Integrating with Oracle Commerce

F59936-02
March 2023

Integrating with Oracle Commerce,

F59936-02

Primary Authors: (primary author), (primary author)

Contributing Authors: (contributing author), (contributing author)

Contributors: (contributor), (contributor)

Contents

1

Integrate with Oracle CPQ

Integrate Oracle CPQ with Oracle Commerce.

When you integrate Oracle CPQ with Commerce enable a number of features that your shoppers can use, including allowing a shopper to configure products, request quotes, or purchase configurable services.

Introduction to Integrating with Oracle CPQ

The Oracle Commerce/Oracle Configure, Price, Quote integration lets you configure complex products for purchase in Commerce by using the Oracle Configure, Price, Quote configurator.

Self-service users in Oracle Commerce (formerly Oracle Commerce Cloud) can configure complex products for purchase in Commerce using the Oracle Configure, Price, Quote configurator.

They can also request an Oracle Configure, Price, Quote quote, thereby initiating an Oracle Configure, Price, Quote Transaction a sales specialist can modify, reconfigure, or discount. Once finalized in Oracle Configure, Price, Quote, the quote returns to Commerce for acceptance and ordering by the self-service user. For additional information, refer to Appendix A: Configurator Flow and Appendix B: Request for Quote Flow.

Note: The integration of Commerce with Oracle Configure, Price, Quote uses the Oracle Integration Cloud Service (OIC) to provide pre-built integrations for the two user flows.

Purpose

The purpose of this implementation guide is to provide the steps that administrators must complete in Oracle Configure, Price, Quote, OIC, and Oracle Commerce to prepare for a Commerce and Oracle Configure, Price, Quote integration.

Audience

This implementation guide is for administrators who are setting up and configuring the integration. The guide assumes administrators have prior Commerce, Oracle Configure, Price, Quote, and OIC administration experience.

Prerequisites

The following is a list of integration prerequisites:

- A Commerce 19D or later site setup as described in this implementation guide.
- An Oracle Configure, Price, Quote 19C or later Base Ref App site set up as described in this implementation guide. The integration between Commerce and Oracle Configure, Price, Quote adds attributes to the Base Ref App site that correspond to required Commerce order data.
- A synchronized product catalog to ensure that products in the Commerce catalog map to corresponding items in the Oracle Configure, Price, Quote catalog.
- Oracle Integration Cloud Service (OIC) 18.3.5 or later.

Note: For information about how to obtain any of the above prerequisites, contact an Oracle sales representative.

Set Up OIC Integrations

To begin setting up your integration, you must first import an OIC Integration Package to the OIC environment that connects Commerce and Oracle Configure, Price, Quote through a common configuration.

You must import an OIC Integration Package to an OIC environment that connects Commerce and Oracle Configure, Price, Quote through a common configuration.

The OIC Integration Package creates web service connections that allow users to adjust order and quote details in Oracle Configure, Price, Quote, approve or reject changes in Commerce, and complete or cancel orders in Commerce. This section contains the steps you must complete to set up and activate the OIC integrations.

Download the integration packages

To begin the OIC set up portion of the integration, you need to download the OIC Integration Package.

Complete the following procedure to download the OIC Integration Package.

1. Go to the Integrating Oracle CX and Oracle Configure, Price, Quote article on My Oracle Support.
2. If you want to implement the integration between Commerce and the Oracle Configure, Price, Quote Configurator, download `OCCS-CPQ_CONFIGURATION_INTEGRATION_X.X.par` to a location where it is accessible from OIC.
   Note: `OCCS_CPQ_GETCONFIGBOM_X.X.par` is only needed if you are enabling Asset Based ordering.
3. If you want to implement the integration between Commerce and Oracle Configure, Price, Quote Quoting, download `OCCS-CPQ_QUOTE_INTEGRATION_X.X.par` to a location that is accessible from OIC.
4. If you want to enable Asset Based Ordering (ABO) through the integration between Commerce and Oracle Configure, Price, Quote, download `OCCS_CPQ_ASSET_INTEGRATION_X.X.par`, `OCC_CPQ_Get_Asset_Upgrade_Options_X.X.par`, and `OCCS_CPQ_GETCONFIGBOM_1.0.par` to a location that is accessible from OIC.
   Note: `OCCS_CPQ_GETCONFIGBOM_X.X.par` is only needed if you are enabling Asset Based ordering.

Import the integration package

You must import the OIC Integration Package into OIC to create an integration between Commerce and Oracle Configure, Price, Quote.

To import the OIC Integration Package:

1. Log in to OIC as an admin user.
2. Click the **Packages** icon.

3. Click the **Import** button.
4. Click **Browse**.
5. Select the integration package archive (PAR) file you want to import.
6. Click **Import**. The package is added to the Packages list.

The `OCCS-CPQ_CONFIGURATION_INTEGRATION` package includes the *OCCS-CPQ Get Configurations* integration flow. This flow is invoked for retrieving a list of configurationIds fromOracle Configure, Price, Quote of regular configured items (non-ABO items) and ABO items with actionCodes other than Suspend and Terminate. This integration is required for the configuration flow and is available to import into OIC. The name of the target connection for this integration is Oracle Configure, Price, Quote. The target connection identifier is Oracle Configure, Price, Quote, and the target connection description is Oracle Configure, Price, Quote ICS Adapter Connection.

The `OCCS-CPQ_QUOTE_INTEGRATION` package includes the following three integration flows: *OCCS-CPQ Create Quote*, *OCCS-CPQ Update Quote*, and *OCCS-CPQ Sync Quote*.

- The *OCCS-CPQ Create Quote* integration sends quote request information to Oracle CPQ.
- The *OCCS-CPQ Sync Quote* integration allows Oracle Configure, Price, Quote to send information to Commerce at the end of the quoting process and synchronize this information in Commerce. This ensures that the order information in Commerce matches the related order information in Oracle Configure, Price, Quote.
- The *OCCS-CPQ Update Quote* integration sends information to Oracle Configure, Price, Quote related to accepting, rejecting, or re-requesting a quote.

The `OCCS-CPQ_ASSET_INTEGRATION` package includes two integration flows: OCCS-CPQ Get Assets and OCCS-CPQ Asset Actions. This integration is required for Asset Based ordering.

- The *OCCS-CPQ Get Assets* integration returns information about assets and services associated with the shopper’s account(s)
- The *OCCS-CPQ Asset Actions* integration enables Commerce to modify, renew, and terminate actions on assets and services associated with the shopper’s account(s).

The `OCC_CPQ_Get_Asset_Upgrade_Options` package is needed to retrieve all upgrade options that are available for an asset. If you want to show upgrade options to an assets shopper, this integration needs to be configured. When a call is made for the GetService(s) endpoint, this integration is called from the Services SSE to get all upgrade options. This call can only be made if `expand=occ_upgradeOptions` is passed as a queryparam for the GetService(s) endpoint.

The `OCCS_CPQ_GETCONFIGBOM` package contains the following OIC integration flow which is also used in Asset Based ordering:

- GetConfigBom - If an item is an ABO item with actionCode of Terminate or Suspend, getConfigBom calls are required to be made for each configuratorID of these filtered items to retrieve a saved Configuration BOM Instance of the item on Oracle Configure, Price, Quote.

Note: Importing and setting up the OIC Integration Package is a prerequisite to completing the Sync Quote action in Oracle Configure, Price, Quote. After all setup procedures are completed, regenerate the OCCS-CPQ Create Quote integration to ensure it accurately reflects the current state of the *Oracle Quote to Order* process.

Configure Oracle Configure, Price, Quote connections

You must configure Oracle Configure, Price, Quote connections to correspond to different SOAP or REST APIs for Oracle Configure, Price, Quote web services used in the integration.

Administrators must configure connections from the integrations referenced in the previous section to Oracle Configure, Price, Quote.

The following Oracle Configure, Price, Quote connections are part of the integrations: Oracle Configure, Price, Quote, Oracle Commerce, Oracle Configure, Price, Quote getConfigurations, Oracle Configure, Price, Quote Quote, Oracle Configure, Price, Quote Get Assets, and Oracle Configure, Price, Quote Asset Actions. Each connection corresponds to different SOAP or REST APIs for Oracle Configure, Price, Quote web services. Setting a connection to use the wrong API will cause the integrations to fail.

To configure the Oracle Configure, Price, Quote connections:

1. Log in to OIC as an admin user.
2. Click the **Connections** icon.
3. Click the **Oracle CPQ** connection.
4. Click **Configure Connectivity**.
5. Add the WSDL or REST metadata URL for the Oracle Configure, Price, Quote getConfigurations API.
   Note: The Oracle Configure, Price, Quote Asset Actions, Get Assets, and GetConfigBom connections are REST based and use the REST Catalog URL. The Oracle Configure, Price, Quote getConfigurations and Oracle Configure, Price, Quote SOAP connections are SOAP based and use WSDL URLs. The WSDL endpoint for getConfigurations is /v2_0/receiver/configuration?wsdl and the endpoint for Oracle Configure, Price, Quote SOAP varies by Commerce Process. For example, the Oracle Quotes and Orders endpoint is /v2_0/receiver/commerce/oraclecpqo?wsdl.
6. Click **OK**.
7. Click **Configure Security**. The Oracle Configure, Price, Quote connection uses the Basic security policy, so you must enter the login details for your Oracle Configure, Price, Quote account.
8. Click **OK**.
9. Click **Test** to test the connection.
10. Click **Save**.
    The Oracle Configure, Price, Quote connection is now configured for the integration. Repeat steps 1-10 for each of the remaining Oracle Configure, Price, Quote connections.

Generate security token for Commerce connections

A security token must be generated to support the Commerce REST web service APIs used to access Commerce data.

You must generate a security token to support the Commerce REST web service APIs used to access Commerce data in the integration. Use the following steps:

1. Log in to Commerce.
2. Click the **Menu** icon.
3. Select **Settings** from the menu.
4. Click **Web APIs** from the sidebar menu.
5. Click **Registered Applications** from the **Web APIs** panel.
6. Click **Register Application**.
7. Enter a name for the integration. Since you are registering OIC, choose a meaningful name that reflects the integration.
8. Click **Save**. The Application ID and Application Key are automatically generated. The application displays on the **Registered Applications** page.
9. Click the name of the application you created.
10. Select **Click to reveal** to display the application key.

Note: You need the application key when configuring the Commerce connection in OIC. Copy the registration key, so that it is available when you complete the Configure the Commerce Connection procedure.

Configure the Commerce connection

You must configure the connection from the OIC integrations to Commerce for the integration to run successfully.

An administrator must complete the following steps to configure the connection from the OIC integrations to Commerce. Use the following steps to do this:

1. Log in to OIC as an admin user.
2. Click the **Connections** icon.
3. Click the **Oracle Commerce** connection.
4. Click **Configure Connectivity**.
5. Enter the Connection base URL, which is derived using the below structure, where `<siteURL>` is the base URL of the Commerce site that integrates with OIC.

   ```
   Connection base URL: https://<hostname>:<port>/ccadmin/v1
   ```

6. Click **Configure Security**. The Commerce connection uses the OAuth security policy, so you must enter the security token for the connection. The security token was generated in the *Generate Security Token* section.
7. Click **OK**.
8. Click **Test**.
9. Click **Save**.
   Your Commerce connection is now configured for the integration.

Activate the OIC integrations

Once your integrations are configured, you must activate them using the OIC admin user interface.

Once the Oracle Configure, Price, Quote, Commerce, Oracle Configure, Price, Quote Quote, Oracle Configure, Price, Quote Configure, and Oracle Configure, Price, Quote getConfigurations connections are configured, you must activate these integrations.

To activate the OIC (Oracle Integration Cloud) integrations:

1. Log in to OIC as an admin user.
2. Click the **Integrations** icon to display the **Integrations List**.
3. Use the **Activate** slide switch to activate the integrations.
4. Decide whether you want to switch on detailed tracing, which collects information about messages processed by the integration flow. Administrators may find detailed tracing helpful when troubleshooting issues with the integration flow, but it may impact performance.
   To switch on detailed tracing, select the **Enable detailed tracing** check box.

   Note: Once an integration flow is active, administrators must deactivate and then reactivate the flow to switch detailed tracing on or off.
5. Click **Activate**.

Create Sync Quote Action in Oracle Configure, Price, Quote

The Sync Quote Action needs to be created for the Oracle Configure, Price, Quote/ Commerce integration to work successfully.

Use the following code to create the following Commerce action at the Commerce quote level:

```
Label(Sync Quote), Variable Name(syncQuote), Action Type(Modify).
```

Set up OIC integration on Oracle Configure, Price, Quote site

You must complete some preliminary OIC integration setup steps on the Oracle Configure, Price, Quote site for the integration to run successfully.

You must set up the OIC integration on the Oracle Configure, Price, Quote site by completing the following steps:

1. Click **Admin** to go to the Admin Home page.
2. Navigate to **Integration Platform** > **Integration Center**. The **Integration Center** opens.
3. From the **Type** drop-down menu, select **Integration Cloud Service**.
4. In the **Name** field, enter Sync Quote integration. The Variable Name field will auto-populate.
5. In the **Discovery URL** field, enter the OIC domain.
6. In the **Username** field, enter a valid username.
7. In the **Password** field, enter a valid password.
8. Click **Create Integration**.

Create the Sync Quote Integration

You must configure the integration of the Sync Quote system.

Administrators must create the Sync Quote integration by completing the following steps:

1. Click **Admin** to go to the Admin Home page.
2. Under the **Navigation** dropdown, select **Integrations** and click **List**.
3. Click **Add**.
4. For **Select Integration Types**, select **Integration Cloud Service**.
5. Click **Next**.
6. Name the action "Sync Quote" (`varname:syncQuote`)
7. Set **timeout** as **60000**.
8. set **Action** as **Import**.
9. For **Services**, choose **OCCS-CPQ Sync Quote** from the dropdown.
10. Click **Apply/Update**.

Set Sync Quote Action to run Advanced Modify

You must set the Sync Quote action to run Advanced Modify for the integration to run successfully.

Complete the following steps to set the Sync Quote action to run Advanced Modify:

1. Open the Admin Home page.
2. Navigate to **Process and Documents** > **Process Definition**. The **Processes** page opens with **Documents** displaying by default in the **Navigation** drop-down menu.
3. Click **List**. The **Document List** page opens.
4. From the **Navigation** drop-down menu, select **Actions** for the Transaction or Transaction Line.
5. Click **List**. The **Action List** page opens.
6. Click the **syncQuote** link. The **Admin Action** page opens.
7. Under the **General Tab** > **Advanced Modify** > **Before Formulas** >, select **Define Advanced Modify - Before Formulas**.
8. Click **Define Function**.
9. Select the attributes shown in the following tables:

Variable Name for (Transaction)	Type	Description
`cC_LineItem_Data`	String	`CC_LineItem_Data`

Variable Name for (Transaction Line)	Type	Description
`_document_number`	String	Document Number

Variable Name for (Transaction Line)	Type	Description
`_model_variable_name`	String	Model Variable Name
`cC_ProductId_l`	String	Product ID
`cC_CommerceItemId_l`	String	Commerce Item ID

10. Insert the sample BML provided in Appendix F: Understand the SyncQuote BML.
11. Update and click **Save**.
12. Navigate to the **Integration** tab and move **Sync quote** above **Modify Functions**.
13. Update and click **Save**.
14. Place the “syncQuote” action on the layout.

Configure Commerce webhooks

You must configure webhooks in Commerce Administration in order to support the REST API generated by the activation of the OIC integration.

The REST API generated by the activation of the OIC integration can be configured as webhooks in Commerce Administration. These include the following:

- **Request Quote**: This webhook is triggered when a request or a re-request for a quote is submitted by a Commerce self-service user. The webhook pushes notifications using the *OCCS-CPQ Create Quote* integration flow.
- **Update Quote**: This webhook is triggered when a response to a requested quote is accepted or rejected or the quote order is canceled by the Commerce self-service user. This webhook pushes notifications using the *OCCS-CPQ Update Quote* integration flow.
- **External Price Validation**: This webhook is triggered at check out when the order contains one or more items configured by Oracle Configure, Price, Quote. The webhook validates the configuration and the price provided for configured items.
- **Contact Accounts Retrieval**: This webhook returns a list of service account IDs for the shopper.
  ***Note:** This webhook has been deprecated.*
- **Services Retrieval**: This webhook returns information about a service or asset associated with the shopper and uses the OCCS-CPQ Get Assets integration flow. This webhook calls the Contact Accounts Retrieval webhook, so that webhook must also be configured for the Services Retrieval webhook to function correctly.
  ***Note:** This webhook has been deprecated.*

Note: Administrators must configure the *Production* and *Preview* versions of the webhooks to ensure they work in all environments. The *Production* webhooks send information from the live Commerce store to the production environments of your live systems. The *Preview* webhooks send information from the preview environment to the test or sandbox environments of external systems.

To configure Request Quote, Update Quote, External Price Validation, Services Retrieval (deprecated), or Services (deprecated) webhooks:

1. Log in to OIC as an admin user.
2. Click the **Integrations** icon.
3. Click the **Integration Details** icon to display information about the integration flow.

- If configuring the **Request Quote** webhook, display information for the *OCCS-CPQ Create Quote* integration flow.
- If configuring the **Update Quote** webhook, display information for the *OCCS-CPQ Update Quote* integration flow.
- If configuring the **External Price Validation** webhook, display information for the *OCCS-CPQ GetConfigurations* integration flow.
- If configuring the **Services Retrieval** webhook, display information for the *OCCS-CPQ Get Assets* integration flow.
  ***Note:** This webhook has been deprecated.*
- If configuring the **Services** webhook, display information for the *OCCS-CPQ Asset Actions* integration flow.

***Note:** This webhook has been deprecated.*

4. Copy the Endpoint URL for the integration.
5. Log in to Commerce.
6. Click on the **Menu** icon.
7. Select **Settings** from the menu.
8. Select **Web APIs** from the sidebar menu.
9. Click the webhook you want to configure.
10. Paste the Endpoint URL that was copied into the URL field for the webhook.
11. Remove the “metadata” text from the end of the URL.
12. Enter your OIC user name and password.
13. Click **Save**.

The webhook is now configured and is triggered each time the relevant event occurs, which in turn triggers the relevant integration flow.

Note: It is not possible to edit webhooks differently for different sites. Updating webhooks applies changes regardless of the site selected.

Understand the Services SSE

The Services SSE enables integration with third party asset management systems to retrieve and execute operations available to a shopper. This SSE also serves as the API for the integration with Oracle Configure, Price, Quote asset management.

The Modify, Renew, Terminate, Suspend, Resume, and Upgrade actions performed on a service or asset are done using the Services SSEs (server side extensions); one set for Storefront and one for Agent.

The Services SSEs call the integrations in `OCCS_CPQ_ASSET_INTEGRATION_X.X.par` and `OCC_CPQ_Get_Asset_Upgrade_Options_X.X.par` for the asset Upgrade feature.

See the section Configure the Commerce Server Side Extensions in this document for more information on these actions.

For more information about Commerce webhooks, refer to the Use Webhooks chapter of the *Extending Oracle Commerce* book.

For more information on understanding and using the asset Upgrade feature, refer to the Use Asset Based Ordering section of the *Using Oracle Configure, Price, Quote Features with Oracle Commerce* book.

Note: You can also customize configurations of complex assets in Commerce without being redirected to an Oracle Configure, Price, Quote hosted iFrame which may have a separate and distinct user interface look and feel that creates a disjointed user experience. This capability is known as the Direct API Configuration feature and can be used as another option for the Modify and Upgrade actions. For more information on the Direct API configuration feature, refer to the Customize configurations in Commerce using the Oracle Configure, Price, Quote Configuration API section of Use Oracle CPQ Cloud Features.

Configure the Commerce server-side extensions

To perform specific functions relating to asset-based orders, you need to install and configure the related Commerce server-side extensions (SSEs).

Commerce includes some server-side extensions (SSEs) that you can configure to perform specific functions relating to asset-based orders.

For more complete information on server-side extensions and how to develop them for use with Commerce, refer to Develop server-side extensions section in the *Extending Oracle Commerce* book found in the Commerce Help Library.

The next sections in this topic explain the purpose and configuration of each available SSE as well as provide information on the inputs required for their respective endpoints.

Note: Address information is something used extensively in Commerce transactions. For all procedures and SSEs that require address information for endpoint inputs, in addition to using Commerce's default address formats, you can also use the REST API to create multi-country custom address formats. This lets you create country-specific address formats to ensure that your address formats align with the requirements of any external service that you might use. This means that addresses appearing in profiles, accounts, registration requests, order addresses and more can be customized. For more complete information on creating custom addresses and understanding how to use custom address formatting, refer to the following:

- Customize Address Formats using the API in *Extending Oracle Commerce*
- Work with address types in *Extending Oracle Commerce*
- Account Details in *Using Oracle Commerce*
- Work with account addresses in *Using Oracle Commerce*
- Work with account registration requests in *Using Oracle Commerce*

Configure the Credit Check SSE

Since Commerce does not provide a pre-built integration with any particular credit checking system, the Credit Check SSE is used to connect to a third-party credit check system so that you can perform a credit check on the logged-in shopper.

As written, this SSE generates outbound calls to a master credit checking system. This means that the Credit Check SSE calls out to an external system to perform the credit check. In order to use this SSE to connect to the external checking of your choice, you must modify the SSE code to provide the specific calls needed to connect to the correct credit checking system.

You can configure the available SSEs, CheckCredit-store.zip and CheckCredit-agent.zip, by first downloading the SSE packages.

To complete installing and configuring the SSE, refer to Understanding the general procedure for installing and configuring the integration SSEs.

Understand the Check Credit endpoint

The Check Credit endpoint is triggered whenever a credit check is requested by Commerce. The inputs for this endpoint are:

- Amount information
- Recurring amount frequency
- Recurring amount duration
- Recurring amount
- Contact information
  - First Name
  - Last Name
  - Email Address
  - Telephone Number
- Address information
  - Address line 1
  - Address line 2
  - City
  - State
  - Country
  - Postal code

The return for this endpoint is either a TRUE or FALSE value depending on whether the shopper passed the credit check or not.

Configure the Customer Account Model SSE

This SSE is used to return information about the customer account model for a registered shopper or to update the customer account model when required.

You can configure the available SSEs, CustomerAccountModel-store.zip and CustomerAccountModel-agent.zip, by first downloading the SSE package.

To complete installing and configuring the SSE, refer to Understanding the general procedure for installing and configuring the integration SSEs.

The subsection(s) that follows describe the relevant endpoint(s) for this SSE.

Understand the Create Accounts endpoint

This endpoint is triggered if the Query Accounts endpoint does not return any accounts for the shopper.

The inputs for this endpoint are:

- User Token for the logged-in shopper.
- Account Type
- Account Name
- Primary Contact
- Billing Profile(s)
- Address(es)
- Contact ID(s)
- Contact Role(s)

The returns for this endpoint are the accounts, roles, addresses, and business profiles now associated with the shopper.

Understand the Create Contact endpoint

This endpoint is triggered when a shopper logs in to Commerce.

The input for this endpoint is the User Token for the logged-in shopper.

The return for this endpoint is the new External Contact ID created for the shopper.

Understand the Query Accounts endpoint

This endpoint is triggered when a shopper logs in to Commerce and when they go to Checkout for an order that contains service items.

The input for this endpoint is the User Token for the logged-in shopper.

The returns for this endpoint are the accounts, roles, addresses, and business profiles associated with the shopper.

Understand the Query Contacts endpoint

This endpoint is triggered when a shopper logs in to Commerce.

The input for this endpoint is the User Token for the logged-in shopper.

The return for this endpoint is the External Contact ID for the shopper.

Understand the Update Accounts endpoint

This endpoint is triggered when a shopper saves an account address.

The inputs for this endpoint are:

- User Token for the logged-in shopper.
- The Account ID of the account to which the billing profile is linked.
- The new address as provided by the shopper.

The returns for this endpoint are the accounts, roles, addresses, and business profiles associated with the shopper.

Configure the Order Qualification SSE

This SSE is used to perform any final checks on an order before payment is authorized and the order is submitted to downstream systems for processing and fulfillment.

It also validates that for any item in the order which is based on a SKU where the configurable property is TRUE and the assetable property is TRUE the quantity must be 1 and, if not, return an error indicating that this item can only be purchased one at a time. This check is done by looking to see if the root item has an assetKey value. For more information, refer to Use Asset Based Ordering.

You can configure the available SSEs, OrderQualification-store.zip and OrderQualification-agent.zip, by first downloading the SSE package.

To complete installing and configuring the SSE, refer to Understanding the general procedure for installing and configuring the integration SSEs.

The subsection(s) that follows describe the relevant endpoint(s) for this SSE.

Understand the Order Qualification endpoint

This endpoint is triggered by the Order Validation webhook when any order containing a configured item is submitted.

The input for this endpoint is the order containing the configured item.

The return for this endpoint is either a TRUE or FALSE value depending on whether the order passed the validation check or not. If the value is FALSE the return also includes information about which item(s) in the order failed validation.

Configure the Order Qualification Pipeline SSE

This SSE is used to ensure that an order is valid. It enables an order qualification step in the purchasing process that can be invoked via the Order Qualification webhook. The extension can be configured to execute custom order qualification processes such as checking whether the shopper is eligible to purchase the items in the cart. It contains a pre-built algorithm to validate that the Customer, Billing, and Service accounts as well as the Billing Profile assigned to the items in the cart are valid for the logged in shopper.

You can configure the available SSEs, OrderQualificationPipeline-store.zip and OrderQualificationPipeline-agent.zip, by first downloading the SSE package.

To complete installing and configuring the SSE, refer to Understanding the general procedure for installing and configuring the integration SSEs.

The subsection(s) that follows describe the relevant endpoint(s) for this SSE.

Understand the Order Qualification Pipeline endpoint

This endpoint is triggered when a shopper goes to checkout for an order that contains configured items.

The inputs for this endpoint are:

- Contact record for the shopper
- Order containing configured items.

The return for this endpoint is either a TRUE or FALSE value depending on whether the order passed the validation check or not. If the value is FALSE the return also includes information about which item(s) in the order failed validation.

Configure the Order Validation Pipeline SSE

This SSE enables an order qualification step in the purchasing process that can be invoked via the Order Validation webhook. The extension can be configured to execute any final checks particular to the purchasing model before the order payment is authorized and the order is submitted to the downstream systems for fulfillment and provisioning.

You can configure the available SSEs, OrderValidationPipeline-store.zip and OrderValidationPipeline-agent.zip, by first downloading the SSE package.

To complete installing and configuring the SSE, refer to Understanding the general procedure for installing and configuring the integration SSEs.

Configure the Services SSE

The Services SSE enables integration with third party asset management systems to retrieve and execute operations available to a shopper. This SSE also serves as the API for the integration with Oracle Configure, Price, Quote asset management. It can be used to retrieve all the services/assets linked to a shopper's profile or it can also be used to retrieve details of just one asset at a time.

The Modify, Renew, Terminate, Suspend, Resume, and Upgrade actions on a service or asset are performed using the Services SSEs (server side extensions), one set for Storefront and one for Agent.

The Services SSEs call the integrations in `OCCS_CPQ_ASSET_INTEGRATION_X.X.par` and `OCC_CPQ_Get_Asset_Upgrade_Options_1.0.par` for the asset Upgrade feature.

You can configure the available SSEs, `Services-store.zip` and `Services-agent.zip`, by first downloading the SSE package.

To complete installing and configuring the SSE, refer to Understanding the general procedure for installing and configuring the integration SSEs.

The subsection(s) that follows describe the relevant endpoint(s) for this SSE.

Understand the Services SSE endpoints

The endpoints for the Services SSE are the following:

- getServices - Calls Get OEC Account Details for OCC Profile OIC flow (to retrieve the account model for the shoppers OCC Profile), OCCS_CPQ_GET_ASSETS (6.0) OIC flow, and OCC_CPQ_Get_Asset_Upgrade_Options_1.0 OIC Flow. This endpoint returns the list of services for the shopper based on their service account(s) and any upgrade options available for those services.
- getService - Calls Get OEC Account Details for OCC Profile OIC flow (to retrieve the account model for the shoppers OCC Profile), OCCS_CPQ_GET_ASSETS (6.0) OIC flow, and OCC_CPQ_Get_Asset_Upgrade_Options_1.0 OIC Flow. This endpoint returns the details for a *single* service for the shopper based on their services account(s) and any upgrade options available for that service.
- Terminate - Calls Get OEC Account Details for OCC Profile (to retrieve the account model for the shoppers OCC Profile), OCCS_CPQ_GET_ASSETS (6.0) OIC flow, and OCCS_CPQ_ASSET_ACTIONS (5.0) OIC flow.
- Renew - Calls Get OEC Account Details for OCC Profile (to retrieve the account model for the shoppers OCC Profile), OCCS_CPQ_GET_ASSETS (6.0) OIC flow, and OCCS_CPQ_ASSET_ACTIONS (5.0) OIC flow.
- Suspend - Calls Get OEC Account Details for OCC Profile (to retrieve the account model for the shoppers OCC Profile), OCCS_CPQ_GET_ASSETS (6.0) OIC flow, and OCCS_CPQ_ASSET_ACTIONS (5.0) OIC flow.

- Resume - Calls Get OEC Account Details for OCC Profile (to retrieve the account model for the shoppers OCC Profile), OCCS_CPQ_GET_ASSETS (6.0) OIC flow, and OCCS_CPQ_ASSET_ACTIONS (5.0) OIC flow.
- Modify - Calls Get OEC Account Details for OCC Profile (to retrieve the account model for the shoppers OCC Profile), OCCS_CPQ_GET_ASSETS (6.0) OIC flow, retrieves iFrame URL from CPQ, and loads the Oracle Configure, Price, Quote hosted iFrame.
- Upgrade - Calls Get OEC Account Details for OCC Profile (to retrieve the account model for the shoppers OCC Profile), OCCS_CPQ_GET_ASSETS (6.0) OIC flow, retrieves iFrame URL from CPQ, and loads the Oracle Configure, Price, Quote hosted iFrame.
- Modify (v2) - Calls Get OEC Account Details for OCC Profile (to retrieve the account model for the shoppers OCC Profile), OCCS_CPQ_GET_ASSETS (6.0) OIC flow, and CPQ /rest/v9/config{prodFamVarName}.{prodLineVarName}.{modelVarName}/actions/_configure. This endpoint supports a directOracle Configure, Price, Quote API Modify action and lets you bypass the use of an iFrame.
- Upgrade (v2) - Calls Get OEC Account Details for OCC Profile (to retrieve the account model for the shoppers OCC Profile), OCCS_CPQ_GET_ASSETS (6.0) OIC flow, and Oracle Configure, Price, Quote /rest/v9/config{prodFamVarName}.{prodLineVarName}.{modelVarName}/actions/_configure. This endpoint supports a direct Oracle Configure, Price, Quote API Upgrade action and lets you bypass the use of an iFrame.

These endpoints are triggered when a shopper performs an operation on an asset.

Note: You can customize configurations of complex assets in Commerce without being redirected to a an Oracle Configure, Price, Quote hosted iFrame which may have a separate and distinct user interface look and feel that creates a disjointed user experience. This capability is known as the Direct API Configuration feature and can be used as another option for the Modify and Upgrade actions. For more information on the Direct API configuration feature, refer to Customize configurations in Commerce using the CPQ Configuration API.

The inputs for these endpoints are:

- Logged in User Token.
- AssetKey, the unique ID for the asset for this operation. This may be a root, branch or leaf asset.

The returns for the endpoints are a BOM (Bill of Materials) or an Error.

Note: For more information about C endpoints, refer to the Use the REST APIs chapter of the *Extending Oracle Commerce* book.

For more information about Commerce webhooks, refer to the Use Webhooks chapter of the *Extending Oracle Commerce* book.

For more information on understanding and using the asset Upgrade feature, refer to Use Asset Based Ordering.

Configure the Configuration Validation SSE

The Configuration Validation SSE (cpq-config-validation-app) plays an important role in Asset Based Ordering and validating asset configuration. This specific SSE performs a configuration validation between items in a shopper's cart and the items captured in response to configuration validation end points. For more complete information on Asset Based Ordering, refer to the Using the Integration Functionality section of this document.

To use this SSE, you should first have the External Pricing webhook set to /ccstorex/custom/v1/validateCPQConfigurations. This is done on the Settings page of the Administration user interface.

You should also have the following endpoints configured:

- GET_CONFIGBOM_URI – This is available when OCCS_CPQ_GETCONFIGBOM is configured.
- GET_CONFIG_URI - This is available when OCCS-CPQ_CONFIGURATION_INTEGRATION is configured.

The GET_CONFIGBOM_URI URL gets triggered for the Suspend and Terminate Services. The GET_CONFIG_URI URL gets triggered for the Renew, Modify, and Resume Services. The SSE does validation between items in cart and items captured in the response of these two end points.

The SSE package is named cpq-config-validation-app and is downloadable by this name from the Commerce Administration user interface.

To complete installing and configuring the SSE, refer to Understanding the general procedure for installing and configuring the integration SSEs.

Set Up Oracle Configure, Price, Quote

You must complete some general, configuration, and Commerce steps in Oracle Configure, Price, Quote to begin working with your integration.

This section contains the general, configuration, and Commerce steps you must complete in Oracle Configure, Price, Quote.

Understand general set up for Oracle Configure, Price, Quote

Some general set up procedures for Oracle Configure, Price, Quote need to be completed for the integration to run successfully.

You must complete the following Oracle Configure, Price, Quote general set up procedures:

- Enable Guest Access to Oracle Configure, Price, Quote
- Add Template Dependencies to File Manger
- Make Oracle Configure, Price, Quote Stylesheet Edits
- Synchronize Oracle Configure, Price, Quote Parts with Commerce SKUs

Enable Guest Access to Oracle Configure, Price, Quote

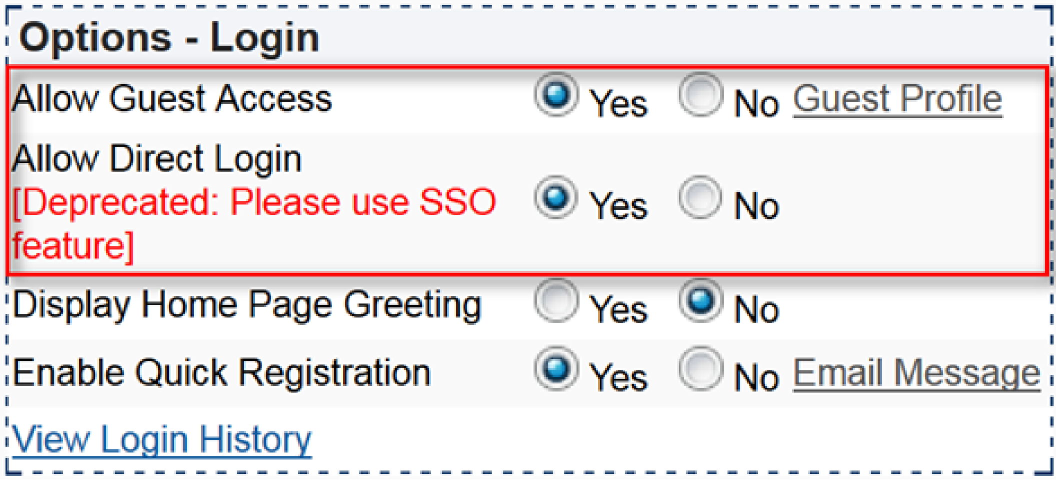
Administrators can allow multiple self-service users in Commerce to access an Oracle Configure, Price, Quote site as a guest user from an iFrame displaying within Commerce. When Commerce punches in to Oracle Configure, Price, Quote for configuring items, the system uses sessions for unregistered users (i.e. guest users). When self-service users access an Oracle Configure, Price, Quote site, their session parameters pass from Commerce to Oracle Configure, Price, Quote. This provides a seamless user experience and eliminates the need for Commerce self-service users to enter login credentials when entering an Oracle Configure, Price, Quote site from Commerce.

Note: You can now customize the configurations of complex products in Commerce without being redirected to an Oracle Configure, Price, Quote hosted iFrame. This capability, known as the Direct API Configuration feature, builds out support in Commerce for direct API driven product configurations where the user interface experience is controlled instead by Commerce and can be customized by Commerce partners rather than relying on the Oracle Configure, Price, Quote hosted iFrame. Refer to the Using Oracle Configure, Price, Quote Features with Oracle Commerce guide in the Oracle Commerce Doc Library for complete details.

Session parameters include currency, language, and locale preferences such as number format, units, and date format. For example: If a Commerce self-service user's language preference is set to German, the text in the Oracle Configure, Price, Quote interface displays in German when the user accesses Oracle Configure, Price, Quote. The user's currency and locale preferences are also passed from Commerce and display in Oracle Configure, Price, Quote.

To enable guest access to Oracle Configure, Price, Quote:

1. Open the Admin Home page.
2. Under **General**, select **General Site Options**. The **Options – General** page opens.
3. Under **Options – Login**, set **Allow Guest Access** to **Yes**.
   This setting allows Commerce to punch in to Oracle Configure, Price, Quote.
4. If multi-currency support from Commerce is required, set Allow Direct Login [Deprecated: Please use SSO feature] to Yes.

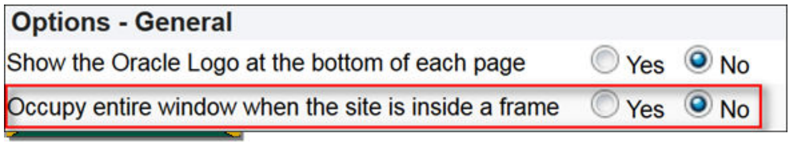


Options - Login

Allow Guest Access	[✓ Yes] No	Guest Profile
Allow Direct Login [Deprecated: Please use SSO feature]	[✓ Yes] No	
Display Home Page Greeting	Yes [✓ No]	
Enable Quick Registration	[✓ Yes] No	Email Message

View Login History

5. Under Options – General, set Occupy entire window when the site is inside a frame to No. This setting improves usability when punching in to Oracle Configure, Price, Quote from Commerce.



Options - General

Show the Oracle Logo at the bottom of each page	Yes [✓ No]
Occupy entire window when the site is inside a frame	Yes [✓ No]

Add Template Dependencies to File Manager

The "Add to Cart" action sends items to a Commerce cart via an **Add to Cart** button, which displays on the Commerce integrated Oracle Configure, Price, Quote site following configuration. Use the information provided in this section to add payload template files to File Manager. If Commerce requires additional information from Oracle Configure, Price, Quote during the “Add to Cart” action, administrators can add the information by creating configurable attributes and modifying the payload templates. Administrators can then export the configurable attributes as key-value pairs from Oracle Configure, Price, Quote to Commerce.

Payload template files (i.e. `Recommended_Items_Payload-Cloud.txt` and `AddToCartPayload-Cloud.txt`) form the payload structure for sending a configured item to the Commerce shopping cart. The template files support the “Add to Cart” action and include configuration information such as config id, quantity, and BOM items. BML reads the template files and replaces the values in brackets, such as {{bomitems}}, with dynamic values.

Complete the following steps to add the payload template files to File Manager:

1. Open the Admin Home page.
2. Navigate to **Utilities** > **File Manager**. **File Manager** opens.
3. Create a new folder named **CommerceCloud**.
4. Under **Add Files**, click **Browse.** The **Choose File to Upload** dialog opens.
5. Navigate to the **Recommended_Items_Payload-Cloud.txt** file and click **Open**.
6. Click **Add File**. The **Recommended_Items_Payload-Cloud.txt** file displays in **File Manager**.
7. Complete steps 1-6 for **AddToCartPayload-Cloud.txt**.

Shown below is the content of each of the payload template files.

Recommended_Items_Payload-Cloud.txt

```
{
 "quantity": "{{quantity}}",
 "catalogRefId": "{{part}}",
 "price": "{{price}}",
 "recurringCharge": { "amount":"{{recurringPrice}}",
 "frequency":"{{pricePeriod}}",
 "duration":"{{duration}}" }
}
```

AddToCartPayload-Cloud.txt

```
{
 "messageType": "Configuration_Details",
 "quantity": "1",
 "catalogRefId": "{{model}}",
 "amount": "{{totalPrice}}",
 "price": "{{basePrice}}",
 "currencyCode": "{{currency}}",
 "configurationId": "{{ConfigId}}",
 "childItems": [
    {{ChildItems}}
 ],
 "bomItems": [
    {{BomItems}}
 ]
}
```

For an example of an `AddToCartPayload-Cloud.txt` file, refer to Appendix G.

Make Oracle Configure, Price, Quote Stylesheet Edits

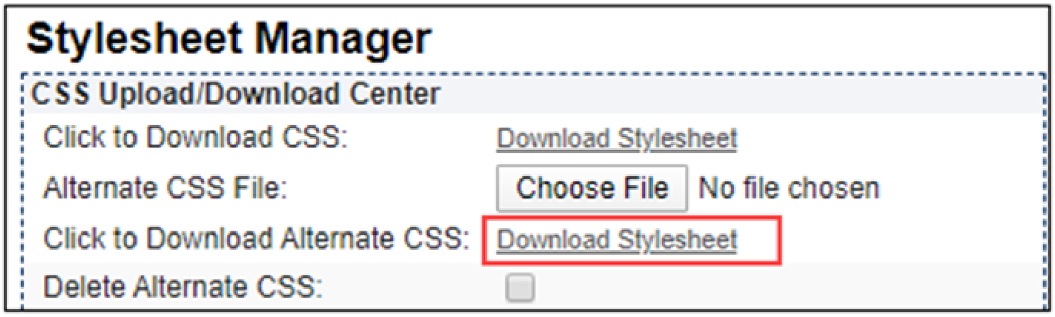
Oracle recommends administrators hide Oracle Configure, Price, Quote navigation options outside the scope of the integration from Commerce self-service users.

