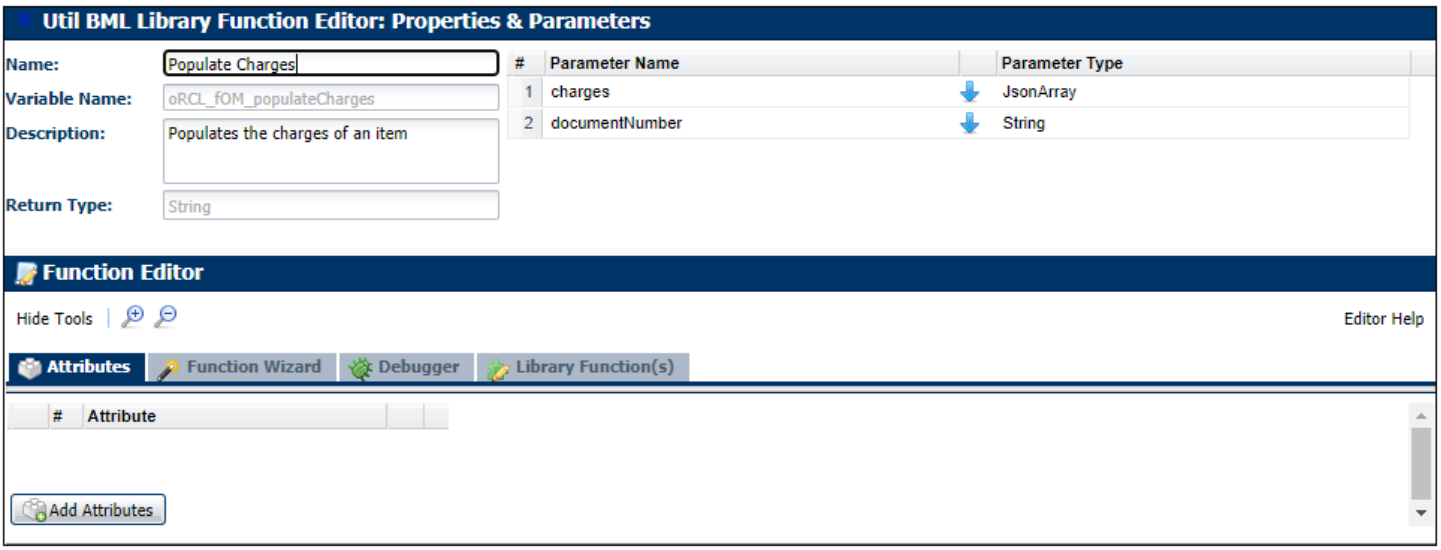
```
//Name : PostDefaultOnLineItemSM
//Variable Name : oRCL_sm_postDefaultOnLineItem
//Description:This BML is used for populating root asset Key, auto adjustments and charges for lines.
//Invoked from sub doc ->Advanced Default->After formula
//Imported Sub doc attributes : _part_number, _document_number,oRCL_pRC_useUsageLock,
_price_calculation_info
//Input Parameters:
//      "lineHierInfo" - dict("anytype"); result of AboBuildLineItemHierarchy execution
// Output: String
// Dependency : None

returnString = "";
docNumToRootDocNum = get(lineHierInfo, "DocNumToRootDocNum", "dict<string>");
docNumToAttrs = get(lineHierInfo, "DocNumToAttrs", "dict<anytype>");

for line in transactionLine {
    if(line.oRCL_pRC_useUsageLock == false AND line._price_calculation_info <> "") {
  calcInfo = jsonarrayget(jsonarray(line._price_calculation_info),0,"json");
  charges = jsonget(calcInfo,"charges", "jsonArray");
  returnString = returnString + "|"+util.oRCL_fOM_populateCharges(charges, line._document_number);
  returnString = returnString + commerce.calculateRollUpCharges(charges,  line._document_number);
    } elif (line._part_number == "" AND line._model_name <> "") {
      tierFrom = integer[1];
  tierListPrice = float[1];
  startDate = commerce.convertLibraryFunctionDateStringToDate(line.contractStartDate_l);
  endDate = commerce.convertLibraryFunctionDateStringToDate(line.contractEndDate_l);
      calculationInfo = util.oRCL_fOM_oraclePricingFOMIntegration(line._model_name,
line.requestedQuantity_l, "", 0, 0, 0.0, tierFrom, tierFrom, tierListPrice, tierFrom, false, startDate,
endDate);
  calcInfo =  jsonget(calculationInfo, "calculationInfo", "json");
  charges =  jsonget(calcInfo, "charges", "jsonarray");
  returnString = returnString + "|"+util.oRCL_fOM_populateCharges(charges, line._document_number);
```

```
  returnString = returnString + commerce.calculateRollUpCharges(charges,  line._document_number);
    }
    autoAdjustments = jsonArray();
    partNumber = line._part_number;
    if (line._part_number == "" AND line._model_name <> "") {
        partNumber = line._model_name;
    }
    adjustmentList = bmql("select Charge_Name,
Adjustment_Name,Adjustment_Amount,Adjustment_Type,Adjustment_Reason,Adjustment_Type,Adjustment_Basis,Adjus
t_Effectivity,Charge_Periods,Auto_Adjustment,Charge_Type from ORCL_PRC_ADJUSTMENTS where
Product=$partNumber");
    for adjustment in adjustmentList {
    adjustmentJson = json();
    jsonput(adjustmentJson,"oRCL_auto_adjustmentSequence_l",1);
    jsonput(adjustmentJson,"oRCL_auto_adjustmentName_l",get(adjustment,"Adjustment_Name"));
    jsonput(adjustmentJson,"oRCL_auto_adjustmentChargeName_l",get(adjustment,"Charge_Name"));
    jsonput(adjustmentJson,"oRCL_auto_adjustmentType_l",get(adjustment,"Adjustment_Type"));
    jsonput(adjustmentJson,"oRCL_auto_adjustmentBasis_l",get(adjustment,"Adjustment_Basis"));
    jsonput(adjustmentJson,"oRCL_auto_adjustmentEffectivity_l",get(adjustment,"Adjust_Effectivity"));
    jsonput(adjustmentJson,"oRCL_auto_numberOfPeriods_l",get(adjustment,"Charge_Periods"));
    jsonput(adjustmentJson,"oRCL_auto_adjustmentValue_l",get(adjustment,"Adjustment_Amount"));
      jsonput(adjustmentJson,"oRCL_autoAdjustmentReason",get(adjustment,"Adjustment_Reason"));
      jsonput(adjustmentJson,"oRCL_autoAdjustmentChargeType_l",get(adjustment,"Charge_Type"));
    jsonarrayappend(autoAdjustments, adjustmentJson);
  }
  returnString = returnString +"|"+ line._document_number + "~oRCL_autoAdjustments~" +
jsonarraytostr(autoAdjustments) + "|";

  //populate rotAssetKey_l
  rootAssetKey="";
  if(containskey(docNumToRootDocNum , line._document_number)){
      rootDocNumber = get(docNumToRootDocNum, line._document_number);
      lineAttrMap = get(docNumToAttrs, rootDocNumber ,"dict<string>");

      rootAssetKey =  get(lineAttrMap, "itemInstanceId_l");

  }
  if(isnull(rootAssetKey) OR rootAssetKey == ""){
    //Simple product case
    if ( len(line._model_variable_name) < 1 AND len(line._parent_doc_number) < 1 AND
              len(line._line_bom_id) > 1  ){
              lineAttrMap = get(docNumToAttrs, line._document_number ,"dict<string>");
              rootAssetKey =  get(lineAttrMap, "itemInstanceId_l" );
          }
  }
  returnString = returnString +"|"+ line._document_number+"~rootAssetKey_l~"+rootAssetKey;
}
return returnString;
```

The Post Default on Line Item FOM BML internally calls the following BMLs.