Hide the Oracle Configure, Price, Quote Home Button

By hiding the Oracle Configure, Price, Quote Home button, the Oracle Configure, Price, Quote configurator opens whenever users access Oracle Configure, Price, Quote. Users cannot navigate away from the original model that opens in the configurator, which prevents them from configuring a different model or adding a different model to Commerce.

To hide the Oracle Configure, Price, Quote Home button:

1. Open the Admin Home page.
2. Under **Style and Templates**, select **Stylesheet**. The **Stylesheet Manager** page opens.
3. Select **Download Alternate Stylesheet** next to **Click to Download Alternate CSS** from the CSS Upload/Download Center.



4. When the alternate CSS file opens, update the CSS to include the following CSS snippet to hide the **Home** button within the iFrame.

```
.nav-links>a img[title="Home"]{
    display: none;
}
```

Note: If the **Home** button shows both a label and an icon, administrators cannot hide the label using only CSS. From the Admin Home page, navigate to **Style and Templates** > **Navigation Menus** > **Subheader** > **Home** > **Edit**. Choose **Icon** for **Display**. The **Home** button is then hidden with the CSS change.

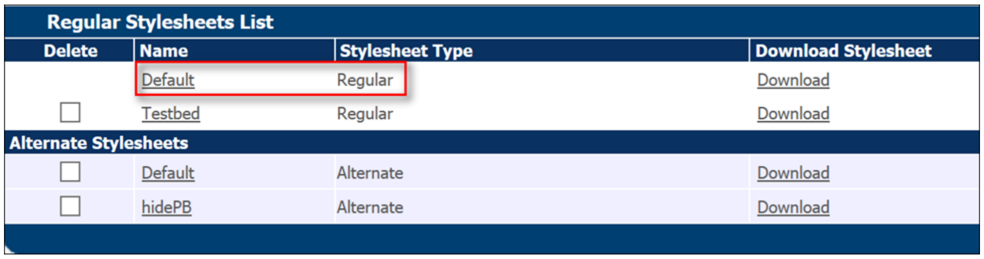
Hide Price Books

Oracle Configure, Price, Quote uses PriceBooks as a way to associate parts with a price. Oracle recommends hiding Price Book information from users.

To hide Price Books:

1. Open the Admin Home page.
2. Under **Products**, select **Catalog Definition.**The **Supported Products** page opens.

3. From the **Navigation** drop-down menu, select **Stylesheets**.
4. Click **List**. The **RegularStylesheets List** page opens.
5. Download the **DefaultRegular Stylesheet**.



Regular Stylesheets List

Delete	Name	Stylesheet Type	Download Stylesheet
	Default	Regular	Download
☐	Testbed	Regular	Download
Alternate Stylesheets			
☐	Default	Alternate	Download
☐	hidePB	Alternate	Download

6. Copy the contents of the **DefaultRegular Stylesheet**.
7. Create a new stylesheet with a name indicative of the stylesheet's purpose. For example: Hide Price Books
8. Paste the contents of the **Default Regular Stylesheet** into the new stylesheet and add the following CSS:

```
.pricebook-container {
            display:
           none;
}
```

9. Save the stylesheet.
10. On the **Regular Stylesheets List** page, click **Add Alternate**. The **Configuration Stylesheet Editor** opens.
11. Click **Browse**.
12. Use the **File Upload** dialog to locate and select the new stylesheet.
13. Click **Open**. The stylesheet displays in the **Regular Stylesheets List** page under the list of **Alternate Stylesheets**.

Synchronize Oracle Configure, Price, Quote Parts with Commerce SKUs

In Commerce, SKUs represent a purchasable instance of a product on a Commerce storefront. Administrators must synchronize Oracle Configure, Price, Quote parts with Commerce SKUs to ensure the pricing information associated with a part is the same in both Oracle Configure, Price, Quote and Commerce.

To synchronize Oracle Configure, Price, Quote parts with Commerce SKUs:

1. Open the Admin Home page.
2. Under **Products**, select **Parts**. The **Part Administration** page opens.
3. Add new parts in Oracle Configure, Price, Quote with part numbers that match SKUs in Commerce.
4. Add Part Custom fields for recurring charge price type, frequency, duration, and cost.

Notes:

- The client-side BML sample included in the *Configure Client-Side Integration, Add To Cart Button, and JSON Response* section of this implementation guide assumes part custom fields 4, 5, 6, and 8 represent recurring period, cost, duration, and type respectively. In order to use other part custom fields, the Add to Cart BML and OIC mappings will have to be adjusted accordingly.

- If a non-configurable SKU is later added to Commerce and intended for use by the Oracle Commerce and Oracle Configure, Price, Quote integration, repeat the above procedure to add the corresponding part in Oracle Configure, Price, Quote.
- In addition toOracle Configure, Price, Quote parts, configurable models must also have a corresponding SKU in Commerce. The SKU number in Commerce should match the model's label and variable name.

Understand Oracle Commerce set up

You must complete preliminary Commerce set up steps in Oracle Configure, Price, Quote for the integration to run successfully.

This topic contains the Commerce set up steps that you must complete in Oracle Configure, Price, Quote.

Note: Request for Quote and Sync Quote flows do not currently support Asset/Subscription based orders.

Create Commerce Attributes at the Transaction Level

You must create the Commerce attributes shown in the following table at the Transaction level and can adjust the attribute labels, as desired.

Note: An asterisk (*) next to the attribute label indicates the attribute should already exist as part of the Base Reference Application.

Attribute Label	Variable Name	Attribute Type	Additional Settings
CC Order Id	cC_OrderId_t	Text Field	none
Discount Info	cC_DiscountInfo_t	Text Field	none
Requestor Note	cC_RequesterNote_t	Text Area	none
Request Date	cC_RequestDate_t	Date	Default Value: System Variable: Current Date
Customer*	customer_t	Additional Address Set	none
Reject Explanation*	rejectExplanation_t	Text Area	none
Rejection Date	cC_RejectionDate_t	Date	none
Provider Note	cC_ProviderNote_t	Text Field	none
Price Expiration Date*	priceExpirationDate_t	Date	none
CC External Id	cC_ExternalId_t	Text Field	none
CC External Order Price	cC_ExternalOrderPrice_t	Currency	**Auto Update: Yes** **Modify: Revert to Default** **Default Value:** Use the formula provided in the *Apply Formulas* section.
CC External Order Price Quantity	cC_ExternalOrderPrice Quantity_t	Integer	none
CC Expiration Date	cC_ExpirationDate_t	Date	none
CC Agent Id	cC_AgentId_t	Text Field	none

Attribute Label	Variable Name	Attribute Type	Additional Settings
CC Subtotal	cC_Subtotal_t	Currency	none
CC Order Discount	cC_OrderDiscount_t	Float	**Auto Update:** Yes **Default Value:** Enter a non-blank default value to ensure the value sent to Commerce during Sync Quote (i.e. externalOrderPrice) is populated.
CC Order Discount Type	cC_OrderDiscountType_t	Menu	**Auto Update: Yes** **Menu Options: Percent Off, Amount Off, Price Override** **Default Value:** Enter a non-blank default value to ensure the value sent to Commerce during Sync Quote (i.e. externalOrderPrice) is populated.
CC_LineItem_Data	cC_LineItem_Data_t	Text Area	none
CC Total Net Price	cC_TotalNetPrice_t	Currency	**Auto Update: Yes** **Modify: Revert to Default** **Document View: Hide** **Default Value:** Use the formula provided in the *Apply Formulas* section.
Order Discount Total	cC_OrderDiscountTotal_t	Currency	**Auto Update: Yes** **Document View: Hide** **Default Value:** Use the formula provided in the *Apply Formulas* section.
Total (Net)*	totalOneTimeNetAmount_t	Currency	**Default Value:** Use the formula provided in the *Apply Formulas* section.
Total Discount*	totalOneTimeDiscount_t	Currency	**Default Value:** Use the formula provided in the *Apply Formulas* section.
CC Order Total	cC_Order_Total_t	Currency	none
CC Organization Id	cC_OrgId_t	Text Field	none
CC Site Id	cC_SiteId_t	Text Field	none
CC Site name	cC_SiteName_t	Text Field	none
Ship To Attributes*	shipTo_t	Additional Address Set	none
Invoice To Attributes*	invoiceTo_t	Additional Address Set	none

Note: For all procedures and SSEs that require address information for endpoint inputs, in addition to usingCommerce's default address formats, you can also use the Commerce REST API to create multi-country custom address formats. Refer to the

Configure the Commerce Server-Side Extensions topic in this guide for more information on address formatting.

Modify the Existing "Status" Transaction Level Attribute

The Status ("status_t") attribute is an existing Transaction-level attribute that should already exist on Base Ref App environments. You must modify this attribute as described below.

- Add the following options:
  - Rejected [REJECTED]
  - Synced [SYNCED]
- Under **Modify**, set the attribute to "**Use Specified Value**" for the following actions:
  - Create Order: ORDERED
  - Customer Rejection: REJECTED
  - Save: CREATED
  - Sync Quote: SYNCED
  - Cancel Transaction: CANCELED

Create Attributes at the Commerce Line Level and Add Them to the Commerce Layout

Create the Commerce attributes shown below at the Commerce line level. Once created, add the attributes to the Commerce layout.

Attribute Label	Variable Name	Attribute Type	Additional Settings
Commerce Item Id	cC_CommerceItemId_l	Text Field	none
Product Id	cC_ProductId_l	Text Field	none

Catalog Ref Id	cC_CatalogRefId_l	Text Field	**Default Value:** Function
			if(_model_variable_name <> ""){
			return _model_variable_name;
			}
			return _part_number;
			Note: When creating the Default value Function, `_model_variable_name` and `_part_number` need to be selected from the Variable Name for that Transaction Line tab.

External Price	cC_ExternalPrice_l	Currency	none
External Price Quantity	cC_ExternalPriceQuantity_l	Integer	none
CC Net Price	cC_NetPrice_l	Currency	none
Quantity*	requestedQuantity_l	Currency	none
Price (List)*	listPrice_l	Currency	**Default Value:** Use the formula provided in the *Apply Formulas* section.

Attribute Label	Variable Name	Attribute Type	Additional Settings
n/a	oRCL_ABO_ActionCode_l	Single Select Menu	**This menu attribute comes from the ABO installation package and is a requirement for the Sync Quote action.**

Apply Formulas

The following Commerce attributes should already exist on Base Ref App environments. Apply the listed formulas to the attributes.

Table 1-1 Attributes for Base Ref App environments

Variable Name	Formula
cC_ExternalOrderPrice_t	if((cC_OrderDiscountType_t = "amountOff"), (cC_TotalNetPrice_t - cC_OrderDiscount_t), if((cC_OrderDiscountType_t = "percentOff"), (cC_TotalNetPrice_t - (cC_TotalNetPrice_t * (cC_OrderDiscount_t / 100))), if((cC_OrderDiscountType_t = "priceOverride"), cC_OrderDiscount_t,cC_TotalNetPrice_t)))
totalOneTimeNetAmount_t*	cC_ExternalOrderPrice_t
totalOneTimeDiscount_t*	sumIf((priceType_l NOT= "Recurring"), discountAmount_l) + cC_OrderDiscountTotal_t
cC_OrderDiscountTotal_t	if((cC_OrderDiscountType_t = "amountOff"), cC_OrderDiscount_t,if((cC_OrderDiscountType_t = "percentOff"), (cC_ExternalOrderPrice_t - (cC_OrderDiscount_t / 100)), if((cC_OrderDiscountType_t = "priceOverride"), (cC_ExternalOrderPrice_t - cC_OrderDiscount_t), 0)))
cC_TotalNetPrice_t	sumIf((priceType_l NOT= "Recurring"), netAmount_l)

listPrice_l*	if((_model_base_price NOT= 0), _model_base_price,_price_list_price_each)
	if((_model_base_price NOT= 0), _model_base_price,if((_pricing_rule_price_each NOT= 0), _pricing_rule_price_each,_price_list_price_each))

Note: An asterisk (*) next to the variable name indicates that a formula for the attribute already exists on Base Ref App environments. You must update the existing formulas as opposed to creating new formulas.

Set Up Commerce Actions

Complete the following steps to set up Commerce actions.

1. Create the following Commerce action at the Transaction level.

Table 1-2 Commerce action

Label	Variable Name	Action Type	Integration	Advanced Modify (Before Formulas)
Sync Quote	cC_syncQuote	Modify	CPQ-OCCS Sync Quote	**Transaction Attribute:**CC_LineItem_Data **Transaction Line Attributes:**_document_number _model_variable_name cC_ProductId_l cC_CommerceItemId_l **BML: Refer to** Appendix F: Understand the SyncQuote BML

2. Place the Sync Quote action on the Commerce layout.
3. Set the quote level actions “cleanSave_t" and " _remove_transactionLine" to define the following attributes based on their formula definitions:
   - Quote Level Attributes:
     – Total Contract Value
     – Total Discount Per Month
     – Total (List) Per Month
     – Total (Net) Per Month
     – Total Discount
     – Total (List)
     – Total (Net)
     – Annual Contract Value
     – Transaction Total
     – Total Contract Discount
     – Annual Contract Discount
     – CC External Order Price
   - Line Level Attributes
     – Actual Amount
     – Annual Value
     – Contract Value
     – Amount (List)
     – Amount (Net)
     – Price (Net)

   - Quantity

4. Set the line level action “save_l" to define the following line level attributes based on their formula definitions:
   - Actual Amount
   - Annual Value
   - Contract Value
   - Amount (List)
   - Amount (Net)
   - Price (Net)
   - Quantity

Notes:

- The “Save” action is already setup to use formulas for a majority of these attributes in the Base Ref Application.
- The Request for Quote and Sync Quote flows do not support the “Copy Line Items” action. The action is not accessible for Commerce integrated Transactions.

(Optional) Create Commerce Validation Rule

You have the option of creating a Commerce validation rule that blocks users from editing the quantity of child items.

1. Open the Admin Home page.
2. Under **Commerce and Documents**, click **Process Definition**. The **Processes** page opens with **Documents** displaying by default in the **Navigation** drop-down menu.
3. Click **List** next to the *Oracle Quote to Order* Commerce process. The **Document List** page opens.
4. At the Transaction Line level, select **Rules** from the **Navigation** drop-down menu.
5. Click **List**.
6. From the **Add** menu, select **Validation**. The **Validation: New Rule** page opens.
7. In the **Name** field, enter a name for the validation rule.
8. Click in the **Variable Name** field to auto-populate the field.
9. For the **Condition Type**, select **Advanced**.
10. Click **Define Function**. The **Select Attributes** dialog opens.
11. Select the attributes shown in the following tables.

System Variable Name	Type	Description
_system_current_document_number	String	Current Document Number

Variable Name for (Transaction Line)	Type	Description
_model_variable_name	String	Model Variable Name

Variable Name for (Transaction Line)	Type	Description
_price_quantity	Integer	Quantity

12. Click **Next**.
13. Enter the following BML:

```
oldvalue = getoldvalue("_price_quantity",
atoi(_system_current_document_number));

if((_model_variable_name == "") AND (_price_quantity <> atoi(oldvalue)))
{

return true;

}
return false;
```

14. Click **Save and Close**.
15. On the **Validation: New Rule** page, select **Advanced** as the **Action Type**.
16. Click **Define Function**. The **Select Attributes** dialog opens.
17. Select the **Variable Name for (Transaction Line)** tab.
18. Select the "_price_quantity" attribute.
19. Click **Next**.
20. Enter the following BML.

```
attributeDict = dict("dict<string>");

    // inner dictionary for attr2
    attr2ActionDict = dict("string");
    // assembling the constraint action
    put(attr2ActionDict, BM_CM_RULES_MESSAGE, "Please re-configure the
item to change quantity of sub-item");

    // put the inner dictionary into the outer dictionary
    put(attributeDict, "_price_quantity", attr2ActionDict);


// return the outer dictionary
return attributeDict;
```

21. Click **Save and Close**
22. In the **Components** list add the **Quantity (_price_quantity)** attribute.
23. Click **Save** to save the Validation Rule.

Set Up Steps

You must use Oracle Configure, Price, Quote to create a Synced step as well as step transitions.

1. Create a new “Synced” step.

2. Create a step transition for the "Sync Quote" action to move from the "In Progress" step to the "Synced" step.
3. Create a step transition for the “Save" action to move from the "Synced" step to the "In Progress" step.
4. Create a step transition for the "Customer Rejection" action to move from the "Synced" step to the "Rejected by Customer" step.
5. Create a step transition for the “Create Order” action to move from the “Synced” step to the “Ordered” step.
6. Create a step transition for the “Cancel Transaction” action to move from the “Synced” step to the “Canceled” step.
7. Hide the "Sync Quote" action from the following steps:
   - Fulfilled
   - Canceled
   - Rejected By Customer
8. Hide all Modify actions from the “Synced” step EXCEPT the following:
   - Save
   - Customer Rejection
   - Create Order
   - Cancel Transaction

Notes:

- Make sure all of the attributes used in the Request for Quote flow have read/write access at the Start step.
- For instructions on how to create Commerce attributes, actions, and step transitions, refer to the Oracle Configure, Price, Quote Administration Help.

Modify Process Manager View

You must complete the following procedure to modify a process manager view.

1. Add a data column named "CC Order Id".
2. Map the data column to the "CC Order Id" quote level attribute.
3. Add a Process Manager column using the "CC Order Id" data column.

Understand Oracle Configure, Price, Quote configuration set up

Specific set up procedures must be completed for the Commerce/Oracle Configure, Price, Quote Configuration integration to run successfully.

This topic contains the configuration set up procedures that you must complete in Oracle Configure, Price, Quote.

Configure Client-Side Integration, Add To Cart Button, and JSON Payload Response

You must configure a client-side integration to add the **Add to Cart** button on a Commerce site. The client-side integration enables the sharing of data between Oracle Configure, Price, Quote and Commerce.

Note: Ensure the appropriate Commerce Product Families and Product Lines are created in Oracle Configure, Price, Quote prior to starting the Client-Side Integration. Refer to the Configuration > Product Families articles within the Oracle Configure, Price, Quote Online Help for instructions.

To configure a client-side integration:

1. Open the Admin Home page.
2. Under **Products**, click **Catalog Definition**. The **Supported Products** page opens. **Product Families** displays by default in the **Navigation** drop-down menu.
3. Click **List**. The **Supported Product Families** page opens.
4. Click **Integrations** from the **Navigation** drop-down menu for the product of the Commerce product family.
5. Click **List**. The **Edit Integration** page opens.
6. Use the **Edit Integration** page to create a “Client-side” integration using the following settings:
   - Name: **Add To Cart**
   - Integration Type: **Client-side**
   - Hide in Reconfiguration: **No**
   - Action: **Define Advanced Function**
7. Click **Define Function** for the Action and use the sample BML from one of the following to add the **Add to Cart** button to the Commerce site:
   - Appendix D: Understand the Add to Cart BML – Customized Integrations (19C and Earlier) - this sample BML is for legacy integration sites who have previously customized their Add to Cart BML. This sample includes site-specific reference file locations.
   - Appendix E: Understand the Add to Cart BML – Customized Integrations and Multi-Site Set Up (19D and Later) – this sample BML is for new integrations and in cases where the setup needs to be duplicated on multiple sites. This sample does not reference site-specific file locations.
8. Select Simple for the **End-Point URL**.
   Enter the URL of the Commerce site to integrate with Oracle Configure, Price, Quote. The value entered should include the basic URL or Commerce’s storefront and administration pages. You can add multiple Commerce sites for a single integration by listing each site delimited by the pipe delimiter (|) character.

   For example:

   ```
   http://cc-store.oracle.com|http://cc-admin.oracle.com|http://
   second-store.oracle.com|http://second-admin.oracle.com
   ```
9. Click **Apply**.

Note: Ensure that all partner site lists of allowed URLs are properly addressed within Oracle Configure, Price, Quote. These include domains that are allowed to load the Oracle Configure, Price, Quote in an iFrame and domains that Oracle Configure, Price, Quote is allowed to connect to in the Integration Center. You may need to file a Service Request (SR) on My Oracle Support to include these domains on the site list of allowed URLs.

Configure Oracle Configure, Price, Quote Models Corresponding to Products in Commerce

You must create Oracle Configure, Price, Quote models corresponding to SKUs in Commerce.

To configure models corresponding to products in Commerce:

1. Open the Admin Home page.
2. Under **Products**, select **Catalog Definition**. The **Supported Products** page opens with **Product Families** displaying by default in the **Navigation** drop-down menu.
3. Click **List**. The **Supported Product Families** page opens with **Product Lines** displaying by default in the **Navigation** drop-down menu.
4. Click **List**. The **Product Line Administration List** page opens with **Models** displaying by default in the **Navigation** drop-down menu.
5. Click **List**. The **Model Administration List** page opens.
6. Click **Add**.
7. Use the **Model Administration** page to create a new model with both the variable name and label matching the configurable root SKU in Commerce.
8. Create a pricing rule on the model with a price matching the root SKU in Commerce.

Configure Child Line Items Corresponding to SKUs in Commerce

For information about setting up BOM Mapping items for a model, refer to the Oracle Configure, Price, Quote Administrator Online Help.

Note: Quantity for the root BOM should use a configurable integer attribute in BOM Attribute Mapping. Otherwise, incorrect quantities may be populated during reconfigure.

Create Configurable Attributes

Configurable attributes define the characteristics of product families. Oracle Configure, Price, Quote uses configurable attributes in search flows, Configuration flows, and every type of Configuration rule.

To create configurable attributes:

1. While you can create the following configurable attributes at any level, Oracle recommends creating the attributes at the Product Family level.

Label	Variable Name	Attribute Type	Additional Settings
Currency Code	currencyCode	Text Field	none
CC Site ID	cC_SiteId_t	Text Field	none
Quantity	quantity	Integer	Required, Default = 1, Positive Number Validation

2. Create a recommendation rule configured as follows:

Condition	Apply Rule To	Action Type	Action Attribute	Values to Set	Set Type
Always True	Configuration	Standard	currencyCode	Edit Function: `return _BM_USER_CURRENCY;`	Forced Set

3. Create any additional attributes that suit your organization’s needs and place them on the Configuration flow layout.
   - You must place “currencyCode”, “cC_SiteId_t”, and “quantity” on the layout, but they do not need to display them.
   - For information about configurable attributes and the steps to create them, refer to the Oracle Configure, Price, Quote Administration Help.
4. Create a hiding rule configured as follows:

Condition	Action Attribute

Advanced:
 	
`if (_transaction_id == "-1")`
`{    return  true;}`
`        return false;`
	quantity

Set Up Subscription Ordering in Oracle Configure, Price, Quote

The subscription ordering feature requires some set up when integrating Oracle Configure, Price, Quote and Commerce.

The following features require specific attention when integrating Oracle Configure, Price, Quote and Commerce and running the subscription ordering feature.

For information about setting up Subscription or asset based orders within Oracle Configure, Price, Quote, refer to the ABO implementation guide and the Oracle Configure, Price, Quote Administrator Online Help.

Create an authentication certificate integration type

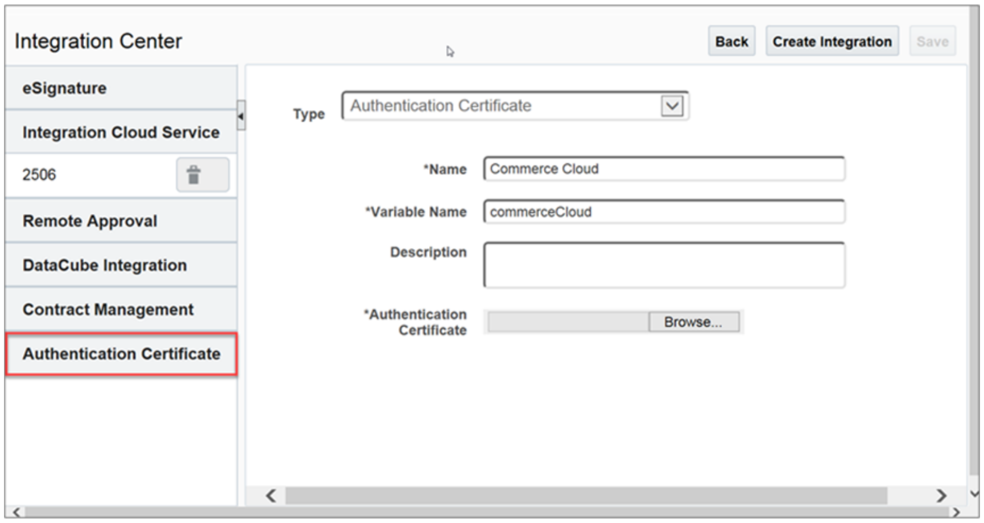
You need to create an Authentication Certificate integration type in the Integration Center to support access token-based authentication in the Commerce/Oracle Configure, Price, Quote integration.

Oracle Configure, Price, Quote provides an Authentication Certificate integration type in the Integration Center to support access token-based authentication. This integration type allows Oracle Commerce self-service users to securely access Oracle Configure, Price, Quote to modify or reconfigure a Subscription Ordering asset-based Configuration without an Oracle Configure, Price, Quote user session.

When administrators create a new integration of type Authentication Certificate, they provide a name and variable name for the authentication certificate and upload the Commerce authentication certificate. A temporary session is created for the Commerce self-service user, allowing the user to access the**Model Configuration** page via an iFrame within Commerce to modify or reconfigure a specific asset.

To create an Authentication Certificate integration type, perform the following steps:

1. Open the Admin Home page.
2. Select **Integration Center** under **Integration Platform**. The **Integration Center** opens.
3. Click **Create Integration**.
4. From the **Type** drop-down, select **Authentication Certificate**.
5. In the **Name** field, enter a name that describes the authentication certificate. For example: Commerce
6. The **Variable Name** field auto-populates upon clicking in or tabbing to the field.
7. (Optional) In the **Description** field, enter a description of the authentication certificate.
8. Click **Browse** next to the **Authentication Certificate** label.
9. Select the Oracle Commerce authentication certificate and click **Open**.
10. Click**Save**. The Authentication Certificate integration appears in the left pane of the Integration Center.



Type: Authentication Certificate
*Name: Commerce Cloud
*Variable Name: commerceCloud
Description:
*Authentication Certificate:

Notes:

- The **Save** button is disabled upon successfully saving the integration. If the changes are made after the save is performed, the button is enabled.
- Administrators can modify the name of the integration but not the variable name. They can also replace the authentication certificate but cannot remove it.
- A single Oracle Configure, Price, Quote site can have any number of Authentication Certificate integrations. There is no limit.

Work with in-flight cancellations

Custom asset fields must be created in Oracle Configure, Price, Quote to support in-flight cancellations of orders.

In order to support in-flight cancellations of orders, the following custom asset fields must be created in Oracle Configure, Price, Quote

Note: Refer to the *Custom Asset Attributes* article within the Oracle Configure, Price, Quote Online Help for instructions on adding a custom asset.

Label	Variable Name	Data Type
Order Id	_asset_custom_orderId	String
Line Id	_asset_custom_lineId	String
Source Site	_asset_custom_source	String

Upgrade an asset

With Asset Based Ordering, the ability to upgrade an existing asset is supported when you complete some preliminary set up work.

With Asset Based Ordering, the ability to upgrade an existing asset is supported.

Oracle Configure, Price, Quote maintains a custom upgrade options table for Commerce to query in order to know which upgrades are available for a given asset. The sections that follow in this topic provide information on how to set up the required tables and how to complete some basic Oracle Configure, Price, Quote configuration steps to support asset based ordering.

Oracle Configure, Price, Quote Data Table Set Up

Create a data table named "`INT_UPGRADE_OPTIONS`" with the following schema:

Column Name	Data Type
currentOffer	String
currentModel	String
upgradeName	String
upgradeProductId	String

The data table column mapping information for this data table is as follows:

- **currentModel** – Maps to the variable name of the root config model in Oracle Configure, Price, Quote which the upgrade offer applies to.
- **currentOffer** – Maps to a configurable attribute on the root config model in Oracle Configure, Price, Quote. This needs to be stored as an attribute mapping onto the root asset as well. This value is sent from Oracle Commerce while retrieving the upgrade options.
- **upgradeName** – Maps to the _config_upgrade_name that is passed from Oracle Commerce to Oracle Configure, Price, Quote, which drives recommendation rules on the upgrade. Not used by Commerce for any other purpose.
- **upgradeProductID** – Maps to the Product Id of the upgrade offer in Commerce. Used to show upgrade details (for example, product display name, description, images, etc.) to the shopper.

Note: We recommend you index the currentModel and/or currentOffer columns.

The INT_UPGRADE_OPTIONS data table is queried by Oracle Commerce to help identify what upgrades are available for a given asset and present those upgrade options to the shopper.

For example:

currentOffer	currentModel	upgradeName	upgradeProductId
4ForUDeal	nPlay	4ForUDealPlus	prod102

Oracle Configure, Price, Quote Upgrade Asset Configuration Set Up

1. Create a configurable text attribute named "currentOffer**"**. This attribute should have either a default value set or have its value recommended based on specific criteria on the configuration; however, the value should not be editable directly by the user. The value of the "currentOffer" attribute is used in the INT_UPGRADE_OPTIONS data table that Commerce queries.
2. Use (Bulk) Recommendation Rules that run when the value of the **"**_config_upgrade_name" attribute matches the value of the "upgradeName" column in the "`INT_UPGRADE_OPTIONS`" data table. Part of the rule should update the "currentOffer" attribute from its previous value to the "upgradeName" as well. Unlike normal configurable attributes, the value of "`_config_upgrade_name`" persists within all models of a system, so inter-model rules are not required to reference "`_config_upgrade_name`" and use them in Recommendation Rules on child models. The value of "`_config_upgrade_name`" also does not persist on the configurations, like other attributes do, so whether "`_config_upgrade_name`" has a value or not distinguishes asset upgrades from a typical asset modify.

Note: For more information on understanding and using the asset Upgrade feature in Commerce, refer to Use asset-based ordering.

You can also customize configurations of complex assets in Commerce without being redirected to an Oracle Configure, Price, Quote hosted iFrame which may have a separate and distinct user interface look and feel that creates a disjointed user experience. This capability is known as the Direct API Configuration feature and can be used as another option for the Modify and Upgrade actions. For more information on the Direct API configuration feature, refer to Customize configurations in Commerce using the CPQ Configuration API.

Enable Integrations in Commerce

To enable the features of this integration, you must configure some settings and storefront widgets in Commerce.

You must complete the procedures in this section to enable the Oracle Configure, Price, Quote Configurator integration, the Oracle Configure, Price, Quote Request For Quote integration, and the Asset Based Ordering (ABO) integration in Commerce.

This section describes how to configure Storefront Classic widgets to support the integration. To learn how to configure Open Storefront Framework widgets instead, see Design Configure-Price Components.

For additional information about these integrations, refer to Appendix A: Configurator Flow and Appendix B: Request for Quote Flow.

Enable Oracle Configure, Price, Quote configuration integration

Some feature configuration procedures must be completed to enable the Oracle Configure, Price, Quote Configuration integration.

To enable the Oracle Configure, Price, Quote Configuration integration, do the following:

1. Log in to Commerce.
2. Navigate to the **Settings** icons from the sidebar menu.
3. Select **Oracle Integrations** from the sidebar menu.
4. Select **CPQ Configuration** from the drop-down menu.
5. Select the **Enable Integration** check box.
6. Click **Preview Confirmation.** You need to do this to display the URL fields.
7. Enter the Configuration URL using the following structure: `https://<cpq_domain>/commerce/new_equipment/products/model_configs.jsp`
8. Enter the Reconfiguration URL using the following structure: `https://<cpq_domain>/commerce/new_equipment/products/external_reconfig.jsp`
9. Enter the Modification URL using the following structure: `https://<cpq_domain>/commerce/new_equipment/products/model_configs.jsp.`
10. Click **Product Configuration**. You need to do this to display the URL fields.
11. Enter the Configuration URL using the following structure: `https://<cpq_domain>/commerce/new_equipment/products/model_configs.jsp`.
12. Enter the Reconfiguration URL using the following structure: `https://<cpq_domain>/commerce/new_equipment/products/external_reconfig.jsp`
13. Enter the Modification URL using the following structure: `https://<cpq_domain>/commerce/new_equipment/products/model_configs.jsp`.
    Note: Enter the Configuration URL and the Reconfiguration URL for both the Production and Preview environments.
14. Click **Save**. If you are using a multisite environment you must follow these instructions for each site that uses the Oracle Configure, Price, Quote Configuration integration.

Identify configurable products in the product catalog

It is important to understand which products are configurable in the product catalog to use this integration..

Before a Commerce self-service user can use the Oracle Configure, Price, Quote Configurator to configure complex products for purchase in Commerce, you must identify the products as configurable in the product catalog.

Before doing so, it is important to have a synchronized product catalog to ensure that products in the Commerce catalog map to corresponding items in the Oracle Configure, Price, Quote catalog.

To identify a product as configurable:

1. Log in to Commerce.

2. Click on the **Menu** icon.
3. Select the product you wish to identify as configurable from the Catalog Settings icon in the sidebar menu.
4. Click on the **SKUs** tab of the product detail pop-up frame.
5. Click on the SKU link of the product you wish to identify as configurable. You need to do this in order to select the SKU and see the SKU details.
6. Check the **Externally Configurable SKU** checkbox. This displays three further fields you must complete.
7. Enter the **Model** variable name. This should match the Model variable name of a configurable product in the Oracle Configure, Price, Quote catalog.
8. Enter the **Product Line** variable name. This should match the Product Line variable name of a configurable product in the Oracle Configure, Price, Quote catalog.
9. Enter the **Product Family** variable name. This should match the Product Family variable name of a configurable product in the Oracle Configure, Price, Quote catalog.
10. Click Save. This returns you to the SKU frame where the SKU you updated should be marked with an asterisk to identify it as a configurable SKU.

Note: Administrators can also perform the above setup steps in bulk by using the SKU import program. From the **Catalog** tab in Commerce, click **Manage Catalog** and select **Import**. In the **Import** dialog, click **Browse** and locate the CSV file to import. Click **Upload File,** click **Validate,** and then click **Import.**

Add Customize Button to the Product Details widget

A Customize button must be added to the Product details widget to allow product customization.

You must add a Customize button to the Product Details widget so that the button is visible to Commerce self-service users from the Product Details page for a customizable product.

To add a Customize button to the Product Details widget:

1. Log in to Commerce.
2. Click on the **Menu** icon.
3. Select **Design** from the menu.
4. Select **Product Layout** from the layout list.
5. Delete the **Product Details** widget from the layout.
6. Place a new product details widget on the layout.
7. Click the **Settings** icon for the new **Product Details** widget.
8. From the **Element Library**, place a **Customize** button on the new **Product Details** widget.
9. Publish the changes.

Enable Oracle Configure, Price, Quote quoting integration

Some feature configuration procedures must be completed to enable the Oracle CPQ Quoting integration.

To enable the Oracle Configure, Price, Quote quoting integration, do the following

1. Log in to Commerce.
2. Click on the **Menu** icon.
3. Select **Settings** from the menu.
4. Select **Oracle Integrations** from the sidebar menu.
5. Select **CPQ Quoting** from the drop-down menu.
6. Select the **Enable Integration** check box.

If you are using a multi-site environment you must follow these instructions for each site that uses the Oracle Configure, Price, Quote Quoting integration.

Add Quote Button to Checkout and Order Details pages

You must add a Quote button to the Checkout layout and the Quote Details widget to make quoting capability available.

To make the Oracle Configure, Price, Quote quoting capability available to Commerce self-service users, you must add the Request Quote widget to the Checkout layout and the Quote Details widget to the Order Details layout.

The **Request Quote** widget adds a **Quote Notes** text box and a **Request Quote** button to the **Checkout** layout.

The **Quote Details** widget adds a **Quote Notes** text box populated with any notes associated with the order to the **Order Detail** layout. The widget also adds a **Reject Quote**, **Request Re-Quote**, and **Accept Quote** buttons to the to the **Order Detail** layout.

The **Quote Details** and **Request Quote** widgets do not display on the layouts by default. The administrator must first make the widgets available and then place them on the **Checkout** and **Order Detail** pages.

To add quote buttons to the **Checkout** and **Order Details** pages:

1. Log in to Commerce.
2. Click the **Menu** icon.
3. Select **Design** from the menu.
4. Select the **Components** tab on the **Design** page.
5. Click **Show Hidden**.
6. Click the **Show** icon for the **Quote Details Widget** and the **Request Quote Widget**.
7. Within the **Design** page, select the **Layouts** tab.
8. From the layout list, select **Checkout Layout**.
9. Drag and drop the **Request Quote** widget from the **Components** menu to the desired location on the **Checkout** layout.

10. From the layout list, select **Order Details**.
11. Drag and drop the **Quote Details** widget from the **Components** menu to the desired location on the **Order Details** layout.
12. Publish the changes.

Enable Asset Based Ordering

The asset based ordering feature of the integration needs to be enabled before it can be used.

To enable Asset Based Ordering, you must make sure that you have set up the right integration webhooks and/or SSEs mentioned in the Configure the Commerce Webhooks and Configure the Commerce Server Side Extensions sections of this document.

Enable Subscription Cloud integration

Information about the integration of Oracle Commerce and Subscription Cloud using CPQ which supports Self-Service subscriptions for configurable products.

Integration includes using CPQ, OSS and OCC Support complex OCC-CPQ-OSS subscription flows such as:

- Create Subscription
- View A Subscription
- Modify/Upgrade/Downgrade a Subscription
- Cancel/Termination a Subscription
- Renew Subscription - this feature is dependent on subscription management system to provide the renewal details of the subscription products

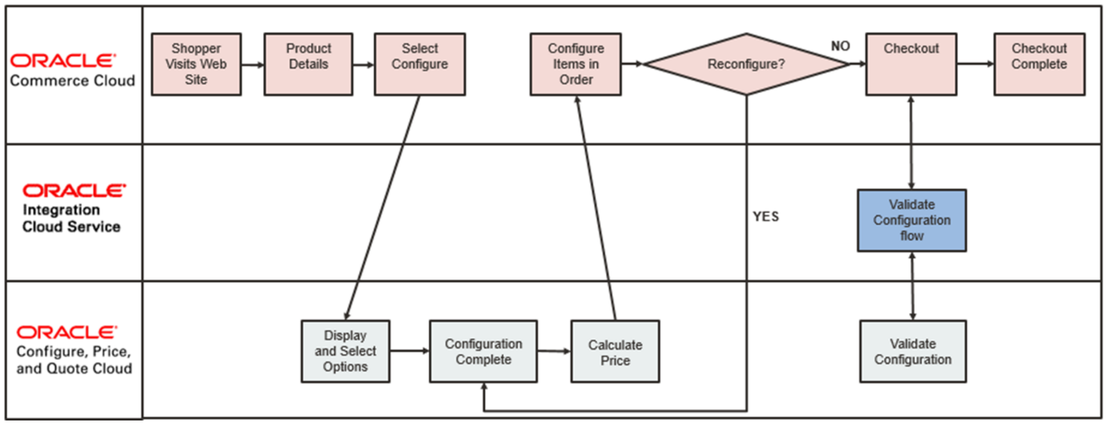
For the above integration with Subscription Cloud, a Store user must be available in Customer Data Management System (CDM). The reference which is `PrimaryPartyId` would be shared with Subscription System in all functional conversations as mentioned above. The primary party id is stored as a dynamic property in user profile.

Only the configured product, which has external recurring charge details is considered as a subscription line items in OIC layer and the rest of the items in the order are filtered out.

Appendix A: Understand the Configurator Flow

A Configurator process flow occurs between Oracle Configure, Price, Quote and Commerce during the integration.

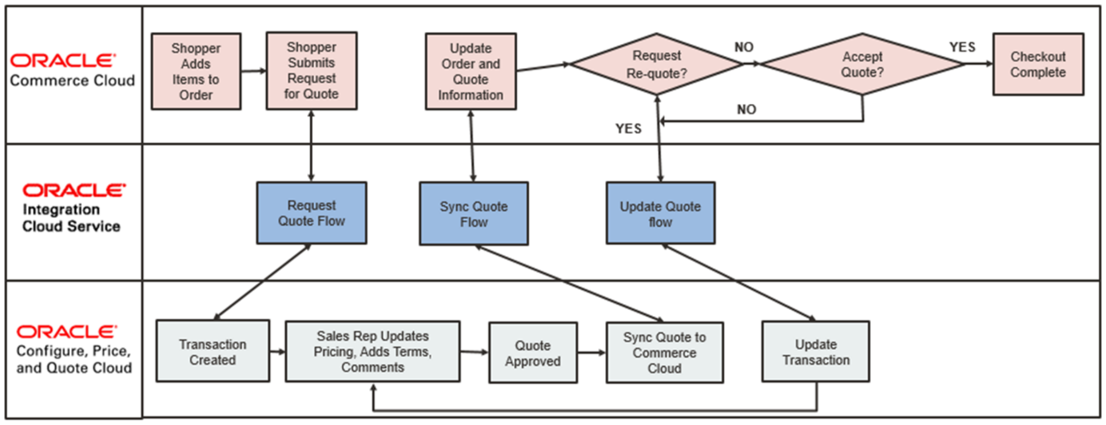
The following presents a diagram of the integration Configurator Flow:



Appendix B: Understand the Request for Quote Flow

A Request for Quote process flow occurs between Oracle Configure, Price, Quote and Commerce during the integration.

The following presents a diagram of the integration Request for Quote flow:


Commerce Cloud
Integration Cloud Service
Configure, Price, and Quote Cloud
Shopper Adds Items to Order
Shopper Submits Request for Quote
Update Order and Quote Information
Request Re-quote?
NO
Accept Quote?
YES
Checkout Complete
NO
YES
Request Quote Flow
Sync Quote Flow
Update Quote flow
Transaction Created
Sales Rep Updates Pricing, Adds Terms, Comments
Quote Approved
Sync Quote to Commerce Cloud
Update Transaction

Appendix C: Understand the OIC Integration Mappings

You must be able to understand the variable mappings for each integration as a requirement to complete the Sync Quote action in Oracle Configure, Price, Quote.

Importing and setting up the OIC package is a prerequisite to completing the Sync Quote action in Oracle Configure, Price, Quote.

After all Oracle Configure, Price, Quote setup is completed, regenerate the OIC integration flows to ensure they accurately reflect the current state of the *Oracle Quote to Order* Commerce process.

Note: Mappings in bold indicate complex, conditional mappings. Mappings in italics indicate the mappings are a static text value instead of a source attribute.

Integration Flow	Target Variable Name	Mapping	Comments
OCCS-CPQ Create Quote > New_Transaction		-	None
-	`cC_RequesterNote_t`	`requesterNote`	None

Integration Flow	Target Variable Name	Mapping	Comments
-	`cC_OrgId_t`	*organizationId*	None
-	`cC_OrderId_t`	*id*	None
-	`cC_SiteId_t`	`siteId`	None
-	`cC_RequesterNote_t`	`requesterNote`	None
-	`currencyCode`	*currencyCode*	None
-	`_customer_t_address`	***shippingGroups >address1***	None
-	`_customer_t_state`	***shippingGroups > state***	None
-	`_customer_t_address_2`	***shippingGroups > address2***	None
-	`_customer_t_company_name`	*shippingGroups > companyName*	None
-	`_customer_t_country`	***shippingGroups > country***	None
-	`_customer_t_city`	**`shippingGroups > city`**	None
-	`_customer_t_zip`	***shippingGroups > postalCode***	None
-	`_customer_t_phone`	***shippingGroups > phoneNumber***	None
-	`_customer_t_email`	*email*	None
-	`_customer_t_last_name`	*lastName*	None
-	`_customer_t_first_name`	*firstName*	None
-	`items`	*commerceItems*	None
-	*_price_book_var_name*	*_default_price_book*	None
-	`_configuration_id`	*configuratorId*	None
-	`cC_CommerceItemId_l`	*id*	None
-	`_part_number`	*catalogRefId*	None
-	`cC_CatalogRefId_l`	*catalogRefId*	None
-	`_price_quantity`	*quantity*	None
-	`cC_ProductId_l`	*productId*	None
-	`cC_NetPrice_l > value`	***priceInfo > amount > quantity***	None
-	`cC_NetPrice_l > currency`	*priceInfo > currencyCode*	None
-	`_modify_action`	*cleanSave_t*	None
OCCS-CPQ Create Quote > Update_Quote		-	None
-	`id`	*id*	None
-	`externalId`	*bs_id*	None
OCCS-CPQ Create Quote > Re-Request_Quote		-	None
-	`cC_RequesterNote_t`	*requesterNote*	None
-	`id`	*externalId*	None
OCCS-CPQ Sync Quote	-	-	None

Integration Flow	Target Variable Name	Mapping	Comments
-	`id`	*cC_OrderId_t*	None
-	`providerNote`	*cC_ProviderNote_t*	None
-	`agentId`	*cC_AgentId_t*	None
-	`externalId`	*id*	None
-	`expirationDate`	***`cC_ExpirationDate_t`***	None
-	`externalPrice`	*totalOneTimeNetAmount_t*	None
line-item	-	-	None
-	`productId`	*cC_ProductId_l*	None
-	`catalogRefId`	*cC_CatalogRefId_l*	None
-	`configuratorId`	*_configuration_id*	None
-	`externalPrice`	`netPrice_l`	None
-	`externalPriceQuantity`	*-1*	None
-	`id`	`cC_CommerceItemId_l`	None
-	`actionCode`	`oRCL_ABO_ActionCode_l`	None
-	`quantity`	**`requestedQuantity_l`**	None

-	`externalData`	`configattrinfo`	XSL manipulations to feed `config` attributes as an array of maps.
			Format:
			`<externalData>`
			`<name></name>`
			`<values>`
			`<name></name>`
			`<variableName></variableName>`
			`<label>Id</label>`
			`<displayValue></displayValue>`
			`<value></value>`
			`</values>`
			`</externalData>`
			`< externalData>`
			`<name></name>`
			`<values>`
			`<name></name>`
			`<variableName></variableName>`
			`<label></label>`
			`<displayValue></displayValue>`
			`<value></value>`
			`</values>`
			`</ externalData>`

Integration Flow	Target Variable Name	Mapping	Comments
OCCS-CPQ Update Quote > Accept Quote		-	None
-	`id`	`externalId`	None
-	`cC_AgentId_t`	`agentId`	None
OCCS-CPQ Update Quote > Reject Quote		-	None
-	`id`	`externalId`	None
-	`cC_AgentId_t`	`agentId`	None
-	`cC_RejectionDate_t`	`date`	None
-	`rejectExplanation_t`	`note`	None
OCCS-CPQ Update Quote > Cancel Quote		-	None
-	`id`	`externalId`	None
-	`cC_AgentId_t`	`agentId`	None
-	`cC_RejectionDate_t`	`date`	None
-	`rejectExplanation_t`	`note`	None
OCCS-CPQ Get Configurations		-	None
-	`locale`	`locale`	None
-	`currency`	`currencyCode`	None
-	`configurationId`	**`configuratorId`**	None
-	`price`	*true*	None
-	`spare`	*true*	None
-	`bomMapping`	*true*	None
OCCS-CPQ Get Assets			
-	`limit`	`limit`	None
-	`offset`	`offset`	None
-	`q`	`for-each(id), for-each(id), for-each(recordId), "{$and:[{$or:[", "{id:{$eq: "", recordId, ""}}", ""}},", "]}", ",", "{$and:[", "{$or:[", "{customer:{$eq:"", id, ""}}", ""}},", "]}", ",", "{$or:[", "{serviceAccount:{$eq:"", id, ""}}", ""}},", "]}", "]}"`	None
-	`expand`	*descendantAssets*	None
OCCS-CPQ Asset Actions (for all flows)			
-	`id`	recordId	None
-	`sourceIdentifier`	`sourceIdentifier`	None
-	`transactionDate`	`transactionDate`	None
-	`transactionId`	`transactionId`	None
OCCS-CPQ Asset Actions (CpqModifyAsset flow)			
-	`productLine`	`product_line`	None

Integration Flow	Target Variable Name	Mapping	Comments
-	`configContextKey`	`configContextKey`	None
-	`configuratorUrl`	`configuratorURL`	None
-	`bomKey`	`bomkey`	None
-	`segment`	`segment`	None
-	`model`	`model`	None
OCCS-CPQ Asset Actions (CpqRenewAsset, CpqTerminateAsset, CpqSuspendAsset, CpqResumeAsset flows)			
-	`configId`	`lineId`	None
-	`serviceAccountId`	`serviceAccount`	None
-	`deactivationDate`	`endDate`	None
-	`amount`	`amount`	None
-	`quantity`	`quantity`	None
-	`parentServiceId`	`parentId`	None
-	`externalRecurringCharge`	`field5`	Corresponds to part `custom field 5` in Oracle CPQ
-	`externalData`	`attributes`	None
-	`billingAccountId`	`billingAccount`	None
-	`externalRecurringChargeFrequency`	`field4`	Corresponds to part `custom field 4` in Oracle Configure, Price, Quote
-	`childItems`	`for-each(children), for-each(partNumber)`	None
-	`catalogRefId`	`partNumber`	None
-	`configuratorId`	`lineId`	None
-	`externalRecurringDuration`	`field6`	Corresponds to part custom field 6 in Oracle Configure, Price, Quote
-	`externalPrice`	`_price_unit_price_each`	None
-	`assetId`	`id`	None
-	`actionCode`	`oRCL_ABO_ActionCode_l`	None
-	`serviceId`	`id`	None
-	`activationDate`	`startDate`	None

Appendix D: Understand the Add to Cart BML – Customized Integrations (19C and Earlier)

Users with legacy integration sites (19C and earlier) who have previously customized their Add to Cart BML need to modify their BML to include site-specific reference file locations.

The following provides the Add to Cart BML for Customized Integrations 19C and Earlier:

```
// Rec Item Properties
part = String[1];
quantity = String[1];
price = String[1];
selected = String[1];
sparepaths = String[1];
sparepaths[0] = "/configuration/configureResponse/spare/rule/item/
part";
sparepaths[1] = "/configuration/configureResponse/spare/rule/item/
quantity";
sparepaths[2] = "/configuration/configureResponse/spare/rule/item/
price";
sparepaths[3] = "/configuration/configureResponse/spare/rule/item/
selected";

// BOM Item Properties
bomItem = String[1];
bomItem[0] = "/configuration/configureResponse/bomItem";

// Model/Price Properties
models = string[1];
configIdSearch = string[1];
currpath = String[1];
totalPrices = string[1];
bomTotals = string[1];
models[0] = "/configuration/configureResponse/item/model";
configIdSearch[0] = "/configuration/configureResponse/item/
@configurationId";
currpath[0] = "/configuration/configureResponse/attributes/
attribute[@_variableName='currencyCode']/value";
totalPrices[0] = "/configuration/configureResponse/price/totalPrice";
bomTotals[0] = "/configuration/configureResponse/price/bomPrice";
priceTotal = 0.0;
baseModelPrice = 0.0;
recurringSubtotal = 0.0;

// Extract data from configXML
outputModel = readxmlsingle(configXML, models);
outputConfigIds = readxmlsingle(configXML, configIdSearch);
currXML = readxmlsingle(configXML, currpath);
currency = get(currXML, currpath[0]);
outputPrices = readxmlsingle(configXML, totalPrices);
bomPrices = readxmlsingle(configXML, bomTotals);
output1 = readxmlmultiple(configXML, sparepaths);
bomItemXMLDict = readxmlsingle(configXML, bomItem);
bomItemString = get(bomItemXMLDict, "/configuration/configureResponse/
bomItem");

payloadTemplate = urldatabyget("https://cpq-046.us.example.com/bmfsweb/
slc10xgj/image/CommerceCloud/AddToCartPayload-Cloud.txt", "", "");
model1 = "";
totalPrice1 = "";
```

```
// Get Model data
for model in models {
  model1 = get(outputModel, model);
}

// Get Price data
for totalPrice in totalPrices {
  totalPrice1 = get(outputPrices, totalPrice);
  totalPrice0 = replace(totalPrice1, ",", "");
  if (isnumber(substring(totalPrice0, 1))) {
    totalPrice2 = getcurrencyvalue(totalPrice1, currency);
    priceTotal = priceTotal + totalPrice2;
  }
}
baseModelPrice = priceTotal;

// Add BOM total price
if (containskey(bomPrices, bomTotals[0])) {
  for bomPrice in bomTotals {
    bomTotal = get(bomPrices, bomPrice);
    bomTotalReplace = replace(bomTotal, ",", "");
    if (isnumber(substring(bomTotalReplace, 1))) {
      bomTotalPrice = getcurrencyvalue(bomTotal, currency);
      priceTotal = bomTotalPrice + priceTotal;
    }
  }
}

// Get ConfigID
configId = "";
for id in configIdSearch {
  configId = get(outputConfigIds, id);
}

// Get Recommended Items
for sparepath in sparepaths {
  if (find(sparepath, "part") <  > -1) {
    part = get(output1, sparepath);
  }
  elif(find(sparepath, "quantity") <  > -1) {
    quantity = get(output1, sparepath);
  }
  elif(find(sparepath, "price") <  > -1) {
    price = get(output1, sparepath);
  }
  elif(find(sparepath, "selected") <  > -1) {
    selected = get(output1, sparepath);
  }
}

// Format Rec Items payload
recItemList = "";
if (isnull(part)) {
  print("No Recommended Items");
```

```
} else {
  recItems = sizeofarray(part);
  recItemsInt = integer[recItems];

  i = 0;
  for recItem in recItemsInt {
    if (selected[i] == "true") {
      //recurring price from parts BMQL
      part_num = part[i];
      partCustomFields = bmql("SELECT part_number, custom_field5,
custom_field4, custom_field6, custom_field8 FROM _parts WHERE
part_number = $part_num");
      recItemPayloadTemplate = urldatabyget("https://
cpq-046.us.example.com/bmfsweb/slc10xgj/image/CommerceCloud/
Recommended_Items_Payload-Cloud.txt", "", "");
      recItemPayloadTemplate = replace(recItemPayloadTemplate,
"{{quantity}}", quantity[i]);
      recItemPayloadTemplate = replace(recItemPayloadTemplate,
"{{part}}", part[i]);

      for each in partCustomFields {
        if (get(each, "custom_field8") == "Recurring") {
          recItemPayloadTemplate = replace(recItemPayloadTemplate,
"{{pricePeriod}}", get(each, "custom_field4"));
          recItemPayloadTemplate = replace(recItemPayloadTemplate,
"{{recurringPrice}}", get(each, "custom_field5"));
          recItemPayloadTemplate = replace(recItemPayloadTemplate,
"{{duration}}", get(each, "custom_field6"));
          //recurringSubtotal = recurringSubtotal + get(each,
"custom_field5");
        } else {
          childPayloadJson = json(recItemPayloadTemplate);
          jsonremove(childPayloadJson, "recurringCharge");
          recItemPayloadTemplate = jsontostr(childPayloadJson);
        }
      }

      //remove region specific formatting for price
      sPrice0 = substring(price[i], 1);
      sPrice0 = replace(sPrice0, ",", "");

      if (isnumber(sPrice0)) {
        priceTotal = priceTotal + atof(sPrice0);
        recItemPayloadTemplate = replace(recItemPayloadTemplate,
"{{price}}", sPrice0);
      } else {
        recItemPayloadTemplate = replace(recItemPayloadTemplate,
"{{price}}", "0");
      }
      if (recItemList == "") {
        recItemList = recItemPayloadTemplate;
      } else {
        recItemList = recItemList + "," + recItemPayloadTemplate;
      }
    }
    i = i + 1;
  }
}

// Get the BOM Items
if (isnull(bomItemString)) {
  print "No BOM Items";
  bomItemString = "";
  payloadTemplate = replace(payloadTemplate, "{{BomItems}}", bomItemString);
} else {
  // Get part numbers for each BOM item, convert to string array for bmql
  bomJson = json(bomItemString);

  // Remove extraneous BOM fields (may have to revert if CC was expecting to
use them)
  jsonpathremove(bomJson, "$..variableName");
  jsonpathremove(bomJson, "$..definition");
  jsonpathremove(bomJson, "$..category");

  // Replacing all 0 prices with actual number 0
  bomPriceArray = jsonpathgetmultiple(bomJson, "$.._price_unit_price_each");
  replace_lookup = boolean[];
  bomPricesString = jsonarraytostr(bomPriceArray);
  bomPricesString = replace(replace(replace(bomPricesString, "\"", ""), "[",
""), "]", "");
  bomPricesStringArray = split(bomPricesString, ",");

  i = 0;
  for each in bomPricesStringArray {
    append(replace_lookup, isnumber(each));
    i = i + 1;
  }

  i = 0;
  for each in replace_lookup {
    if (i == 0 and each == false) {
      jsonpathset(bomJson, "$.fields._price_unit_price_each", "0");
    }
    elif(each == false) {
      str = "$.children[" + string(i - 1) +
"].fields._price_unit_price_each";
      jsonpathset(bomJson, str, "0");
    }

    i = i + 1;
  }

  bomItemString = jsontostr(bomJson);
  bomPartsArray = jsonpathgetmultiple(bomJson, "$..partNumber");
  bomPartsString = jsonarraytostr(bomPartsArray);
  bomPartsString = replace(replace(replace(bomPartsString, "\"", ""), "[",
""), "]", "");
  bomPartsStringArray = split(bomPartsString, ",");
  bomParts = bmql("SELECT part_number, custom_field5, custom_field4,
custom_field6, custom_field8 FROM _parts WHERE part_number
```

```
IN $bomPartsStringArray");

  // Get path for each part, add recurringCharge to them all
  for each in bomParts {
    partField = "\"partNumber\":\"" + get(each, "part_number") + "\",";
    recurringTemplate = "\"recurringCharge\":
{ \"amount\":,\"frequency\":,\"duration\":},";

    if (get(each, "custom_field8") == "Recurring") {
      recurringTemplate = replace(recurringTemplate, "frequency\":",
"frequency\":\"" + get(each, "custom_field4") + "\"");
      recurringTemplate = replace(recurringTemplate, "amount\":",
"amount\":\"" + get(each, "custom_field5") + "\"");
      recurringTemplate = replace(recurringTemplate, "duration\":",
"duration\":\"" + get(each, "custom_field6") + "\"");
    } else {
      recurringTemplate = "";
    }
    bomItemString = replace(bomItemString, partField, partField +
recurringTemplate);
  }

  // Unflatten
  bomItemString = replace(bomItemString, "\"partNumber\":",
"\"catalogRefId\":");
  bomItemString = replace(bomItemString, "On Request", "0"); // This
may only fix English users
  bomJson = convertbomtohier(json(bomItemString));
  payloadTemplate = replace(payloadTemplate, "{{BomItems}}",
jsontostr(bomJson));
}

// Format main template with subcomponents and properties
payloadTemplate = replace(payloadTemplate, "{{commerceItemId}}", "");
payloadTemplate = replace(payloadTemplate, "{{ConfigId}}", configId);
payloadTemplate = replace(payloadTemplate, "{{model}}", model1);
payloadTemplate = replace(payloadTemplate, "{{totalPrice}}",
string(priceTotal));
payloadTemplate = replace(payloadTemplate, "{{basePrice}}",
string(baseModelPrice));
payloadTemplate = replace(payloadTemplate, "{{currency}}", currency);
payloadTemplate = replace(payloadTemplate, "{{ChildItems}}",
recItemList);
return payloadTemplate;
```

Appendix E: Understand the Add to Cart BML – Customized Integrations and Multi-Site Set Up (19D and Later)

Users with customized integrations and multi-site set ups (19D and later) who have previously customized their Add to Cart BML need to modify and update their BML.

The following provides the Add to Cart BML for Customized Integrations and Multi-Site Set Up 19D and later:

```
// Initialize variables
MODEL_PATH = "/configuration/configureResponse/item/model";
CONFIG_ID_PATH = "/configuration/configureResponse/item/@configurationId";
CURRENCY_CODE_PATH = "/configuration/configureResponse/attributes/
attribute[@_variableName='currencyCode']/value";
TOTAL_PRICE_PATH = "/configuration/configureResponse/price/totalPrice";
SPARE_PART_PATH = "/configuration/configureResponse/spare/rule/item/part";
SPARE_QUANTITY_PATH = "/configuration/configureResponse/spare/rule/item/
quantity";
SPARE_PRICE_PATH = "/configuration/configureResponse/spare/rule/item/price";
SPARE_SELECTED_PATH = "/configuration/configureResponse/spare/rule/item/
selected";
BOM_ITEM_PATH = "/configuration/configureResponse/bomItem";
BOM_PRICE_PATH = "/configuration/configureResponse/price/bomPrice";

CART_TEMPLATE_LOCATION = "$BASE_PATH$/CommerceCloud/AddToCartPayload-
Cloud.txt";
SPARE_TEMPLATE_LOCATION = "$BASE_PATH$/CommerceCloud/
Recommended_Items_Payload-Cloud.txt";

payload = "";
sparesList = "";
priceTotal = 0.0;
baseModelPrice = 0.0;
sparePart = String[1];
spareQuantity = String[1];
sparePrice = String[1];
spareSelected = String[1];
singleSpareDict = dict("string");
configDict = dict("string");

// Create array of XML paths:
pathArray = string[];
sparePathArray = string[];

// For Model/Price Properties
append(pathArray, MODEL_PATH);
append(pathArray, CONFIG_ID_PATH);
append(pathArray, CURRENCY_CODE_PATH);
append(pathArray, TOTAL_PRICE_PATH);

// For BOM Item Property
append(pathArray, BOM_ITEM_PATH);
append(pathArray, BOM_PRICE_PATH);

// For Rec Item Properties (needs its own array)
append(sparePathArray, SPARE_PART_PATH);
append(sparePathArray, SPARE_QUANTITY_PATH);
append(sparePathArray, SPARE_PRICE_PATH);
append(sparePathArray, SPARE_SELECTED_PATH);

// Extract data from configXML
```

```
pathDict = readxmlsingle(configXML, pathArray);
spareDict = readxmlmultiple(configXML, sparePathArray);

model = get(pathDict, MODEL_PATH);
configId = get(pathDict, CONFIG_ID_PATH);
currency = get(pathDict, CURRENCY_CODE_PATH);
totalPrice = get(pathDict, TOTAL_PRICE_PATH);
bomPrice = get(pathDict, BOM_PRICE_PATH);
bomItem = get(pathDict, BOM_ITEM_PATH);

// Convert totalPrice (which is a misleading name) to numeric value,
set as baseModelPrice
totalPrice = replace(totalPrice, ",", "");
if (isnumber(substring(totalPrice, 1))) {
    totalPriceNum = getcurrencyvalue(totalPrice, currency);
    priceTotal = priceTotal + totalPriceNum;
}
baseModelPrice = priceTotal;

// Add BOM total price to priceTotal (which is the REAL total price),
with the same conversion as the base price
if (NOT(isnull(bomPrice))) {
    bomPrice = replace(bomPrice, ",", "");
    if (isnumber(substring(bomPrice, 1))) {
        bomPriceNum = getcurrencyvalue(bomPrice, currency);
        priceTotal = bomPriceNum + priceTotal;
    }
}

// Get Recommended Items
for sparepath in sparePathArray {
    if (find(sparepath, "part") <> -1) {
        sparePart = get(spareDict, sparepath);
    }
    elif(find(sparepath, "quantity") <> -1) {
        spareQuantity = get(spareDict, sparepath);
    }
    elif(find(sparepath, "price") <> -1) {
        sparePrice = get(spareDict, sparepath);
    }
    elif(find(sparepath, "selected") <> -1) {
        spareSelected = get(spareDict, sparepath);
    }
}

// Format Rec Items payload
if (isnull(sparePart)) {
    print "No Recommended Items";
} else {
    spareListSize = sizeofarray(sparePart);
    spareArray = integer[spareListSize];

    i = 0;
    for eachSpare in spareArray {
        if (spareSelected[i] == "true") {
            //Convert price, similar to Base and BOM prices above
            priceString = substring(sparePrice[i], 1);
            priceString = replace(priceString, ",", "");
            if (isnumber(priceString)) {
                sparePrice[i] = string(getcurrencyvalue(priceString, 
currency));
                priceTotal = priceTotal + atof(sparePrice[i]);
            } else {
                sparePrice[i] = "0";
            }

            // Add basic part fields to dictionary from array dictionary
            put(singleSpareDict, "part", sparePart[i]);
            put(singleSpareDict, "quantity", spareQuantity[i]);
            put(singleSpareDict, "price", sparePrice[i]);

            // Generate template and set values from dictionary
            singleSparePayload = applytemplate(SPARE_TEMPLATE_LOCATION, 
singleSpareDict);

            // Get Recurring Charge fields
            part_num = sparePart[i];
            partCustomFieldsDict = bmql("SELECT part_number, custom_field5, 
custom_field4, custom_field6, custom_field8 FROM _parts WHERE part_number 
= $part_num");

            for each in partCustomFieldsDict {
                if (get(each, "custom_field8") == "Recurring") {
                    singleSparePayload = replace(singleSparePayload, 
"{{pricePeriod}}", get(each, "custom_field4"));
                    singleSparePayload = replace(singleSparePayload, 
"{{recurringPrice}}", get(each, "custom_field5"));
                    singleSparePayload = replace(singleSparePayload, 
"{{duration}}", get(each, "custom_field6"));
                } else {
                    childPayloadJson = json(singleSparePayload);
                    jsonremove(childPayloadJson, "recurringCharge");
                    singleSparePayload = jsontostr(childPayloadJson);
                }
            }

            // Add Item to List
            if (sparesList == "") {
                sparesList = singleSparePayload;
            } else {
                sparesList = sparesList + "," + singleSparePayload;
            }
        }
        i = i + 1;
    }
}

// Get the BOM Items
if (isnull(bomItem)) {
    print "No BOM Items";
    bomItem = "";
} else {
    // Get part numbers for each BOM item, convert to string array for
bmql
    bomJson = json(bomItem);

    // Remove extraneous BOM fields (may have to revert if CC was
expecting to use them)
    jsonpathremove(bomJson, "$..variableName");
    jsonpathremove(bomJson, "$..definition");
    jsonpathremove(bomJson, "$..category");

    // Replacing all 0 prices with actual number 0
    bomPriceArray = jsonpathgetmultiple(bomJson,
"$.._price_unit_price_each");
    replace_lookup = boolean[];
    bomPricesString = jsonarraytostr(bomPriceArray);
    bomPricesString = replace(replace(replace(bomPricesString, "\"",
""), "[", ""), "]", "");
    bomPricesStringArray = split(bomPricesString, ",");

    i = 0;
    for each in bomPricesStringArray {
        append(replace_lookup, isnumber(each));
        i = i + 1;
    }

    i = 0;
    for each in replace_lookup {
        if (i == 0 and each == false) {
            jsonpathset(bomJson, "$.fields._price_unit_price_each",
"0");
        }
        elif(each == false) {
            str = "$.children[" + string(i - 1) +
"].fields._price_unit_price_each";
            jsonpathset(bomJson, str, "0");
        }

        i = i + 1;
    }

    bomItem = jsontostr(bomJson);
    bomPartsArray = jsonpathgetmultiple(bomJson, "$..partNumber");
    bomPartsString = jsonarraytostr(bomPartsArray);
    bomPartsString = replace(replace(replace(bomPartsString, "\"",
""), "[", ""), "]", "");
    bomPartsStringArray = split(bomPartsString, ",");
    bomParts = bmql("SELECT part_number, custom_field5, custom_field4,
custom_field6, custom_field8 FROM _parts WHERE part_number
IN $bomPartsStringArray");

    // Get path for each part, add recurringCharge to them all
    for each in bomParts {
        partField = "\"partNumber\":\"" + get(each, "part_number") +
"\",";
        recurringTemplate = "\"recurringCharge\":
{ \"amount\":,\"frequency\":,\"duration\":},";

        if (get(each, "custom_field8") == "Recurring") {
            recurringTemplate = replace(recurringTemplate, "frequency\":",
"frequency\":\"" + get(each, "custom_field4") + "\"");
            recurringTemplate = replace(recurringTemplate, "amount\":",
"amount\":\"" + get(each, "custom_field5") + "\"");
            recurringTemplate = replace(recurringTemplate, "duration\":",
"duration\":\"" + get(each, "custom_field6") + "\"");
        } else {
            recurringTemplate = "";
        }
        bomItem = replace(bomItem, partField, partField + recurringTemplate);
    }

    // Handle 0 prices in configuration (this may only fix English users)
    bomItem = replace(bomItem, "\"partNumber\":", "\"catalogRefId\":");
    bomItem = replace(bomItem, "On Request", "0");

    // Unflatten
    bomJson = convertbomtohier(json(bomItem));
    bomItem = jsontostr(bomJson);
}

// Format main template with subcomponents and properties
put(configDict, "commerceItemId", "");
put(configDict, "model", model);
put(configDict, "ConfigId", configId);
put(configDict, "currency", currency);
put(configDict, "totalPrice", string(priceTotal));
put(configDict, "basePrice", string(baseModelPrice));
put(configDict, "ChildItems", sparesList);
put(configDict, "BomItems", bomItem);
payload = applytemplate(CART_TEMPLATE_LOCATION, configDict);
payload = replace(payload, "&quot;", "\""); // encoding bug on applytemplate

return payload;
```

Appendix F: Understand the SyncQuote BML

You must modify the function BML to set the Sync Quote action to run Advanced Modify for the integration.

The following provides the SyncQuote BML used in the integration:

```
str = "";

for each in transactionLine{
    if (each._model_variable_name <> ""){
        lineItem_array = split(cC_LineItem_Data_t, "|");
        for lineItem in lineItem_array {
            row = split(lineItem, "~");
            if(row[0] == each._document_number){
                str = str + each._document_number + 
"~cC_CommerceItemId_l~" + row[1]+"|";
                str = str + each._document_number + "~cC_ProductId_l~" 
+ row[2]+"|";
            }
        }
    }
}

return str;
```

Appendix G: AddToCartPayload-Cloud

Example of the `AddToCartPayload-Cloud.txt` file.

The following is an example of the `AddToCartPayload-Cloud.txt` file.

```
// Initialize variables
MODEL_PATH = "/configuration/configureResponse/item/model";
CONFIG_ID_PATH = "/configuration/configureResponse/item/
@configurationId";
CURRENCY_CODE_PATH = "/configuration/configureResponse/attributes/
attribute[@_variableName='currencyCode']/value";
TOTAL_PRICE_PATH = "/configuration/configureResponse/price/totalPrice";
SPARE_PART_PATH = "/configuration/configureResponse/spare/rule/item/
part";
SPARE_QUANTITY_PATH = "/configuration/configureResponse/spare/rule/
item/quantity";
SPARE_PRICE_PATH = "/configuration/configureResponse/spare/rule/item/
price";
SPARE_SELECTED_PATH = "/configuration/configureResponse/spare/rule/
item/selected";
BOM_ITEM_PATH = "/configuration/configureResponse/bomItem";
BOM_PRICE_PATH = "/configuration/configureResponse/price/bomPrice";
DELTA_PRICE_PATH = "/configuration/configureResponse/price/deltaPrice";

CART_TEMPLATE_LOCATION = "$BASE_PATH$/CommerceCloud/AddToCartPayload-
Cloud.txt";
SPARE_TEMPLATE_LOCATION = "$BASE_PATH$/CommerceCloud/
Recommended_Items_Payload-Cloud.txt";

payload = "";
sparesList = "";
priceTotal = 0.0;
baseModelPrice = 0.0;
totalDeltaPrice = 0.0;
sparePart = String[1];
spareQuantity = String[1];
sparePrice = String[1];
spareSelected = String[1];
singleSpareDict = dict("string");
configDict = dict("string");
```

```
// Create array of XML paths:
pathArray = string[];
sparePathArray = string[];

// For Model/Price Properties
append(pathArray, MODEL_PATH);
append(pathArray, CONFIG_ID_PATH);
append(pathArray, CURRENCY_CODE_PATH);
append(pathArray, TOTAL_PRICE_PATH);
append(pathArray, DELTA_PRICE_PATH);

// For BOM Item Property
append(pathArray, BOM_ITEM_PATH);
append(pathArray, BOM_PRICE_PATH);

// For Rec Item Properties (needs its own array)
append(sparePathArray, SPARE_PART_PATH);
append(sparePathArray, SPARE_QUANTITY_PATH);
append(sparePathArray, SPARE_PRICE_PATH);
append(sparePathArray, SPARE_SELECTED_PATH);

// Extract data from configXML
pathDict = readxmlsingle(configXML, pathArray);
spareDict = readxmlmultiple(configXML, sparePathArray);

model = get(pathDict, MODEL_PATH);
configId = get(pathDict, CONFIG_ID_PATH);
currency = get(pathDict, CURRENCY_CODE_PATH);
totalPrice = get(pathDict, TOTAL_PRICE_PATH);
bomPrice = get(pathDict, BOM_PRICE_PATH);
deltaPrice = get(pathDict, DELTA_PRICE_PATH);
bomItem = get(pathDict, BOM_ITEM_PATH);

// Convert totalPrice (which is a misleading name) to numeric value, set as
baseModelPrice
totalPrice = replace(totalPrice, ",", "");
if (isnumber(substring(totalPrice, 1))) {
    totalPriceNum = getcurrencyvalue(totalPrice, currency);
    priceTotal = priceTotal + totalPriceNum;
}
baseModelPrice = priceTotal;

// Convert deltaPrice to numeric value, set as totalDeltaPrice
if (NOT(isnull(deltaPrice))) {
    deltaPrice = replace(deltaPrice, ",", "");
    if (isnumber(substring(deltaPrice, 1))) {
        totalDeltaPrice = getcurrencyvalue(deltaPrice, currency);
    }
}

// Add BOM total price to priceTotal (which is the REAL total price), with
the same conversion as the base price
if (NOT(isnull(bomPrice))) {
    bomPrice = replace(bomPrice, ",", "");
    if (isnumber(substring(bomPrice, 1))) {
        bomPriceNum = getcurrencyvalue(bomPrice, currency);
        priceTotal = bomPriceNum + priceTotal;
    }
}

// Get Recommended Items
for sparepath in sparePathArray {
    if (find(sparepath, "part") <> -1) {
        sparePart = get(spareDict, sparepath);
    }
    elif(find(sparepath, "quantity") <> -1) {
        spareQuantity = get(spareDict, sparepath);
    }
    elif(find(sparepath, "price") <> -1) {
        sparePrice = get(spareDict, sparepath);
    }
    elif(find(sparepath, "selected") <> -1) {
        spareSelected = get(spareDict, sparepath);
    }
}

// Format Rec Items payload
if (isnull(sparePart)) {
    print("No Recommended Items");
} else {
    spareListSize = sizeofarray(sparePart);
    spareArray = integer[spareListSize];

    i = 0;
    for eachSpare in spareArray {
        if (spareSelected[i] == "true") {
            //Convert price, similar to Base and BOM prices above
            priceString = substring(sparePrice[i], 1);
            priceString = replace(priceString, ",", "");
            if (isnumber(priceString)) {
                sparePrice[i] = string(getcurrencyvalue(priceString,
currency));
                priceTotal = priceTotal + atof(sparePrice[i]);
            } else {
                sparePrice[i] = "0";
            }

            // Add basic part fields to dictionary from array
dictionary
            put(singleSpareDict, "part", sparePart[i]);
            put(singleSpareDict, "quantity", spareQuantity[i]);
            put(singleSpareDict, "price", sparePrice[i]);

            // Generate template and set values from dictionary
            singleSparePayload =
applytemplate(SPARE_TEMPLATE_LOCATION, singleSpareDict);

            // Get Recurring Charge fields
            part_num = sparePart[i];
            partCustomFieldsDict = bmql("SELECT part_number,
custom_field5, custom_field4, custom_field6, custom_field8 FROM _parts WHERE
part_number = $part_num");

            for each in partCustomFieldsDict {
                if (get(each, "custom_field8") == "Recurring") {
                    singleSparePayload = replace(singleSparePayload,
"{{pricePeriod}}", get(each, "custom_field4"));
                    singleSparePayload = replace(singleSparePayload,
"{{recurringPrice}}", get(each, "custom_field5"));
                    singleSparePayload = replace(singleSparePayload,
"{{duration}}", get(each, "custom_field6"));
                } else {
                    childPayloadJson = json(singleSparePayload);
                    jsonremove(childPayloadJson, "recurringCharge");
                    singleSparePayload = jsontostr(childPayloadJson);
                }
            }

            // Add Item to List
            if (sparesList == "") {
                sparesList = singleSparePayload;
            } else {
                sparesList = sparesList + "," + singleSparePayload;
            }
        }
        i = i + 1;
    }
}

// Get the BOM Items
if (isnull(bomItem)) {
    print "No BOM Items";
    bomItem = "";
} else {
    // Get part numbers for each BOM item, convert to string array for bmql
    bomJson = json(bomItem);

    // Remove extraneous BOM fields (may have to revert if CC was expecting
to use them)
    jsonpathremove(bomJson, "$..variableName");
    jsonpathremove(bomJson, "$..definition");
    jsonpathremove(bomJson, "$..category");

    // Replacing all 0 prices with actual number 0
    bomPriceArray = jsonpathgetmultiple(bomJson,
"$.._price_unit_price_each");
    replace_lookup = boolean[];
    bomPricesString = jsonarraytostr(bomPriceArray);
    bomPricesString = replace(replace(replace(bomPricesString, "\"", ""),
"[", ""), "]", "");
    bomPricesStringArray = split(bomPricesString, ",");

    i = 0;
    for each in bomPricesStringArray {
        append(replace_lookup, isnumber(each));
```

```
            i = i + 1;
        }

        i = 0;
        for each in replace_lookup {
            if (i == 0 and each == false) {
                jsonpathset(bomJson, "$.fields._price_unit_price_each", 
"0");
            }
            elif(each == false) {
                str = "$.children[" + string(i - 1) + 
"].fields._price_unit_price_each";
                jsonpathset(bomJson, str, "0");
            }

            i = i + 1;
        }


        // Replacing all 0 delta with actual number 0
        bomDeltaArray = jsonpathgetmultiple(bomJson, "$.._delta_price");
        replace_lookupDelta = boolean[];
        bomDeltaString = jsonarraytostr(bomDeltaArray);
        bomDeltaString = replace(replace(replace(bomDeltaString, "\"", 
""), "[", ""), "]", "");
        bomDeltaStringArray = split(bomDeltaString, ",");

        i = 0;
        for each in bomDeltaStringArray {
            append(replace_lookupDelta, isnumber(each));
            i = i + 1;
        }

        i = 0;
        for each in replace_lookupDelta {
            if (i == 0 and each == false) {
                jsonpathset(bomJson, "$.fields._delta_price", "0");
            }
            elif(each == false) {
                str = "$.children[" + string(i - 1) + 
"].fields._delta_price";
                jsonpathset(bomJson, str, "0");
            }

            i = i + 1;
        }


        bomItem = jsontostr(bomJson);
        bomPartsArray = jsonpathgetmultiple(bomJson, "$..partNumber");
        bomPartsString = jsonarraytostr(bomPartsArray);
        bomPartsString = replace(replace(replace(bomPartsString, "\"", 
""), "[", ""), "]", "");
        bomPartsStringArray = split(bomPartsString, ",");
        bomParts = bmql("SELECT part_number, custom_field5, custom_field4, 
    custom_field6, custom_field8 FROM _parts WHERE part_number
    IN $bomPartsStringArray");

        // Get path for each part, add recurringCharge to them all
        for each in bomParts {
            partField = "\"partNumber\":\"" + get(each, "part_number") + "\",";
            recurringTemplate = "\"recurringCharge\":
    { \"amount\":,\"frequency\":,\"duration\":},";

            if (get(each, "custom_field8") == "Recurring") {
                recurringTemplate = replace(recurringTemplate, "frequency\":",
    "frequency\":\"" + get(each, "custom_field4") + "\"");
                recurringTemplate = replace(recurringTemplate, "amount\":",
    "amount\":\"" + get(each, "custom_field5") + "\"");
                recurringTemplate = replace(recurringTemplate, "duration\":",
    "duration\":\"" + get(each, "custom_field6") + "\"");
            } else {
                recurringTemplate = "";
            }
            bomItem = replace(bomItem, partField, partField + recurringTemplate);
        }

        // Handle 0 prices in configuration (this may only fix English users)
        bomItem = replace(bomItem, "\"partNumber\":", "\"catalogRefId\":");
        bomItem = replace(bomItem, "On Request", "0");

        // Unflatten
        bomJson = convertbomtohier(json(bomItem));
        bomItem = jsontostr(bomJson);
    }

    // Format main template with subcomponents and properties
    put(configDict, "commerceItemId", "");
    put(configDict, "model", model);
    put(configDict, "ConfigId", configId);
    put(configDict, "currency", currency);
    put(configDict, "totalPrice", string(priceTotal));
    put(configDict, "basePrice", string(baseModelPrice));
    put(configDict, "deltaPrice", string(totalDeltaPrice));
    put(configDict, "ChildItems", sparesList);
    put(configDict, "BomItems", bomItem);

    payload = applytemplate(CART_TEMPLATE_LOCATION, configDict);
    payload = replace(payload, "&quot;", "\"");    // encoding bug on
    applytemplate

    return payload;
```

2 Use Oracle CPQ Cloud Features

Use Oracle CPQ features in conjunction with Oracle Commerce.

Oracle CPQ allows you to create quote-to-cash processes, and guides users towards product options and configurations. Integrating these features with Commerce allows you to offer shoppers a method to successfully interact with your business, improving their contact experience and increasing shopper satisfaction.

Introduction

Many important Oracle Configure, Price, Quote features are available via an integration solution between Oracle Configure, Price, Quote andOracle Commerce.

This document is intended to provide the instructions on how to use Oracle Configure, Price, Quote features with Oracle Commerce - via an integration supported by the two solutions.

Oracle Commerce is an eCommerce solution designed specifically to run in the Oracle Cloud. The service provides you with a range of powerful tools to build a flexible, feature-rich storefront for your shoppers.

Activities you can perform with Oracle Commerce include the following:

- Customize the design and layout of your storefront pages and preview your changes
- Display your store content in different languages
- Create or import catalog items
- Manage inventory
- Offer promotions
- Manage shopper accounts
- Allow shoppers to set up wish lists
- View reports about your store
- Test the visual elements of your store to determine which design shoppers prefer
- Develop custom features for your store through the Oracle Commerce web services API

Oracle Configure, Price, Quote is the only cloud solution to support the complete quote-to-cash process - from shopper inquiry to order fulfillment. It guides users to optimal product options and configurations from simple to complex, automatically applying discounts and relevant up-sell and cross-sell opportunities.

Integrating these solutions brings together the capabilities of Oracle Commerce and Oracle Configure, Price, Quote to provide a unified solution that enables businesses to offer shoppers a method of interacting meaningfully with the business during the purchasing process, and to provide agents with the means to be flexible with shoppers, improving their contact experience and maximizing shopper satisfaction.

Objective

By integrating Oracle Commerce and Oracle Configure, Price, Quote, you increase the number of supported available commerce shopper features.

The integration of Oracle Commerce and Oracle Configure, Price, Quote targets support for the following shopper commerce activity:

- **Product configuration**: The shopper or agent can configure any product that has been identified as configurable in the product catalog.
- **Shopper quote request**: The shopper can request a quote for an order.
- **Agent quote request**: An agent dealing with a shopper contact can request a quote for a discount on behalf of the shopper.
- **Asset Based Ordering -** Asset based ordering (ABO) allows you to sell tangible assets or subscription services delivered over a period of time; for example mobile phone call and data plans, television and broadband packages, cloud storage service, music streaming service, etc.