Populate Charges (oRCL_fOM_populateCharges)

Populates the charges array:

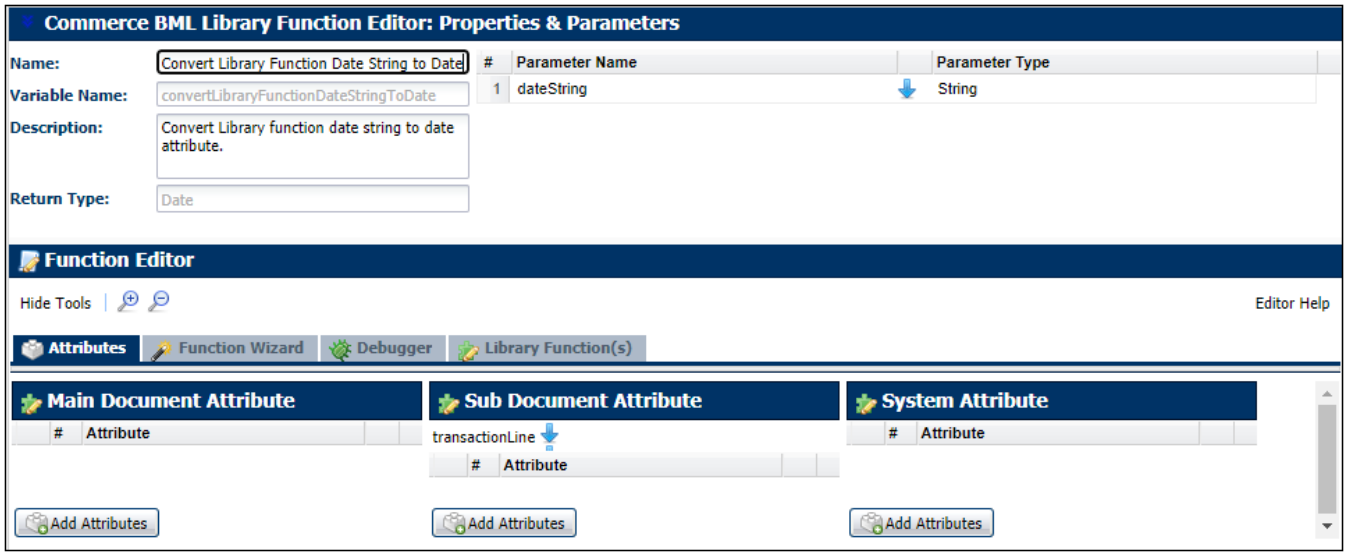


Script:

```
// Populate the charges array and tierInfo arrays from calculated info
//charges, documentNumber
//util.oRCL_fOM_populateTiers(tierList, documentNumber)

returnString = "";
chargesArray = jsonArray();

chargesSize = jsonarraysize(charges);
itr = string[chargesSize];
chargecount=0;
for i in itr {
  charge = jsonArrayget(charges, chargecount, "json");
  chargecount = chargecount + 1;
  chargeJson = json();
  jsonput(chargeJson,"oRCL_chargeName",jsonget(charge,"chargeName","string"));
  jsonput(chargeJson,"oRCL_chargeType",jsonget(charge,"chargeType","string"));
  if(jsonget(charge,"periodicity","string") <> ""){
    jsonput(chargeJson,"oRCL_periodicity",jsonget(charge,"periodicity","string"));
  }
  if(jsonpathcheck(charge,"$.Allowance")){
    jsonput(chargeJson,"oRCL_allowance",jsonget(charge,"Allowance","integer"));
  }
  if(jsonpathcheck(charge,"$.BlockSize")){
    jsonput(chargeJson,"oRCL_blockSize", jsonget(charge,"BlockSize","integer"));
  }
  if(NOT(isnull(jsonget(charge,"unitPrice")))){
    jsonput(chargeJson,"oRCL_unitPrice",jsonget(charge,"unitPrice","float"));
  }
  if(NOT(isnull(jsonget(charge,"listPrice")))){
    jsonput(chargeJson,"oRCL_listPrice",jsonget(charge,"listPrice","float"));
  }
  if(NOT(isnull(jsonget(charge,"unitListPrice")))){
    jsonput(chargeJson,"oRCL_unitListPrice",jsonget(charge,"unitListPrice","float"));
```

```
  }
  if(NOT(isnull(jsonget(charge,"netPrice")))){
    jsonput(chargeJson,"oRCL_netPrice",jsonget(charge,"netPrice","float"));
  }
  if(NOT(isnull(jsonget(charge,"sequenceNumber")))){
    jsonput(chargeJson,"oRCL_chargeSequenceNumber",jsonget(charge,"sequenceNumber","integer"));
  }
  if(NOT(isnull(jsonget(charge,"unitNetPrice")))){
    jsonput(chargeJson,"oRCL_unitNetPrice",jsonget(charge,"unitNetPrice","float"));
  }
  if(NOT(isnull(jsonget(charge,"applyTo")))){
    jsonput(chargeJson,"oRCL_applyTo",jsonget(charge,"applyTo","string"));
  }
  if(NOT(isnull(jsonget(charge,"manualAdj")))){
    jsonput(chargeJson,"oRCL_manualAdjustment",jsonget(charge,"manualAdj","float"));
  }
  if(NOT(isnull(jsonget(charge,"autoAdj")))){
    jsonput(chargeJson,"oRCL_autoAdjustment",jsonget(charge,"autoAdj","float"));
  }
  if(NOT(isnull(jsonget(charge,"periods")))){
    jsonput(chargeJson,"oRCL_periods",jsonget(charge,"periods","float"));
  }
  if(NOT(isnull(jsonget(charge,"totalAutoAdj")))){
    jsonput(chargeJson,"oRCL_totalAutoAdjustment",jsonget(charge,"totalAutoAdj","float"));
  }
  if(NOT(isnull(jsonget(charge,"totalManualAdj")))){
    jsonput(chargeJson,"oRCL_totalManualAdjustment",jsonget(charge,"totalManualAdj","float"));
  }
  if(NOT(isnull(jsonget(charge,"primaryCharge")))){
    jsonput(chargeJson,"oRCL_primaryCharge",jsonget(charge,"primaryCharge","boolean"));
  }
  tiered = "N";
  if(NOT(isnull(jsonget(charge ,"tiered")))){
    tiered = jsonget(charge ,"tiered", "string");
    jsonput(chargeJson,"oRCL_tiered",tiered );
  }else{
    jsonput(chargeJson,"oRCL_tiered","N");
  }
  if(tiered == "Y") {
    jsonput(chargeJson,"oRCL_tierType",jsonget(charge,"tierType"));
    tierList = jsonget(charge, "tierList", "jsonArray");
    if(isnull(tierList) == false){
      returnString = util.oRCL_fOM_populateTiers(tierList, documentNumber);
    }
    returnString = returnString + "|" + documentNumber + "~oRCL_pRC_tierd_l~" + tiered + "|";
  }
  jsonarrayappend(chargesArray, chargeJson);
}
returnString = returnString + documentNumber + "~oRCL_charges~" + jsonarraytostr(chargesArray) + "|";
return returnString;
```

Convert Library Function Date String to Date (convertLibraryFunctionDateStringToDate)