This document provides instructions on how to set up an integration between Oracle Commerce and Oracle Configure, Price, Quote so that relevant Commerce information is automatically passed to Oracle Configure, Price, Quote, ensuring that the decision process has all the required information and increasing the speed at which a reply is delivered to the shopper or agent.

This document describes the setup tasks that must be performed in Oracle Commerce and Oracle Integration Cloud in order to use this integration flow. There are additional setup tasks that must be performed in Oracle Configure, Price, Quote so that the integration works as expected. Full information about these tasks that must be performed in Oracle Configure, Price, Quote can be found in the Integrating Oracle CX Commerce with Oracle Configure, Price, Quote article on My Oracle Support.

Chapter 2 – Configuring the Integration: provides technical instructions on the following topics:

- How to download the Oracle Integration Cloud Integration Flows.
- How to configure the Oracle Integration Cloud Integration Flows.
- How to setup the connection to Oracle Configure, Price, Quote.
- How to setup the connection to Oracle Commerce.
- How to configure the webhooks to trigger the integration flows.
- How to configure the SSEs (Side-Server Extension) necessary for the integration flows.

Chapter 3 – Using the Integration Functionality: provides instructions on how to use the functionality supported by this integration.

Audience

You must follow product-provided documentation to set up and configure the integration between Oracle Commerce and Oracle Configure, Price, Quote systems.

This document is written for Oracle Commerce and Oracle Configure, Price, Quote administrators who need to set up and configure the integration between these two systems.

Readers of this document should have experience with Oracle Commerce, Oracle Configure, Price, Quote and Oracle Integration Cloud (OIC) administration. This document does not provide instructions on configuring aspects other than the integration for Oracle Commerce and Oracle Configure, Price, Quote.

Prerequisites

In order to configure and use the Oracle Commerce/Oracle Configure, Price, Quote integration, there are specific software, account, and data prerequisites that must be met.

For the purposes of this document, it is assumed that you already have:

- An Oracle Commerce account and access to the Oracle Commerce 19.1 or later with necessary SSEs enabled (see sections that follow).
- An Oracle Configure, Price, Quote account and access to Oracle Configure, Price, Quote 19.1 or later.
- An Oracle Integration Cloud account and access to Oracle Integration Cloud Service 18.4.5 or later.
- A synchronized product catalog to ensure that products in the Commerce catalog map to corresponding items in the Oracle Configure, Price, Quote catalog.
- Pricing Base pricing data which is synchronized from the primary PIM (Product Information Management)/ERP (Enterprise Resource Planning system to both Oracle Commerce and Oracle Configure, Price, Quote.
- Profiles Shopper/Account data which is synchronized from the primary CRM (Customer Relationship Management) system to both Oracle Commerce and Oracle Configure, Price, Quote.
- An extension server to support any required Serve-Side Extensions for the integration.

If you do not have one or more of these, please contact an Oracle sales representative for information on how to acquire one: http://www.oracle.com/us/corporate/contact/index.html.

Additional Resources

Addition information about Oracle Commerce can be found through the Oracle Help Center page for Oracle Commerce.

If you require further information regarding Oracle Commerce, you can access the latest product documentation and training videos through the Oracle Help Center page for Oracle Commerce.

If you require further information regarding Oracle Configure, Price, Quote, you can access the latest product documentation through the for Oracle Help Center page Oracle Configure, Price, Quote.

The documentation mentioned contains links to blogs, developer communities, and Support. (Please note that some of these resources require an account for access.)

Configure the Integration

Several stages are required to configure this integration.

Five stages are required to configure the integration between Oracle Configure, Price, Quote and Commerce. Each stage is covered in this chapter.

Configure the Integration Package

In order to use this integration, you must first download the integration package(s) and then import the package(s) into Oracle Integration Cloud.

This section provides detail about where the integration package(s) can be downloaded and how to import the integration package.

Importing the integration package in Oracle Integration Cloud (OIC) creates connections between Oracle Commerce and Oracle Configure, Price, Quote in OIC. It also creates an integration between Commerce and Oracle Configure, Price, Quote with some default mappings in place.

Download the integration package

Follow these steps to download the integration package:

1. Go to the Integrating Oracle Commerce and Oracle Configure, Price, Quote with Oracle Configure, Price, Quote article on My Oracle Support.
2. If you want to implement the integration between Commerce and the Oracle Configure, Price, Quote Configurator, download `OCCS-CPQ_CONFIGURATION_INTEGRATION_X.X.par` to a location where it is accessible from OIC.
   Note: `_X.X.par` refers to the most recent version of all downloadable files described.
3. If you want to implement the integration between Commerce and Oracle Configure, Price, Quote Quoting, download `OCCS-CPQ_QUOTE_INTEGRATION_X.X.par` to a location that is accessible from OIC.
4. If you want to enable Asset Based Ordering (ABO) through the integration between Commerce and Oracle Configure, Price, Quote, download the following packages to a location that is accessible from OIC:
   - `OCCS_CPQ_ASSET_INTEGRATION_X.X.par`
   - `OCCS_CPQ_GETCONFIGBOM_X.X.par`
   - `OCCS_CPQ_CONFIGURATION_INTEGRATION_X.X.par`
   - `OCC_CPQ_Get_Asset_Upgrade_Options_X.X.par`

Import the integration package(s)

Import the OIC Integration Package into OIC to create an integration between Commerce and Oracle Configure, Price, Quote through OIC.

To import the OIC Integration Package:

1. Log on to OIC as an admin user.
2. Click the **Packages** icon.
3. Click the **Import** button.
4. Click **Browse** to open a navigation pane.
5. Select the integration package archive (.PAR) file you want to import.

6. Click **Import**. The package is added to the **Packages** list.

The `OCCS-CPQ_CONFIGURATION_INTEGRATION` package includes the OCCS-CPQ Get Configurations integration flow. The GetConfigurations integration flow is used for the following Asset Based Ordering operations:

- Modify
- Upgrade
- Renew
- Resume

This integration is required for the configuration flow. The name of the target connection for this integration is “Oracle CPQ”. The target connection identifier is “Oracle_CPQ”, and the target connection description is “Oracle CPQ ICS Adapter Connection.”
The `OCCS-CPQ_QUOTE_INTEGRATION` package includes the following three integration flows: OCCS-CPQ Create Quote, OCCS-CPQ Update Quote, and OCCS-CPQ Sync Quote.

- The OCCS-CPQ Create Quote integration sends quote request information to Oracle Configure, Price, Quote.
- The OCCS-CPQ Update Quote integration sends information to Oracle Configure, Price, Quote related to accepting, rejecting, or re-requesting a quote.
- The OCCS-CPQ Sync Quote integration allows Oracle Configure, Price, Quote to send information to Commerce at the end of the quoting process and synchronize this information in Commerce. This ensures that the order information in Commerce matches the related order information in Oracle Configure, Price, Quote.

The `OCCS_CPQ_ASSET_INTEGRATION` package includes two integration flows: OCCS-CPQ Get Assets and OCCS-CPQ Asset Actions. This integration is required for Asset Based ordering. The name of the target connection for this integration is “Oracle CPQ”. The target connection identifier is “Oracle_CPQ”, and the target connection description is “Oracle CPQ ICS Adapter Connection.”

Note: The OCCS-CPQ Get Assets integration returns information about assets and services associated with the shopper’s account(s).

The `OCCS_CPQ_GETCONFIGBOM` package contains the following OIC integration flow which is also used in Asset Based ordering:

- `GetConfigBom` - This flow is invoked for the following Asset Based Ordering operation flows:
  - Suspend
  - Terminate

`GetConfigBom` calls are required to be made for each `configuratorID` of these filtered items to retrieve a saved Configuration BOM Instance of the item on Oracle Configure, Price, Quote.

The name of the target connection for this integration is “Oracle CPQ”. The target connection identifier is “Oracle_CPQ”, and the target connection description is “Oracle CPQ ICS Adapter Connection.”

Configure the Oracle Commerce Connection

For the integration to run successful, you need to configure the connection from the integrations imported to OIC to Commerce.

You must complete the following steps to configure the connection from the OIC integrations to Commerce.

1. Log on to OIC as an admin user.
2. Click the **Connections** icon.
3. Click the Oracle Commerce connection.
4. Click the **Configure Connectivity** button.
5. Enter the Connection base URL. The Connection base URL is derived using the following structure where <siteURL> is the base URL and port number of the Oracle Commerce site that integrates with OIC. For example:

   ```
   Connection base URL: https://<siteURL>/ccadmin/v1
   ```

6. Click the **Configure Security** button.
7. The Oracle Commerce connection uses the OAuth security policy, so you must enter a Security token for the connection. This token is generated in Oracle Commerce. Instructions on generating the token can be found in the next Generate a Security Token section of this document.
8. Click **OK**.
9. Click **Test** to test that the connection is working.
10. Click **Save**.

Your Oracle Commerce connection is now configured for the integration.

Generate a Security Token

This integration uses the Oracle Commerce REST web services APIs to access Oracle Commerce data. You must register the integration within Oracle Commerce and generate a security token in order for the integration to be granted access to the data.

Follow these instructions in order to generate a security token:

1. Log onto Oracle Commerce.
2. Click the **Menu** icon.
3. Select **Settings** from the menu.
4. Click **Web APIs** from the sidebar menu.
5. Click **Registered Applications** from the Web APIs panel.
6. Click the **Register Application** button.
7. Enter a name for the integration. The application you are registering is OIC, so you should choose a meaningful name that reflects this.
8. Click **Save**. The Application ID and Application Key are automatically generated and the application is added to the Registered Applications page.
9. Click on the name of the application you created.
10. Click on **Click to reveal** to display the application key. You can copy the application key to use as the security token for the Oracle Commerce connection.

For more information on managing an application within Oracle Commerce, please refer to Register applications.

Activate the Integrations

Once your integrations are configured, you must activate them using the OIC admin user interface.

Once the Oracle Configure, Price, Quote, Commerce, Oracle Configure, Price, Quote Quote, Oracle Configure, Price, Quote Configure, and Oracle Configure, Price, Quote getConfigurations connections are configured, you must activate these integrations.

Follow these instructions to activate the OIC integrations:

1. Log on to OIC as an admin user.
2. Click on the **Integrations** icon to display the Integrations list.
3. Click on the **Activate** button for the integration you wish to activate.
4. Decide whether you want to switch on detailed tracing, which collects information about messages processed by the integration flow. Administrators may find detailed tracing helpful when troubleshooting issues with the integration flow, but it may impact performance.

   To switch on detailed tracing, select the **Enable detailed tracing** check box.

   Note: Once an integration flow is active, administrators must deactivate it and activate it again to switch detailed tracing on or off.
5. Click **Activate**.

Configure the Commerce Webhooks

You must configure webhooks in Commerce Administration in order to support the REST API generated by the activation of the OIC integration.

The REST API generated by activating the OIC integration can be configured as a Webhook in Commerce Administration. These webhooks include the following:

- **Request Quote**: This webhook is triggered when a request or re-request for a quote is submitted by a Commerce self-service user. This webhook pushes notifications using the OCCS-CPQ Create Quote integration flow.
- **Update Quote**: This webhook is triggered when a response to a requested quote is accepted, rejected, or the quote is canceled by a Commerce self-service user. This webhook pushes notifications using the OCCS-CPQ Update Quote integration flow.
- **External Price Validation**: This webhook is triggered at checkout when the order contains one or more items configured by Oracle Configure, Price, Quote. This webhook should point to the SSE app URL configured later. The webhook validates the configuration and price provided for the configured items. It also includes the commerce item ID data in the request payload and updates the external price information of the commerce items. Finally, it invokes a re-pricing operation at order checkout.
- **Contact Accounts Retrieval**: This webhook has been deprecated. The corresponding SSE endpoints are invoked from the widget. It returns a list of service account IDs for the shopper. Formerly, this webhook called the Contact Accounts Retrieval webhook, so that webhook also had to be configured for the Services Retrieval webhook to function correctly.
- **Services Retrieval**: This webhook has been deprecated. The corresponding SSE endpoints are invoked from the corresponding widget. Formerly, this webhook returned

information about a service or asset associated with the shopper and used the OCCS-CPQ Get Assets integration flow. This webhook called the Contact Accounts Retrieval webhook, so that webhook also had to be configured for the Services Retrieval webhook to function correctly.

You must configure the Production and Preview version of these webhooks to ensure that they work in all environments. The Production webhooks send information from your live store to production environments of your live systems, while preview webhooks send information from your preview environment to the test or sandbox environments of your external systems.

Follow these instructions to configure the Request Quote, Update Quote, External Price Validation, Services Retrieval, and Services webhooks:

1. Log on to OIC as an admin user.
2. Click on the **Integrations** icon.
3. Click on the **Integration Details** icon to display information about the integration flow.
   - If you are configuring the Request Quote webhook, you should display information for the OCCS-CPQ Create Quote integration flow.
   - If you are configuring the Update Quote webhook, you should display information for the OCCS-CPQ Update Quote integration flow.
   - If you are configuring the External Price Validation webhook, you should display information for the OCCS-CPQ External Pricing integration flow. For this webhook, you to configure the SSE app endpoint.
   - If you are configuring the Services Retrieval webhook, you should display information for the OCCS-CPQ Get Assets integration flow. This OIC flows requires the Services SSE to be set up and invoked from there.
   - If you are configuring the Services webhook, you should display information for the OCCS-CPQ Asset Actions integration flow. This OIC flows requires the Services SSE to be set up and invoked from there.
4. Copy the Endpoint URL for the integration.
5. Log into Commerce.
6. Click the **Menu** icon.
7. Select **Settings** from the menu.
8. Select **Web APIs** from the sidebar menu.
9. Click the webhook you wish to configure.
10. Paste the Endpoint URL you copied into the URL field for the webhook.
11. Remove the “metadata” text from the end of the URL.
12. Enter the user name and Password for your OIC account.
13. Click the **Save** button.

The webhook is now configured and is triggered each time the relevant event occurs, which in turn triggers the relevant integration flow.

Note: It is not possible to edit webhooks differently for different sites. Updating webhooks applies changes regardless of the site selected.

For more information on Oracle Commerce webhooks, please refer to Configure webhooks.

Understand the Services SSE

Modify, renew, terminate, suspend, and resume actions performed on a service or asset are done using the Services server side extensions, one set for Storefront and one for Agent. Get Assets and Get Asset details are also performed using the endpoints in the Services SSE.

See the topic Use developer tools to customize your store for information.

Configure the Server Side Extensions

To perform specific functions relating to asset-based orders, you need to install and configure the related Commerce server-side extensions (SSEs).

Available Commerce server-side extensions (SSEs) can be installed and configured to perform specific functions relating to asset-based orders.

For more complete information on server-side extensions and how to develop them for use with Commerce, refer to Develop server-side extensions in *Extending Oracle Commerce* found in the Commerce Help Library.

The next sections in this topic explain the purpose and configuration of each available SSE as well as provide information on the inputs required for their respective endpoints. Finally the last section of this topic, **Understand the general procedure for installing and configuring the integration SSEs** , provides general instruction on downloading, installing, and configuring the available SSEs.

Note: Address information is something used extensively in Commerce transactions. For all procedures and SSEs that require address information for endpoint inputs, in addition to using Commerce's default address formats, you can also use the REST API to create multi-country custom address formats. This lets you create country-specific address formats to ensure that your address formats align with the requirements of any external service that you might use. This means that addresses appearing in profiles, accounts, registration requests, order addresses and more can be customized. For more complete information on creating custom addresses and understanding how to use custom address formatting, refer to the following:

- Customize Address Formats using the API in *Extending Oracle Commerce*
- Work with address types in *Extending Oracle Commerce*
- Account Details in *Using Oracle Commerce*
- Work with account addresses in *Using Oracle Commerce*
- Work with account registration requests in *Using Oracle Commerce*

Configure the Credit Check SSE

Since Commerce does not provide a pre-built integration with any particular credit checking system, the Credit Check SSE is used to connect to a third-party credit check system so that you can perform a credit check on the logged-in shopper.

Note: This SSE is optional and can be used if you want a credit check to be done as part of an order submit task.

You can configure the available SSEs, **CheckCredit-store.zip** and **CheckCredit-agent.zip**, by first downloading the SSE packages.

Note: As written, this SSE generates outbound calls to an external credit checking system. This means that the Credit Check SSE calls out to an external system to perform the credit check. In order to use this SSE to connect to the external checking of your choice, you must modify the SSE code to provide the specific calls needed to connect to the correct credit checking system.

To complete installing and configuring the SSE, refer to the **Understand the general procedure for installing and configuring the integration SSEs** section at the end of this topic.

The subsection(s) that follows describe the relevant endpoint(s) for this SSE.

Understand the Check Credit endpoint

The Check Credit endpoint is triggered whenever a credit check is requested by Commerce.

The inputs for this endpoint are:

- Amount information
- Recurring amount frequency
- Recurring amount duration
- Recurring amount
- Contact information
- First Name
- Last Name
- Email Address
- Telephone Number
- Address information
- Address line 1
- Address line 2
- City
- State
- Country
- Postal code

The return for this endpoint is either a TRUE or FALSE value depending on whether the shopper passed the credit check or not.

Configure the Customer Account Model SSE

This SSE is used to return information about the customer account model for a registered shopper or to update the customer account model when required. In detail, this SSE is meant to get account details from CDM masters like OEC Communications and is required in Telco kind of installations

You can configure the available SSEs, **CustomerAccountModel-store.zip** and **CustomerAccountModel-agent.zip**, by first downloading the SSE package.

To complete installing and configuring the SSE, refer to the **Understand the general procedure for installing and configuring the integration SSEs** section at the end of this topic.

The subsection(s) that follows describe the relevant endpoint(s) for this SSE.

Understand the Create Accounts endpoint

This endpoint is triggered if the Query Accounts endpoint does not return any accounts for the shopper.

The inputs for this endpoint are:

- User Token for the logged-in shopper.
- Account Type
- Account Name
- Primary Contact
- Billing Profile(s)
- Address(es)
- Contact ID(s)
- Contact Role(s)

The returns for this endpoint are the accounts, roles, addresses, and business profiles now associated with the shopper.

Understand the Create Contact endpoint

This endpoint is triggered when a shopper logs in to Commerce.

The input for this endpoint is the User Token for the logged-in shopper.

The return for this endpoint is the new External Contact ID created for the shopper.

Understand the Query Accounts endpoint

This endpoint is triggered when a shopper logs in to Commerce and when they go to Checkout for an order that contains service items.

The input for this endpoint is the User Token for the logged-in shopper.

The returns for this endpoint are the accounts, roles, addresses, and business profiles associated with the shopper.

Understand the Query Contacts endpoint

This endpoint is triggered when a shopper logs in to Commerce.

The input for this endpoint is the User Token for the logged-in shopper.

The return for this endpoint is the External Contact ID for the shopper.

Understand the Update Accounts endpoint

This endpoint is triggered when a shopper saves an account address.

The inputs for this endpoint are:

- User Token for the logged-in shopper.
- The Account ID of the account to which the billing profile is linked.

- The new address as provided by the shopper.

The returns for this endpoint are the accounts, roles, addresses, and business profiles associated with the shopper.

Configure the Order Qualification SSE

This SSE is used to perform any final checks on an order before payment is authorized and the order is submitted to downstream systems for processing and fulfillment.

It also validates that for any item in the order which is based on a SKU where the configurable property is TRUE and the `assetable` property is TRUE the quantity must be 1 and, if not, return an error indicating that this item can only be purchased one at a time. This check is done by looking to see if the root item has an `assetKey` value. For more information, see the Use Asset Based Ordering section of this guide.

You can configure the available SSEs, **OrderQualification-store.zip** and **OrderQualification-agent.zip**, by first downloading the SSE package.

To complete installing and configuring the SSE, refer to the **Understand the general procedure for installing and configuring integration SSEs** section at the end of this topic.

The subsection(s) that follows describe the relevant endpoint(s) for this SSE.

Understand the Order Qualification endpoint

This endpoint is triggered by the Order Qualification webhook when any order containing a configured item is submitted.

The input for this endpoint is the order containing the configured item.

The return for this endpoint is either a TRUE or FALSE value depending on whether the order passed the validation check or not. If the value is FALSE the return also includes information about which item(s) in the order failed validation.

Configure the Order Qualification Pipeline SSE

This SSE is used to ensure that an order is valid. It enables an order qualification step in the purchasing process that can be invoked via the Order Qualification webhook. The extension can be configured to execute custom order qualification processes such as checking whether the shopper is eligible to purchase the items in the cart. It contains a pre-built algorithm to validate that the Customer, Billing, and Service accounts as well as the Billing Profile assigned to the items in the cart are valid for the logged in shopper. It also contains a module to check if the cancel in-flight is allowed for a given order.

You can configure the available SSEs, **OrderQualificationPipeline-store.zip** and **OrderQualificationPipeline-agent.zip**, by first downloading the SSE package.

To complete installing and configuring the SSE, refer to the **Understand the general procedure for installing and configuring the integration SSEs** section at the end of this topic.

The subsection(s) that follows describe the relevant endpoint(s) for this SSE.

Understand the Order Qualification Pipeline endpoint

This endpoint is triggered when a shopper goes to checkout for an order that contains configured items.

The inputs for this endpoint are:

- Contact record for the shopper
- Order containing configured items.

The return for this endpoint is either a TRUE or FALSE value depending on whether the order passed the validation check or not. If the value is FALSE the return also includes information about which item(s) in the order failed validation.

Configure the Order Validation Pipeline SSE

This SSE enables an order qualification step in the purchasing process that can be invoked via the Order Validation webhook. The extension can be configured to execute any final checks particular to the purchasing model before the order payment is authorized and the order is submitted to the downstream systems for fulfillment and provisioning.

You can configure the available SSEs, **OrderValidationPipeline-store.zip** and **OrderValidationPipeline-agent.zip**, by first downloading the SSE package.

To complete installing and configuring the SSE, refer to the **Understand the general procedure for installing and configuring the integration SSEs** section at the end of this topic.

Configure the Services SSE

The Services SSE is used to perform modify, renew, terminate, suspend, and resume actions on a service or asset - one SSE for Storefront and one for Agent. The SSE also contains a module to check if the cancel in-flight feature is allowed for a given order and is also used to retrieve the assets and asset details

You can configure the available SSEs, **Services-store.zip** and **Services-agent.zip**, by first downloading the SSE package.

To complete installing and configuring the SSE, refer to the **Understand the general procedure for installing and configuring the integration SSEs** section at the end of this topic.

The subsection(s) that follows describe the relevant endpoint(s) for this SSE.

Understand the Services SSE endpoints

The Server Side Extension Endpoints for the Services SSE are the following:

- Modify
- Renew
- Terminate
- Suspend
- Resume

These endpoints are triggered when a user performs an operation on an asset.

The inputs for these endpoints are:

- Logged in User Token

- `AssetKey`, the unique ID for the asset for this operation. This may be a root, branch or leaf asset.

The returns for this endpoint are BOM (Bill of Materials) and Error.

Configure the Configuration Validation SSE

The Configuration Validation SSE plays an important role in Asset Based Ordering and validating asset configuration. This specific SSE performs a configuration validation between items in a shopper's cart and the items captured in response to configuration validation end points. For more complete information on Asset Based Ordering, refer to the Using the Integration Functionality section of this document.

To use this SSE, you should first have the External Pricing webhook set to `/ccstorex/custom/v1/validateCPQConfigurations`. This is done on the Settings page of the Administration user interface.

You should also have the following endpoints configured:

- `GET_CONFIGBOM_URI`
- `GET_CONFIG_URI`

The `GET_CONFIGBOM_URI` URL gets triggered for the Suspend and Terminate Services. The `GET_CONFIG_URI` URL gets triggered for the Renew, Modify, and Resume Services. The SSE does validation between items in cart and items captured in the response of these two end points

You can configure the available SSEs, **Services-store.zip** and **Services-agent.zip**, by first downloading the SSE package.

To complete installing and configuring the SSE, refer to the **Understand the general procedure for installing and configuring the integration SSEs** section at the end of this topic.

Understand the general procedure for installing and configuring the integration SSEs

To use this integration, you need to install and configure the integration server-side extensions (SSEs). The SSE code logic allows communication between Commerce and Oracle Configure, Price, Quote - via Oracle Integration Cloud as part of the data flow. The Commerce and Oracle Configure, Price, Quote integration functionality/ communication is provided through the configuration of these server-side extensions.

In addition to providing REST APIs and webhooks for integrating with external systems (as well as widgets for extending your storefront), Commerce also includes support for developing server-side extensions written in JavaScript. For more information, refer to Working with Commerce Server-Side Extensions

The general installation and configuration procedure for the integration SSEs uses the following steps:

- Before you configure and install the integration server-side extensions, first make sure your custom **Node.js** server is associated with your Commerce environment.
- Download the integration server-side extension (SSE) files locally, so that you can install and configure them. Select and remember the desired location where you want the SSE .ZIP file(s) to be downloaded. See Integrating Oracle CX Commerce and Oracle CPQ (Doc ID 2214316.1) on the My Oracle Support site for more

information on the required integration SSE .ZIP files and for the links that let you download these files.

- After downloading the required files, you need to install them. Use the `POST / ccadmin/v1/serverExtensions` endpoint to do this. Specify the Content-Type as `multipart/form-data` and include a reference to the file in the body of the request. For example, your request header might look like the following:

```
POST /ccadmin/v1/serverExtensions  HTTP/1.1
Content-Type: multipart/form-data
Authorization: Bearer <access_token>
```

- The request body should consist of the `<YOUR_SSE_NAME>.zip` file, uploaded as `multipart/form data`. The response to the request should look similar to this:

```
{
    "result": {
        "unzipped": false,
        "failedImages": 0,
        "allImagesFailed": false,
        "failedImagesReasons": {},
        "modifiedImages": 0,
        "newImages": 1,
        "assignedImages": 0
    },
    "success": true,
    "links": [
        {
            "rel": "self",
            "href": "http://myserver.example.com:7002/ccadmin/v1/
serverExtensions"
        }
    ],
    "token": "d63c663af7f15_cd3d"
}
```

- Make changes to each server-side extension's `config.json` file by providing the correct URLs to complete the SSE configuration portion of that integration. The typical steps used for working with the SSE code and making changes to the `config.json` file include the following:
  - Obtain and download the correct SSE .ZIP file.
  - Extract the SSE .ZIP file.
  - Edit and save the `config.json` file.
  - Zip the files using the original .ZIP file name as the original.

  The following example shows some configuration information (in **bold**) that must be added to the `config.json` file for both the store and agent models of the SSE:

```
"hostname": "yourhostname.example.com",
"port": "7003",
"timeout": 50000,
"username_env_var": "YOUR_USERNAME",
"password_env_var": "YOUR_PASSWORD",
"QUERY_CONTACTS": "/ic/api/integration/v1/flows/rest/
OCC_OEC_GET_PROFILE_SSE/1.0/contacts",
"CREATE_CONTACT": "/ic/api/integration/v1/flows/rest/
OCC_OEC_CONTACT_CREATE_SSE/1.0/contacts",
"QUERY_ACCOUNTS": "/ic/api/integration/v1/flows/rest/
OCC_OEC_GET_ACCNT_DETLS_PROF_SSE/1.0/accounts",
"CREATE_ACCOUNTS": "/ic/api/integration/v1/flows/rest/
OCC_OEC_ACOUNT_CREATE_SSE/1.0/contacts/{currentContactId}/accounts",
"UPDATE_ACCOUNT": "/ic/api/integration/v1/flows/rest/
OCC_OEC_ACCOUNT_UPDATE_SSE/1.0/contacts/{currentContactId}/accounts/
{accountId}"
```

All of the example endpoint URLs (paths) specified in the example, starting from the "`QUERY_CONTACTS`" to the “`UPDATE_ACCOUNT`" keys, are coming from Oracle Integration Cloud and are necessary for a successful integration activation between Commerce and Oracle Configure, Price, Quote. The paths that you would use when editing your `config.json` files would be the ones specific to your SSE endpoints. The ones shown here are for example purposes only. Refer to each specific SSE section in this topic to obtain the correct SSE and endpoint information.

- Upload the modified SSE .ZIP file. To upload the file, click **Settings** then **Extensions**. On the Extensions page, click **Installed** and then **Upload Extension**. Select the location and name of the ZIP file.

Understand the environment variables supported by the integration SSEs

When communicating with Commerce via its REST APIs, you need to authenticate your requests using confidential information. The need to authenticate is not just limited to Commerce as many 3rd party services require the same. It is recommended that you do not store confidential information in extension files but that you use environment variables to maintain value confidentiality. In the previous example `config.json` file, environment variables are used to make username and password information confidential. Commerce SSEs include the `nconf` package which provides a hierarchical `node.js` configuration with files, environment variables, command-line arguments, and atomic object merging. Use the hierarchy provided by `nconf` to manage your configuration values and maintain different values for different environments used in your integration. You can also use environment variables to pass through API information you want protected. Refer to "REST API authentication" in the Commerce REST API documentation for more info on how to authenticate Commerce API calls.

The specific environment variables supported by the integration SSEs are the following:

Table 2-1 Integration SSE environment variables

SSE name	Supported variable name	Description
CustomerAccountModel-Store	CRM_USERNAME	Specifies the basic authentication username for the accounts integration. In this case, for Oracle Integration Cloud (OIC) which integrates OEC Comms.

Table 2-1 (Cont.) Integration SSE environment variables

SSE name	Supported variable name	Description
CustomerAccountModel-Store	CRM_PASSWORD	Specifies the basic authentication password for accounts integration. In this case, for OIC which integrates OEC Comms.
CustomerAccountModel-Agent	CRM_USERNAME	Specifies the basic authentication username. In this case, for Oracle Integration Cloud (OIC) which integrates OEC Comms.
CustomerAccountModel-Agent	CRM_PASSWORD	Specifies the basic authentication password for accounts integration. In this case, for OIC which integrates OEC Comms.
Services-Store	OIC_USERNAME	Specifies the basic authentication username for the accounts integration and Oracle Configure, Price, Quote integration that proxies via OIC.
Services-Store	OIC_PASSWORD	Specifies the basic authentication password for the accounts integration and Oracle Configure, Price, Quote integration that proxies via OIC.
Services-Store	CPQ_USERNAME	Specifies the basic authentication username for requests that go directly to Oracle Configure, Price, Quote.
Services-Store	CPQ_PASSWORD	Specifies the basic authentication password for requests that go directly to Oracle Configure, Price, Quote.
Services - Agent	OIC_USERNAME	Specifies the basic authentication username for the accounts integration and Oracle Configure, Price, Quote integration that proxies via OIC.
Services-Agent	OIC_PASSWORD	Specifies the basic authentication password for the accounts integration and Oracle Configure, Price, Quote integration that proxies via OIC.

Table 2-1 (Cont.) Integration SSE environment variables

SSE name	Supported variable name	Description
Services-Agent	CPQ_USERNAME	Specifies the basic authentication username for requests that go directly to Oracle Configure, Price, Quote.
Services-Agent	CPQ_PASSWORD	Specifies the basic authentication password for requests that go directly to Oracle Configure, Price, Quote.
Order Qualification Pipeline	ORDER_QUALIFICATION_PIPELINE_USERNAME	Specifies the basic authentication username for securing the `/v1/orderQualification` route.
Order Qualification Pipeline	ORDER_QUALIFICATION_PIPELINE_PASSWORD	Specifies the basic authentication password for securing the `/v1/orderQualification` route.
Order Qualification Pipeline	VALIDATION_USERNAME	Specifies the basic authentication username for accessing the `/v1/crm/accounts` route in the Customer Account Model to retrieve accounts data.
Order Qualification Pipeline	VALIDATION_PASSWORD	Specifies the basic authentication password for accessing the `/v1/crm/accounts` route in the Customer Account Model to retrieve accounts data.
Order Qualification Pipeline	OIC_USER_NAME	Specifies the basic authentication username for accessing the services integration that integrates with Oracle Configure, Price, Quote to retrieve asset/services data.
Order Qualification Pipeline	OIC_PASSWORD	Specifies the basic authentication password for accessing the services integration that integrates with Oracle Configure, Price, Quote to retrieve asset/services data.
Order Validation Pipeline	ORDER_VALIDATION_PIPELINE_USERNAME	Specifies the basic authentication username for securing the /`v1/orderValidation` route.
Order Validation Pipeline	ORDER_VALIDATION_PIPELINE_PASSWORD	Specifies the basic authentication password for the `/v1/orderValidation` route

Table 2-1 (Cont.) Integration SSE environment variables

SSE name	Supported variable name	Description
cpq-configurator-app-store	CPQ_USERNAME	Specifies the basic authentication username for requests that go directly to Oracle Configure, Price, Quote.
cpq-configurator-app-store	CPQ_PASSWORD	Specifies the basic authentication password for requests that go directly to Oracle Configure, Price, Quote.
cpq-configurator-app-agent	CPQ_USERNAME	Specifies the basic authentication username for requests that go directly to Oracle Configure, Price, Quote.
cpq-configurator-app-agent	CPQ_PASSWORD	Specifies the basic authentication password for requests that go directly to Oracle Configure, Price, Quote.

Note: Commerce provides the `admin` endpoint that can be used to set an environment variable on the Commerce server. For additional information on each SSE's supported environment variables, a `README.TXT` file is provided along with the `config.json` file that has additional usage information.

Enable the Integrations

You need to enable the Oracle Configure, Price, Quote Configurator integration, the Oracle Configure, Price, Quote Request For Quote integration, and the Asset Based Ordering (ABO) integration in Commerce for the complete integration to work successfully.

You must complete the procedures in this section to enable the Oracle Configure, Price, Quote Configurator integration, the Oracle Configure, Price, Quote Request For Quote integration, and the Asset Based Ordering (ABO) integration in Commerce.

For additional information, refer to Appendix A: Configurator Flow and Appendix B: Request for Quote Flow.

Enable Oracle Configure, Price, Quote Configuration Integration

Follow these steps to enable the Oracle Configure, Price, Quote Configuration Integration within Oracle Commerce:

1. Log on to Commerce.
2. Click on the **Menu** icon.
3. Select **Settings** from the menu.
4. Select **Oracle Integrations** from the sidebar menu.
5. Select **CPQ Configuration** from the dropdown menu.

6. Check the **Enable Integration** checkbox.
7. Enter the Configuration URL using the following structure:

   ```
   https://<cpq_domain>/commerce/new_equipment/products/
   model_configs.jsp
   ```

8. Enter the Reconfiguration URL using the following structure. You must enter these values for your production and preview environments.

   ```
   https://<cpq_domain>/commerce/new_equipment/products/
   external_reconfig.jsp
   ```

9. Enter the Modification URL using the following structure. You must enter these values for your production and preview environments.

   ```
   https://<cpq_domain>/commerce/new_equipment/products/
   model_configs.jsp
   ```

10. Click the **Save** button.

If you are using multiple sites you must follow these instructions for each site that you operate that uses the Oracle Configure, Price, Quote Configuration integration.

Identify Configurable Products in the Product Catalog

Before a Commerce self-service user can use the Oracle Configure, Price, Quote configurator to configure complex products for purchase in Commerce, you must identify the products as configurable in the product catalog. Before doing so, it is important to have a synchronized product catalog to ensure that products in the Commerce catalog map to corresponding items in the Oracle Configure, Price, Quote catalog.

To identify a product as configurable:

1. Log in to Commerce.
2. Click on the **Menu** icon.
3. Select Catalog from the menu.
4. Select the product you wish to identify as configurable.
5. Click on the **SKUs** tab of the product detail pop-up frame.
6. Select the SKU you wish to identify as configurable.
7. Check the **Configurable** checkbox. This displays three further fields you must complete.
8. Enter the Model information. This should match the Model information of a configurable product in the Oracle Configure, Price, Quote catalog.
9. Enter the Product Line information. This should match the Product Line information of a configurable product in the Oracle Configure, Price, Quote catalog.
10. Enter the Product Family information. This should match the Product Family information of a configurable product in the Oracle Configure, Price, Quote catalog.

11. Click **Save**. This returns you to the SKU frame where the SKU you updated should be marked with an asterisk to identify it as a configurable SKU.

Note: Administrators can also perform the above setup steps in bulk by using the SKU import program. From the Catalog page in Commerce, click **Manage Catalog** and select **Import**. In the **Import dialog**, click **Browse** and locate the CSV file to import. Click **Upload File**, click **Validate**, and then click **Import**.

Add Customize button to the Product Details Widget

Administrators must add a Customize button to the Product Details widget, so the button is visible to Commerce self-service users from the Product Details page for a customizable product.

To add a Customize button to the Product Details widget:

1. Log in to Commerce.
2. Click on the **Menu** icon.
3. Select **Design** from the menu.
4. Select **Product Layout** from the layout list.
5. Delete the Product Details widget from the layout.
6. Place a new product details widget on the layout.
7. Click the **Settings** icon for the new Product Details widget.
8. From the Element Library, place a Customize button on the new Product Details widget.
9. Publish the changes.

Enable Oracle Configure, Price, Quote Quoting Integration

Follow these steps to enable the Oracle Configure, Price, Quote Quoting Integration within Oracle Commerce:

1. Log on to Commerce.
2. Click the **Menu** icon.
3. Select **Settings** from the menu.
4. Select **Oracle Integrations** from the sidebar menu.
5. Select **CPQ Quoting** from the dropdown box.
6. Check the **Enable Integration** checkbox.

Add Quote Buttons to the Checkout and Order Details Pages

To make the Oracle Configure, Price, Quote quoting capability available to Commerce self-service users, you must add the Request Quote widget to the Checkout layout and the Quote Details widget to the Order Details layout.

The Request Quote widget adds a Quote Notes text box and a Request Quote button to the Checkout layout.

The Quote Details widget adds a Quote Notes text box populated with any notes associated with the order to the Order Detail layout. The widget also adds a Reject Quote, Request Re-Quote, and Accept Quote buttons to the to the Order Detail layout.

The Quote Details and Request Quote widgets do not display on the layouts by default. You must first make the widgets available and then place them on the Checkout and Order Detail pages.

To add quote buttons to the Checkout and Order Details pages:

1. Log in to Commerce.
2. Click the **Menu** icon.
3. Select **Design** from the menu.
4. Select the **Components** tab on the Design page.
5. Click the **Show Hidden** button.
6. Click the **Show** icon for the Quote Details Widget and the Request Quote Widget.
7. Within the Design page, select the **Layouts** tab.
8. From the layout list, select **Checkout Layout**.
9. Drag and drop the **Request Quote** widget from the **Components** menu to the desired location on the **Checkout** layout.
10. From the layout list, select **Order Details**.
11. Drag and drop the **Quote Details** widget from the **Components** menu to the desired location on the **Order Details** layout.
12. Publish the changes.

Enable Asset Based Ordering

To enable Asset Based Ordering, you must make sure that you have set up the right integration webhooks and/or SSEs mentioned in the Extending Oracle Commerce.

Use the Integration Functionality

Oracle Configure, Price, Quote provides greater pricing and price quoting features for Oracle Commerce when the two are used together in an integration.

This chapter provides the instructions on how to use this functionality in Oracle Commerce that is supported by the integration with Oracle Configure, Price, Quote.

Configure an item

Items marked as configurable in your catalog can be configured either by an agent via the Commerce Agent Console or by a shopper via the Commerce Storefront.

Items that have been marked as configurable in your catalog may be configured either by an agent via the Commerce Agent Console, or by a shopper via the Commerce Storefront. This section provides instructions for both methods of configuring an item.

Configure an Item by an Agent

These instructions detail how an agent can configure an item via the Agent Console.

1. Log onto Commerce.
2. Using Agent Console, search for the shopper for whom you wish to create a new order.

3. Click **New** to create a new order.
4. Select a configurable product from the catalog.
5. Click on the **Configure** button to open the Oracle Configure, Price, Quote iFrame. **Note**: The Oracle Configure, Price, Quote iFrame is optimized for desktop, laptop, or tablet-size devices and is not recommended for mobile devices. If you need to display on mobile devices, please contact your Oracle Configure, Price, Quote Implementation team and inquire about the CPQ Mobile Layout.
6. Select the configuration options required for the order.
7. Click **Add** to Cart.

Once the configured item has been added to the cart, the agent can complete the order by going through the normal checkout process.

There is a validation check before the order is processed to ensure that the configuration options selected are valid. If they are valid, the order process completes and the order is placed. If they are not valid, an error message is displayed to the agent telling them that the configuration is invalid and that the order cannot be placed.

Configure an Item by a Shopper

These instructions detail how a shopper can configure an item via Commerce Storefront.

1. Shopper selects a configurable item from the product catalog.
2. Shopper clicks on the **Customize** button which opens the CPQ iFrame.
3. Shopper selects their desired configuration options for the item.
4. Shopper adds customized item to their cart.
5. Shopper goes to checkout and provides shipping and payment details.

There is a validation before the order is processed to ensure that the configuration options selected are valid. If they are valid, the order process completes and the order is placed. If they are not valid, an error message is displayed to the shopper telling them that the configuration is invalid and that the order cannot be place. The shopper is then unable to place the order until the configuration options have been changed and the configured item passes the validation check.

Request a Quote

With Oracle Configure, Price, Quote enabled in the integration price quotes may be requested for one or more items.

Quotes may be requested for one or more items on an order either by an agent from within the Agent Console, or by a shopper from the checkout page for their order. If you are also using Oracle Configure, Price, Quote Configuration functionality, the order may contain a combination of configured and non-configured items.

Request a Quote by an Agent

An agent can request a quote on one or more items in an order from the Commerce Agent Console. The agent must follow these instructions to request a quote:

1. Log onto the Commerce Agent Console.
2. Search for the shopper for whom you wish to generate a new quote.
3. Click **New** to create a new order, or select an existing unfulfilled order for the shopper.

4. Once you have an order with items in the cart, click on the **Request Quote** link in the order edit page. You can switch between the **Request Quote** page and the **Create Order** page by clicking on the appropriate link.
5. Add text to the **Quote Notes** text box as desired.
6. Click on the **Request Quote** button.
   Once the agent has submitted the quote request, the Request Quote webhook is triggered and all relevant information is passed to Oracle Configure, Price, Quote for a decision on the quote. The order status is changed to "Pending quote". When an order is in Pending status, the agent cannot perform any operations on the order.

   A confirmation email is sent to the shopper informing them of the status of their order.
7. Once a response is received, the order status changes to "This order is a quote", and the agent then has a number of options about how to proceed. The agent can: The agent can:
   - **Accept the quote**: If the shopper is satisfied with the quoted price returned from Oracle Configure, Price, Quote, the agent can accept the quote on their behalf by clicking on the Accept button and proceeding with the order as normal.
     Once payment information has been entered and the order placed the order status changes to "Submitted for fulfillment". At this point the Update Quote webhook is triggered and Oracle Configure, Price, Quote is informed that the quote has been accepted.

     At this stage the agent can click on the Edit Order button, but the only edits allowed to the quote are changes to the shipping group, or the application of shipping discounts or promotions. The agent may not add or remove items from the cart, or change the quantities of items included in the order. The order status changes to "Order being amended" until the agent clicks on the Complete Order button.
   - **Request a requote**: If desired, the agent can enter more details in the Request Quote textbox and click on the Request Requote button to request an updated quote. When the agent requests a requote the order status changes to "Pending quote". When an order is in Pending status, the agent cannot perform any operations on the order.
   - **Reject the quote**: The agent can click on the Reject Quote button to reject the quote. This cancels the shopper's order and the order status changes to "this quote has been rejected".
     Note: The response to a quote request includes provision for an expiry date for the quote. If the quote has expired the Accept Quote and Reject Quote buttons are disabled, but an agent can request a requote for the order.

     Once the agent responds to the quote a confirmation email is sent to the shopper informing them of the status of their quote.

Order statuses relating to quotes are included in the dropdown list of order statuses in the Order Details section of the Order Search page.

Request a Quote by a Shopper

A shopper can request a quote on one or more items in an order from the checkout page. The shopper must follow these instructions to request a quote.

1. Add the desired items to the shopping cart.

2. Proceed to the checkout page.
3. On the checkout page, enter supporting details in the **Request a Quote** text box.
4. Click the **Request Quote** button.

Once the shopper has submitted their quote request, the Request Quote webhook is triggered and all relevant information is passed to Oracle Configure, Price, Quote for a decision on the quote.

When a decision is made about the quote, the order is updated and the shopper then has three options about how to proceed.

They can:

- **Accept the quote**: This means the shopper is satisfied with the quote and they may continue through the purchase process with the prices provided. The checkout page is displayed and the shopper may enter their shipping details and proceed with payment.
- **Reject the quote**: This means that the shopper has rejected the quote provided by CPQ Cloud, and the order is canceled.
- **Request a requote**: The shopper can use the **Request Requote** text box to provide further information and request an updated quote.

Use account-specific pricing for configured items

Account-specific prices configured on Oracle Configure, Price, Quote can be displayed in Commerce.

Account-based shoppers can obtain account-specific prices configured on Oracle Configure, Price, Quote and display these prices in Commerce. This topic explains the concepts behind this feature.

Formerly, the Oracle Configure, Price, Quote iFrame would open in an item configuration as part of anonymous session. All details in the Oracle Configure, Price, Quote page, then, were independent of the logged in shopper/contact/account. Account-based shoppers can now obtain account-specific prices configured on Oracle Configure, Price, Quote and display these prices in Commerce. This is possible because the iFrame displayed on Commerce obtains the context of the related account as well the contacts associated with it so that the correct account-based pricing information is returned to Commerce.

For example, consider that an account-based shopper logs in and selects to purchase a configurable computer package. The prices that Oracle Configure, Price, Quote returns to the shopper are specific and unique to that shopper's account. The pricing that is specific for one shopper is not visible to another shopper. The shopper then changes the configuration of the computer package as needed, enters the quantities needed, and finally submits the order. In the case of an Agent configuring the package, the agent also sees the account-specific details when configuring a price.

With the Commerce/Oracle Configure, Price, Quote configuration integration enabled, Commerce sends different criteria to determine and obtain the account-based price of the configuration maintained on Oracle Configure, Price, Quote.

Understand a user case as well as the workflow used to obtain correct account-based pricing

An example of a user case where a shopper (or agent) would want to obtain account-based pricing could go something like this:

- The shopper selects a commerce site and browses through the items in the catalog.
- The shopper selects a configurable item and clicks on the Customize button, which opens an iFrame allowing them to customize/reconfigure the item.
- The shopper configures the item and expects to see the prices for the items which only that customer would be allowed to see for that account.
- The shopper places the order with the customer-specific pricing. The shopper, after submitting the order and within the designated remorse period, is able to update the configurations of the items as well as receive the customer specific pricing for those items.
- The shopper can cancel this order containing customer specific priced items (within the remorse period).
- The shopper can carry out returns and refunds.
- The shopper can exchange within the same configured item(s).

Some variations to this use case could include:

- The shopper gets the account specific pricing but when shopper account details change, the adjusted price specific to that account would appear.
- An agent placing an order for an account based shopper gets the account specific pricing specific to that shopper.

The workflow used to obtain the correct account-specific pricing is the following:

- The store sends the `Account ID(Customer ID, Customer Name)` through the Configure Product iFrame. The shopper's `accountId` has been encrypted using the SSE and the encrypted details are what is sent to the Oracle Configure, Price, Quote iFrame. Other properties like model, product line, locale, currency are not encrypted.
- The calls made to Oracle Configure, Price, Quote at this point internally call the Oracle Configure, Price, Quote Price API.
- The iFrame shows the account specific pricing for the account based on the `accountId`.
- The Price API looks for any customer pricing rules defined in Oracle Configure, Price, Quote and returns the correct account-specific pricing for that customer based on the accountId. If there are no prices configured specifically for the customer, then they are presented with the default prices.
- A sample widget can be customized by implementers to encrypt and pass additional properties along with the `accountId`. The re-configuration flow works as it already exists.

The main purpose of this workflow is to pass the customer account/organization details to the Oracle Configure, Price, Quote system and calculate the customer-specific price (if any pricing rules are defined).

The existing integration components should retain their existing functionality (i.e. the customer/system implementer should be able to switch as to whether they are using anonymous or customer specific pricing).

With this workflow, it is assumed that there is data synchronization of Customers (Commerce account-based customer accounts) across Oracle Configure, Price, Quote and Commerce. Oracle Configure, Price, Quote is the mechanism that has the ability to set up rule-based pricing which can be customer specific. The customer specific pricing rule(s) should be the source for the account-based pricing of the item. Finally, there is a check done that is part of the integration which makes sure that the logged-in user is validated.

Note: A customer can use Oracle CDM (Oracle Customer Data Management) to maintain that the accounts (organizations) are synchronized between Commerce and Oracle Configure, Price, Quote or they can just use the Commerce accounts. The `accountId` that is passed in the integration flow varies based on the implementation model.

Understand how Commerce and Oracle Configure, Price, Quote support account-specific pricing

To be able to obtain account-specific prices configured on Oracle Configure, Price, Quote and display it on Commerce via the returned iFrame, you need the iFrame to be extended to handle various attributes as part of getting the price from Oracle Configure, Price, Quote. By extending these attributes, you can then display the account specific pricing given by the Oracle Configure, Price, Quote system.

The `cpq-config-validation-app` SSE now validates the additional `accountID` from the `getConfiguration` call made to Oracle Configure, Price, Quote to find the profile associated with the order before calling the Submit Order endpoint. An appropriate error message is returned if the `accountId` does not match the values of the `orgId` of the profile in `order.json`. By passing these parameters from Commerce to Oracle Configure, Price, Quote during the Configuration Page launch, the Pricing logic in Oracle Configure, Price, Quote can be triggered within the Configuration user interface. Commerce provides the initial ability to pass the Account ID, but an implementer can extend this to pass any other parameters from Commerce that Oracle Configure, Price, Quote can understand.

The integration takes an Access Token Based security approach to ensure that prices meant for users of one account are not visible to users of a different account. The key features of using this approach are the following:

- The authentication into Oracle Configure, Price, Quote continues to be an anonymous/ guest user method as it is today.
- There is no need for user mapping between the Commerce user and Oracle Configure, Price, Quote user as well as no need for additional user maintenance between Commerce and Oracle Configure, Price, Quote.
- The approach follows an established approach based on Assets Modify Punch-In.

SSE flow for Store and Agent

The following describes the SSE flow for Store and Agent

- When an account-based customer clicks on Customize for a product, the SSE endpoint gets triggered.
- The `accountId` of the account-based user and other configurable details like model, product_line, product_family, etc. get passed to the SSE.
- The validation of the `accountId` takes place first whether the logged in customer is associated with the `accountId` being passed or not.
- If the validation is successful, `accessTokenData` is generated containing the `accountId` and the expiration time which is then encrypted and signed with the private key to form the accessToken.

- A query string is formed using `accessToken` and another configurable list of parameters. This is then appended to the base URL and the Oracle Configure, Price, Quote iFrame that is launched with the account-based prices.
- The `accountId` is decrypted by Oracle Configure, Price, Quote using the Public key. The true `accountId` is then determined and prices are shown as per the pricing rule setup for this `accountId`.

The following illustrates a sample request:

```
{
  "accountId" : "or-100001",
  "configurableSkuDetails" : {
    "currency" : "USD",
    "locale" : "en",
    "model" : "sku50001",
    "product_line" : "laptopConfiguration",
    "product_family" : "Laptop"
  }
}
```

The following illustrates a sample response to that request:

```
{ "queryString":
"_bm_session_currency=USD&_bm_session_locale=en&model=sku50001&product_lin
e=laptopConfiguration&product_family=Laptop&segment=laptop&_from_partner=t
rue&accessTokenData=%7B%22expiryTime%22%3A%222019-11-06T15%3A40%3A49%2B05%
3A30%22%2C+
%22configAttrPunchinValues%22%3A%7B%22accountId%22%3A%22or-100001%22%7D%7D
&publicKeyVarName=shagul_rsa_public&accessToken=xboKIL0YMl1R1IERTBKzzfFbyV
AWq5bZgkWX%2Bf71YOJYlBu1GZ5aZay%2B5FS338joCIs8C7B9RrJlRXXkmd1U4zgqfPD2NJnf
bYzxCelhFpbwdau6n88qVH6WI%2BPCLzUJKrwJdNxuTd9O78ZL4pKW8g9mFhpnZcNec%2FRxpH
MrV%2BYm4S2iS5IZt7apTkt%2Bd%2BDDvm3Y0cmyEyfwcbhTjxKho904dJId0pf%2BU3VKcNIh
MRMtoeFFCskhQNiqA8gyjUqamyB1y%2BgZQ9WKqo84rYsPnjCHvOF5z%2BAjMF5FysbGQxLJAF
PAczACuLhn1XrmDjjYMD6T26ey2d%2BQbKlzGgMIsg%3D%3D" }
```

SSE flow for validating the account ID

The `cpq-config-validation-lib` SSE has the functionality to validate the `accountId` (part of External Data) from the `getConfiguration` call to Oracle Configure, Price, Quote with the organization ID of the account-based profile associated with the order before calling the Submit Order endpoint.

Understand best practices for using account-specific pricing

Although the integration allows for account-specific pricing, it does not, however, allow for re-pricing of configured items, when any of the following conditions occur:

- The price list group (currency) is changed by the shopper
- An anonymous user logs in which results in a change of price list group (for example, an anonymous shopper logs in as an account-based shopper)
- The account based shopper changes the current account in context

Simple (i.e., non-configured) items are re-priced but configured items are not. A shopper cannot even re-configure the item to get the updated price. This is because Oracle Configure, Price, Quote does not accept secure punch-in attributes during the process of re-configuring an item. Unfortunately, the only available option is to delete the configured item and add it again as a fresh configuration.

To avoid inconvenience to the shopper, it is recommended that you add an information message to your custom widgets. The message should be seen when an anonymous shopper tries to add a configured item to cart, (for example, when they click the Customize button or when they are on the Product Details page. The message should suggest that the shopper first login and then do the configuration.

Set up and configure Commerce and Oracle Configure, Price, Quote for account-specific pricing

Use the following procedures to set up and configure Commerce and Oracle Configure, Price, Quote for the account-specific pricing feature:

Configure Oracle Integration for account-specific pricing

Use the following steps to configure Oracle Integration for account-specific pricing while using the Commerce/Oracle Configure, Price, Quote integration:

- Download the Oracle Integration packages found in the `OCCS-CPQ_CONFIGURATION_INTEGRATION_4.0.par` package file from Oracle Marketplace or My Oracle Support.
- Import the package into the OIC Environment
- Configure the Oracle Configure, Price, Quote Connection in OIC.
- `getConfigurations` (4.0) is used to validate configuration and pricing from Oracle Configure, Price, Quote.

For more information on using Oracle Integration, refer to the product Help Library.

The next steps you need to complete are downloading and configuring the required Server Side Extensions (SSEs) used for account-specific pricing. There are three SSEs used by the integration to support this. These are the following:

- `cpq-config-validation-app.zip`
- `cpq-config-punchin-store.zip`
- `cpq-config-punchin-agent.zip`

The information that follows describes how to download and configure these SSEs.

Download and configure `cpq-config-validation-app.zip`

Use the following steps to download and configure the `cpq-config-validation-app.zip` Server Side Extension for account-specific pricing while using the Commerce/Oracle Configure, Price, Quote integration:

- Login as Administrator to Commerce
- From the Admin interface, download the Server Side Extension (SSE) `cpq-config-validation-app.zip` by clicking **Design - Developer - Server-Side Extensions**. This SSE triggers the Configuration integration just setup on Oracle Integration.
- Unzip the file.
- Update the `config.json` file with the Oracle Integration Hostname and Port information.
- Also, a the dd extension environment variables for `OIC_USERNAME` and `OIC_PASSWORD` using the `Admin` endpoint

- Run the Node Package Manager to install in the .ZIP contents in the root folder
- Zip the contents in the root folder.
- Upload to the Extension Server using the `serverExtension` endpoint

Download and configure `cpq-config-punchin-store.zip` and `cpq-config-punchin-agent.zip`

Use the following steps to download and configure the `cpq-config-punchin-store.zip` and `cpq-config-punchin-agent.zip` Server Side Extensions for account-specific pricing while using the Commerce/Oracle CPQ integration:

- From the Commerce Admin interface, download the `cpq-config-punchin-store.zip` and `cpq-config-punchin-agent.zip` Server Side Extensions by clicking **Design - Developer - Server-Side Extensions**. These use the `CPQ Punchin Lib` to generate a query string with an encrypted `accountID` that is passed to the CPQ iFrame. This enables CPQ to show account specific prices.
- Unzip the files.
- Generate a public and private key using the OpenSSL utility.
- Place the private key file in the **Keystore** folders.
- Specify the public key in the `config.json` files.
- The public key is also added to the CPQ Integration Center under **Authentication Certificate**.
- Run NMP to install the .ZIP files contents in the root folder
- Zip the contents in the root folder
- Upload to the Extension Server.

Configure Commerce for account-specific pricing with Oracle Configure, Price, Quote

Use the following steps configure Commerce for account-specific pricing while using the Commerce/Oracle Configure, Price, Quote integration:

- Set up the external pricing validation webhook in Commerce: `/ccstorex/custom/v1/validateCPQConfigurations`
  Do this by going to **Settings - Web APIS - Webhook** in the Admin user interface. Choose the **External Price Validation** function API and enter the requested information.
- Enable and configure the Commerce/Oracle Configure, Price, Quote Configuration Integration by going to **Settings - Oracle Integrations** in the Admin user interface. Select the Oracle Configure, Price, Quote integration from the list.
- Modify widgets in the following layouts:
  – Store layout: Product Details page
  – Agent layout: B2B Checkout Layout - Accordion element: Search and Add Items to Cart popup stack

  These take the `accountIDs` of the user and display the appropriate prices for that account.

You also need to have synchronized product catalogs between Oracle Configure, Price, Quote and Commerce. The models available in Oracle Configure, Price, Quote need to have a corresponding externally configurable SKU in Commerce. Also, make sure you have set up the accounts and desired users in Commerce and have set up the proper pricing rules in Oracle Configure, Price, Quote (since it is the provider of the prices to sent to Commerce).

Finally, double check that you have setup pricing rules on Oracle Configure, Price, Quote based on a parameter (for example, Account Id) and have also added the public key to the **Integration Center - Authentication Certificate** in Oracle Configure, Price, Quote.

Use multi-level items

This integration features support for a hierarchical structure for items available to shoppers for purchase.

This integration provides support for a hierarchical structure for items available for shoppers to purchase. Commerce supports an “n-level” hierarchical configuration model. This means that a configured item can contain sub-items that are also configurable items and that can in turn contain sub-items that are configurable items.

An example of this would be a bundled package for a cellphone. The bundle itself would be the top-level item. The cellphone would be a configurable sub-item, but this could then itself have configurable sub-items, such as an SD card. Commerce can provide a top-level price for the bundle, but can also provide a price breakdown for each configurable item within the bundle.

If a shopper adds a multi-level item to their cart, Commerce works with Oracle Configure, Price, Quote to display the information about the multi-level item in the shopper’s cart. The cart displays a total price and an item price for any configurable sub-items. If the shopper changes any of the configurable sub-items, the price displayed for that sub-item changes and the total price is also amended accordingly.

When a shopper clicks on the Place Order button a validation check is carried out to ensure that the prices displayed for the configured items is still applicable. If it is then the order can proceed. If it is not, a message explaining this is displayed to the shopper and the cart is reloaded with up-to-date price information included for the configured items.

You can create a multi-level hierarchy in your catalog using either a recommended items model or a bill of materials model. You must refer to the relevant Oracle Configure, Price, Quote documentation for instructions on how to do this.

Use Quadplay/NPlay items

A standard, or single play, configured item represents a single service, such as Mobile Phone or IPTV that has a single set of configuration information, i.e. is based on a single configuration model in Oracle Oracle Configure, Price, Quote.

A Quadplay or NPlay configured item represents a package or bundle that combines multiple services in a single purchase and contains multiple sets of configuration information, i.e. is based on a single configuration model that also contains other configuration models in Oracle Configure, Price, Quote.

As an example, consider a case where the configured bundle contains 4 separate services (or ‘plays’) such as Landline, Internet, Mobile and IPTV. In this example, the bundle is called the Get4 Bundle. Unlike a standard configured item, the Get4 Bundle, as a Quadplay or NPlay configured item has configuration information at the following levels:

- Root level - in this example, the Get4 Bundle level.
- Branch level - in this example, the Landline, Internet, Mobile and IPTV levels.

With the support of Quadplay/NPlay configured items, the shopper adds the Get4 Bundle to the Oracle Commerce cart as a standard multi-level hierarchical configured item. This item also has the ability to be reconfigured if needed. The item is then validated and checked out as usual. For more detailed information working with Quadplay and NPlay items, refer to CX Communications - How to Customize and Extend – Configure NPlay Bundles with Oracle Configure, Price, Quote System Configuration white paper on the My Oracle Support site.

Understand Commerce Cloud Administration support for configuration metadata

In Oracle Configure, Price, Quote, a single model is also able to support multiple NPlay offers and additional versions of those offers. For example the same Model, Product Line, and Product Family might contain 3 variations on the same NPlay bundle such as the following:

- Starter Home Bundle
- Total Home Bundle
- Friends and Family Bundle

The same model might also support multiple versions of those bundle variations such

- Starter Home Bundle
- Starter Home Bundle 2017
- Starter Home Bundle 2018
  and
- Total Home Bundle
- Total Home Bundle 2017
- Total Home Bundle 2018
  and
- Friends and Family Bundle
- Friends and Family Bundle 2017
- Friends and Family Bundle 2018

To work with these types of variations, when the shopper selects a version of a bundle in Commerce and chooses to configure it, the configuration request needs to include extra information to allow the configurator to load the correct version of the configuration model. This extra information is provided in what is called configuration metadata. This data is passed along as a collection of key value pairs and aid in helping to identify the correct bundle.

Understand configuration metadata details

Where a Oracle Configure, Price, Quote configuration model supports multiple products and product variations, this information may not be sufficient to pre-load the order iframe with the correct starting point. In such cases extra information (i.e., configuration metadata) can included in the iframe URL created by Commerce.

Again, think of configuration metadata as a collection of one or more key value pairs that identifies the correct starting point for the configuration model. Configuration metadata can be static or dynamic. Static configuration metadata is manually entered by the Commerce Cloud Administrator and stored on the SKU record in Commerce. Dynamic configuration metadata can be captured by the PDP UI widget and can be entirely implementation specific.

Note: Dynamic configuration metadata is not restricted to being captured on the PDP UI widget. The dynamic configuration can be derived from any relevant information such as shopper profile.

This means that merchants can decide what dynamic key value pair data they want to capture and pass in the configuration request for any SKU. Dynamic configuration metadata can be mandatory or optional (i.e., in some cases the shopper MUST enter a value for a key and in some cases they may optionally enter a value for a key).

Configuration metadata lets merchants define a single model for all variants of a configurable product and at purchase time pre-load the configuration model at the appropriate starting point based on the shopper’s selection in Commerce.

The configuration metadata feature builds on the already existing support of the NPlay feature. Earlier there was support of the purchase of NPlay products but only where there is a one-to-one relationship between product and model (i.e., each NPlay product had to have its own unique corresponding configuration model in Oracle Configure, Price, Quote).

Enter configuration metadata via the administration user interface

To provide the configuration metadata needed for processing an order, the `configurationMetadata` property is exposed so that you can enter the information in the Commerce Cloud Administration interface. To get there click **Catalogue** then **Product** and finally **SKU**. This Administration panel lets you view, add, delete, and edit the Configuration Metadata values as required. Any request from Commerce to configure an item will include configuration metadata where it is available.

An example of using configuration metadata might be a case where a Commerce Cloud Administrator receives an email from a colleague in Oracle Configure, Price, Quote to advise them that the configuration model with the correct configuration metadata for the Family Plan products SKUs is now complete. The email contains the information to further configure the SKU. The SKU is called sku_fp_001 and the information provided is the following:

- Product Family – mobile
- Product Line - bundles
- Model - sku_fp_001
- Bundle Version - 18.1
- Region - EMEA

The process for entering the configuration metadata via the Admin interface would go something like the following:

1. Navigate to the Commerce Cloud administration user interface panel and select **Catalog**.
2. Select the **Family Plan** product and select the SKU sku_fp_001 which is currently flagged as inactive.
3. Click on **Externally Configurable SKU**. You see the text “Oracle Configure, Price, Quote can configure this as a part of a complex product.”
   Note: For any SKU where you want to add configuration metadata, you must make sure that Externally Configurable SKU is checked when you first begin entering data. A new input property will be displayed which will allow you to begin to enter one or more key value pairs of data.
4. Slide the panel down until you see the Product Family, Product Line, and Model fields appear on the panel. Enter all of the correct metadata details (the ones sent to you in the email from your Oracle Configure, Price, Quote colleague) manually.

5. Slide the panel down to see the Configuration Metadata table, click the **Add** button to add a row.
6. Add Bundle Version to the **Name** field. In the field next to Bundle Version, add 18.1 (as the bundle version number). You can press **Tab** or **Enter** to create a value entry. Click the **Add** button when done. A new row in the metadata value table appears.
7. Add Region to the **Name** field. In the field next to Region, add EMEA. You can press **Tab** or **Enter** to create the value entry.
8. Slide the panel back up to the top of the SKU ID panel, click **Active**, and then click **Save**.

At this point, you have entered all of the details received from your Oracle Configure, Price, Quote contact. This information must be entered correctly. The details that are entered are not seen by the customer. The information is designed to populate the config iframe window with the correct information. As a final step you activate the SKU and save the details.

Note: Since the configuration metadata must be entered manually via the Commerce administration console, keep in mind the following rules:

1. There is no support for versioning of configuration metadata so when an SKU record is imported, make sure it does not contain any configuration metadata that should replace any existing configuration metadata assigned to that SKU.
2. If the imported SKU record includes configuration metadata (columns present in the import file) but there are no values included then any existing configuration metadata will be deleted.
3. If the import SKU record does not include configuration metadata (no columns present in the import file), then any existing configuration metadata should be retained.

A Commerce administrator can view, delete, edit, or add Configuration Metadata key value pairs for any SKU where the _Externally Configurable SKU_ property is selected. A Configuration request from Commerce to Oracle Configure, Price, Quote always includes the configuration metadata set in Commerce for that SKU.

To work with configuration metadata you must have the following prerequisites:

- Oracle Commerce account
- Oracle Integration Cloud account
- Oracle Configure, Price, Quote account

Assign shipping groups to sub-items

You can assign different shipping groups to product configuration sub-items and more.

With a configurable item, you have the ability to assign different shipping groups to product configuration sub-items. Different shipping groups can be assigned to different levels in a multi-level configurable item.

Previously, you could only assign a shipping group at the root level. Now, with a configurable item you have the ability to assign different shipping groups to sub-items of the root item. Different shipping groups can now be assigned to the following levels in a multi-level configurable item:

- An item contained as the root item

- An item that may be contained at one or more branch items of the root item
- An item that may be contained at one or more of leaf items of a specific branch

Formerly, you assigned a shipping group at the root item level. The assumption was that the integration layer managed updates to the "shipping group" relationship object. The result of this was something where "if all sub-items are shipped then set the `shippingGroupItem` status on the root item to the status SHIPPED."

The original method did not work if you sold configured items that are a combination of goods and services and the services that needed to be assigned to separate shipping groups. It is important that customers selling nPlay bundles be able to assign each "play" to its own shipping group. As an example, you should reasonably expect to be able to assign separate shipping groups to, say, a handset (shipped to your office via priority mail), a router (shipped to your home via standard mail) and set top box (shipped to your vacation House via standard mail).

The ability to assign different shipping groups at different levels is also important for the Cancel In-flight order feature which lets you cancel In-Flight orders. An order which has been submitted but not fulfilled is considered to be In-Flight. When an In-Flight order is canceled, the process results in the creation of a new Cancel Order and may also result in the creation of a Return Request for items that may have already shipped to the shopper and must be returned, or at least refunded. In order to determine which items have to be returned, the system must be able to determine the shipping status of each item in the configuration.

In summary, all items in a multi-item configuration hierarchy from the root to the leaf level are assigned to a shipping group. You must also be aware that the assignment of shipping groups is also dependent on other key Commerce product features that directly impact the assignment of shipping groups.

For customers that are migrating to a release of Commerce that has this shipping group feature, the assignment of the shipping group logic is not adapted automatically. They will have to modify their `ClientConfiguration` settings. For first time users of the new release, the logic is adapted automatically.

Finally, logic changes are adapted only when the customer is using the Commerce user interface for all front-end behavior. If the customer is making endpoint calls directly, then they can call the `orders` endpoint with the required payloads without worrying about modifying `ClientConfiguration`.

Understand the details of assigning separate shipping groups to sub-items

When assigning shipping groups to sub-items, keep in mind the following:

- A `shippable` product property is provided and should be set to indicate that a product, whether a hard good or service, can be physically shipped to a purchase. When the `shippable` product property for a product is set to `FALSE`, it can be assigned to a Virtual Shipping Group so that Oracle Commerce does not attempt to calculate shipping charges for this product..
- If a product is a service then the physical address where the service is to be provided is provided by the fulfillment system based on the service account assigned to the item. This information is also provided by the client. Address information is mandatory for virtual shipping groups as it is required for tax calculations.
- If the product is a non-shippable good, (for example, a movie download, extended warranty etc.), address information is again mandatory as it is required for tax calculations.

Note: Address information is something used extensively in Commerce transactions. For all procedures and SSEs that require address information for endpoint inputs, in addition to using Commerce's default address formats, you can also use the REST API to create multi-country custom address formats. This lets you create country-specific address formats to ensure that your address formats align with the requirements of any external service that you might use. This means that addresses appearing in profiles, accounts, registration requests, order addresses and more can be customized. For more complete information on creating custom addresses and understanding how to use custom address formatting, refer to the following:

- Customize Address Formats using the API in *Extending Oracle Commerce*
- Work with address types in *Extending Oracle Commerce*
- Account Details in *Using Oracle Commerce*
- Work with account addresses in *Using Oracle Commerce*
- Work with account registration requests in *Using Oracle Commerce*

An `assetable` product property is provided which identifies those products that are sold as a service or subscription (for example a mobile phone tariff, magazine subscription etc). Assetable products must be assigned to the following when purchased:

- Customer Account
- Service Account
- Billing Account
- Billing Profile

These type of products are then assigned to a Virtual Shipping Group. Even if the product is a good (for example, a physical product), it must be assigned to a virtual shipping group This is because in these circumstances the Commerce purchasing process is not responsible for calculating shipping charges and the physical address where the item must be shipped but will be based on the service account assigned to the item.

The `onlineOnly` property is provided to identify products that can be purchased online but cannot be picked up in store. This means that an item can only be assigned an `inStorePickupShippingGroup` value if the `onlineOnly` property value for that product is `FALSE`.

In summary, configured item shipping groups are assigned at all of the following levels of the configured item:

- Root item of type `configurableCommerceItem`
- Branch items of type `configurableSubSkuCommerceItem`
- Leaf items of type `subSkuCommerceItem`

The assignment of shipping groups to configured items is then dependent on whether individual products are one of (or a combination of) the following types:

- `Shippable`
- `Assetable`
- Available for purchase online only

- Being sold as a package (`soldAsPackage` SKU property). The `soldAsPackage` property default is set to `TRUE`. Setting `soldAsPackage = FALSE` should only be considered for SKUs that have their configurable value set to `TRUE`. If the configurable value of a SKU is set to `FALSE`, the `soldAsPackage` property must be set to `TRUE`.

Understand the store features related to assigning shipping groups to sub-items

The following store features are provided to support the assigning of shipping groups to sub-items:

- The ability to assign all items in a configuration hierarchy to an appropriate shipping group type.
- The ability to change the assignment of shipping groups at all levels in a configuration hierarchy.
- The ability to update the tax calculation process to support shipping group assignment at all levels of a configuration hierarchy.
- The ability to update the shipping charge calculation process to support shipping group assignment at all levels of a configuration hierarchy.
- The ability to update to the proportional application of promotion discounts to all items in a configuration hierarchy

See the rest of the topics in this section for more information.

Understand tax calculation and shipping charges when assigning shipping groups to sub-items

When assigning taxes and shipping charges for shipping groups assigned to sub-items, you assign different calculating processes from normal customer calculation processes.

Understand the tax calculation process when assigning shipping groups to sub-items

The processes for calculating taxes and shipping charges for shipping groups assigned to sub-items differ slightly than the normal customer calculation process. In summary, the method used is that if sub-items are shipped separately, then the root item and the child items are sent as different items to the taxation system which contains the cost of that item alone and no additional item in the package.

This means that there are two ways that tax calculation occurs with a shipping group. The first way is that the price of the fully configured package is sent to the taxation system as all the items in the product configuration have to be delivered to a single place.

In determining the correct amount of tax payable on a product, the four key parameters passed to the tax calculator are the following:

- `amount` - the amount paid by the shopper
- `quantity` - the quantity of the item being purchases
- `shipping charge` - the shipping charge that has been calculated for the item
- `taxCode` - the tax code assigned to the product

For configured items, tax is calculated for each line item in the configuration hierarchy but the amount passed to the tax calculator is always be the external price returned from the configurator for that item, in the context of the overall configuration.

For any configured item, the price of a sub-item may be included in the price of the root item, so that the amount passed to the tax calculator would be zero.

The net result of this is that, although tax will be calculated for each item in the configuration hierarchy, all of the appropriate data will be passed to the tax calculator. It is possible, however, that the amount of tax paid by the shopper may be skewed in circumstances where the configured item contains products with different tax codes.

Understand the shipping charge calculation process when assigning shipping groups to sub-items

To determine the correct shipping charges payable on a product, the key parameters passed to the shipping calculator are the following:

- `shippingMethod` - the shipping method selected for the item.
- `quantity`- the quantity of the item being shipped.
- `amount` - the amount paid by the shopper for the item.

For configured items, shipping charges are calculated for each line item in the configuration hierarchy. The amount passed to the shipping calculator, however, will always be the external price returned from the configurator for that item, in the context of the overall configuration.

For any configured item, the price of a sub-item may be included in the price of the root item, so that the amount passed to the shipping calculator would be zero.

The net result of this is the following:

- Shipping charges are calculated for each item in the configuration hierarchy and all of the appropriate data is passed to the shipping calculator. It is possible, however, that the amount of shipping charges paid by the shopper may be skewed in circumstances where the configured item price does not accurately reflect the actual proportional amount paid for an item in the configuration hierarchy.
- Shipping surcharges are included for any item in the hierarchy where such surcharges have been assigned to that product in Commerce.
  Note: The charges are only included for the root item if the whole configuration is sold as a package
- A merchant can always choose to apply a shipping surcharge for any item where there is a risk that the shopper will be undercharged for shipping when a particular product is purchased as part of a configured item.
- Shipping surcharges are not considered if any item presents itself in a virtual shipping group as those items are non-shippable and are not required to have shipping surcharges.

Understand shipping charge and tax calculation when assigning costs to items sold as a package

When assigning costs to items sold as a package, you assign processes for calculating shipping charges and taxes that differ slightly from normal customer calculation processes.

Understand the shipping charge calculation process when assigning costs to items sold as a package

The processes for calculating shipping charges and taxes when assigning costs to items sold as a package differ slightly from normal customer calculation processes. To determine the correct amount of shipping charges payable on an item configured as a package, the following key parameters are passed to the shipping calculator:

- `shippingMethod` - the shipping method selected for the item
- `quantity` - the quantity of the item being shipped
- `amount` - the amount paid by the shopper for the item

For items sold as a single item (root item) configured as a package, the following occurs:

- The `amount` passed to the shipping calculator is always the total price for the configured item.
- The `shippingMethod` passed to the shipping calculator will always be the shipping method assigned to the root item.
- The `quantity` passed to the shipping calculator is always be the quantity of the root item.

Shipping charges will be calculated accurately, given that you have decided that the configured item must be shipped as a unit. Any shipping surcharges assigned to a sub-item in the configuration hierarchy will not be included in the total shipping charges.

Understand the tax calculation process when assigning costs to items sold as a package

In summary, the way that tax calculation occurs with a shipping group sold as a package is that the price of the fully configured package is sent to the taxation system as all the items in the product configuration have to be delivered to a single place.

To determine the correct amount of taxes payable on an item configured as a package, the following key parameters are passed to the tax calculator:

- `amount` - the amount paid by the shopper
- `quantity` - the quantity of the item being purchased
- `shipping charge` - the shipping charge that has been calculated for the package item
- `taxCode` - the tax code assigned to the product.

For configured items sold as a package (i.e., where the `soldAsPackage` value for the root item = `TRUE`), taxes are calculated based on the root item only. For configured items sold as a package, the following occurs:

- The `amount` passed to the tax calculator is always the total price for the configured item.

- The `taxCode` passed to the tax calculator is always the tax code for the root item. This means that although taxes are calculated for the configured item, the amount is based only on the tax code of the root item.

Understand how promotion discounts are applied to multi-level items

Promotional discounts can be applied proportionally to multi-level items.

For a multi-level configured item, promotion discounts must be applied proportionally across the root and all of the sub-items in the hierarchy.

In Commerce, order level discounts are applied proportionally across all items in the order (unless an item is specifically excluded from benefiting from such a discount). For a configured item, a proportional discount must be applied to all items in the configuration hierarchy. For example if an order level promotion applies a 10% discount then that 10% discount must be applied to any configured item in the order.

For a multi-level configured item, however, the promotion discount must be applied proportionally across the root and all of the sub-items in the hierarchy. This applies only to configured items that are not sold as a package (i.e. where the `soldAsPackage` value on the root item = `FALSE`).

Add payment details to customer billing profile

You must add payment details to customer billing profiles so that this information is passed downstream to fulfillment and provisioning systems.

In Telco transactions there is critical contact information that must be passed downstream to fulfillment and provisioning systems. Based on the Customer Account Model, this information is the following:

- Customer Account
- Service Account
- Billing Account
- Billing Profile

This topic covers the processes involved in the updating of payment details in a Billing Profile.

Understand how billing profiles are handled

A Contact (that is a user, shopper, or customer) may have the following information in Oracle Commerce transactions:

- A Customer Account (Account of type “Customer”).
- A link to other Customer Accounts. This would occur where the merchant supports account models such as “Family Account,” “Household,” or “Family and Friends.”
- A link to one or more Service Accounts (Accounts of type “Service”).
- A link to one or more Billing Accounts (Accounts of type “Billing”).
- A link to one or more Billing Profiles.

In this type of transaction model, Commerce has the important ability to create or update a billing profile with payment details. This is important because the payment information for the billing profile needs to be captured and passed on to the primary

CRM (Customer Relationship Management) system in a PCI compliant manner. For the Oracle Telco CX Solution, the primary CRM system is Oracle Engagement Cloud (OEC).

Note: In an integration like this, transaction payment details that are stored in the CRM system are used for recurring payments. This info is pulled by the billing system from CRM. To compare this with Commerce, Commerce handles one time/upfront payments by interacting with payment gateways.

At this point in time, Commerce is PCI compliant whereas Oracle Integration Cloud and Oracle Engagement Cloud are not. Commerce supports the storing of credit cards against a shopper profile. The card details are captured, however, in the store as part of the checkout flow and subsequently a tokenized version of the card is obtained from an integrated payment system as part of the payment authorization process. This token, along with a masked version of card number and the expiration date are then stored against the shopper's profile so that the shopper can easily use the same card for future purchases.

A feature is now supported that offers a generic horizontal benefit to Commerce and contributes to the Telco specific vertical requirement. This feature uses a store API endpoint that allows a shopper to store credit cards as part of their profile without actually purchasing an order. This endpoint can then be used to enable Commerce to pass credit card information to Oracle Engagement Cloud as part of the shopper billing profile when creating or for updating accounts in OEC.

Understand the Update Profile endpoint and Generic Payment webhook

As mentioned, Commerce provides an Admin and Agent Update Profile store API endpoint that lets you add and store customer credit cards as part of a shopper Billing Profile without actually purchasing an order.

The name of the endpoint is `addCreditCard`. The Admin URI for the endpoint is `POST /ccstoreui/v1/current/creditCards/`. The Agent URI for the endpoint is `POST /ccagentui/v1/profiles/{id}/creditCards`.

The endpoint can be used to invoke Add Card requests multiple times to let you add more than one card to a shopper Profile. Each new card is then stored against the profile. The inputs of this endpoint are:

- `cardType`
- `nameOnCard`
- `cardNumber`
- `expiryMonth`
- `expiryYear`

Both versions of this endpoint trigger the Generic Payment webhook for a Tokenize operation on the payment system. The payment system is expected to return a tokenized value of the card which is then saved against the billing profile. The endpoint then returns back a stored card ID.

The Admin Get Profile endpoint can then be used to get the token value of the card using the stored card ID. See Configure Payment Processing and Create a Generic Payment Gateway Integration for more complete information on this subject.

Add and update a Billing Profile to include a card token

There are two processes/flows that Commerce uses to capture the billing profile information that can be passed on to the primary CRM system. These are the following:

- Commerce creates an account(s) for a contact which includes creating one or more billing profiles.
- Commerce updates an existing billing profile.

To assist in these processes, Commerce provides a Customer Account Model Server Side Extension (SSE). The sections that follow provide the details on the various processes and flows that this SSE supports

Understand the OCC to OEC Account Create flow

The basic information on this flow is the following:

- SSE Name: Customer Account Model
- Endpoint Name: Create Accounts
- Flow Name: OCC to OEC Account Create flow

In this process, the SSE first identifies the payment type. If the payment type is Card, a check is done to see if the token for the card has been passed in (i.e., an existing card stored on the shopper's Commerce profile). If the token has been passed in, then a check is done to see that the basic card information has also been passed in the form of `maskedCardNumber` and `expiryMonth`. If the token has not been passed in (i.e., a new card is being introduced), then a check is made to look for the following "full card" information being passed in:

- Card Type
- Name on Card
- Card Number
- CVN (card verification number)
- Expiry Month
- Expiry Year

There is also a step in the SSE execution whereby an API call can be made to the storefront Profiles/Update Profile endpoint to retrieve a tokenized version of the card. The billing profile information passed to the OCC to OEC Account Create flow includes:

- Payment Method=Card
- Masked Card Number
- Card Expiry Date
- Tokenized representation of the card

The next section provides details on an update process/flow that the SSE handles.

Understand the OCC to OEC Account Update flow

The basic information on this flow is the following:

- SSE: Customer Account Model
- Endpoint Name: Update Accounts
- Flow Name: OCC to OEC Account Update flow

In this process, the SSE first identifies the payment type. If the payment type is Card, a check is made to see if the token for the card has been passed in (i.e., there is an existing card stored on the shopper's Commerce profile). If the token has been passed in, then a check is made that the basic card information has also been passed in the form of `maskedCardNumber` and `expiryMonth`.

If the token has not been passed in (i.e., a new card has been introduced), then a check is made to look for the following “full card” information:

- Card Type
- Name on Card
- Card Number
- CVN (card verification number)
- Expiry Month
- Expiry Year

The billing profile information passed to the OCC to OEC Account Update flow includes:

- Payment Method=Card
- Masked Card Number
- Card Expiry Date
- Tokenized representation of the card

Note: Keep in mind the following additional details regarding the OCC OEC Comms: Account Update flow:

- A check is made to verify that the payment type is credit card/debit card. If it is credit card/debit card, then a check is made to verify whether `creditCardNumber` is masked or non-masked.
- If `creditCardNumber` is masked, an additional check is made to verify that the masked `creditCardNumber` and `creditCardId` values provided are valid. If both are valid, then only `creditCardNickname` is allowed to update. All other fields/properties are not allowed to update.
- If `creditCardNumber` is unmasked, then the card is considered a new card. Tokenization occurs with the provided card details is similar to the Create Account flow. The process ends with the new card and token details stored in CDM.