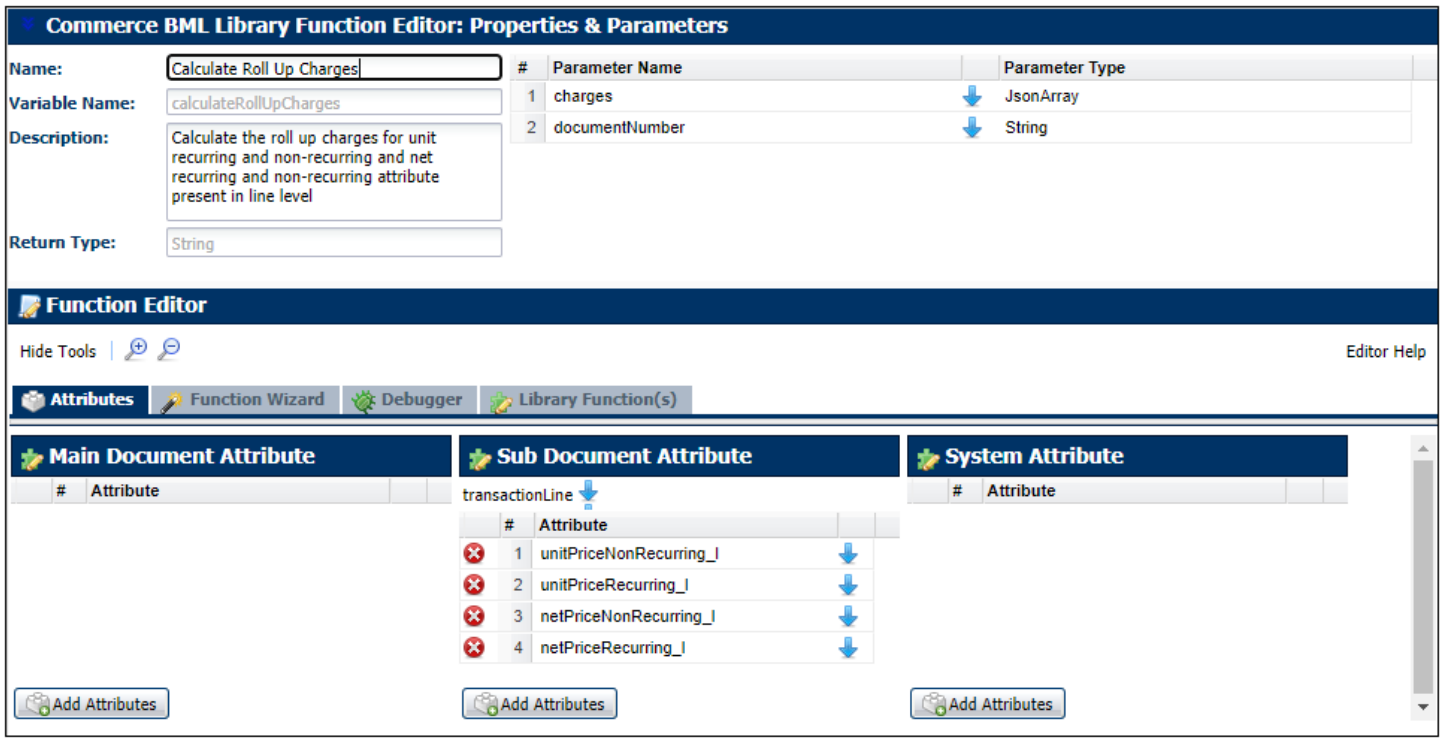


Script:

```
dateInInputFormat = strtodate(dateString,"%Y-%m-%d %H:%M:%S");
dateStringInrequiredFormat = datetostr(dateInInputFormat,"MM/dd/yyyy HH:mm:ss");
returnDate =  strtojavadate(dateStringInrequiredFormat, "MM/dd/yyyy HH:mm:ss");
return returnDate;
```

CalculateRollUpCharges (calculateRollUpCharges)

Calculates roll up charges



Script:

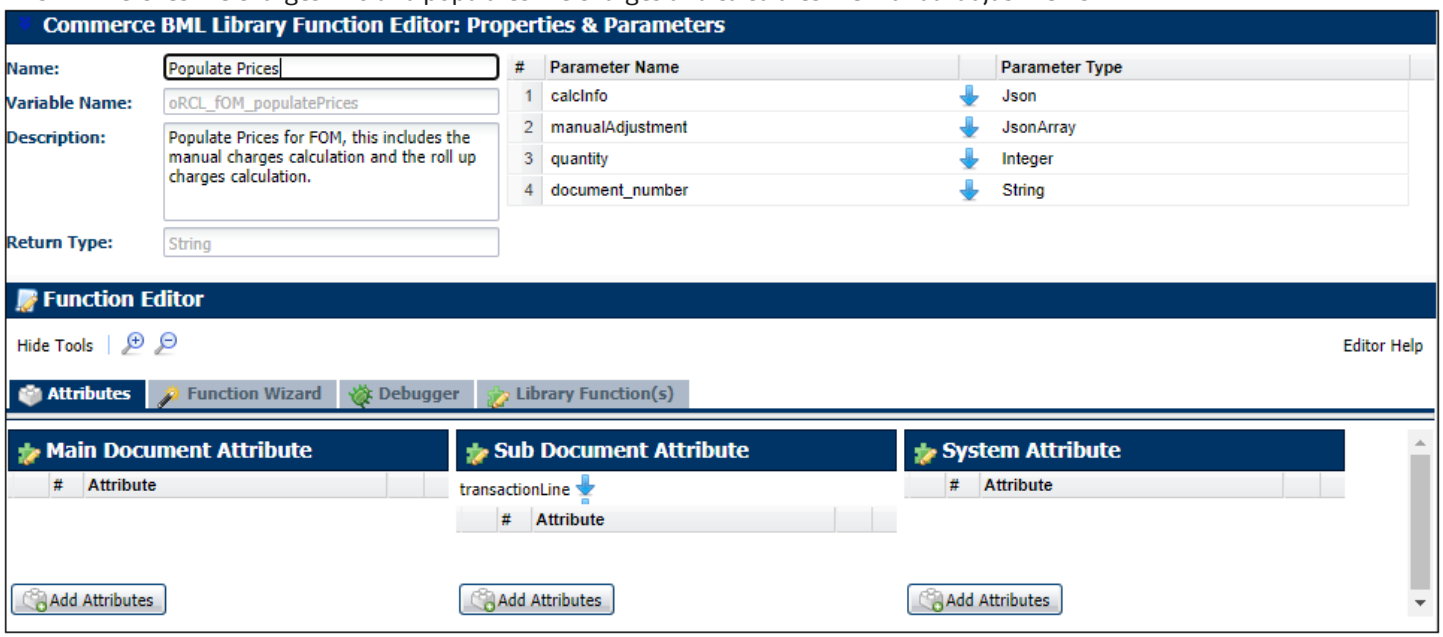
```
returnString = "";
document_number = documentNumber;
unitPriceNonRecurring = 0.0;
netPriceNonRecurring = 0.0;
unitPriceRecurring = 0.0;
netPriceRecurring = 0.0;
chargesSize = string[jsonarraysize(charges)];
count = 0;
for ch in chargesSize {
  charge = jsonarrayget(charges,count,"json");
  chargeType = jsonget(charge,"chargeType");
  unitPrice = jsonget(charge,"unitPrice", "float");
  netPrice = jsonget(charge,"netPrice", "float");
  if(chargeType == "ORA_ONE_TIME") {
    unitPriceNonRecurring = unitPriceNonRecurring + unitPrice;
    netPriceNonRecurring = netPriceNonRecurring + netPrice;
  } else {
    unitPriceRecurring = unitPriceRecurring + unitPrice;
    netPriceRecurring = netPriceRecurring + netPrice;
  }
  count = count + 1;
}
returnString = returnString +"|"+ document_number+"~unitPriceNonRecurring_l~" +
string(round(unitPriceNonRecurring,2));
returnString = returnString +"|"+ document_number+"~netPriceNonRecurring_l~" +
string(round(netPriceNonRecurring,2));
returnString = returnString +"|"+ document_number+"~unitPriceRecurring_l~" +
string(round(unitPriceRecurring,2));
returnString = returnString +"|"+ document_number+"~netPriceRecurring_l~" +
string(round(netPriceRecurring,2));
return returnString;
```

Manual Adjustment Calculations

Manual adjustments are calculated during the Save action. This BML is invoked from the Save action Advanced Default - After Formulas. This BML internally calls BML Populate Prices

Populate Prices (oRCL_fOM_populatePrices)

This BML fetches the charges info and populates the charges and calculates the manual adjustments.