Understand fields and properties supported by the billing profile

For the credit card or debit card payment type, the following fields/properties are supported:

- paymentMethod: “debitcard”/”creditcard”
- creditCardNumber (mandatory field)
- creditCardExpiryMonth (mandatory field)
- creditCardExpiryYear (mandatory field)
- creditCardContactName (mandatory field)
- creditCardType (mandatory field)
- creditCardSecurityCode
- creditCardNickname
- creditCardIin

For the bank transfer payment type, the following fields/properties are supported:

- paymentMethod: “bankTransfer”
- bankAccountNumber (This is the only mandatory field for bankTransfer payment type.)
- bankRoutingNumber
- bankAccountType
- bankAccountName
- bankSortCode
- bankName
- bankAddress
- bankIban
- bankSwiftCode

Understand the process changes as seen in the store interface

With a standard checkout flow where the shopper does not yet have a customer account model and is purchasing a service, the store interface captures the payment card details. This includes the following:

- The shopper is prompted to identify the payment type. If the payment type is “Credit/DebitCard,” the shopper is prompted to select either an existing card or enter new card details.
- If it is a new card the user interface captures the following required card information:
  - Card Type
  - Name on Card
  - Card Number
  - CVN (card verification number)
  - Expiry Month
  - Expiry Year

For a billing profile update checkout flow where the shopper, who does not have a customer account model and is purchasing a service, the store interface captures the payment card details. These details include the following:

- The shopper is prompted to identify payment type.
- If the payment type is “Credit/DebitCard,” the user interface captures the following the required card information:
  - Card Type
  - Name on Card
  - Card Number
  - CVN (card verification number)
  - Expiry Month
  - Expiry Year

Understand the Customer Account Model

For customers using the Customer Account Model SSE, there are a number of different account types that can be associated with a shopper within the Oracle Commerce/Oracle Configure, Price, Quote integration.

If you are using the Customer Account Model SSE, there are a number of different account types that can be associated with a shopper within Oracle Commerce. To configure the Customer Account model, use the provided SSE. To do this, click the **Design** icon in the Administration user interface. Then click **Developer** and **Server-Side Extensions**. Select the CustomerAccountModel-store SSE and/or the CustomerAccountModel-agent SSE.

Both SSEs enable integration with an external CRM system to retrieve and update the following:

- Contacts
- Accounts (Customer Billing and Service accounts)
- Account Roles (Admin, Buyer and User)
- Billing Profiles

Finally, the SSEs serve as the API for the pre-built integration with Oracle Engagement Cloud.

There are three account types available within Commerce relating to billable services, Customer account, Service account, and Billing account.

The details for these three accounts are captured when an order is placed and their relationship with the service is maintained after an order has been fulfilled.

In many instances these three accounts may all refer to the same person or organization, but there may also be instances when they differ, and it is important to understand the relationship between the different types of account.

In addition to the three account types, there is a Billing Profile, which includes information such as billing preferences.

All of the information required for the Customer, Service, and Billing accounts, and for the Billing Profile is captured during the order process in Commerce.

Customer Account

This type of account represents the person or organization that owns the service. It includes basic customer information, such as name and address and can receive both services and bills.

Customer accounts are the highest level in the account hierarchy and can perform all customer, service, or billing functions.

Service Account

This type of account represents the person or organization that receives the service.

The address associated with the Service account defines the physical location where the service must be delivered. This address is used to verify service and ordering eligibility.

Service accounts are required when the location and/or party receiving the service differ from the Customer account. If a Service account is required, it is always a child of a Customer account. There can be multiple Service accounts associated with a single Customer account.

A Service account cannot be used to perform any of the functions of a Customer or Billing account.

Billing Account

This type of account represents the person or organization that pays for the service.

Billing accounts are required when the location and/or party paying for a service differ from the Customer account. If a Billing account is required, it is always a child of a Customer account. There can be multiple Billing accounts associated with a single Customer account.

A Billing account cannot be used to perform any of the functions of a Customer or Service account.

Billing Profile

A billing profile may be associated with either a Customer account or a Billing account. It captures information such as billing preferences, method of payment, and contact details. There may be more than one billing profile associated with a Customer or Billing account, and the shopper must choose which billing profile to use when placing an order for a service.

Note: Address information is something used extensively in Commerce transactions. For all procedures and SSEs that require address information for endpoint inputs, in addition to using Commerce's default address formats, you can also use the REST API to create multi-country custom address formats. This lets you create country-specific address formats to ensure that your address formats align with the requirements of any external service that you might use. This means that addresses appearing in profiles, accounts, registration requests, order addresses and more can be customized. For more complete information on creating custom addresses and understanding how to use custom address formatting, refer to the following:

- Customize Address Formats using the API in *Extending Oracle Commerce*
- Work with address types in *Extending Oracle Commerce*
- Account Details in *Using Oracle Commerce*
- Work with account addresses in *Using Oracle Commerce*
- Work with account registration requests in *Using Oracle Commerce*

Use Recurring Charge Items

This integration provides you with the ability to configure items with a recurring charge that can be passed on in purchase.

This integration enables you to provide items that come with a recurring charge available for shoppers to purchase. Examples of items that include a recurring charge include a service such as a data/call minutes/ text message bundle for a cellphone, or a subscription charge for a cable television package.

Items that include a recurring charge may have just a recurring charge or may have a recurring charge and a price. If an item has a price and a recurring charge, it is assumed that the item is not a service or subscription item. In this case the price represents an upfront payment and the recurring charge is the means by which the outstanding balance is paid.

Identification of items that include a recurring charge must be carried out through your Oracle Configure, Price, Quote Admin account. Please refer to the Synchronize Oracle Configure, Price, Quote Cloud Parts with Commerce SKUs section of the Implementation Guide contained in the Integrating Oracle CX Commerce with Oracle Configure, Price, Quote article on My Oracle Support.

If a shopper adds a recurring charge item to their cart, Commerce works with Oracle Configure, Price, Quote to display full information about the recurring charges associated with the order. This includes how much the recurring charge is for, the frequency of the recurring charge, and the duration for which the recurring charge will be made.

Note: The default value for frequency is monthly and the default value for duration is open-ended. If either of these is not the right value for the item they must be corrected in the Oracle Configure, Price, Quote Part for the item.

Items with a recurring charge are not included in order sub-total passed to the shipping calculator. If a cart contains only recurring charge items the order sub-total passed to the shipping calculator is zero, which means that no shipping charge is applied to the order.

Configure payment for recurring charge items

Commerce includes several built-in integrations with payment gateways that let your store accept credit cards, debit cards, gift cards, and PayPal payments. However, these integrations do not currently support recurring charges. If you wish to sell items with recurring charges you must use one of the methods set out below to configure Commerce payment processing to support recurring charges.

Configure credit card payments

Follow these instructions to configure your credit card payment processing to handle recurring charges:

1. Create a custom credit card payment extension.
   For detailed instructions about performing this step, refer to Create a credit card extension.
2. Install the custom credit card payment extension.
   For detailed instructions about performing this step, refer to Install the extension and configure the gateway.
3. Enable the payment gateway.
   For detailed instructions about performing this task, refer to Create a Credit Card Payment Gateway Integration and Create a Generic Payment Gateway Integration, .
4. Add custom properties to the Credit Card Payment webhook.
   For detailed instructions about performing this task, refer to Install the extension and configure the gateway .

   Note: This webhook is not site-specific. If you are running multiple sites on your Commerce instance, the configuration you supply applies to all sites that use this webhook.

Configure non-credit card payments

Follow these instructions to configure your generic gateway payment processing to handle recurring charges:

1. Create a custom generic payment extension.
   For detailed instructions on performing this task refer to the Supported payment methods and transaction types section of Create a Generic Payment Gateway Integration.
2. Install the generic payment extension.
   For detailed instructions about performing this step, refer to the Install the extension section of Create a Generic Payment Gateway Integration.
3. Enable the payment gateway.
   For detailed instructions about performing this task, refer to Create a Credit Card Payment Gateway Integration and Create a Generic Payment Gateway Integration.
4. Customize the payment details widget to capture payment information other than card details.
5. Add custom properties to the Generic Payment webhook.
   For detailed instructions about performing this task, refer to Send custom properties to a payment gateway.

   Note: This webhook is not site-specific. If you are running multiple sites on your Commerce instance, the configuration you supply applies to all sites that use this webhook.

Use Asset Based Ordering

The Commerce/Oracle Configure, Price, Quote integration features asset based ordering (ABO).

Understand asset definition and related properties

This integration supports an asset based ordering (ABO) model. Asset based ordering lets you sell tangible assets or subscription services delivered over a period of time; for example, mobile phone call and data plans, television and broadband packages. When these orders are subsequently fulfilled, the fulfillment system notifies Oracle Configure, Price, Quote via an asset API, and Oracle Configure, Price, Quote then creates an asset in the Oracle Configure, Price, Quote asset repository. To better understand asset based ordering and its related services, it is important that you first understand asset definition and the related properties.

In the Commerce/Oracle Configure, Price, Quote integration, Commerce acts as the first point of contact for registered and account-based shoppers. Commerce lets a shopper review and select their purchases as needed.

In Telco-related purchases, Oracle Configure, Price, Quote acts as the primary Asset system. Commerce makes a call to Oracle Configure, Price, Quote to retrieve the assets for a particular profile or account. Oracle Configure, Price, Quote then manages the retrieval of assets from multiple systems if necessary.

One of the underlying features of any Telco solution is the ability for a self-service channel (in this case, Commerce) to retrieve and display the complete set of assets owned by the shopper and then to allow the shopper to trigger operations on those assets. In order for this to happen, Oracle Commercesupports the following asset-related information properties at the order item level:

- Asset Key - the `assetKey` property (formerly `assetID`) is a unique identifier that is assigned to potential assets when adding items to a cart. This value is used throughout the asset life cycle by fulfillment, asset management, and order capture systems. In this case, the term "potential" is used meaning that not every item added to a cart gets completely fulfilled, a provisioning system may fail, etc. For configured items, the assetKey value is assigned as part of the asset configuration process in Oracle Configure, Price, Quote.

- Parent Asset Key - Some configured items in an order may be many levels deep in a BOM structure. In order to ensure that the BOM hierarchy is consistent throughout the asset life cycle, each item in the BOM hierarchy must be able to identify its direct parent. The `parentAssetKey` property makes this possible. For root items in a BOM hierarchy, the `parentAssetKey` value is `NULL`.
- Root Asset Key - Again, some configured items in an order may be many levels deep in a BOM structure. In order to ensure that the BOM hierarchy is consistent throughout the asset life cycle, each item in the BOM hierarchy must be able to identify its root asset. The `rootAssetKey` property makes this possible. For root items in a BOM hierarchy, the `assetKey` and the `rootAssetKey` value is the same.

Understand the mapping of an asset key to an item

In Oracle Commerce, a configurable SKU may be flagged as "non-assetable" which means that when this item is configured and purchased it will not be assigned a customer, billing, or service account and will not become an asset for the shopper. When this item is configured, however, Oracle Configure, Price, Quote returns asset key values for each item in the BOM by default.

Note: The flag name is `assetable` and the default value is `False`.

Commerce only maps asset key values to commerce items that are actually "assetable." The rules used in this process are the following:

- If the SKU selected for configuration is based on a product where the property value for `assetable = TRUE`, map the asset key data.
- If the SKU selected for configuration is based on a product where the property value for `assetable = FALSE` do not map the asset key data.

Understand the Asset Root

It is also important to point out that when a shopper chooses to configure a SKU in Commerce, the root item of the BOM returned from Oracle Configure, Price, Quote may not always be that same SKU, that is, the root item part number may not map directly to the selected configurable SKU.

Say, for example, a mobile product bundle that is represented by the "Red Bundle" SKU in Commerce is configured several ways. At the initial step of the configuration process, the shopper may be asked to select either the Standard Package, Student Package, or Value Package. Depending on the selection made, the root item of the configuration will be different.

So, based on this example, it is possible that the SKU selected by the shopper to configure the item will be based on a product where `_assetable _= TRUE` but the root item for the resulting configuration may be based on a product where _`assetable _= FALSE.`

The rule that decides whether a configured item should be assigned `_assetKey _ property` is based on whether the SKU that corresponds to the root item of the configuration is "assetable" and not on whether the item that the shopper selected to be configured in Commerce is "assetable".

Understand asset based ordering and related service operations

As already discussed, asset based ordering lets you sell assets or subscription services delivered over a period of time. When these orders are subsequently fulfilled, the fulfillment system notifies Oracle Configure, Price, Quote via an asset API, and Oracle Configure, Price, Quote then creates an asset in the Oracle Configure, Price, Quote asset repository.

Once created, assets can subsequently be reviewed by shoppers in the My Services management area within the shopper account. The shopper can then administer an asset by creating and placing new commerce orders to perform a number of actions on the asset. These include the following:

- Modify
- Renew
- Terminate
- Suspend
- Resume
- Upgrade

A Services-store SSE and the Services-agent SSE can be configured from the administrator’s user interface. To do this, click the **Design** icon in the Administration user interface. Then click **Developer** and **Server-Side Extensions**. Select the name of the SSE. Both SSEs enable integration with 3rd party asset management systems to retrieve and execute operations and services on assets available to the shopper. They also serve as the API for the pre-built integration with Oracle Configure, Price, Quote asset management.

For each of these operations the operation flow is basically the following:

- The shopper views their list of assets.
- The shopper selects an asset.
- The shopper selects the desired operation:
  - For a Modify operation, the system loads the Oracle Configure, Price, Quote hosted iFrame, the shopper makes their modifications, and selects to add to cart. This is the Oracle Configure, Price, Quote hosted iFrame presented to the shopper when they configure a new purchase prior to adding it the cart, reconfigure a new purchase prior to checking out, or modify an existing asset.
  - For an Upgrade operation, all available upgrade options are displayed on the Storefront Asset list and then the specific Asset Details pages. After you have selected a specific Asset, you can select the Upgrade option to view its upgrade details. When you click on an upgrade option, an iFrame is returned and opens up in the context of the available upgrade options. You can then choose your asset upgrade(s) and add them to your cart.
  - For all other operations, the system only makes a call to Oracle Configure, Price, Quote to execute the operation.
- Oracle Configure, Price, Quote asset records are updated.
- Oracle Configure, Price, Quote returns the required JSON representation of the terminated/renewed/suspended/resumed/modified/upgraded asset.

- Commerce transforms the JSON returned to a commerce item and adds it the cart.
  - For the Modify and Upgrade operations, the transformation is executed in the Commerce client layer.
  - For all other operations the transformation occurs in the Services SSE which uses the Asset Action OIC flow.
- The shopper continues shopping.
- When the shopper places the order, the `cpq-config-validation-app` SSE is triggered through the External Pricing Webhook. This SSE invokes `getConfiguration` for every flow except when the asset actions are Terminate and Suspend. The response received from OIC gets transformed from the `cpq-config-validation-app SSE` as the OIC flows, `getConfigurations`, and `getConfigBom` return a flat structure of items which is converted to a hierarchical structure. Validation is then done in the `cpq-config-validation-app` SSE to verify that data is not manipulated on client-side.
- The order items representing Asset Based Ordering operations are submitted downstream and contain all of the information required to ensure that the operation is fulfilled.

The specific Services actions are described in more detail later in this section. These actions are important for maintaining an efficient self-service channel. When a shopper performs any one of these actions on an asset, the Oracle Configure, Price, Quote asset repository is updated accordingly.

Since Commerce serves as the first point of contact for shoppers, it allows shoppers to review and select their purchases. In the case of a Telco commerce solution, the Oracle Configure, Price, Quote asset repository acts as the primary Asset system in which Commerce makes a call to Oracle Configure, Price, Quote to retrieve the assets for a particular profile or account. Oracle Configure, Price, Quote manages the retrieval of assets from multiple systems.

The Commerce Telco solution gives the shopper the ability to retrieve and display the complete set of Assets owned by the shopper/account as well as carry out the mentioned administration operations that can be performed on those assets.

When a shopper opens the My Services management area within their account they are presented with a list of the assets linked to their account. From here they can select an asset and click on the Details button next to the desired asset to see the detailed view of the service.

It is at this point that the shopper can choose between the Modify, Renew, Terminate, Suspend, Resume, and Upgrade actions.

Modify

If the shopper chooses Modify, Commerce loads the current configuration for the service in question and opens a screen that allows the shopper to modify the service as required. The new monthly charge for the service is updated automatically as the shopper makes their selections. The shopper can then add the modified service to their cart.

When the shopper goes through checkout and completes their order, Commerce submits a service modification request to the fulfillment system.

As mentioned, earlier the steps in this operation are typically the following:

- The shopper views their list of assets.
- The shopper selects an asset.

- The shopper selects a Modify operation. For a Modify operation, the system loads an Oracle Configure, Price, Quote hosted iFrame. The shopper makes their asset modifications and selects to add it to cart.
- Oracle Configure, Price, Quote asset records are updated and Oracle Configure, Price, Quote returns the required JSON representation the terminated/renewed/ suspended/resumed/modified asset.
- Commercetransforms the required JSON returned to a commerce item and adds it the cart. This transformation is executed in the Commerce client layer.
- The shopper continues shopping and then checks out.

The order items representing ABO operations are submitted downstream and contain all of the information required to ensure that the operation is fulfilled.

Renew

If the shopper chooses Renew, Commerce determines the configuration ID that represents a renewal of the service in its current configuration and then adds a renewal instruction to the shopping cart and opens the Shopping Cart Details page.

When the shopper goes through checkout and completes their order, Commerce submits a service renewal request to the fulfillment system. This is handled and invoked via the Services SSE endpoint `/services/{id}/renewService` and the SSE invokes the OIC flow.

Terminate a service

If the shopper chooses Terminate, a configuration ID is sent back by Oracle Configure, Price, Quote that represents the termination of the service in question. A termination instruction is added to the shopping cart and the Shopping Cart Details page is then opened

When the shopper goes through checkout and completes their order, Commerce then submits the service termination request to the fulfillment system. This is handled and invoked via the Services SSE endpoint `/services/{id}/terminateService` which invokes the OIC flow.

Suspend a service

If the shopper chooses Suspend, it allows them to suspend a service. Commerce provides an endpoint that is used to suspend a service. When a shopper selects to suspend a service, they choose the Suspend action and then enter a valid suspend date.

By clicking on the Suspend button, Commerce determines the configuration ID that represents the suspension of the service in question, adds a suspension instruction to the shopping cart, and opens the Shopping Cart Details page. When the shopper goes through checkout and completes their order, Commerce submits a service suspension request to the fulfillment system. Also, when the Suspend action is chosen from the store user interface, the transaction date is set to current date (i.e., the date that the shopper suspended the service. This suspension may be indefinite or for set for a specific period of time by entering a date. A specific shopper use case example might be letting a shopper suspend a data plan for 30 days.

The Services SSEs support the Suspend operation which returns either a Configured Item or an Error. Services is part of the Oracle Integrated Cloud flow.

The Services API has an endpoint called Suspend Service. The endpoint can be triggered when a shopper selects to suspend a service, enters a valid suspend date and time, and selects to proceed. Inputs include the following:

- Asset Key
- Action - Suspend
- Transaction Date - The valid suspend date and time information that the shopper entered. The Suspend date is not equal to or later than the asset end date.

The API returns either a Configured Item or an Error.

Resume a Service

If the shopper chooses Resume, it allows them to resume a service that was previously suspended. Commerce provides an endpoint that is used to resume a service. When a shopper selects to resume a service, they choose the Resume action and then enter a valid resume date and time to resume the service.

By clicking on the Resume button, Commerce determines the configuration ID that represents the service in question that is to be resumed, adds a resume instruction to the shopping cart, and opens the Shopping Cart Details page. When the shopper goes through checkout and completes their order, Commerce submits a resume service request to the fulfillment system. Also, when the Resume action is chosen from the store user interface, the transaction date is set to current date (i.e., the date that the shopper resumed the service).

The Services SSEs also support this Resume operation which returns either a Configured Item or an Error. Services is part of the Oracle Integrated Cloud flow.

The Services API has an endpoint called Resume Service. The endpoint can be triggered when a shopper selects to resume a service, enters a valid resume date and time and selects to proceed. Inputs include the following:

- AssetKey
- Action: Resume
- Transaction Date: The valid Resume date and time that the shopper entered. The Resume date is not equal to or later than the asset end date.

The API returns either a Configured Item or an Error.

Note: An action code of Renew, Terminate, Suspend, and Resume is assigned to an item when that respective operation has been applied to that item.

Upgrade an Asset

With Asset Based Ordering, you have the ability to upgrade an existing asset. If a shopper chooses the Upgrade operation, they can upgrade an asset to one of the upgrades available for the product. Any Root asset may have one or more upgrade options available at any time. Commerce SSE endpoints `getServices` and `getServices/{id}` return the upgrade options for each of the asset if the query param "`expand=occ_upgradeOptions`" is passed. Once the shopper selects the Upgrade action and clicks the Upgrade button, this action invokes the `upgradeService` SSE endpoint which gets the upgrade name as input and returns the query string that is to be used as punchin URL to launch the Oracle Configure, Price, Quote iFrame. From the user interface point of view, a shopper selects to upgrade an asset, choose the Upgrade action in the Asset Details view and then select the asset upgrade that they desire. Oracle Configure, Price, Quote maintains a custom upgrade options table for Commerce to query in order to know which upgrades are available for a given asset. The key parameter controlling the operation is the SKU of one or more items that are part of a current asset bundle. The response received after initiating this operation includes all of the eligible SKUs that an asset can be upgraded to.

Commerce has an Upgrade endpoint to fetch all available upgrade options. The input for this endpoint is `currentModel` and `currentOffer`. The following presents the details on the information needed to retrieve the upgrade options table from Oracle Configure, Price, Quote:

- Oracle Configure, Price, Quote Table Name: `INT_UPGRADE_OPTIONS`
- Input (via URL parameter): `occ_Upgrade_options` query parameter which is a list of `currentSku` plus `currentModel` for the assets. Type: String. This query parameter is passed from Commerce to the SSE endpoint (described further in this section). After a `getAssets` call, you then pick the `currentModel` and `currentOffer` from each asset and invoke the Oracle Configure, Price, Quote upgrade options table.
- Output: `upgradeName, upgradeProductId(OCC)`

The following presents the details on the basic schema of the upgrade options table maintained by Oracle Configure, Price, Quote that contains the specified upgrade information:

Table 2-2 Oracle Configure, Price, Quote Upgrade Table

Column Name	Data Type and Description
currentSku	String. This value defines the current offer. This needs to be stored as an attribute of an asset record. This value is sent from Commerce while retrieving the upgrade options.
currentModel	String. The model name for which the upgrade offer is valid.
upgradeName	String. This value is passed to he Oracle Configure, Price, Quote iFrame while upgrading and is used by Oracle Configure, Price, Quote to default and render the upgrade options. This is not be used by Commerce for any purpose.
upgradeProductId	String. This is used by Commerce to identify the product corresponding to upgrade option. The product display name, description, images, etc. can be used to show upgrade details to the shopper.

Note: A combination of currentSku and currentModel is used as the parameter to find the matching upgrade options

The Oracle Configure, Price, Quote upgrade table is queried by Commerce to help identify the upgrades that are available for a given asset. These upgrade options are then presented to the shopper. An example of what the upgrade information would contain includes the following:

Table 2-3 Example of upgrade options returned to Commerce

currentSku	currentModel	upgradeName	upgradeProductID
4ForUDeal	nPlay	4ForUDeal	prod102

It is recommended that the currentSku column is indexed. The following presents additional details on each returned upgrade option:

- currentOffer - Maps to a configurable attribute on the root config model in Oracle Configure, Price, Quote. This needs to be stored as an attribute mapping onto the root asset as well. This value is sent from Commerce while retrieving the upgrade options.
- currentModel - Maps to the variable name of the root config model in Oracle Configure, Price, Quote which the upgrade offer applies to.
- upgradeName - Maps to the `_config_upgrade_name` that is passed from Commerceto Oracle Configure, Price, Quote, which drives recommendation rules on the upgrade. This is not used by Commerce for any other purpose.
- upgradeProductId - Maps to the Product Id of the upgrade offer in Commerce. This is used to show upgrade details (product display name, description, images, etc) to the shopper.

As mentioned, Commerce provides an Upgrade endpoint that is used in the operation to upgrade the asset. This endpoint is part of the Services SSE which works to complete multiple service operations (already mentioned in the above sections) via the Services API. For this operation, the Services API has an endpoint called Upgrade. The following information provides more detail on what is required by the API to upgrade an asset using this endpoint:

- SSE name: Services
- Endpoint name: `Upgrade`
- Endpoint trigger: The endpoint is triggered when the user clicks **Upgrade** against an upgrade option
- Inputs:
  - Logged in User Token
  - AssetID
  - upgradeName (returned from Oracle Configure, Price, Quote)
- Returns: Upgrade URL Query String. This is the string of data that is appended to the base Modify URL to ensure that the upgrade iFrame is correctly pre-populated based on the product that the shopper is upgrading from and the product that they are upgrading to.

The activity that occurs at the store user interface level during the Upgrade operation is the following:

- Select the Asset List view. This lets you view information about all of your assets/ services. This view will also show the upgrade options (if available) for the asset. You cannot trigger an Upgrade operation from this view as the actual upgrade URL is not yet determined until the asset details are retrieved.
- Select the asset you wish to view and click the Details button so that you can view the asset details and as well as possible upgrade details.

- When you click the Details button of any asset, the asset details page is displayed which shows all of the details associated with that asset along with available asset action options.
- The asset details page also has a section showing the upgrade options available for that asset. When you display the asset details page, the product details of that SKU/Product are displayed. The Upgrade button is displayed next to any upgrade available for that asset.
- Click the Upgrade button in the Asset view of the asset that you want upgraded. Your upgrade option details are then displayed in the Asset Details view. You can also get the same results by clicking the link for the available upgrade from the Service list.
- Click Upgrade. When the Upgrade operation is initiated, the following occurs:
  - If there are upgrades available for the asset, the SSE endpoint returns an Upgrade URL Query String and creates the upgrade punch-in URL to load the iFrame containing the information about the available upgrades.
  - When you select to upgrade you are finally presented with a pre-configured modification to your asset bundle.
  - If the SSE endpoint returns an error, this means there are no upgrades available for this asset and an appropriate error message is displayed.
- Add the upgrade to your cart and submit the order to complete the upgrading process.

Finally, each of the Asset Based Ordering services operations described earlier may be carried out by a shopper or by an agent acting on the shopper’s behalf.

Additional information related to using the Upgrade feature with Commerce

The following additional details should be kept in mind when using the Upgrade feature:

- In Commerce you can start a configuration upgrade from a configurable SKU (for example, "4ForU Deal") which in turn maps to a model "nPlay" in Oracle Configure, Price, Quote.
- Configuration metadata is set with key "offer" and value "4ForU Deal" for the above SKU and is passed to Oracle Configure, Price, Quote
- After the configuration upgrade is completed, the BOM returned from the Oracle Configure, Price, Quote for that configuration may have a different rootSKU (i.e., "nPlay") and that is what is added to cart. "4ForUDeal" may be a child of "nPlay".
- In Commerce, there is another SKU for "nPlay" that is configurable and maps to the same model "nPlay" in Oracle Configure, Price, Quote.
- After the order is submitted, an asset with "nPlay" is created which has an asset attribute of `Offer`. `Offer` then has a value of `4ForUDeal`.

Also, via the CommerceAdmin, you can create products with an `upgradeProductId` as the `productId` value, and mark them as '`notForIndividualSale`.' This lets you do the following:

- Have a unique name for each upgrade that can be displayed in the store
- Have a unique description to describe what the upgrade is
- Support locale specific translations

- Have the ability to upload images related to the `upgradeOption`.

Handle further upgrades to an asset that has already been upgraded

In some use cases, you may have a situation where you have an asset with `currentOffer=sku1234` that is being upgraded to **Upgrade 101**. When you then visit the **Asset Details** page again you are presented with the same upgrade option of **Upgrade101.** This can occur because the upgrade does not modify the `currentOffer` and it is still `sku1234` and its corresponding upgrade options are being fetched during the getAssets/getAsset flow.

The following details show how you can solve this type of situation:

Table 2-4 Example of how to handle further upgrades to an asset

currentOffer	currentModel	upgradeName	upgradeProductID
4ForUDeal	nPlay	4ForUDealPlus	4ForUDealPlus

- Let's say a shopper starts an upgrade configuration from the SKU "4ForUDeal" by passing the configuration metadata `offer=4ForUDeal`.
- After upgrading the configuration, the BOM sent from Oracle Configure, Price, Quote may have a different root SKU id such as "nPlay." "4ForUDeal" may be a child of it. It will also contain an attribute "offer" with value "4ForUDeal"
- An asset with "nPlay" as the currentModel gets created and the getAssets/getAsset flows return the asset details along with asset attribute `offer=4ForUDeal.`
- The offer attribute is sent as the currentOffer to the Oracle Configure, Price, Quote while retrieving the upgrade option 4ForUDealPlus.
- Once the upgrade has been performed by passing the upgrade name 4ForUDealPlus to Oracle Configure, Price, Quote in the queryString, the BOM returned from Oracle Configure, Price, Quote will have the attribute "offer" with value "4ForUDealPlus".
- After submitting the order and updating the asset, the asset attribute "offer" value now gets updated to "4ForUDealPlus".
- In subsequent getAssets and getAsset calls the asset attribute offer value will be returned as "4ForUDealPlus", so that there are no matching records for that currentOffer in upgrade options table in Oracle Configure, Price, Quote.

Understand the Disable Reconfiguration feature

Regarding these operations, the Oracle Commerce and Oracle Configure, Price, Quote integration also has the ability to prevent shoppers from attempting to reconfigure items in their cart that have been added by any of the following operations:

- Renew
- Terminate
- Suspend
- Resume

To assist in disabling reconfiguration on already configured items added by any of these actions, an action code of Renew, Terminate, Suspend, and Resume is assigned to an item when that respective operation has been applied to that item.

This code is assigned to make sure that shoppers are prevented from attempting to reconfigure an asset. The purpose of the code is to make sure the reconfigure session(s) fails, either at reconfiguration or order validation time.

Differentiate between new order items and ABO order items

To identify items in an order that are the result of an operation on an existing asset (Terminate, Renew, Suspend, Resume, Modify, Upgrade), Commerce has checked to see if there was an `assetId` value. If there was, Commerce assumed that the item is the result of an ABO and not a net new purchase. This approach worked on the assumption that an asset identifier would only be assigned when the asset record was created in Oracle Configure, Price, Quote.

Asset identifier values are now assigned at the time when a shopper adds an item to the cart. To ensure that Commerce can always reliably differentiate between new order items and ABO order items when an ABO item is added to the cart, a `lineType` property for each item in the configuration hierarchy is set to ASSET.

The rule used to differentiate between new order items and ABO order items is the following: If `assetKey` value is present and `_lineType = NULL` then the item is a new purchase and not an operation on an existing asset.

Retrieve assets for an order with an asset key

For the cancel in-flight feature, Commerceneeds a mechanism for retrieving all of the assets derived from a particular order. Commerce used to retrieve the assets for a particular order based on `assetID` (stored on the asset record in Oracle Configure, Price, Quote). Commerce now uses the `assetKey` value.

For any given order Commerce queries the Oracle Configure, Price, Quote assets API to retrieve the assets for the order based on the collection of `assetKey` values. This query is limited to the `assetKey` values for the root items in the order only

Understand restricting the quantity of assetable items

A shopper used to be able to increase the item quantity for a configured item in the cart in the same way as any other purchase. This action does not work where an asset key value has been assigned.

Asset keys are assigned to net new purchases as part of the configuration process. Oracle Configure, Price, Quote assigns an `assetKey` for the root and all child items in the configuration. If an item has been assigned an asset key then this asset key is used to identity a single instance of this asset throughout the fulfillment, provisioning and asset management processes. As a result, the quantity of an item cannot be greater than one.

Customize configurations in Commerce using the CPQ Configuration API

You can customize the configurations of complex products in Oracle Commerce by using the Oracle Configure, Price, Quote Configuration API to avoid being redirected to a Oracle Configure, Price, Quote hosted iFrame.

You can now customize the configurations of complex products in Oracle Commerce without being redirected to a Oracle Configure, Price, Quote hosted iFrame.

You can now customize the configurations of complex products in Commerce without being redirected to a Oracle Configure, Price, Quote hosted iFrame which may have a separate and distinct user interface look and feel that creates a disjointed user experience. This capability, known as the Direct API Configuration feature, is provided to build out support in Commerce for API driven product configurations where the user interface experience is controlled by Commerce and can be customized by Commerce partners. At a high level, this feature lets you do the following:

- Create brand specific configuration user interfaces and controls at the global level.
- Create a specific user interface experience for individual customizable products at the product level.

The goal of this feature is to provide full support of the Oracle Configure, Price, Quote Configuration API on Commerce Storefront frameworks. This includes providing a mechanism to dynamically create user interface elements that let shoppers select customizable products. It then presents them with the appropriate user interface elements to complete the customization process and add the each item to the cart. These user interface elements are generated dynamically in response to the selections made by the shopper at each step of the customization. The functionality of this feature is fully compliant with current Commerce Storefront frameworks.

The principal benefits of the Direct API Configuration feature are the following:

- iFrame is not required - The current functionality requires that the configuration system (Oracle Configure, Price, Quote) perform all of the configuration tasks. This means that the shopper's user interface experience is managed in 2 separate applications. Up to the point where the shopper selects a customizable product, their user interface experience is driven by Commerce. On the other hand, the configuration user experience is managed by Oracle Configure, Price, Quote and when the shopper adds the configured item to the cart the user experience reverts back to the control of Commerce. The addition of this feature means that customers will not be required to execute product configuration via an iFrame. This lets shoppers experience a consistent user interface with common look and feel across their storefront.
- Decoupling of the user interface and the configuration process - This feature ensures that the user interface framework is decoupled from the configuration process. This lets customers do the following:
  - Build configuration user interface components using the Commerce Design Page based on the Store Front 1.0 Framework.
  - Build configuration user interface components using a non-Commerce design user interface framework.
- Performance improvements - The use of the iFrame pattern also creates a performance concern. The former integration with Oracle Configure, Price, Quote functions well and the disjointed user experience can be managed to some extent with user interface customization. However, there is also no reliable evidence that this design pattern performs at the levels required for high volume customer-based Telco implementations, where hundreds of thousands of shoppers may be configuring complex Telco bundles at the same time. This feature attempts to address this concern.

The roles that Commerce and Oracle Configure, Price, Quote now take with this feature are the following:

- Oracle Configure, Price, Quote remains the primary configurator and controls the following:
  - What needs to be configured

  - The sequence in which components/attributes are presented
  - The configuration values that are required or accepted
- The Commerce client is responsible for how the configurator is displayed (without an iFrame).

Additional topics in the current chapter provide you with detailed use cases for this feature.

Understand the support of the Oracle Configure, Price, Quote Configuration APIs

This feature provides a downloadable extension to the Commerce application component that provides a collection of endpoints which lets the Storefront UI (regardless of which user interface framework you are using) do the following:

- Retrieve the end to end UI flow for a given Oracle Configure, Price, Quote Configuration Model
- Retrieve sufficient metadata to identify the user interface elements required for each attribute of the model. These elements include the following:
  - Input Controls (Radio Buttons, List Boxes, Toggles, Date/Time Pickers etc.)
  - Navigational Components (Breadcrumbs, Sliders, Image Carousels etc.)
  - Information Components (Progress Bars, Tool Tips etc.)
  - Containers such as accordion elements
- Retrieve data required to correctly populate each user interface element. This includes Label Names, Selectable Options, and more
- Create a product configuration
- Update a product configuration
- Update user interface flow
- Update a user interface elements
- Modify a product configuration
- Upgrade a product configuration
- Save a product configuration
- Transform a BOM (bill of materials) to a Commerce cart item
- Reconfigure a saved product configuration

This extension also handles the following management tasks:

- Maintains the state of the configuration until such time as it is saved.
- Makes sure that calls made from the user interface framework to the Commerce Extension are authorized.
- Makes sure that calls made from the Commerce Extension to Oracle Configure, Price, Quote Configuration REST APIs are authorized.
- Ensures that connections are made from the user interface framework to extension to Oracle Configure, Price, Quote REST APIs without OAIC (Oracle Integration) integration flows.

- Manages BOM (Bill of Materials) data objects returned from Oracle Configure, Price, Quote when the configuration is saved.

This Commerce Extension supports any user interface client, including those built on Commerce Storefront 1.0.

Understand supported integration-specific configuration APIs

The Oracle Configure, Price, Quote (Configure, Price and Quote) Cloud solution supports the complete quote-to-cash process from customer inquiry to order fulfillment. It guides users to optimal product options and configurations from simple to complex, automatically applying discounts and relevant up-sell and cross-sell opportunities. Oracle Configure, Price, Quote exposes objects and data through REST APIs. By exposing objects and data through REST APIs, Oracle Configure, Price, Quote promotes simpler API calls and more robust integration using HTTP standards. For the Direct API Configuration feature and current Oracle Configure, Price, Quote Integration support, the following configuration APIs are mostly used:

- Configuration Run-Time Data Services APIs - These endpoints expose information and perform an action for a configuration model. All Configuration Run-Time Data REST APIs follow a required product hierarchy starting with the product family then product line followed by the model. A variable name for the product entity is required. For example, `/config{prodFamVarName}.(prodLineVarName}.{modelVarName}/` is the standard Configuration Run-Time Data product path for an endpoint URL.
- Configuration Administration REST APIs - These APIs provide product configuration endpoints that expose configuration definition information for Configuration Product Families, Product Lines, Models, attributes, array sets, menu items, and translations. The information for these items is organized in a hierarchical structure. The Configuration Administration REST API query parameters are supported to include and exclude child resources in a given resource. The response for each level in the hierarchy can include the details of the sub resources based on the query parameter passed in the request.

Customer Configuration flows dictate how users go through the site pages and the options available as they create a Transaction. Configuration flow rules consist of a condition and flow attributes. Actions display based on which node in the flow that the user has available based on defined criteria. Beginning in Oracle Configure, Price, Quote Release 18D,Oracle Configure, Price, Quote transformed the current configuration definition as REST endpoints to support UI interfaces. These services are available v7 and higher RESTful services.

Refer to the Oracle Configure, Price, Quote REST API documentation for more complete information.

Understand how the Direct API Configuration feature enhances Asset Operations

As mentioned, this feature provides Commerce with support of the Oracle Configure, Price, Quote Configuration API Layer while using the Commerce and Oracle Configure, Price, Quote integration. This means providing functionality that lets customers, using any user interface framework, configure and/or reconfigure customizable products by invoking the following from Oracle Configure, Price, Quote:

- Configuration Run-Time Data Services APIs
- Configuration Administration REST APIs

Building on this foundation, the feature further supports some asset-based operations whereby the configuration model retrieved from Oracle CPQ represents an existing asset. This lets the shopper execute the following configuration-related Asset Operations via direct API calls to the Oracle Configure, Price, Quote Configuration API:

- Modify
- Upgrade

Available Storefront and Agent endpoints for this feature let you modify and upgrade assets via direct API calls to Oracle Configure, Price, Quote thus removing the need to include an iFrame in this part of the shopping experience as well. This feature is limited to API only and customers will need to build their own UI elements to invoke these new endpoints.

By creating your own Modify and Upgrade user interface elements, you can deliver a seamless and consistent user experience even when modifying or upgrading complex products or services. The shopper user interface experience while modifying or upgrading a service can then be consistent with the rest of the site navigation experience as configuration user interface controls can be created in compliance with the Site Theme and CSS being used.

To fully implement the Asset Operations portion of this feature you must:

- Download and install the CpqConfiguratorStoreApp and CpqConfiguratorAgentApp SSEs
- Create a ‘Modify’ user interface element which can be coded into the Asset Details widget (which is not elementized)
- Create an ‘Upgrade’ user interface element which can also be coded into the Asset Details widget

The creation of the user interface elements should be a straightforward process for any developer partner with a working knowledge of Commerce development and `knockout.js`.

Refer to Use Asset Based Ordering for more information on these Asset Operations.

Understand Sys Config model support via Commerce and the Oracle Configure, Price, Quote Configuration API

In Oracle Configure, Price, Quote, certain parts of customizable (configurable) products are based on "Sys Config" models that are accessible via the Oracle Configure, Price, Quote Configuration API. The "Sys Config" model consists of a hierarchy of components and associated classes that are used to model the hierarchical nature of the Product and Promotion structure of that configurable product.

When products in Oracle Configure, Price, Quote are structured hierarchically, Product Families are created first. Families provide the broad classifications of products. The next parts created are Product Lines which are used to describe more specific product areas of Product Families. Finally, Models are created to provide detail about the most specific product traits.

Note: In a "Sys Config" model, an attribute of a model can also be another model so it is important that you fully understand the structured hierarchy of each product family.

Examples of the product hierarchies just described might look something like the following:

- Product Family: "Business Laptop"
  – Product Line: “EZCompute”
    * Model: "EZ"
    * Model: "EZ Pro"

- Product Family: ""Gamer Laptop""
  - Product Line: "Avenger"
    - Model: "Novice"
    - Model: "EZ Pro"

Note: In a "Sys Config" model, an attribute of a model can also be another model so it is important that you fully understand the structured hierarchy of each configured product family. For example, the "Novice" model in the "Gamer Laptop" product family could have its own "sub-model" that had a variation of the features (more memory, better graphics card, and so on) offered in the basic configuration of the parent "Novice" model. To summarize, this feature lets you reload the configurator with a model which can be an attribute of the root/parent.

For more complete information on models and Oracle Configure, Price, Quote REST APIs, refer to the Oracle Configure, Price, Quote documentation.