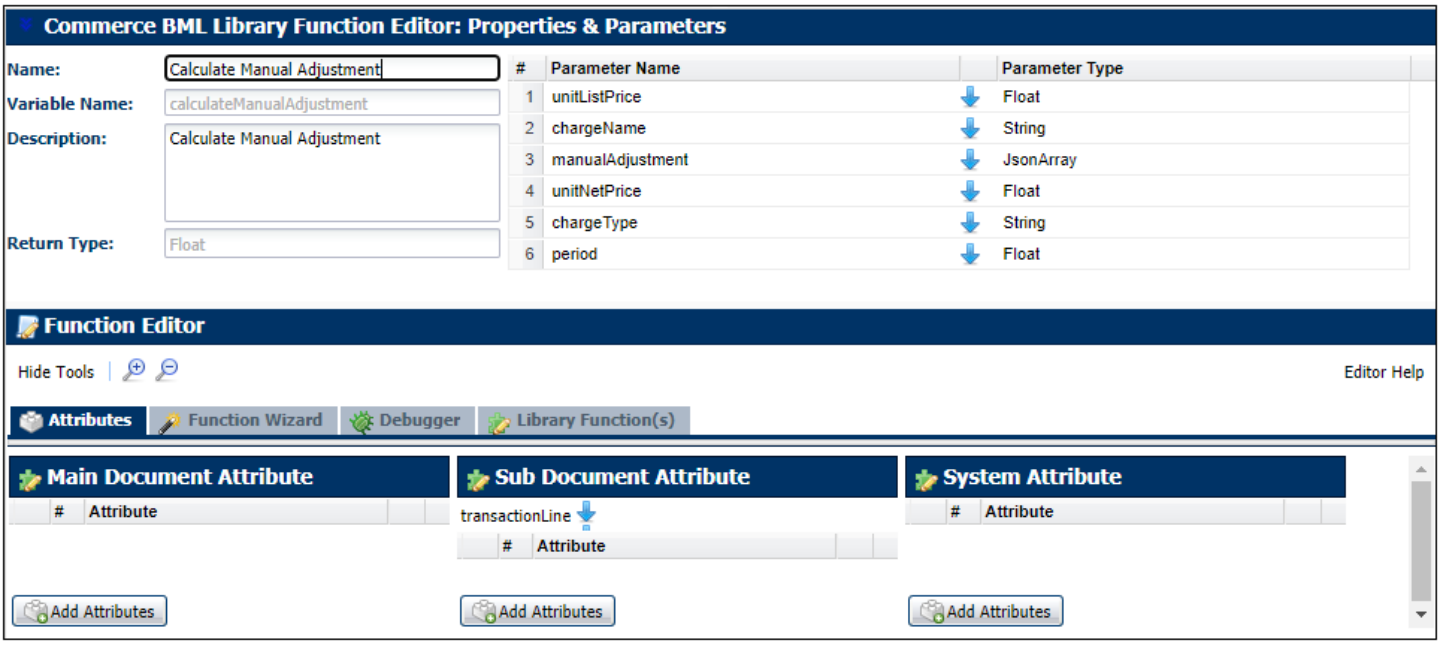
Script:

```
//At TrnsactionLine level save
//System Variable Name    Type    Description
//_system_current_document_number String  Current Document Number

//Variable Name for (Transaction Line)    Type    Description
//transactionLine Collection of Sub Documents
//      _document_number  String  Document Number
//      _price_calculation_info   String  Calculation Information
//      oRCL_pRC_useUsageLock Boolean Use Usage Lock
//      oRCL_pRC_tierd_l  String  Tiered

//Imported Util Functions
//String _SM.oRCL_pRC_populateCharges(JsonArray charges, String documentNumber)

returnString = "";
for line in transactionLine {
  if(((line.oRCL_pRC_useUsageLock == false OR line.oRCL_pRC_tierd_l == "N") AND
line._price_calculation_info <> "")) {
    calcInfo = jsonarrayget(jsonarray(line._price_calculation_info),0,"json");
    print calcInfo;
    returnString = commerce.oRCL_populateFOMPrices(calcInfo, line.oRCL_manualAdjustments,
line.requestedQuantity_l, line._document_number);
  } elif (line._part_number == "") {
    tierFrom = integer[1];
    tierListPrice = float[1];
    startDate = strtojavadate(line.contractStartDate_l , "MM/dd/YYYY HH:mm:ss");
    endDate = strtojavadate(line.contractEndDate_l , "MM/dd/YYYY HH:mm:ss");
        calculationInfo = util.oRCL_fOM_oraclePricingFOMIntegration(line._model_name,
line.requestedQuantity_l, "Sale Price", 0, 0, 0.0, tierFrom, tierFrom, tierListPrice, tierFrom, false,
startDate, endDate);
    calcInfo =  jsonget(calculationInfo, "calculationInfo", "json");
    returnString = commerce.oRCL_populateFOMPrices(calcInfo, line.oRCL_manualAdjustments,
line.requestedQuantity_l, line._document_number);
  }
}
return returnString;
```

Manual Adjustments are calculated based on the following BMLs

Calculate Manual Adjustment (calculateManualAdjustment)

Calculates the total manual adjustment.