In the Commerce and Oracle Configure, Price, Quote Integration, Commerce works with the Oracle Configure, Price, Quote Configuration API to let you execute the configuration of complex "Sys Config" models via API calls to the Oracle Configure, Price, Quote Configuration API. The Commerce support of the Oracle Configure, Price, Quote Configuration API lets you open and customize desired models within a bundle configuration. An example of this might be a product bundle consisting of a Mobile service attribute as well as a Cable TV service attribute. In this example, each service attribute (Mobile and Cable TV) is its own model. Commerce support of the Oracle Configure, Price, Quote Configuration API lets you open a product configurator directly on either of those service models.

Note: Keep in mind that a shopper can only interact with a model starting from the root asset of the configured product. Every Configure, Reconfigure, Modify, or Upgrade operation is an operation carried out on the root asset. Having retrieved the root asset (the complete product model), the shopper may then navigate to any attributes of the root. In some cases, an attribute may well be an attribute that is a sub-model.

As far as user cases go, this feature lets you (the developer) build out specific user interface experiences dealing with the configuration of customizable products from a desired catalog. In doing so, it lets you apply global, site, or even product-specific user interface template changes as well as control the user interface flow of the configuration process for each product. For customers, this feature lets them enjoy a seamless product customization experience without any indication that multiple applications are working together as part of an integration to handle the product configuration.

Implement configuration customization via the CPQ Configuration API.

You need to complete some initial tasks to implement the functionality that directly customizes configurations using the Oracle Configure, Price, Quote Configuration API for the first time on a customer storefront.

The Direct API Configuration feature lets you directly customize configurations using the Oracle Configure, Price, Quote Configuration API. This topic describes the tasks which a developer and designer would work together to implement this functionality for the first time for a customer storefront.

This feature lets you directly customize configurations using the Oracle Configure, Price, Quote Configuration API. This topic describes the tasks which a developer and designer would work together to implement this functionality for the first time for a customer storefront. This would be the set of tasks that would be carried out first to allow you to use the feature.

In this case, the customer does not want to use the hosted iFrame model for executing product customization on their site but would prefer customization via the Direct API Configuration feature. The specific reasons the customer is requesting the implementation of this feature are the following:

- The customer wants the customization user experience to be as seamless as possible.
- The customer wants their merchandising team to have as much control over the customization user interface "look and feel" as possible.
- The customer would prefer that the merchandising team manage the user interface experience in their design tools as much as possible.

The details for implementing and using the Oracle Configure, Price, Quote Configuration API feature are described in the sections that follow. In these descriptions, it is assumed that the Commerce and Oracle Configure, Price, Quote Integration is already configured and enabled.

Understand the role of the Commerce Configurator SSE in the Direct API Configuration feature

The Direct API Configuration feature uses a Commerce server-side extension (SSE) to provide a collection of endpoints which lets the storefront UI (regardless of the UI framework used) to configure products and services. The SSE accepts a configurator request, invokes the corresponding requests in Oracle Configure, Price, Quote, and processes the Oracle Configure, Price, Quote response before returning an optimized payload.

The SSE performs the following configurator actions:

- Configure - This action corresponds to the Oracle Configure, Price, Quote `_configure` endpoint and is the starting point for configuring a model. It returns all the necessary layout data, attribute, and configuration state data for a user interface to display a configurator model. Also, where a layout contains Pick Lists and/or Array Sets, it returns all data required for those components to be rendered.
- Update - This action corresponds to the Oracle Configure, Price, Quote `_update` endpoint. It will accept an updated configuration state from the client and return an new configuration state based on the changes made.
- Next - This action corresponds to the Oracle Configure, Price, Quote `_next` endpoint. This action is available when the model configuration is spanned across multiple nodes/configuration flow layouts. It works similarly to the initial configure action as it also returns all the necessary layout data, attributes, configuration state, and pick list/array set data to display the particular layout for a stage in the flow.
- Previous - This action corresponds to the Oracle Configure, Price, Quote `_next` endpoint. This action is available when the model configuration is spanned across multiple nodes/configuration flow layouts. It works similarly to the initial configure action as it also returns all the necessary layout data, attributes, configuration state, and pick list/array set data to display the particular layout for a stage in the flow.
- Add to Cart - This action corresponds to the Oracle Configure, Price, Quote`_integration_addToCart` endpoint. This action returns a Commerce commerce item (cart item). It transforms a `Configuration_Details` response

(returned from Oracle Configure, Price, Quote) to a Commerce commerce item (cart item). With the embedded configurator, approach the `Configuration_Details` response is returned to Commerce and it is the responsibility of the Commerce client to transform the response to a Commerce commerce item.

- Reconfigure - This action corresponds to the Oracle Configure, Price, Quote `_reconfigureClient` endpoint. It is similar to the Configure action but rather than starting a brand new configuration, it returns all the necessary layout data, attributes, configuration state, and pick list/array set data for a user interface to display a configurator model for an existing configuration. A `configId` parameter is used to identify the existing configuration.
- Interact - This action corresponds to the Oracle Configure, Price, Quote `_interact` endpoint. It is typically triggered by the user interface in response to a change to an attribute value when `ajaxEnabled` has been set to true for the user interface component. It takes the value for the attribute that has changed and returns a new configuration state based on the change made.
- Array Set - The action supports the following:
  - Add Row - This action corresponds to the Oracle Configure, Price, Quote `_set<arraySetVarName>/actions/_add` endpoint. It accepts a `cacheInstanceId` and adds a row to the `arraySet`.
  - Delete Row - This action corresponds to the Oracle Configure, Price, Quote `_set<arraySetVarName>/actions/_delete` endpoint. It accepts a `cacheInstanceId` and `removeIndex` and removes the row from the supplied index in the `arraySet`.
- Layout - This action retrieves the full `layoutCache` for a particular product and flow.
- Pick Lists - This action retrieves all options available for a particular pick list.
- UI Settings - This action retrieves all general/base user interface configuration settings from Oracle Configure, Price, Quote.
- Templates - The action retrieves configuration templates that are to be used for rendering a BOM table and a recommended parts table.

Implement the Direct API Configuration feature

To implement the Direct API Configuration feature in Commerce you must:

- Download and install the Oracle Configure, Price, Quote Configurator (Storefront/Agent) Server-Side Extension in Commerce.
- Create a "Customize Button for Direct API" user interface element for direct API configuration.
- Create a "Reconfigure Button for Direct API" user interface element for direct API reconfiguration."
- Create a JavaScript Library of user interface components that will be used to render the Layout response from Oracle Configure, Price, Quote (this could be Knockout Components, React, Commerce elements etc.).
- Include the "Customize Button for Direct API" element (button) in the Product Details widget in order to trigger a customization session.
- Include the "Reconfigure Button for Direct API" element (button) in the Shopping Cart widget in order to trigger a reconfigure session.
- Bundle the user interface elements and JavaScript library into a single extension that can be uploaded in a single step.

- Log in to Commerce Admin and navigate to **Settings** → **Extensions**.
- Upload the Oracle Configure, Price, Quote Configurator server-side extension.
- Upload the new extension containing the user interface elements and JavaScript library.

Implement the Direct API Configuration feature for Configure

If you decide to implement the Direct API Configuration feature for Configure do the following:

- Log in to Commerce Admin and navigate to **Design** → **Layout** → **Product Layout** → **Layout Settings**.
- Select **Product Layout** → **Grid View** and then select the **Product Details** widget.
- Select the **Element Library**. You should see three "Customize Button" user elements available. These include the following:
  - Customize Button - Supports the iFrame customization flow by using the iFrame URL stored in Commerce Admin and appending values for Product Line, Product Family and Model to load the iFrame and kick off the configuration process.
  - Customize Button with Configuration Metadata - Supports the iFrame customization flow by using the iFrame URL stored in Commerce Admin and appending values for Product Line, Product Family, Model and a collection of one or more static or dynamic key value pairs of configuration metadata to load the iFrame in the correct state and kick off the configuration process.
  - Customize Button for Direct API - Supports the API driven customization flow. **Note:** You created this as directed in the previous section as the "Customize Button for Direct API" element.
- Add the **Customize Button for Direct API** to the Product Details widget.
- Save your changes.
- Navigate back to **Layout** → **Product Layout** → **Layout Settings**.
- Set the Layout Preview Product ID for 4ForU Deal offer. This is a configurable product that lets you buy services for Landline, Mobile, Internet and TV in a single bundle at a steep discount.
- Save your changes. Select **Product Layout** → **Preview**. You are presented with a preview of the product layout for the 4ForU Deal offer.
- Select to customize the offer. You are presented with the customizable options for the offer in a combination of user interface components including the following:
  - Panels
  - Tabs
  - Input fields
  - Radio buttons
  - Checkboxes
  - Multi-select lists
  - Single select lists
  - Date pickers

  - Pick Lists

  These components are presented as the default mapping for the corresponding Oracle Configure, Price, Quote model attributes and layout.
- Publish your changes.

Implement the Direct API Configuration feature for Reconfigure

If you decide to implement the Direct API Configuration feature for Reconfigure do the following:

- Select **Layout** → **Cart Layout** → **Grid View** and select the **Shopping Cart** widget.
- Select **Go to widget code**.
- Add the **Reconfigure Button for Direct API** to the Shopping Cart widget.
  Note: You created this as directed in the earlier section as the "Reconfigure Button for Direct API" element.
- Save your changes.
- Navigate back to **Layout** → **Cart Layout** → **Layout Settings**.
- Set the Layout Preview Product ID for the 4ForU Deal offer with a quantity of 1. This is a configurable product which lets you buy services for Landline, Mobile, Internet and TV in a single bundle at a steep discount.
- Save your changes.
- Select **Product Layout** → **Preview**. You are presented with a preview of the product layout for the 4ForU Deal offer.
- Select to customize the offer and add it to the cart.
- Select the cart and choose to edit the configurable item.
- You are presented with customizable options for the offer in a combination of user interface components including the following:
  - Panels
  - Tabs
  - Input fields
  - Radio buttons
  - Checkboxes
  - Multi-select lists
  - Single select lists
  - Date pickers
  - Pick Lists

  These components are presented as per the default mapping for the corresponding Oracle Configure, Price, Quote model attributes and layout.
- Publish your changes.

Commerce is now configured to use the direct API configuration process for customizable products.

Control user interface look and feel using the CPQ Configuration API

The Direct API Configuration feature lets you control user interface "look and feel" using the Oracle Configure, Price, Quote Configuration API.

You can use the Direct API Configuration feature to control user interface "look and feel" using the Oracle Configure, Price, Quote Configuration API. This ability lets you do things like the following:

- Apply a site-specific "Look and Feel" product customization to the user interface experience.
- Apply site-specific user interface components for a custom user interface experience.
- Add a new UI component to the configuration flow.
- Remove tabs from the product customization user interface experience.
- Apply a product type specific set of user interface components to the configuration flow.

Before you can accomplish these tasks, you must first make sure that the API driven configuration feature has been implemented (described in the previous topic). Also, it is assumed that the Commerce and Oracle Configure, Price, Quote Integration has already been configured and enabled.

In the sections that follow, you are provided with details for using this feature to carry out these customization tasks.

Apply a site-specific "Look and Feel" product customization to the user interface experience

Consider this situation. Say a customer wants a new custom user interface look and feel for their site. The customer's in-house design and brand management team have provided specifications as to:

- Color Schemes
- Style Header and Footer
- Navigation
- Buttons, input fields, check boxes, Multi-select Lists, single select Lists, date pickers, pick lists
- Component Sizes
- Component Styles
- Component Colors
- Component Fonts

You are instructed to change the user site interface look and feel so that it reflects the customer product customization changes. This is done by completing the following tasks:

- Refer to the Customizing your store layouts section on the Oracle Help Center. You can see that it is possible to apply the required user in look and feel by cloning and customizing a Commerce theme.

Note: The included version of the JavaScript Library of Knockout UI Components used to render the Layout response from Oracle Configure, Price, Quote uses OOTB theme/ styles, (i.e., Bootstrap Forms and Components). Also, by making changes at the provided Theme level, you can change the look and feel of the configuration UI experience without making any changes directly to the UI elements or JS Library.

- Clone the provided the theme and apply the required specifications for:
  - Backgrounds
  - Buttons
  - Navigation Menu
  - Menu
  - Typography

  This is done directly in the Design page.
- Use the Design page to access the theme's CSS and apply all of the remaining UI specifications.
- Save all your changes.
- Navigate to **Layout** → **Product Layout** → **Layout Settings**.
- Set the Layout Preview Product ID for 4ForU Deal offer, this is a configurable product which allows shoppers to buy services for Landline, Mobile, Internet and TV in a single bundle at a steep discount.
- Save your changes.
- Select **Product Layout** → **Preview**. You are presented with a preview of the product layout for the 4ForU Deal offer.
- The system displays the configurable options available in a combination of UI components such as the following:
  - Panels
  - Tabs
  - Input fields
  - Radio buttons
  - Checkboxes
  - Multi -select lists
  - Single select lists
  - Date pickers
  - Pick Lists

You can now see that all of the user interface components are displayed in accordance with the new theme that you have created and are in accordance with the rest of the site.

Apply site-specific user interface components for a custom user interface experience

A case may arise where a customer wants the customization user interface experience to be slightly different from the rest of the site to convey the feeling of personal design when they are building their tailored product.

The customer's in-house design and brand management team has provided specifications to make changes to the following user interface elements:

- Buttons - Primary Buttons should contain an icon
- Input Fields - Should all have labels
- Check boxes - Should be rendered as sliders
- Multi-select lists - Should be displayed as a collection of check boxes
- Single select lists - Should be displayed as drop down lists
- Date pickers - Should be displayed as Tumbler Scrolls
- Color pick list - Should be displayed as a swatch matrix with a tone slider

As a member of the SI user interface design team, you are instructed to implement the new product customization user interface look and feel. You see that in order to change how the Oracle Configure, Price, Quote model user interface components are rendered inCommerce, changes must be made to the JavaScript Library of Knockout user interface components used to render the Layout response from Oracle Configure, Price, Quote. This JavaScript Library is part of the Oracle Configure, Price, Quote Configurator user interface extension which was uploaded at feature implementation time.

To implement the new product customization user interface look and feel, complete the following tasks:

- Log in to Commerce Admin and navigate to **Settings** → **Extensions**.
- Deactivate the Oracle Configure, Price, Quote Configurator user interface extension.
- Delete the Oracle Configure, Price, Quote Configurator user interface extension. This extension includes the Direct API versions of the Configure and Reconfigure user interface elements as well as a common JavaScript Library that defines the mapping of Oracle Configure, Price, Quote user interface components to Commerce Knockout Components.
- Create new versions of the following:
  - Configure element (if you want the button to appear differently or launch the configuration in a new widget)
  - Reconfigure element (if you want the button to appear differently or launch the configuration in a new widget)
  - JavaScript Library (In the JavaScript library for each component that is to be rendered differently modify the HTML, JavaScript and define new styles which must also be added to the global stylesheet).
- Bundle the user interface elements and JavaScript library into a single extension that can be uploaded in a single step.
- Navigate to **Settings** → **Extensions** and upload the new version of Oracle Configure, Price, Quote Configurator user interface extension.
- Reapply the "Customize via direct API" for Configure by doing the following:
  - Navigate to **Design** → **Layout** → **Product Layout** → **Layout Settings**.
  - Select **Product Layout** → **Grid View** and select the Product Details widget.
  - Select the Element Library.
  - Add the Customize Button for Direct API to the Product Details Widget.
  - Save your changes.

  - Navigate back to **Layout** → **Product Layout** → **Layout Settings**.
  - Set the Layout Preview Product ID for 4ForU Deal offer. This is a configurable product which allows shoppers to buy services for Landline, Mobile, Internet, and TV in a single bundle at a steep discount.
  - Save your changes.
  - Select **Product Layout** → **Preview**. You are presented with a preview of the product layout for the 4ForU Deal offer.
  - Select to customize the offer. You are presented with the customizable options for the offer in a combination of user interface components including the following. These are presented as per the new Knockout user interface components.
    - Buttons
    - Input Fields
    - Checkboxes
    - Multi-select lists
    - Single select list
    - Date pickers
    - Color pick list
- Add the customized offer to the cart.
- Select the cart and chooses to edit the configure item. You are presented with the customizable options for the offer in a combination of user interface components. These are presented as per the new Knockout user interface components. These include the following:
  - Buttons
  - Input Fields
  - Checkboxes
  - Multi-select lists
  - Single select list
  - Date pickers
  - Color pick list
- Publish your changes.

Upon completing these tasks, you will see that the product customization user interface look and feel and components are now distinct from the store design theme and in accordance with the customer's specifications.

Add a new user interface component to the configuration user interface flow

Sometimes a customer may want new to add a new user interface component that shoppers will use to select an image that will be imprinted on the shopper's mobile phone case.

In this example, the customer's in-house design and brand management team have developed a new "Image Carousel" user interface component that shoppers will use to select the image to be imprinted. This new user interface component is used as the user interface control for Oracle Configure, Price, Quote model attributes which require the shopper to select an image.

As a member of the SI user interface design team, you are instructed to ensure that this new user interface component is displayed correctly in Commerce. To add the new user interface component to the configuration user interface experience via direct API, complete the following tasks:

- Log in to Commerce Admin and navigate to **Settings** → **Extensions** and do the following:
  - Deactivate the Oracle Configure, Price, Quote Configurator user interface extension.
  - Delete the Oracle Configure, Price, Quote Configurator user interface extension. This extension includes the direct API versions of the Configure and Reconfigure user interface elements as well as a common JavaScript Library of user interface Components used to render the Layout response from Oracle Configure, Price, Quote.
- Create new versions of the JavaScript Library to include the new 'Image Carousel' user interface component, including HTML, JavaScript and Style Definitions which must also be added to the global stylesheet.
- Bundle the user interface elements and new JavaScript library into a single extension that can be uploaded in a single step.
- Navigate to **Settings** → **Extensions** and upload the edited version of the Oracle Configure, Price, Quote Configurator user interface extension.
- Reapply the "'Customize via direct API" for Configure.
- Preview the product layout and make sure that the new image carousel user interface component renders correctly when customizing a product.
- Preview product layout and make sure that the new image carousel user interface component renders correctly when reconfiguring a product.
- Publish your changes.

Upon completing these tasks, you should see that the product customization user interface now includes a new user interface component in accordance with the customer's specifications.

Remove tabs from the product customization user interface experience

In this case, the customer's in-house design and brand management team have requested that all tabs be removed from the product customization user interface as they have received feedback from customers that they are confusing.

As a member of the user interface design team, you are instructed to remove all tabs from the customization user interfaces. To remove all tabs, complete the following tasks:

- Login to Commerce Admin and navigate to **Settings** → **Extensions.**
- Deactivate the Oracle Configure, Price, Quote Configurator user interface extension.
- Delete the Oracle Configure, Price, Quote Configurator user interface extension. This extension includes the direct API versions of the Configure and Reconfigure user interface elements as well as the JavaScript Library of user interface Components.
- Edit the JavaScript Library to change how tabs are rendered (stacked, side by side etc.)

- Navigate to **Settings** → **Extensions** and upload the edited version of Oracle Configure, Price, Quote Configurator user interface Extension.
- Reapply the "Customize via direct API" for Configure Preview the product layout and make sure that there are no tabs displayed when customizing a product.
- Preview the product layout and make sure that there are no tabs displayed when reconfiguring a product.
- Publish you changes.

Upon completion of these tasks, you will note that the product customization user interface no longer displays any tabbed layout in accordance with the customer's specifications.

Apply a product type specific set of user interface components to the configuration flow

In this case the, the customer's in-house design and brand management team want the shopper's configuration experience to be different when they customize shippable goods (for example, "Build your own laptop") and when they customize services such as the Phones4All offer.

For this, a new set of "Service Configuration user interface Components" has been developed by the in-house design and brand management team for the following:

- Buttons
- Input Fields
- Checkboxes
- Multi-select lists
- Single select list
- Date pickers
- Color pick list

As a member of the user interface design team, you are instructed to ensure that when a shopper is customizing a service these new user interface components will be displayed. This is done by completing the following tasks:

- Log in to Commerce Admin.
- Navigate to **Settings** → **Extensions**.
- Deactivate the Oracle Configure, Price, Quote Configurator user interface extension.
- Delete the Oracle Configure, Price, Quote Configurator user interface Extension. This extension includes the direct API versions of the Configure and Reconfigure user interface elements as well as the common JavaScript Library.
- Edit the JavaScript Library by adding conditional `IF` statements that map the Oracle Configure, Price, Quote user interface components to the new "Service Configuration user interface Components" where `Product Type = Service`.
- Navigate to **Settings** → **Extensions** and upload the edited version of Oracle Configure, Price, Quote Configurator user interface extension.
- Reapply the "Customize via direct API" for Configure.
- Publish your changes.
- Create a new "Services Product Layout" for products where `Product Type = Service`.

- Create a new "Service Product Details Widget."
- Add the "Customize Button for direct API" user interface element to the Product Details Widget.
- Add the "Service Product Details Widget" to the "Services Product Layout."
- Save your changes.
- Navigate back to **Layout** → **Services Product Layout** → **Layout Settings**.
- Set the Layout Preview Product ID for 4ForU Deal offer.
- Select **Product Layout** → **Preview**. You are presented with a preview of the product layout for the 4ForU Deal offer.
- Select to customize the offer. You are presented with the customizable options for the offer in a combination of user interface components. This includes each of the new "Service Configuration user interface Components." This includes the following:
  - Buttons
  - Input Fields
  - Checkboxes
  - Multi-select lists
  - Single select list
  - Date pickers
  - Color pick list

  These are now presented correctly.
- Publish your changes.

Upon completing these tasks, the product customization user interface now displays the new product type specific user interface components in the configuration flow.

Customize and reconfigure a product by direct use of the CPQ Configuration API

You can customize and reconfigure a product by directly using of the Oracle Configure, Price, Quote Configuration API.

You can use the Direct API Configuration feature to customize a product by implementing and using the Oracle Configure, Price, Quote Configuration API. This feature give you the ability to do the following:

- Customize a product where the "Customize via direct API" feature has implemented in Commerce
- Reconfigure a product before checking out

Before you can accomplish these tasks, you must first make sure that the Direct API Configuration feature has been implemented (described in a previous topic of this section of the guide). Also, it is assumed that the Commerce and Oracle Configure, Price, Quote Integration is already configured and enabled. In the section that follows, you are provided with details for using the feature to carry out these specific customization tasks as just described.

Apply customizations to a product by directly using the Oracle Configure, Price, Quote Configuration API

The list of tasks that follow describe a situation where a shopper customizes a product where the Direct API Configuration feature has been implemented in Commerce.

In this case, a System Integration Partner has already implemented the feature and the SI user interface design team may have already done some user interface customizations by directly using the Oracle Configure, Price, Quote Configuration API.

For this example, it is assumed that the Commerce and Oracle Configure, Price, Quote Integration is already configured and enabled.

Use the following guidelines to accomplish the specified goals.

- As an example, let us say that the shopper has noticed a lot of web advertising by their cell phone service for their new Phones4All offer which allows them to buy a single deal with phones and plans for up to 6 people at huge savings on handsets, accessories and monthly bills.
- The shopper navigates to their cell phone service and selects the Phones4All offer. The shopper selects to customize the offer. The UI element **Customize Button for Direct API** invokes the `.../v1/configurations` SSE endpoint passing the following parameters:
  - `productFamily`
  - `productLine`
  - `model`
  - `locale`
  - `currency`
  - `configurationMetadata`
- The `.../v1/configurations` SSE endpoint triggers the following Oracle Configure, Price, Quote API endpoints:
  - `GET_configUISettings`
  - `GET_pageTemplates`
  - `POST_config`
  - `GET_Layout_ Cache`
- The .../v1/configurations SSE collates the data returned from Oracle Configure, Price, Quote, strips out all extraneous information, and returns a "combined configuration data response."
- The shopper is presented with a set of customization options that they can use to tailor the offer to their specific needs.
- The first option the shopper is presented with is the number of lines required.
- The shopper selects 4 lines.
- The shopper selects **Next**.
- The UI element **Customize Button for Direct API** invokes the `.../v1/configurations/{cacheInstanceId}/page` SSE endpoint (where `cacheInstanceId` represents the current

reconfiguration instance in Oracle Configure, Price, Quote) by passing the following parameters:

  - `productFamily`
  - `productLine`
  - `model`
  - `locale`
  - `currency`
  - `op: next`

- The `.../v1/configurations/{cacheInstanceId}/page` SSE endpoint triggers the following Oracle Configure, Price, Quote API endpoints:
  - `POST_next`
  - `GET_Layout_ Cache`
- The `.../v1/configurations/{cacheInstanceId}/page` SSE collates the data returned from Oracle Configure, Price, Quote, strips out all extraneous information, and returns a "combined configuration data response."
- The shopper is presented with the configuration options for Handset and Plan for Line 1 including:
  - Handset - including Capacity, Color, Tablet, and Watch
  - Plan - Silver or Gold
- The shopper selects the "Samsung S10" handset
- The UI element **Customize Button for Direct API** checks the `isUpdatable` property for the handset attribute.
- The `isUpdatable` property value is TRUE (this means that when an option is selected for this attribute, the configuration model must be updated as this selection impacts other model attributes).
- The UI element **Customize Button for Direct API** invokes the `.../v1/configurations/{cacheInstanceId}` SSE endpoint (where `cacheInstanceId` represents the current reconfiguration instance in Oracle Configure, Price, Quote) passing the following parameters:
  - `productFamily`
  - `productLine`
  - `model`
  - `locale`
- The `.../v1/configurations/{cacheInstanceId}` SSE endpoint triggers the `POST_update` Oracle Configure, Price, Quote API endpoint.
- The `.../v1/configurations/{cacheInstanceId}` SSE collates the data returned from Oracle Configure, Price, Quote, strips out all extraneous information and returns a "combined configuration data response."
- The shopper sees that some of the options that were previously available for capacity, color, table and watch have been updated and that they are now limited to those compatible with their selected Samsung S10 handset.

- The shopper selects the 256GB capacity option for the handset.
- The UI element **Customize Button for Direct API** checks the `isUpdatable` property for the handset attribute. The `isUpdatable` property value is FALSE (this means that when an option is selected for this attribute the configuration model need not be updated as this selection does not impact other model attributes).
- The shopper completes the customization for Line 1 and moves on to line 2.
- When the shopper is part way through the customization of Line 2, they decide that they may need to make a change to the handset capacity for Line 1.
- The shopper selects **Previous**.
- The UI element **Customize Button for Direct API** invokes the `.../v1/configurations/{cacheInstanceId}/page` SSE endpoint (where `cacheInstanceId` represents the current reconfiguration instance in Oracle Configure, Price, Quote) passing the following parameters:
  - `productFamily`
  - `productLine`
  - `model`
  - `locale`
  - `currency`
  - `op: previous`
- The `.../v1/configurations/{cacheInstanceId}/page` SSE endpoint triggers the `POST_previous` and `GET_Layout_ Cache` Oracle Configure, Price, Quote API endpoints.
- The `.../v1/configurations/{cacheInstanceId}/page` SSE collates the data returned from Oracle Configure, Price, Quote, strips out all extraneous information, and returns a "combined configuration data response."
- The shopper is presented with the configuration options that they selected for Line 1.
- The shopper changes the capacity for the Line 1 handset and continues to customize the rest of the lines.
- The shopper completes the customization of their Phones4All offer.
- The shopper selects **Add to Cart**.
- The UI element **Customize Button for Direct API** invokes the `.../v1/configurations/{cacheInstanceId}/add-to-cart` SSE endpoint (where `cacheInstanceId` represents the current reconfiguration instance in Oracle Configure, Price, Quote) passing the following parameters:
  - `productFamily`
  - `productLine`
  - `model`
- The `.../v1/configurations/{cacheInstanceId}/add-to-cart` SSE endpoint triggers the POST_integration_add_to_cart Oracle Configure, Price, Quote API endpoint.
- The `.../v1/configurations/{cacheInstanceId}/add-to-cart` SSE transforms the Oracle Configure, Price, Quote response to a Commerce cart item and adds the configured item to the Commerce cart.
- The shopper proceeds to checkout.

When all of this has completed, a multi-level configured item is added to Commerce cart.

Reconfigure a customized product before checking out

In this situation, a shopper decides to make a change to a customized product after adding it to the cart but before checking out.

Say, for example, the customer has customized their Phones4All offer and has added it to the cart. Before checking out, however, the shopper reviews their choices and realizes that by including the Apple Watch with Line 4 the offer is more than $200 over their budget. The following details illustrate what occurs if a typical shopper wishes to reconfigure an already customized product before checking out:

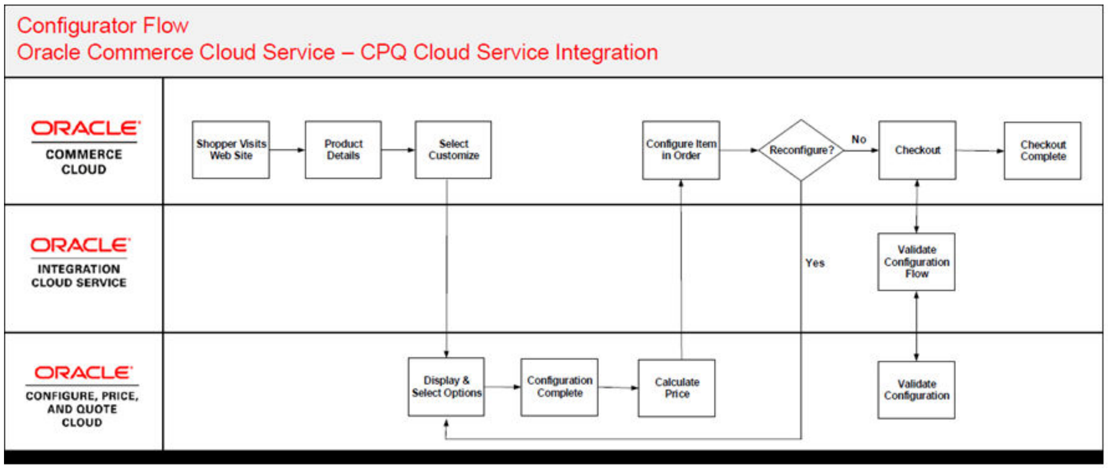
- The shopper selects to edit the Phones4All item in her cart.
- The user interface Shopping Cart widget with a Reconfigure Button for Direct API invokes the `.../v1/configurations` SSE endpoint passing the following parameters:
  - `productFamily`
  - `productLine`
  - `model`
  - `locale`
  - `currency`
  - `configId` (identifies the specific instance of configuration in Oracle Configure, Price, Quote which is to be reconfigured)
- The `.../v1/configurations` SSE endpoint triggers the following Oracle Configure, Price, Quote API endpoints:
  - `GET_configUISettings`
  - `GET_pageTemplates`
  - `POST_config`
  - `GET_Layout_ Cache`
- The `.../v1/configurations` SSE collates the data returned from Oracle Configure, Price, Quote, strips out all extraneous information, and returns a "combined configuration data response."
- The shopper is presented with all of the customization options and selections that they have made.
- The shopper navigates to Line 4 and removes the Apple Watch selection.
- The shopper selects to save and their cart is updated.

The Commerce cart is now updated with the newly reconfigured item.

Appendix A: Configurator Flow

A Configurator process flow occurs between Oracle Configure, Price, Quote and Commerce during the integration.

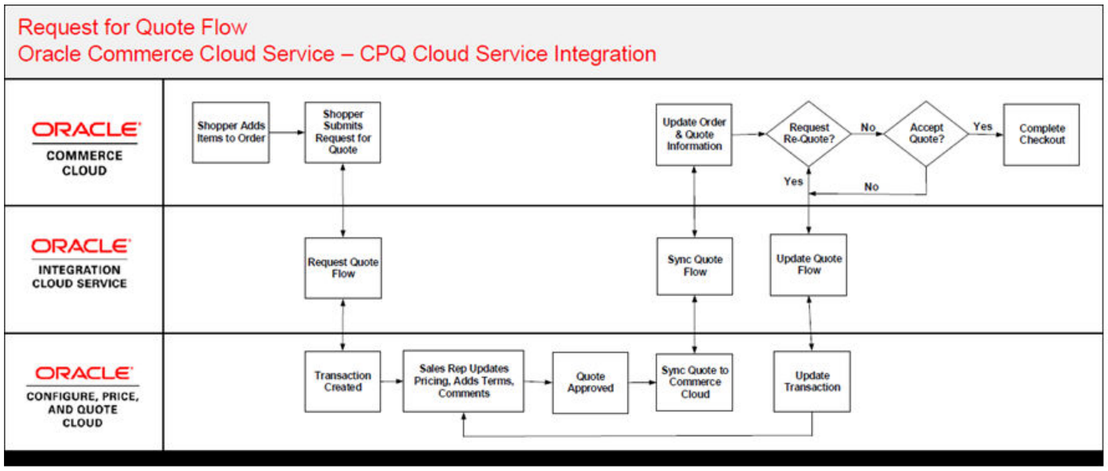
The following presents a diagram of the integration Configurator Flow:



Appendix B: Request for Quote Flow

A Request for Quote process flow occurs between Oracle Configure, Price, Quote and Commerce during the Quote integration.

The following presents a diagram of the Request for Quote integration flow between Commerce, OIC, and Oracle CPQ Cloud when using theOracle Commerce-Oracle CPQ Quote integration


Request for Quote Flow
Oracle Commerce Cloud Service – CPQ Cloud Service Integration
ORACLE COMMERCE CLOUD
ORACLE INTEGRATION CLOUD SERVICE
ORACLE CONFIGURE, PRICE, AND QUOTE CLOUD
Shopper Adds Items to Order
Shopper Submits Request for Quote
Update Order & Quote Information
Request Re-Quote?
No
Accept Quote?
Yes
Complete Checkout
Yes
No
Request Quote Flow
Sync Quote Flow
Update Quote Flow
Transaction Created
Sales Rep Updates Pricing, Adds Terms, Comments
Quote Approved
Sync Quote to Commerce Cloud
Update Transaction

3 Integrate with Oracle Content Management

Oracle Commerce provides an integration with Oracle Content Management that you can use to display content items such as blog posts and articles on your storefront.

Oracle Content Management is a cloud-based hub for managing your content and other digital assets. The integration enables you to create and manage a wide range of content, and make it available on your Commerce sites. Possible uses include:

- Creating a blog or article template so any content published is automatically available and is formatted correctly for access by shoppers and by search engines.
- Creating pages to dynamically display lists of content targeted to specific focus areas; for example, items that mention boots.

The Oracle Content Management features you can access, and the UIs you can view, depend on your assigned role. For more information, see the Oracle Content Management documentation available in the Oracle Help Center.

Enable the integration with Oracle Content Management

You can enable the integration with Oracle Content Management using the Settings page in the administration interface.

To enable the integration, perform the following steps:

1. Open the **Settings** page and select **Oracle Integrations**.
2. Choose **Content and Experience** from the dropdown list.
3. Check the **Enable Integration** checkbox, and expand the **Product Configuration** options.
4. Enter the Server URL, Channel Token, and Channel ID, the details of which you can locate within your content management system.
   A channel ID and a channel token are assigned to a channel when it is created within Oracle Content Management. Refer to the Oracle Content Management documentation for further details.
5. Click **Add User** and enter the username and password of the Oracle Content Management user you want to add.
   Note: These user credentials are provided within Oracle Content Management along with the appropriate permissions for the dedicated integration user. The username must be entered exactly as provided within Oracle Content Management; it is not the same as the user's email address.
6. Click **Save**.

Once the configuration is saved, a newly created webhook enables communication between Oracle Commerce and your content management system, and retrieves all content items from the specified channel. Each channel can contain a variety of different content types. (An example of a content type might be a blog or an article.) These content types and items are available in storefronts built using either the Open Storefront Framework or the Storefront

Classic framework. The next two sections describe how to use the **Design** page in the administration interface to enable access to this content from each storefront framework.

Configure Storefront Classic to display content items

You use layouts and widgets to configure pages for selecting and displaying content items in Storefront Classic.

You can find more information about configuring and customizing layouts and widgets in Design Your Store Layout.

Create UI controls for selecting content items

You use the Content Listing widget to display a dropdown menu for shoppers to select individual items to display detailed information about.

Note that you can associate only one content type with an instance of the Content Listing widget. If you want to allow shoppers to select different content types, you will need to create multiple instances of this widget. For example, if you have a blog content type and a recipe content type, you need separate widget instances for listing the items of each type.

To configure a page containing an instance of the Content Listing widget, perform the following steps:

1. Open the **Design** page and select Storefront (Classic) from the dropdown menu at the top of the page.
2. Select the layout you want to clone.
3. Configure the settings for the newly cloned layout, and click **Save**.
4. Open grid view and drag the Content Listing widget to the layout.
5. Open the Content Listing widget’s settings and select the content type from the dropdown list.
6. Click **Save**.
   You can now edit the widget’s code to ensure that the content item fields match those on your own content management system, and to tailor the look of the list as required.
7. Open the **About** tab and click **Go to widget code**, which enables you to go directly to the widget’s template. From here you can update the code references for the content item fields. **Note:** You must ensure the content identifier is up to date so it matches your own fields.
8. Click **Save**.
9. Publish the changes in order to see the content pages, containing links to the content details, displayed on the storefront.

Display content items on the storefront

When a shopper selects a content item from a Content Listing dropdown, the item details are displayed on the Content Item Layout page. The page URL for the content item corresponds to the content mapping in Oracle Content Management. If you make updates to the content and then publish the changes, the new mappings are automatically sent to Commerce.

By default, all content items are rendered using the default Content Item Layout. However, you can create another version of that layout for specific content types by cloning the Content Item Layout and configuring the cloned layout to associate it with a specific content type. You may need to modify the instance of the Content Item widget in the cloned layout to ensure content items are displayed properly on the page.

Configure an OSF storefront to display content items

If your storefront is built using the Open Storefront Framework (OSF), the integration with Oracle Content Management provides additional features not available in Storefront Classic.

These features include:

- The ability to use search queries to tailor content listings and to show the same content types in different combinations.
- Dynamic metadata for improving SEO results.
- A product-aware mode for the Content Item widget that enables you to enrich products by including additional content, media, and documentation.

This section describes how to configure your OSF storefront to access these features.

You can find more information about configuring and customizing OSF layouts and widgets in Developing Open Storefront Framework Applications for Oracle Commerce.

Create UI controls for selecting content items

You use the Content Listing widget to display a dropdown menu for shoppers to select individual items to display detailed information about.

Note that you can associate only one content type with an instance of the Content Listing widget. If you want to allow shoppers to select different content types, you will need to create multiple instances of this widget. For example, if you have a blog content type and a recipe content type, you need separate widget instances for listing the items of each type.

To configure a page containing an instance of the Content Listing widget, perform the following steps:

1. Open the **Design** page and select your OSF application from the dropdown menu at the top of the page.
2. Select the layout you want to clone.
3. Configure the settings for the newly cloned layout, and click **Save**.
4. Open grid view and drag the Content Listing widget to the layout.
5. Open the Content Listing widget's settings and select the content type from the dropdown list.
6. Click **Save**.
7. Publish the changes in order to see the content pages, containing links to the content details, displayed on the storefront.

Query-driven content listing

You can tailor content listings by specifying one or more queries to apply when populating the list. When you configure your Content Listing widget, after you select the content type, click **Add New Parameter**. A dialog box displays controls for constructing the query:

1. In the **Content Field** dropdown, select the field to use in the query. (The values in the dropdown are taken directly from the item type.)
2. In the **Operator** dropdown, select the operator to use. (The operators listed are ones that are appropriate for the type of data in the field.)
3. In the **Value** input, enter the value to apply the operator to.
4. Click **Save**.

For example, if the content type is Article, you might select the Title field, then select the Contains operator, and enter the value `baseball`. If this query is applied, the items listed by the widget will be articles whose title contains "baseball."

You can create multiple queries for a single Content Listing widget instance. If you do this, a toggle switch appears for specifying how to combine them. Click **All** to list only those items that match all of the queries, or **Any** to list all items that match any of the queries.

Display content items on the storefront

When a shopper selects a content item from a Content Listing dropdown, the item details are displayed on the Content Item page. The page URL for the content item corresponds to the content mapping in Oracle Content Management. If you make updates to the content and then publish the changes, the new mappings are automatically sent to Commerce.

By default, all content items are rendered using the default Content Item page. However, you can create another version of that layout for specific content types by cloning the Content Item page and configuring the cloned layout to associate it with a specific content type. You may need to modify the instance of the Content Item widget in the cloned layout to ensure content items are displayed properly on the page.

SEO configuration

The Content Item page configuration include item-specific settings you can use to improve the ranking of pages with web search engines:

1. Use the **Page Title Field** dropdown to specify the content item field whose value will be displayed as the title of the rendered page.
2. Click **Add Meta Tags** to construct SEO meta tags that incorporate data from content item fields you specify.

Content Item widget

The Content Item widget is a container with content field widgets that you can arrange on a layout. Each content field widget is named for the type of data it can display:

- Content Text
- Content Date
- Content Image

You can use these content field widgets in any combination. For example, a page might have a Content Image widget to display an image associated with the content item, and several Content Text widgets for displaying different text fields from the item.

The Content Item container has configuration for specifying the source of the data. You select one of these options from the **Widget Type** dropdown:

- Layout – The data is derived from the layout. (See below.)
- Static – The data is a specific item that you select from the **Asset Name** dropdown that appears if you select this option.
- Product Aware – Enrich product detail pages with additional content from Oracle Content Management. (See below.)

Content Item layout mode

The Layout setting is intended for use with a Content Item container that is placed on a Content Item page. The Content Item page is accessed in the storefront using the URL slug. You can also create a content-type specific layout by cloning the Content Item layout and selecting a content type.