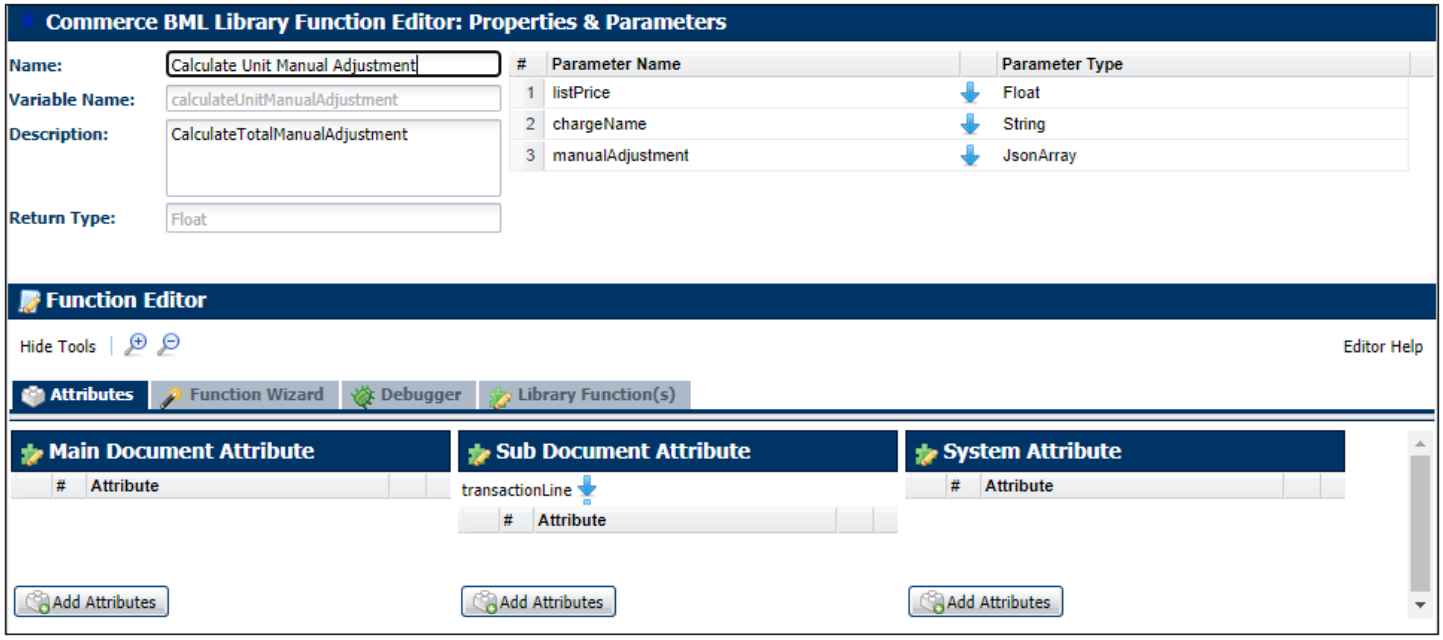


Script:

```
returnString = "";
charges = jsonget(calcInfo,"charges", "jsonArray");
chargesSize = string[jsonarraysize(charges)];
count = 0;
netPriceTotal = 0.0;
listPriceTotal = 0.0;
totalManualAdj = 0.0;
sumManualAdj = 0.0;
sumAutoAdj = 0.0;
manualAdjustments = manualAdjustment;
for ch in chargesSize {
    charge = jsonarrayget(charges,count,"json");
    netPrice = jsonget(charge,"unitNetPrice", "float");
    listPrice = jsonget(charge,"listPrice", "float");
    unitListPrice = jsonget(charge,"unitListPrice", "float");
    unitNetPrice = jsonget(charge,"unitNetPrice", "float");
    chargeName = jsonget(charge,"chargeName");
    chargeType = jsonget(charge,"chargeType");
    period = jsonget(charge,"periods", "float");
    if(chargeType == "ORA_RECURRING" OR chargeType == "ORA_RECURRING_USAGE") {
      listPrice = listPrice * period;
    }
    manualAdjSize = jsonarraysize(manualAdjustment);
    if(manualAdjSize >= 1){
      unitNetPrice = commerce.calculateManualAdjustment(unitListPrice/period , chargeName,
manualAdjustment, (unitNetPrice/period), chargeType, period);
    }
    netPrice = unitNetPrice * quantity;
    netPriceTotal = netPriceTotal + netPrice;
    listPriceTotal = listPriceTotal + listPrice;
    jsonput(charge,"netPrice",netPrice);
    jsonput(charge,"unitNetPrice",unitNetPrice);
    unitManualAdj = commerce.calculateUnitManualAdjustment(unitListPrice/period, chargeName,
manualAdjustment);
    jsonput(charge,"manualAdj", unitManualAdj);
    totalManualAdj = commerce.calculateTotalManualAdjustment(unitListPrice/period, chargeName,
manualAdjustment, period, chargeType, quantity);
    jsonput(charge,"totalManualAdj", totalManualAdj);
    count = count + 1;
}
manualAdjustments = commerce.getChargeTypeForManualAdjustment(manualAdjustments, charges);
returnString = returnString + util.oRCL_fOM_populateCharges(charges, document_number) + "|";
returnString = returnString + document_number + "~oRCL_manualAdjustments~" +
jsonarraytostr(manualAdjustments) + "|";
returnString = returnString + commerce.calculateRollUpCharges(charges, document_number);
return returnString;
```

Calculate Unit Manual Adjustment (calculateUnitManualAdjustment)

Calculates the manual adjustment per unit.

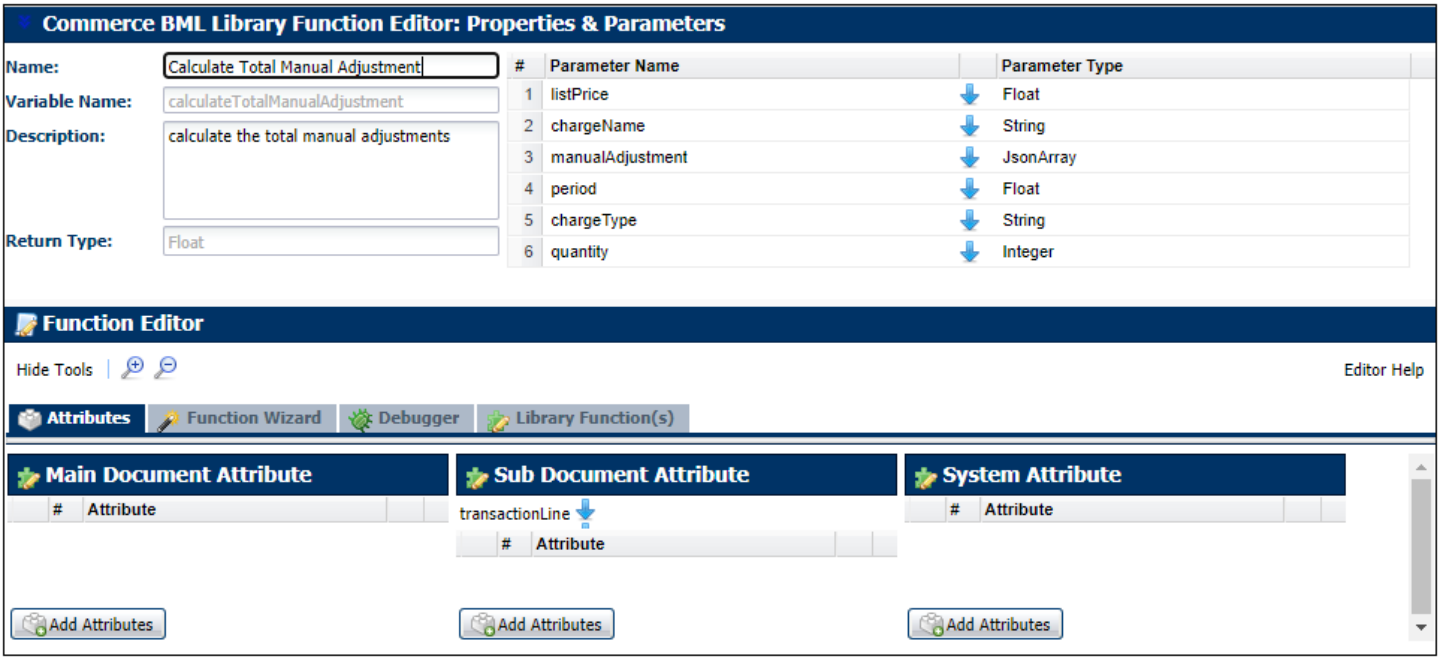


Script:

```
totalAdjustment = 0.0;
index = 0;
manualAdjSize = string[jsonarraysize(manualAdjustment)];
for adj in manualAdjSize {
  adjustment = jsonarrayget(manualAdjustment,index,"json");
  if(chargeName == jsonget(adjustment,"oRCL_manual_adjustmentChargeName_l")) {
    adjustedAmount = jsonget(adjustment,"oRCL_manual_adjustmentValue_l", "float");
    adjustmentType = jsonget(adjustment,"oRCL_manual_adjustmentType_l");
    if (adjustmentType == "ORA_DISCOUNT_PERCENT") {
      adjustedAmount = fabs(adjustedAmount) * listPrice /100;
    }
    totalAdjustment = totalAdjustment + adjustedAmount;
  }
  index = index + 1;
}
return totalAdjustment;
```

Calculate Total Manual Adjustment (calculateTotalManualAdjustment)

Calculates total manual adjustments

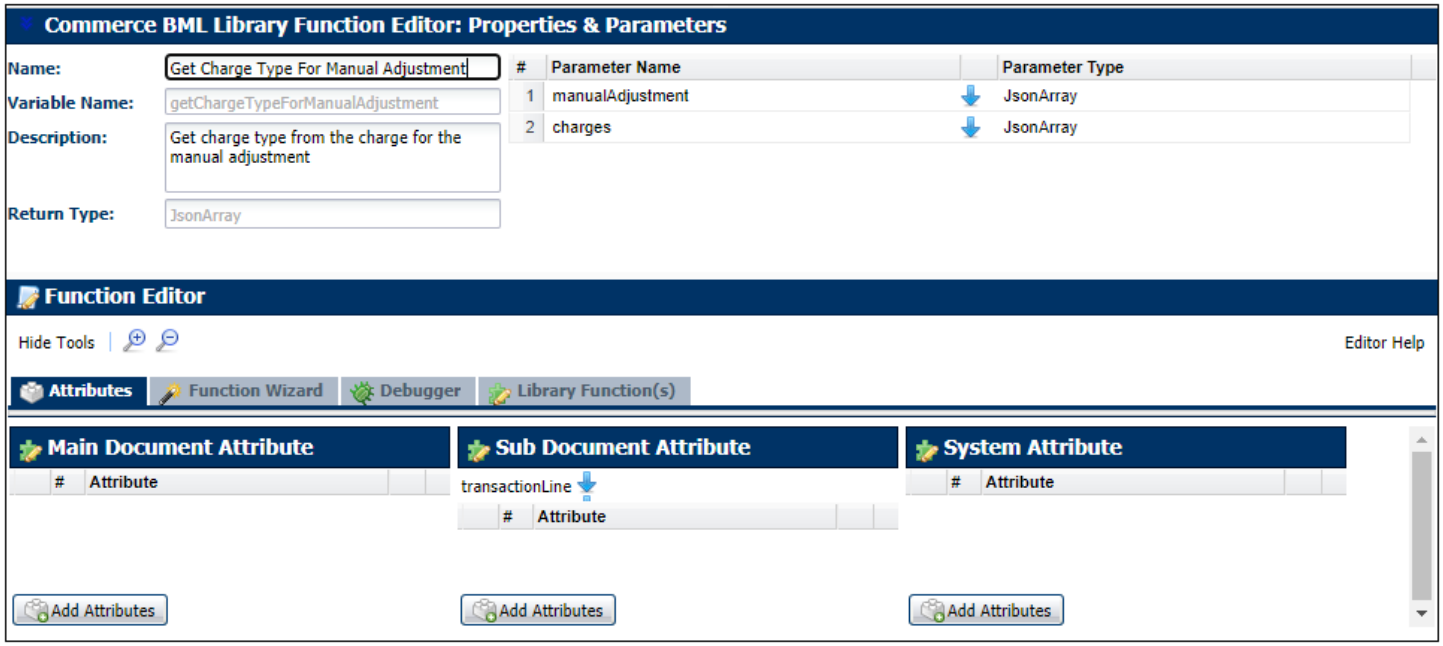


Script:

```
totalAdjustment = 0.0;
index = 0;
manualAdjSize = string[jsonarraysize(manualAdjustment)];
for adj in manualAdjSize {
  adjustment = jsonarrayget(manualAdjustment,index,"json");
  if(chargeName == jsonget(adjustment,"oRCL_manual_adjustmentChargeName_l")) {
    adjustedAmount = jsonget(adjustment,"oRCL_manual_adjustmentValue_l", "float");
    adjustmentType = jsonget(adjustment,"oRCL_manual_adjustmentType_l");
    adjustmentEffectivity = jsonget(adjustment,"oRCL_manual_adjustmentEffectivity_l");
    adjustmentPeriod = 0;
    if (adjustmentType == "ORA_DISCOUNT_PERCENT") {
      adjustedAmount = fabs(adjustedAmount) * listPrice /100;
    }
    totalAdjustment = totalAdjustment + adjustedAmount;

    if(chargeType == "ORA_RECURRING" OR chargeType == "ORA_RECURRING_USAGE") {
      if(adjustmentEffectivity == "ORA_ALL_TERM") {
        totalAdjustment = totalAdjustment * period;
      } else {
        adjustmentPeriod  =jsonget(adjustment,"oRCL_manual_numberofPeriods_l", "integer");
        totalAdjustment = totalAdjustment * adjustmentPeriod;
      }
    }
  }
  index = index + 1;
}
return totalAdjustment;
```

Get Charge Type for Manual Adjustment (getChargeTypeForManualAdjustment)

Fetches the charge type from the charges and populates the charge Type in manual adjustment



Script:

```
manualAdjustments = manualAdjustment;
manualAdjustmentsSize = string[jsonarraysize(manualAdjustments)];
count = 0;
for adj in manualAdjustmentsSize {
  adjustment = jsonarrayget(manualAdjustments,count,"json");
  adjChargeName = jsonget(adjustment,"oRCL_manual_adjustmentChargeName_l");
  adjChargeType = "";
  chargesSize = string[jsonarraysize(charges)];
  index = 0;
  for ch in chargesSize {
    charge = jsonarrayget(charges,count,"json");
    chargeName = jsonget(charge,"chargeName");
    if(adjChargeName == chargeName) {
      chargeType = jsonget(charge,"chargeType");
      jsonput(adjustment,"oRCL_manual_adjustmentChargeType_l",chargeType);
    }
    index = index + 1;
  }
  count = count + 1;
}
return manualAdjustments;
```

Appendix G: Payload Template Files

The BML associated with the payload template files referenced in Add Template Dependencies to File Manager follows.

findOrganizationPayload.txt

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body
xmlns:ns1="http://xmlns.oracle.com/apps/cdm/foundation/parties/organizationService/applicationModule/types/"
>
    <ns1:findOrganization>
      <ns1:findCriteria xmlns:ns2="http://xmlns.oracle.com/adf/svc/types/">
        <ns2:filter>
          <ns2:conjunction>And</ns2:conjunction>
          <ns2:group>
            <ns2:conjunction>And</ns2:conjunction>
            <ns2:item>
              <ns2:conjunction>And</ns2:conjunction>
              <ns2:attribute>PartyName</ns2:attribute>
              <ns2:operator>=</ns2:operator>
              <ns2:value>{{customerCompanyName_t}}</ns2:value>
            </ns2:item>
          </ns2:group>
        </ns2:filter>
      </ns1:findCriteria>
      <ns1:findControl xmlns:ns3="http://xmlns.oracle.com/adf/svc/types/">
        <ns3:retrieveAllTranslations/>
      </ns1:findControl>
    </ns1:findOrganization>
  </soap:Body>
</soap:Envelope>
```

customerAccountPayload.txt

```
<?xml version="1.0"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns1="http://xmlns.oracle.com/apps/cdm/foundation/parties/customerAccountService/applicationModule/types
/" xmlns:ns2="http://xmlns.oracle.com/adf/svc/types/">
  <soapenv:Body>
    <ns1:findCustomerAccount>
      <ns1:findCriteria>
        <ns2:filter>
          <ns2:conjunction>And</ns2:conjunction>
          <ns2:group>
            <ns2:conjunction>And</ns2:conjunction>
            <ns2:item>
              <ns2:conjunction>And</ns2:conjunction>
              <ns2:attribute>PartyId</ns2:attribute>
              <ns2:operator>=</ns2:operator>
              <ns2:value>{{returnPartyId}}</ns2:value>
            </ns2:item>
          </ns2:group>
        </ns2:filter>
      </ns1:findCriteria>
      <ns1:findControl>
        <ns2:retrieveAllTranslations>false</ns2:retrieveAllTranslations>
      </ns1:findControl>
    </ns1:findCustomerAccount>
  </soapenv:Body>
</soapenv:Envelope>
```

Appendix H: Miscellaneous Commerce Library Functions

The Oracle CPQ Order Management package adds several library functions to the Commerce process. This topic provides the BML associated with each of the library functions.

String getTemplateLocation(String system, String operation)

The code for this library function is provided the following for reference.

```
//1. Get Template File
templateUrl = "";
//bmql query
resultSet = bmql("Select Template from INT_SYSTEM_TEMPLATES where System = $system and Operation =
$operation");
//loop through the records
for record in resultSet {
templateUrl = get(record,"Template");
}
temp=split(templateUrl,"image")
return temp[1];
```

String invokeWebService(String system, String soapReq)

The code for this library function is provided the following for reference.

```
//1. Get webservice endpoint for the system
resultSet = bmql("Select Endpoint,Username,Password from INT_SYSTEM_DETAILS where System = $system");
endpoint = "";
username = "";
password = "";
//loop through the records
for record in resultSet {
 endpoint = get(record,"Endpoint");
 username = get(record,"Username");
 password = get(record,"Password");
}
//2. Invoke the web serviceheader
Values = dict("string");
put(headerValues, "Content-Type", "text/xml; charset=utf-8");
encodeCredential = encodebase64(username+":"+password);auth = "Basic " + encodeCredential;
put(headerValues,"Authorization",auth);
errorString = "Error in "+system+" invocation";
soapResponse= urldatabypost(endPoint , soapReq,errorString,headerValues,true);
// sends the soap call and returns response to variable.
print "going to print soapResponse";
//3. Return the response
return soapResponse;
```

Appendix I: CPQ-OM Status Mapping

CPQ Main Document Order Status

The CPQ - Order Management Package adds the following order status values to the main document (e.g. Transaction) `status_t` menu attribute.

Display Value	Variable Name
Draft	`DOO_DRAFT`
Processing	`OPEN`
Approval Pending	`DOO_APPROVAL_PENDING`
Closed	`CLOSED`

CPQ Sub-Document Line Status

The CPQ - Order Management Package adds the following line status values to the sub-document (e.g. Transaction Line) `status_l` menu attribute.

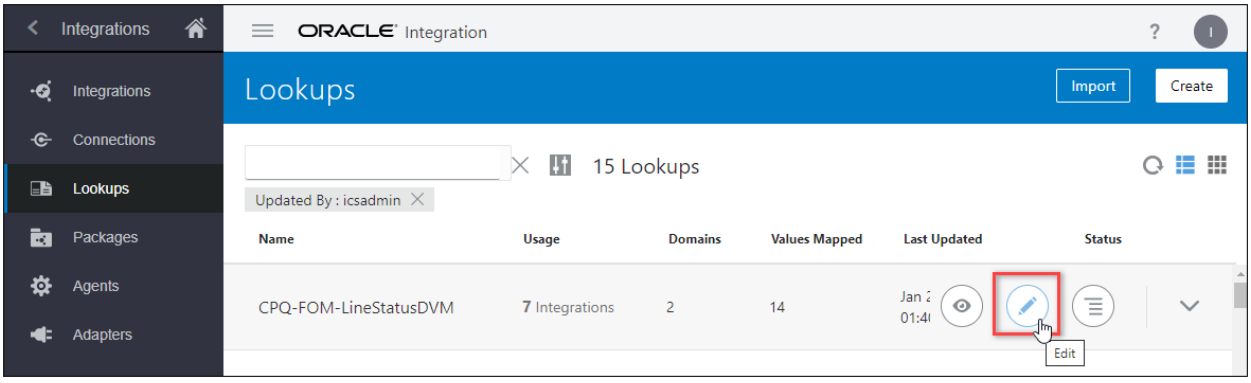
Display Value	Variable Name
Cancel Requested	`CANCEL_REQUESTED`
Deactivated	`DEACTIVATED`
Closed	`CLOSED`
Waiting	`WAITING`

Map CPQ Line Status to Order Management Line Status

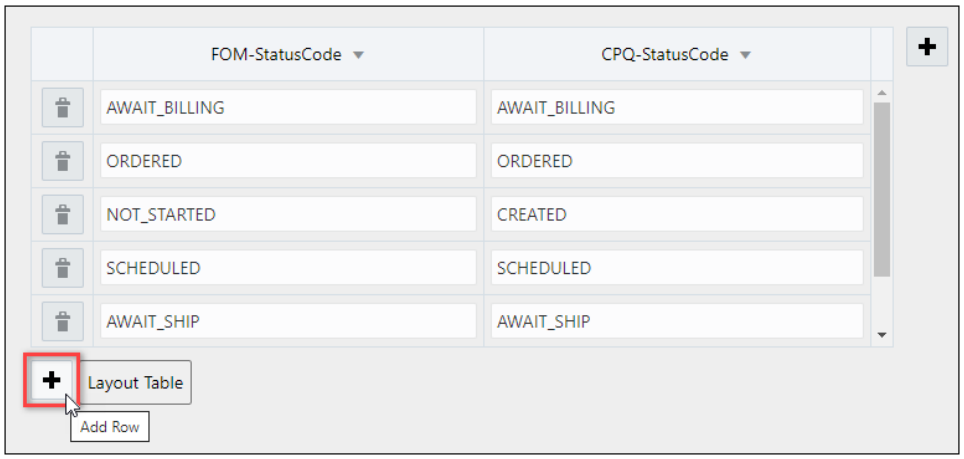
Administrators can use the CPQ-FOM-LineStatusDVM Lookup in OIC to map existing CPQ status codes with the Order Management status codes.

Perform the following steps to modify Lookups defined in OIC

1. Log in to the OIC application.
2. Select **Integrations** in the left side navigation panel, and then select **Lookups**.
3. If required, search for the applicable Lookup.
4. Click the Edit icon  for the corresponding lookup.



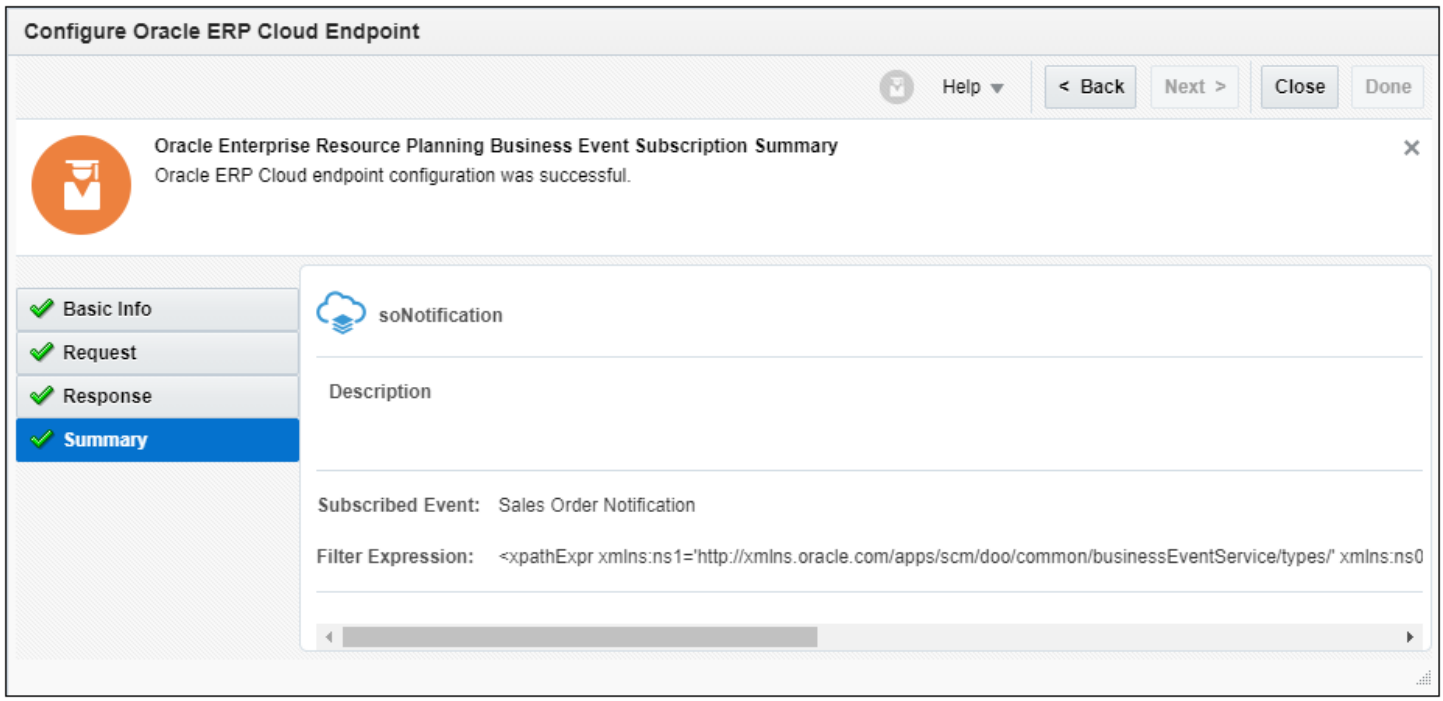
8. Click on the Add Row icon **(+)** at the bottom of the mapping table to add a new mapping.



9. Add the FOM Status code in the 'FOM-StatusCode' column and CPQ status code in the 'CPQ-StatusCode' column.
10. Click **Save** after all modification are complete.
11. A dialog window will open and list the integration which are using this lookup.
    Record those integrations, and then reactivate these integrations to apply the changes.

Appendix J: OIC Integration - UPDATESOSTATUSFROMFOM

This integration uses Oracle ERP Cloud adapter to subscribe to order management **Sales Order Notification** business event.



There is a filter expression defined to subscribe to the events for only those orders that were originated from CPQ by specifying the Source System 'ORA_BM_CPQ' in the filter expression.

In Order Management, Sales Order Notification business event is raised for an order each time any of the following condition occurs:

- Update order header status.
- Update fulfillment line status.
- Close fulfillment line.

Whenever the above events are raised, this integration is invoked and quote header status and line status are updated in CPQ. Additionally, if the quote line status matches any of the status in 'Trigger Update Asset for Line Status', the Update Asset action REST API is invoked to create an Asset in CPQ Asset Repository and update the CPQ quote line Fulfillment Status.

To enable business events in Order Management for different Fulfillment Status of order orchestration process, refer to the **Order Management Setup** section

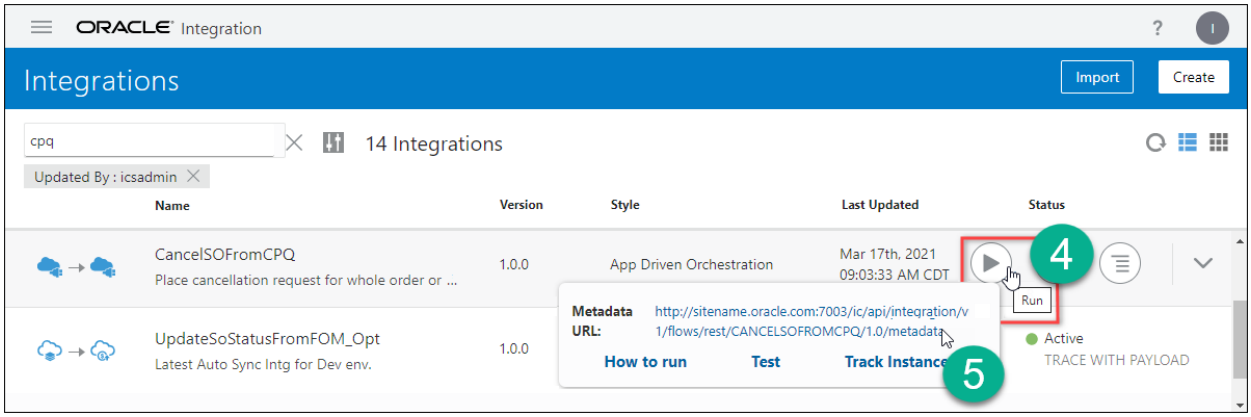
> **Notes:**
>
> - Fulfilment Status for which business events are enabled can only be synched back to CPQ.
> - Status mentioned in CPQ attribute 'Trigger Update Asset for Line Status' should have the business event enabled for corresponding Fulfilment Status in Order management. Without this Asset cannot be created in CPQ.
> - The CPQ Integration User should have Read/Write permissions on the status attributes which are being updated through this integration.

Appendix K: Retrieve OIC Integration Endpoint URL

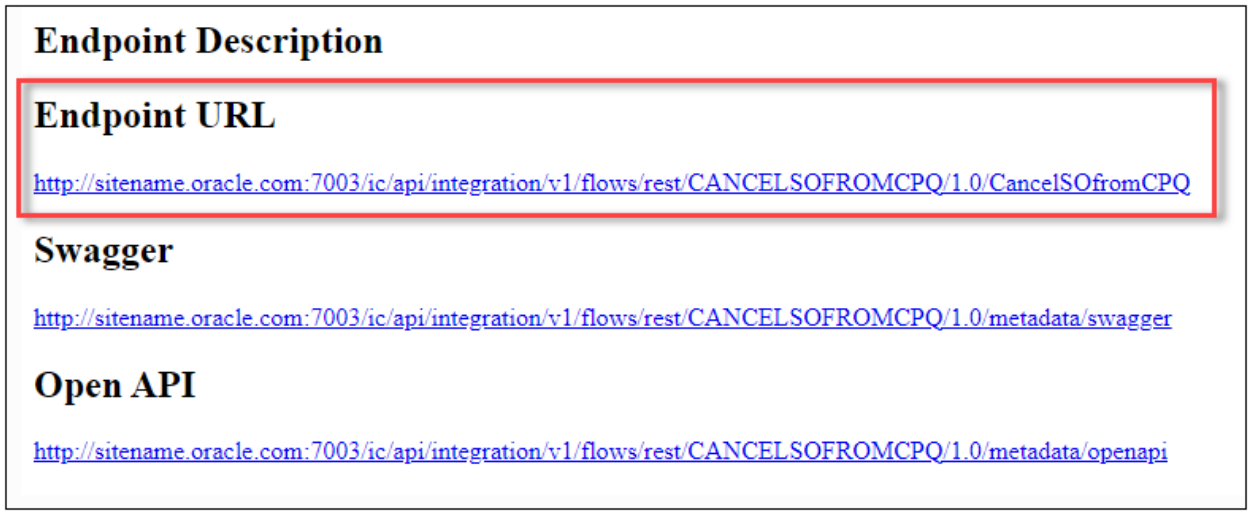
Integration endpoints are required for defining the Generic Integration in the Integration Center for the cancel order and get order status flows.

Perform the following steps to retrieve an integration endpoint.

1. Log in to the OIC application.
2. Select **Integrations** in the left side navigation panel, and then select **Integrations**.
3. If required, search icon to find your integration (e.g. CancelSOFromCPQ).
4. Click the Run icon for the applicable integration. A pop-up window displays with the Metadata URL.



5. Click on the Metadata URL link. A page will display with the Endpoint URL. Record the Endpoint URL.



Appendix L: Limitations and Troubleshooting

Appendix L1: Limitations

OIC integration using the Rest Adapter has a 10MB limit in the size of a message that can be received by the adapter. If the request payload for create order exceeds 10MB in OIC, you will see the following error message.

> "Content received of length <payload size> exceeds the maximum allowed threshold of 10485760 bytes "

Appendix L2: Troubleshooting

Resolve Issues with Create Order Action

If installing the CPQ Order Management package results in issues with the Create Order action, a workaround is available.

Perform the following steps to resolve issues with the Create Order action.

1. Open Oracle CPQ.
2. Select **Integration Center** under **Integration Platform**. The Integration Center opens.
3. Select the **ICS integration** from the left pane.
4. Unmark the **Enable Integration** check box to disable the integration.
5. Click **Save**.
6. Re-enable the ICS integration.
7. Click **Save**.

Assets Not Created in CPQ Assets Repository

- Check whether the business events are enabled for the Fulfilment status corresponding to status values mentioned in the CPQ attribute 'Trigger Update Asset for Line Status'.
- Check integration UpdateSOStatusFromFOM instance in OIC for errors. If there are not any you do not find any UpdateSOStatusFromFOM instance errors, there is an issue with the event subscription setup in OIC.

Blank Line Status on click of 'Get SO Status from FOM' button

If a line status becomes blank after clicking **Get SO Status from FOM**, it means that either the order management line status lookup code is not populated in the `status_1` attribute or the status mapping doesn't exist in OIC lookup **CPQ-FOM-LineStatusDVM** for the Order Management line status of that particular line.

To resolve this, either add a value in the status_l attribute or add mapping to the **CPQ-FOM-LineStatusDVM** lookup in OIC for that status. Refer to Appendix I: CPQ-OM Status Mapping.

CONNECT WITH US

Call +1.800.ORACLE1 or visit oracle.com.
Outside North America, find your local office at oracle.com/contact.

 blogs.oracle.com  facebook.com/oracle  twitter.com/oracle