Product-aware mode

The Product Aware setting is intended for use with a Content Item container that is placed on a Product page. When you select this option from the **Widget Type** dropdown, controls appear that you use to associate the Commerce product on the page with a corresponding item in Oracle Content Management, enabling the product details to be enhanced with additional content:

1. Use the Content Type dropdown to specify the type of content item to associate with the Commerce product.
2. Use the Content Field dropdown to specify the field in the content item that represents the product ID in Oracle Content Management.

4 Integrate with Customer Data Management

Integrate Oracle Customer Data Management with Oracle Commerce.

You can configure your Commerce environment to integrate with Oracle Customer Data Management (CDM), a cloud-based application for managing organizations and contacts.

Integrate with Customer Data Management

Integrate your Oracle Commerce environment with Oracle Customer Data Management.

Oracle Commerce can be configured to integrate with Oracle Customer Data Management (CDM), a cloud-based application for managing organizations and contacts. (CDM is also referred to as Oracle Engagement Manager or OEM.) CDM can store organization and contact records that are consolidated from several different applications deployed throughout your environment. You can use CDM to identify potential duplicate records and take the necessary actions to edit, remove or validate your data. For information on obtaining, installing and configuring CDM, refer to `https://docs.oracle.com/en/cloud/saas/customer-data-management/20d/books.html`.

Accounts, contacts and their relationships and addresses can be synchronized from CDM to Oracle Commerce. Similarly, accounts and contacts that are created either through self-registration or a delegated administrator can be synchronized with CDM in real time. The integration between Oracle Commerce and CDM occurs by both applications communicating through Oracle Integration Cloud (OIC), a cloud-based communication platform.

For information on obtaining, installing and configuring OIC, refer to `https://docs.oracle.com/en/cloud/paas/integration-cloud/index.html`.

Integrating between CDM and Oracle Commerce allows you to create scheduled jobs that identify changes to data and then perform the following actions:

- Synchronize in bulk or individually accounts that have been created or updated in Commerce to CDM in real time.
- Synchronize in bulk or individually contacts and profiles that have been created or updated in Commerce to CDM in real time. Additionally, you can associated contacts to an account during the synchronization process.
- Maintain organization hierarchy between synchronizations.

Steps and requirements for the integration

Before you can configure the integration, ensure that you have the following:

- An Oracle Commerce account and access to Oracle CX Commerce 21A or later.
- An Oracle Customer Data Management account and access to CDM 21A or later.
- An Oracle Integration Cloud (OIC) account and access to the Oracle Integration Cloud Service.

If you require one or more of these applications, please contact your Oracle sales representative: `http://www.oracle.com/us/coporate/contact/index.html`.

The integration is delivered as a `.par` file. To download and import the integration, perform the following:

1. Open the integration package `OCC-OEC_Integration`.
2. Import the package by logging into OIC as an admin user.
3. Click the **Packages** button.
4. Click the **Import** button.
5. Click **Browse** to open the navigation pane.
6. Select the `OCC-OEC_Integrations` package.
7. Click **Import**.

The package is added to the packages list.

Understand integrations

The `OCC-OEC_Integrations` package contains three connections and six integrations.

The connections used are:

- Oracle Customer Data Management (CDM), also known as Oracle Engagement Cloud (OEC) - You must provide a CDM Services Catalog URL and an Interface Catalog URL. You must also provide the `Username` and `Password` for access to the OEC.
- Oracle Export Download - This connection is a REST API Base URL that requires a connection URL that points to the Bulk Export Activities resource. You must also provide the CDM `Username` and `Password` for access to CDM.
- Oracle Commerce Cloud - This requires a connection to a Base URL as well as a security token.

The six integrations configured within the package are:

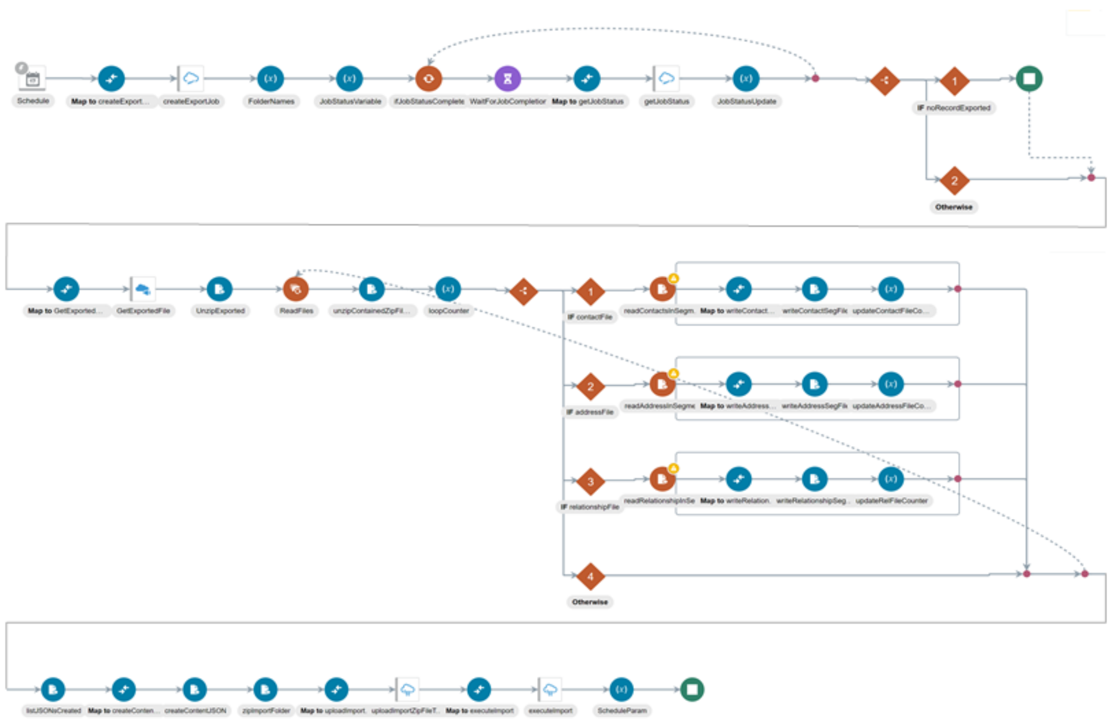
Bulk Profile Sync from OEC to Commerce

This scheduled flow synchronizes profiles in bulk from CDM to Commerce. The identifier is `BULK_PROFILE_SYNC_OEC_TO_OCC`.

When OEC encounters a file that contains more than 50 thousand records, it splits the records into multiple CSV files. However, the OIC integration does not support the conversion of multiple CSV files into JSON files. Should your export file contain more than 50 thousand records, it will be divided into multiple files, however these files will not be converted. To prevent this from occurring, ensure that you do not export more than 50 thousand records at a time.

You should also ensure that CDM is configured to store states using the abbreviated state format, such as CA or VT. This is required because Commerce stores the state values in the abbreviated format.

The following diagram shows the flow of the integration:

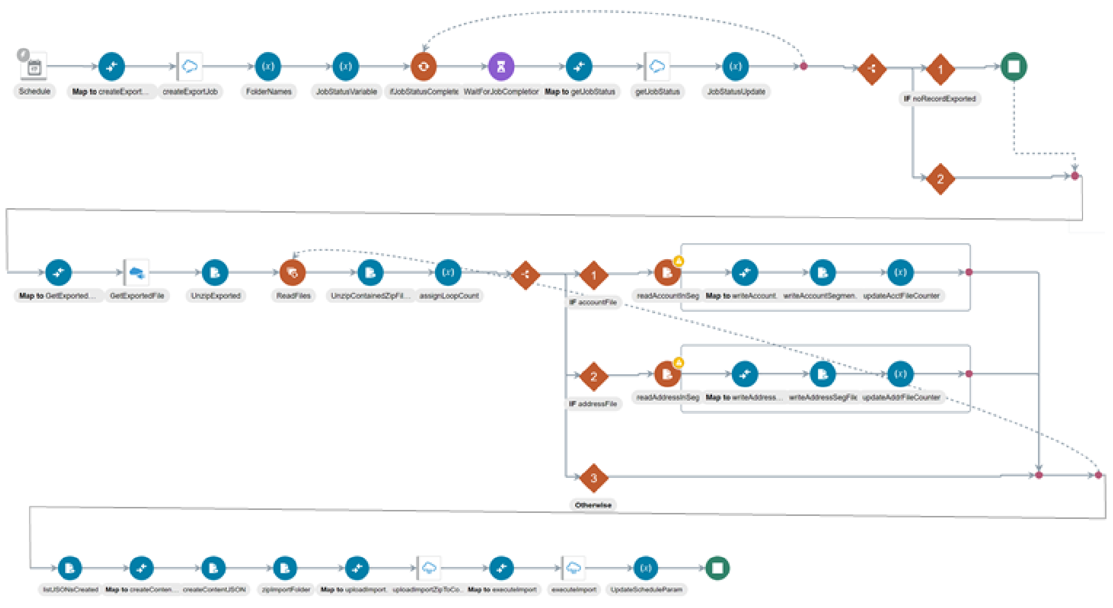


Bulk Account Sync from OEC to Commerce

This scheduled flow synchronizes account data in bulk from CDM to Commerce. Its identifier is `BULK_ACCOUNT_SYNC_OEC_TO_OCC`. When you are synchronizing addresses, the primary address in CDM is marked as the default shipping address in Commerce.

Note that OEC supports multiple accounts with the same name. If a CSV file has to account records with the same name or email address, only the first instance will create a record, the second instance will then update the record. Therefore it is important that you define the appropriate restrictions in CDM to ensure that account names and profile email addresses are unique.

The following diagram shows the flow of the integration:

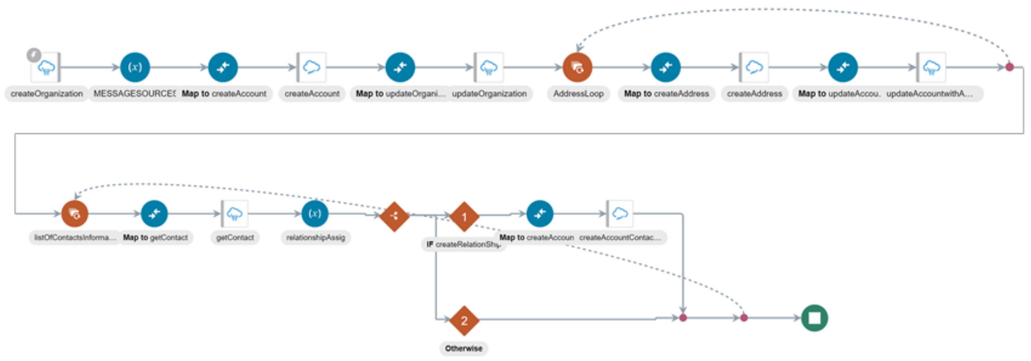


Create Account From Commerce to OEC

The following integrations perform individual synchronizations of things such as profiles, accounts and addresses. This event flow is triggered whenever an account is created in Commerce. It synchronizes the new account data with CDM. Its identifier is `CREATE_ACCOUNT_OCCS_TO_OEC`.

Note that when synchronizing account and contact data from Commerce to CDM, the default shipping address in Commerce is marked as the primary shipping address in CDM. Additionally, accounts that are synchronized from Commerce to OEC are marked as `type = CUSTOMER` in OEC.

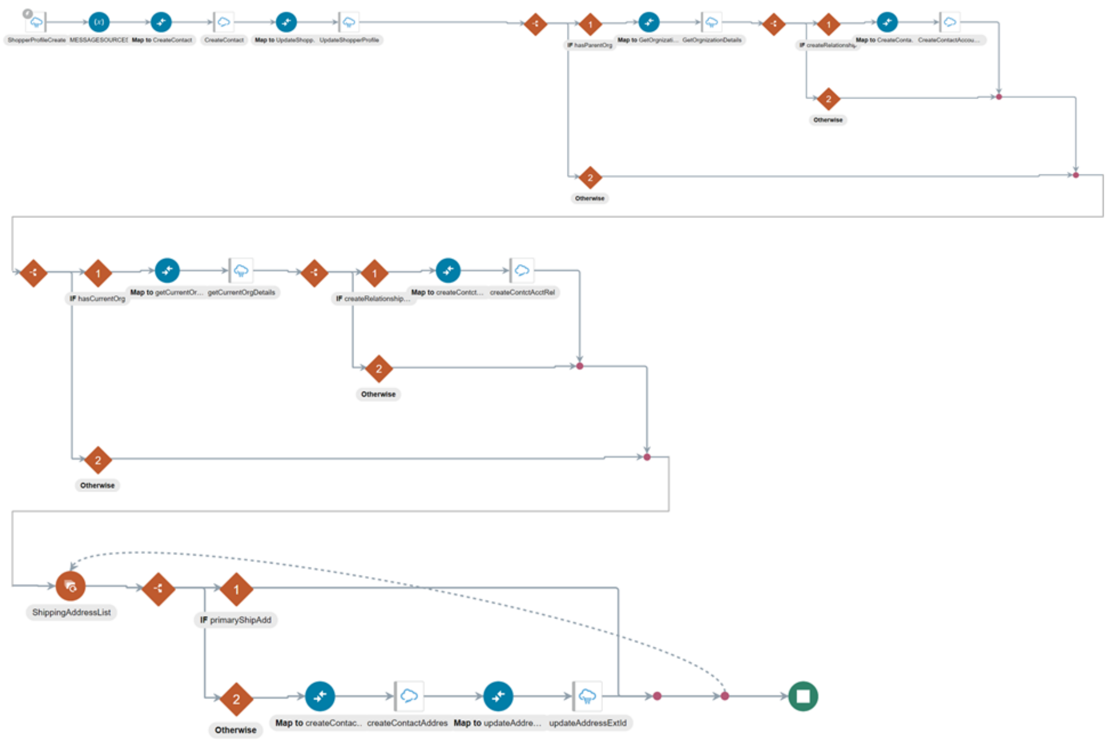
Inherited attribute values are not synchronized to OEC. If an account is a sub-account and it inherits the Tax and DUNs values from its parent, these values are not synchronized, and the Tax and DUNs values will be set to `NULL`. This occurs because inheritance is not recognized in CDM. The integration uses the following architecture:



Create Profile sync from Commerce to OEC

This even flow is triggered whenever a profile is created in Commerce. It synchronizes the new profile data with CDM. Its identifier is `CREA_PROF_SYNC_FOM_OCC_TO_OEC`.

This integration uses the following architecture:



Update Account From Commerce to OEC

This event flow is triggered whenever an account is updated in Commerce. It synchronizes the new account data with CDM. Its identifier is `UPDATE_ACCOUNTS_OCC_TO_OEC`.

Note that when synchronizing account and contact data from Commerce to CDM, the default shipping address in Commerce is marked as the primary shipping address in CDM. Additionally, accounts that are synchronized from Commerce to OEC are marked as `type = CUSTOMER` in OEC.

Update Profile sync from Commerce to OEC

This event flow is triggered whenever a profile is updated in CDM. It synchronizes the new profile data with Commerce. Its identifier is `UPDA_PROF_SYNC_FROM_OCC_TO_OEC`.

Understand account-based contact address synchronization

Commerce supports roles at the account-contact relationship level. However, CDM does not provide such a dynamic use of roles. Whenever an account or contact is synchronized from CDM to Commerce, the default role of Buyer is assigned to all relationships. Because of this, Commerce is unable to assign the Address Manager role and cannot assign addresses to account-based contacts who only have the role of Buyer.

Register the integration with Commerce and generate a security token

This integration uses the Commerce REST APIs to access Commerce data. You must register the integration within Commerce and generate a security token in order for the integration to be granted access to the data.

To generate a security token:

- Log into the Commerce administration interface.
- Click the **Settings** menu and select **Web APIs**.

- Click **Registered Applications** from the **Web APIs** panel.
- Click the **Register Application** button.
- Enter a name for the integration application. Create a meaningful name that reflects the purpose of the application.
- Click **Save**. The Application ID and Application Key are automatically generated and the application is added to the Registered Applications page.
- Click on the name of the application you created.
- Click on **Reveal link** to display application key. You can copy the application key to use as the security token for the Oracle Commerce Cloud connection.

For more information on managing an application within Commerce, refer to Register applications.

Configure the source system reference

Whenever contacts are synchronized from Commerce to CDM, a source system reference is required in CDM. Source system references allow you to identify the source of the data. When you create a source system code, ensure that it has a unique identifier.

Configure the source system code in OIC to pass the value to CDM as part of the integration flow. For information on setting up OIC mappings, refer to the OIC documentation.

To configure a Commerce system, log into your CDM application and perform the following steps:

- Navigate to the **Setup and Maintenance** tab.
- Select **Customer Data Management** from the **Setup** options.
- Select **Trading Community Foundation**. From there, select the **Manage Trading Community Source Systems**.
- Create a Commerce Cloud system with the code `COMMERCE_CLOUD`. The `Type` of the code is `Spoke`. Provide a full name in the **Name** field, such as Oracle Commerce Cloud. Enable the code for **Trading Community Members**.
- When you have finished, save your changes by publishing the sandbox by using the drop down menu to select **Manage Sandboxes**. Select the currently active sandbox and click **Publish**.

Configure the Commerce webhook

When an account or profile is created in Commerce, it is synced to OEC. These synchronizations are triggered by the account, shopper and `CreateAnUpdate` webhooks. The webhooks then trigger the integration workflows. You must configure the profile and account webhook to point to the correct URLs. Follow these steps to configure the webhooks in the Commerce administration interface:

- Log into the Commerce administration interface.
- Click the **Settings** icon.
- Click **Web APIs** and then click the **Webhook** tab.
- Click the `production-updateProfile` webhook. Provide the endpoint URL for the integration:

```
.../ic/api/integration/v1/flows/rest_oraclecommercecloud/
UPDA_PROF_SYNC_FROM_OCC_TO_OEC/1.0/
```

- Update the OIC `username` and `password` under **Basic Authorization**.
- Click the `production-registerProfile` webhook. Enter the integration endpoint URL in the **URL** box:

```
.../ic/api/integration/v1/flows/rest_oraclecommercecloud/
CREA_PROF_SYNC_FROM_OCC_TO_OEC/1.0/
```

- Update the OIC `username` and `password` under **Basic Authorization**.
- Click the `production-createAccount` webhook. Enter the integration endpoint URL in the **URL** box:

```
.../ic/api/integration/v1/flows/rest_oraclecommercecloud/
CREATE_ACCOUNT_OCCS_TO OEC/1.0/
```

- Update the OIC `username` and `password` under **Basic Authorization**.
- Click the `production-updateAccount` webhook. Enter the integration endpoint URL in the **URL** box:

```
.../ic/api/integration/v1/flows/rest_oraclecommercecloud/
UPDATE_ACCOUNT_OCCS_TO_OEC/1.0/
```

- Update the OIC `username` and `password` under **Basic Authorization**.
- Click **Save**.

Configure the connections

Once you have installed the package, you must configure the connections used in the integration.

- Log in to OIC as an admin user.
- Select **Integration** and then **Connections**.
- Select **Oracle Engagement Cloud**. The **Connection Properties** dialog appears.

Enter the **OEC Services Catalog URL** and an **Interface Catalog URL**. The OEC Services Catalog URL is: `https://`*`hostname`*`/fscmService/ServiceCatalogService?wsdl`

The **Interface Catalog URL** is: `https://`*`hostname`*`/helpProfalApi/otherResources/latest/interfaceCatalogs`

- Enter the `Username` and `Password` for access to the OEC.
- Enter the security token value, which you can find in the Commerce administration settings and click **OK**.
- Select **OEC Export Download**. The **Connection Properties** dialog appears.
- The `connection type` for this property is `restUrl`.

Enter the connection URL that points to the Bulk Export Activities resource. For example, the URL would be: `https://`*`CDMServer`*`/crmRestApi/resources/CDMServer/bulkExportActivities`

or

`https://`*`CDMServer`*`/crmRestApi/resources/latest/bulkExportActivities`.

- Select Oracle Commerce Cloud.
  - Enter the **Connection base URL**, which would be `https://`*`CommerceHost`*`/ccadmin/v1`
  - The security token is the application key created in **Register the application** and **Create a security key**.

Activate the integration flows

After you configure the Oracle CDM and Commerce connections, you must activate the integrations that were created when the integration package was imported to Oracle Integration Cloud. To do this, follow these steps:

- Log in to Oracle Integration Cloud (OIC) as an admin user.
- Click the **Integrations** icon to display the Integrations list.
- Click the **Activate** button for each of the following integrations:
  - `Bulk Profile Sync from OEC to OCC`
  - `Bulk Account Sync from OEC to OCC` – Note that activating both of these bulk integrations also requires creating a schedule that then runs the integration.
  - `Update Account From OEC to OCC`
  - `Update Profile sync from OEC to OCC`
  - `Create Account From OCC to OEC`
  - `Create Profile sync from OCC to OEC`

OIC displays a message to indicate that the integration flow was successfully activated.

Mapping for CDM and Commerce

The following table shows the relationships between the CDM properties and the Commerce properties. For details on the properties, refer to each product's documentation:

Property in CDM	Property in Commerce
`Account`	`Account`
`Address`	`Address`
`Address ID`	`Id`
`Address Line 1`	`address1`
`Address Line 2`	`address2`
`City`	`city`
`Country`	`country`
`DateOfBirth`	`dateOfBirth`
`DoNotEmailFlag`	`not(receiveEmail)`
`emailAddress`	`email`
`FirstName`	`firstName`
`LastName`	`lastName`
`MiddleName`	`middleName`

Property in CDM	Property in Commerce
`Party Number`	Whenever an account, contact or address entity is synchronized between CDM and Commerce, the `Party Number` information is stored in `externalOrganizationId` property. The `Party Number` property also maps to the `customerContactId` and the `externalAddressId` properties.
`PartyId` (Generated automatically by CDM)	None
`Person`	`Profile`
`Postal Code`	`postalCode`
`Primary address`	`Default shipping address`
`Primary contact`	Commerce accounts can have multiple contacts, and do not recognize a primary contact.
`Province`	None
`Relationship (account-account)`	`Parent Organization`
`Relationship (account-person of type contact)`	`Contact`, or `Secondary Contact` (There is no distinction between contact or secondary contact in CDM.)
`SourceSystemReferenceValue`	`profileId`
`State`	`state`

5

Integrate with an External Product Configurator

Integrate an external product configurator with Oracle Commerce.

When your store is configured to sell configurable products, you may want to integrate with an external product configurator. The recommended configuration is to integrate with Oracle CPQ, however, you can also integrate with a third-party configurator application.

Enable the integration

This topic shows how to enable the integration with the third-party configurator within Oracle Commerce.

1. In the Commerce administration interface, select **Settings**.
2. Select **Oracle Integrations** from the sidebar menu.
3. Select your configurator from the dropdown menu.
4. Check the **Enable Integration** checkbox.
5. Enter the Configuration URL.
6. Enter the Reconfiguration URL.

   Note: You must enter these values for your production and preview environments.
7. Click **Save**.

If you are using multiple sites, you must follow these instructions for each site that you operate.

Mark products as configurable

To identify a product as configurable:

1. In the Commerce administration interface, select **Catalog**.
2. Select the product you wish to identify as configurable.
3. Click on the **SKUs** tab of the product detail pop-up frame.
4. Select the SKU you wish to identify as configurable.
5. Check the **Configurable** checkbox. This displays three further fields you must complete.
6. Enter the Model information. This should match the Model information of a configurable product in the catalog on your configurator.
7. Enter the Product Line information. This should match the Product Line information of a configurable product in the catalog on your configurator.
8. Enter the Product Family information. This should match the Product Family information of a configurable product in the catalog on your configurator.

9. Click **Save**. This returns you to the SKU frame, where the SKU you updated should be marked with an asterisk to identify it as a configurable SKU.

   Note: Administrators can also perform the above setup steps in bulk by using the SKU import program. From the Catalog page in Commerce, click **Manage Catalog** and select Import. In the Import dialog, click Browse and locate the CSV file to import. Click **Upload File**, click **Validate**, and then click **Import**.

Add Customize button to Product Details widget

Add a Customize button to the Product Details widget so the button is visible to Commerce self-service users from the Product Details page for a customizable product.

To add a Customize button to the Product Details widget:

1. In the Commerce administration interface, Select **Design**.
2. Select **Product Layout** from the layout list.
3. Delete the Product Details widget from the layout.
4. Place a new product details widget on the layout.
5. Click the **Settings** icon for the new Product Details widget.
6. From the Element Library, place a **Customize** button on the new Product Details widget.
7. Publish the changes.

Configure the webhooks

A number of webhooks within Commerce provide support for configured items. These must be set up appropriately for your external configurator.

The following webhooks support configuration:

- Approval
- Cart Idle
- External Price Validation
- Order Submit
- Order Submit for PCI Compliant Target Systems
- Quote Request
- Quote Update
- Return Request Update

Ensure that each of these webhooks is configured to work with your external configurator. This means providing appropriate URLs, usernames, and passwords to each of these webhooks.

6

Integrate with Oracle Infinity to collect data

Through an integration between Oracle Commerce and Oracle Infinity, the Commerce Data Ingestion feature lets you use a Universal JavaScript tag that ingests all Commerce Storefront events and sends the data to the Infinity data repository for analytic purposes.

By using this feature, Oracle provides the Customer Data Platform (CDP) system with data that lets marketers dynamically generate audience segments based on current and past behaviors and data attributes.

As a critical part and foundation of the CDP, the Oracle Management Cloud (OMC) Universal Data Ingestion Framework (DIF), by integrating with the Oracle Infinity technologies, establishes the common data ingestion framework for collecting Commerce product behavioral data.

Integrate Commerce with Infinity

This integration establishes a common data ingestion framework for collecting product behavioral data.

This topic explains how the Commerce and Infinity integration establishes a common data ingestion framework for collecting product behavioral data.

To integrate Oracle Infinity with Commerce, events are used as starting point. There are a lot of events which are published from the current store front framework whenever an event takes place in the store user interface. The events used in the integration revolve around actions such as Registration, Login, Cart events, Search, Products viewed, Order placement, and others. These events are subscribed to and are used to send data to Infinity whenever they occur. Specific examples of the data that can be collected include the following:

- Page analytics data (URL, referrer, time on page, browser, device operating system, etc.)
- Commerce specific data
- Products viewed
- Products added to cart
- Categories viewed
- Search terms used
- Order data
- Wish list data

Note: The integration collects data for both account-based and anonymous shoppers.

To collect this data, the presence of an Infinity tag in a site page initiates the download of an Infinity JavaScript. For that to occur, the Infinity tag has to be in a "require" dependency. Infinity provides a long list of event parameters which can accept Commerce data and send it to the Infinity API. Commerce then subscribes to a particular set of these events and provides the mappings to send the data to Infinity.

In summary, the integration works as follows:

- Commerce loads Infinity JavaScript to site pages through a "require" dependency from the Infinity viewmodel.
- Commerce subscribes to particular Infinity events. These are then tracked and bound with methods.
- Commerce data is mapped with Infinity parameters in the methods and this is sent to Infinity for collection.

A new setting for the Infinity integration is provided under the Integrations tab available to Commerce Administrators. After enabling this setting, you must provide the Infinity tag in the Production URL field required for this setting.

A new viewmodel, `infinity.js`, is also provided. The Infinity viewmodel loads the Infinity script into the browser. Subscriptions to the events to be tracked are added in this viewmodel along with their corresponding methods for correct data mapping. The methods are kept as prototype methods which makes them extendable if you want to add more parameters apart from the provided mappings.

For more complete details on using Infinity and its capabilities, refer to the Oracle Infinity documentation.

Understand the role of the Infinity platform in data ingestion

Infinity provides a platform for data ingestion when integrated with Commerce.

With the Commerce data ingestion feature enabled, the Oracle Infinity Tag used in Commerce site pages initiates the collection of data from online systems capable of executing JavaScript. This data is then saved in the Oracle Infinity data repository. Though initial configuration is very simple (by using the Infinity tag features), complex behaviors and site content can be tracked and delivered to the Oracle Infinity reporting environment. Data collected by using the tag can then be used to drive marketing activities of any conceivable type, and integrations with Oracle Marketing Cloud applications.

Oracle Infinity provides the following capabilities:

- Data collection - Collects web and mobile app activity data that interests you. As data is collected, it is organized in sessions, augmented, and evaluated to identify if someone is a previously known user or a new user. All data is collected quickly, processed, and made available for analysis using Infinity's reporting user interface and APIs. This lets you get immediate feedback on campaigns or new content you just launched on your site.
- Reports - Analyzes your data and prepares reports immediately. Unlimited swappable dimensions reduce the need for one-off reports.
- Streams - Gains real-time insights into a continuous flow of visitor activity data.
- Action Center - Integrates in-session, customer-level data with action systems such as email service providers, CRM systems, and marketing automation platforms. Action Center enables creation, monitoring, stopping, and starting of connections.
- Integrations - Provides APIs that let you integrate with your business and marketing applications.
- Account settings - Defines roles, groups, user privileges, and more.

- Library - The Library application provides you with a way to administer reports, measures, dimensions, segments, and any other objects that you can administer.

You may encounter the following Infinity terminology when trying to work with the Commerce and Infinity integration to successfully collect the data that best works for you:

- Account GUID – A unique value used to identify your account. All collected data is stored in one place for an account. All tags on an account use the same account GUID.
- Tag Id – Tag identifier used to put your tags into a hierarchical format. Each tag has a unique ID that may be set at creation time.
- Context – A Context tag is a unique tag configuration selectable by query parameter. You may only have one active context at a time for a tag, though you may have multiple contexts configured for an individual tag.
- Plugin – An add-on to the tag that enables tracking libraries for functions outside of what the base tag tracks.

For more complete details on using Infinity and its capabilities, refer to the Oracle Infinity documentation.

Tag site pages to use the Infinity data ingestion feature

The presence of an Infinity tag in site pages initiates data collection from online systems capable of executing JavaScript.

As mentioned, complex behaviors and site content can be tracked and delivered to the Oracle Infinity reporting environment by using the special Commerce Infinity tag in your store pages. This data is ingested and sent to the Infinity data repository for analysis.

To use this tag in your store site pages, contact your Oracle account representative to obtain a base tag for your site. A tag URL will be returned to you that looks something like this: `c.oracleinfinity.io/acs/account/account_guid/my_tagid/odc.js`.

A setting for Infinity is available in Commerce Admin application under the **Integrations** tab. After enabling the setting, provide the Infinity Tag URL obtained from Infinity in the **Production URL** field.

Note: The `GUID` and `tagID` are unique strings for your site and tag.

For more complete details on using Infinity and its capabilities, refer to the Oracle Infinity documentation.

Understand Infinity integration parameter mapping

Commerce subscribes to particular events in the Storefront which are then tracked and bound with related methods. Infinity parameters are mapped in these related prototype methods with Commerce data.

A site page containing the Infinity tag loads an Infinity script through a "require" dependency. Commerce then subscribes to particular Infinity events to be tracked and bound with specific methods. The Commerce data is then mapped with Infinity parameters in the methods and this is sent to Infinity for collection. The available Commerce/Infinity parameter mappings are the following:

Note: If for some data field a provided parameter is not available in Infinity, you can create custom parameters as "`wt.z_<yourName>`."

Table 6-1 Commerce/Infinity parameter mappings

Event	Event Details	Data tracked	Infinity Parameters

`USER_PROFILE_UPDATE_SUCCESSFUL`	Published when the user profile is updated. When the REST call for profile update is a success, it publishes an event.	• page URI
		• user-id
		• content-group name ("User Profile")
		• step name ("Update Successful")
			`page-uri`
			`wt.dcsvid`
			`wt.cg_n`
			`wt.si_p`

`USER_PROFILE_UPDATE_SUBMIT`	Published from `order.js` while placing an order, before placing an order it validates registered user.
	Published from `user.js` when the user locale is updated if it's not part of supported locales. On successful update to profile, this event is published.
	Published from the Customer Profile widget via the widget's `customerProfile.js` when the user profile is updated.
	Published from Header widget via the widget's `element.js` when user locale is updated.
	Published from the Checkout Registration widget via the widget's `checkoutRegistration.js`, when a place order button is clicked and it publishes a `CHECKOUT_VALIDATE_NOW` event. If the user login is not valid, it publishes an event to this topic.
		• page URI
		• user-id
		• content-group name ("User Profile")
		• step name ("Update Submit")
			`page-uri`
			`wt.dcsvid`
			`wt.cg_n`
			`wt.si_p`

Table 6-1 (Cont.) Commerce/Infinity parameter mappings

Event	Event Details	Data tracked	Infinity Parameters

`USER_LOGOUT_SUBMIT`	Published from the Logon Registration widget (Login-Registration-v2 -> `element.js`) when the user clicks logout or clicks Cancel on login.	• page URI
		• user-id
		• content-group name ("User Profile")
		• step name ("User Logged Out")
			`page-uri`
			`wt.dcsvid`
			`wt.cg_n`
			`wt.si_p`

`USER_LOGIN_SUCCESSFUL`	Published from `user.js` when a user login is successful or a SAML callback is successful.	• page URI
		• user-id
		• GDPR cookie consent
		• content-group name
		• ("User Profile") step name ("Logged In")
			`page-uri`
			`wt.dcsvid`
			`wt.ce`
			`wt.cg_n`
			`wt.si_p`

`USER_LOGIN_SUBMIT`	Published from Login-Registration-v2 -> `element.js` (header) widget and Checkout-Registration -> `checkoutRegistration.js` (checkout page), while the user logs in.	• page URI
		• content-group name ("User Profile")
		• step name ("Log In Submit")
			`page-uri`
			`wt.cg_n`
			`wt.si_p`

`USER_AUTO_LOGIN_SUCCESSFUL`	Published from `user.js` when the user autologin is successful.	• page URI
		• user-id
		• GDPR cookie consent
		• content-group name ("User Profile")
		• step name ("Registered")
			`page-uri`
			`wt.dcsvid`
			`wt.vt_f`
			`wt.ce`
			`wt.cg_n`
			`wt.si_p`

`SEARCH_RESULTS_UPDATED`	Published from `search.js` page layout after a search request is completed.
	If the search request is a success then it publishes with the search results otherwise it publishes with an error message.
		• page URI
		• user-id
		• content-group name ("Search")
		• search text total
		• records found
		• search facet selected (sent as <facet name>-<facet value>)
			`page-uri`
			`wt.dcsvid`
			`wt.cg_n`
			`wt.oss`
			`wt.oss_r`
			`wt.z_selectedSearchFacet`

Table 6-1 (Cont.) Commerce/Infinity parameter mappings

Event	Event Details	Data tracked	Infinity Parameters

`PRODUCT_VIEWED`	Published from the Product Details widget when a product is viewed from PDP or quick view.	• page URI
		• user-id
		• content-group name ("Purchase List")
		• step name ("Add to Purchase List")
		• product id
		• SKU id
		• quantity
		• price
		• product type
		• brand
		• transaction event ("w")
		• currency
			`page-uri`
			`wt.dcsvid`
			`wt.cg_n`
			`wt.si_p`
			`wt.pn_sku`
			`wt.tx_u`
			`wt.tx_s`
			`wt.pn_fa`
			`wt.pn_ma`
			`wt.tx_e`
			`wt.z_currency`

`PRODUCT_ADDED_TO_PURCHASE_LIST_SUCCESS`	Published from Purchase Lists widget (add-to-purchase-list -> `element.js`) when an item is added to the purchase list.	• page URI
		• user-id
		• content-group name ("Purchase List")
		• step name ("Add to Purchase List") product id
		• SKU id
		• quantity
		• price
		• product type
		• brand transaction event ("w")
		• currency
			`page-uri`
			`wt.dcsvid`
			`wt.cg_n`
			`wt.si_p`
			`wt.pn_sku`
			`wt.tx_u`
			`wt.tx_s`
			`wt.pn_fa`
			`wt.pn_ma`
			`wt.tx_e`
			`wt.z_currency`

`PAYMENT_AUTH_SUCCESS`	Published from payment-auth-response view model when the response from the paymentAuthResponse endpoint returns the state of payment accepted and the order status has not failed.	• page URI
		• user-id
		• content-group name ("Payment")
		• step name ("Payment Success")
			`page-uri`
			`wt.dcsvid`
			`wt.cg_n`
			`wt.si_p`

Table 6-1 (Cont.) Commerce/Infinity parameter mappings

Event	Event Details	Data tracked	Infinity Parameters

`PAYMENT_AUTH_DECLINED`	Published from payment-auth-response view model when the response from paymentAuthResponse endpoint returns a state like "removed" or when a payment is authorized but the order failed.	• page URI
		• user-id
		• content-group name ("Payment")
		• step name ("Payment Fail")
			`page-uri`
			`wt.dcsvid`
			`wt.cg_n`
			`wt.si_p`

`PAGE_CHANGED`	Published from `layout-container.js` after the layout is loaded.
	It publishes with pageEventData such as page, pageId, path, pageRepositoryId, etc.
		• page URI
		• user-id
		• page id
		• wt-dcsvid
		• In the case of the of a confirmation page, the following is published:
		• shipping method
		• shipping cost
		• payment gateway name
		• gateway transaction amount
		• content-group name(depending on the page)
			`page-uri`
			`wt.cg_n`
			`wt.dcsvid`
			`wt.z_shippingMethod`
			`wt.z_shippingCharges`
			`wt.z_gatewayName`
			`wt.z_gatewayTransactionAmount`

Table 6-1 (Cont.) Commerce/Infinity parameter mappings

Event	Event Details	Data tracked	Infinity Parameters

`ORDER_SUBMISSION_SUCCESS`	Published from the `order.js` view model when the order details of the initial order created during checkout with PayPal/ PayU is fetched. If the transaction is done via PayU and the status is settled/ approved, this event is published.
	
	Published from the `order.js` view model when an order is created or updated successfully and the status is submitted or is pending approval then this event is published.
	
	Published from the `order.js` view model when the payment is authorized. This is triggered when it receives a PAYMENT_AUTH_SUCCESS event.
		• page URI
		• user-id
		• content-group name ("Order")
		• step name ("Order Submission Success")
		• SKU id
		• product type
		• brand
		• quantity
		• price
		• transaction event ("p")
		• invoice date
		• invoice time
		• invoice number (UUID)
		• order id
		• conversion ("Purchase")
		• campaign id
		• campaign source
			`page-uri`
			`wt.dcsvid`
			`wt.cg_n`
			`wt.si_p`
			`wt.pn_sku`
			`wt.pn_fa`
			`wt.pn_ma`
			`wt.tx_u`
			`wt.tx_s`
			`wt.tx_e`
			`wt.tx_id`
			`wt.tx_it`
			`wt.tx_i`
			`wt.tx_cartid`
			`wt.conv`
			`wt.z_campaignId`
			`wt.z_campaignSource`

`ORDER_SUBMISSION_FAIL`	Published from the `order.js` view model when order submission fails due to any of these reasons: Payment Auth timeout, Payment declined, and/or order creation/ update failure.
	
	Published from the CyberSource Payment Authorization widget if there is an error while generating the signature in a payment iFrame.
		• page URI
		• user-id
		• content-group name ("Order")
		• step name ("Order Submission Fail")
			`page-uri`
			`wt.dcsvid`
			`wt.cg_n`
			`wt.si_p`

Table 6-1 (Cont.) Commerce/Infinity parameter mappings

Event	Event Details	Data tracked	Infinity Parameters

`ORDER_COMPLETED`	Published from the `payment-auth-response.js` view model when a payment authorization is accepted.
	Published from the `order.js` view model when the order status is either submitted or pending approval.
	Published from the Split Payments widget when the order state is either pending approval or a template (i.e., the order is a scheduled order) or pending scheduled order approval.
		• page URI
		• user-id
		• content-group name ("Order")
		• step name ("Order Completed")
			`page-uri`
			`wt.dcsvid`
			`wt.cg_n`
			`wt.si_p`

`COUPON_APPLY_SUCCESSFUL`	Published from the cart view model when a cart is updated from the server after a coupon is applied successfully.	• page URI
		• user-id
		• content-group name ("Coupon")
		• coupon id
			`page-uri`
			`wt.dcsvid`
			`wt.cg_n`
			`wt.mc_id`

`CHECKOUT_SHIPPING_METHOD`	Published from the `cart.js` view model with shippingOption when the shipping methods are loaded.
	Published from the Cart Shipping widget when a shipping option is reset or if a shipping address and shipping method has changed.
		• page URI
		• user-id
		• content-group name ("Shipping Method")
		• step name ("Shipping Method Selected")
			`page-uri`
			`wt.dcsvid`
			`wt.cg_n`
			`wt.si_p`

`CHECKOUT_SAVE_SHIPPING_ADDRESS`	Published from `order.js` view model with the shipping address when the Place Order button is clicked.	• page URI
		• user-id
		• content-group name ("Address")
		• country
		• state
		• city
		• postal code
			`page-uri`
			`wt.dcsvid`
			`wt.cg_n`
			`wt.z_country`
			`wt.z_region`
			`wt.z_city`
			`wt.z_zip`

Table 6-1 (Cont.) Commerce/Infinity parameter mappings

Event	Event Details	Data tracked	Infinity Parameters

`CHECKOUT_REGISTER_USER`	Published from the Checkout Order Details widget when all validations for creating the order have passed.	• page URI
		• user-id
		• content-group name ("User Profile")
		• step name ("Checkout Register")
		• GDPR cookie consent
			`page-uri`
			`wt.dcsvid`
			`wt.cg_n`
			`wt.si_p`
			`wt.ce`

`CART_UPDATE_QUANTITY`	Published in the Shopping Cart widget with the `commerceItemId` when the Quantity Update button is clicked.	• page URI
		• user-id
		• content-group name ("Cart")
		• step name ("Update Cart")
		• product id
		• SKU id
		• updated quantity
		• price
		• product type
		• brand transaction event ("a")
		• currency
			`page-uri`
			`wt.dcsvid`
			`wt.cg_n`
			`wt.si_p`
			`wt.pn_id`
			`wt.pn_sku`
			`wt.tx_u`
			`wt.tx_s`
			`wt.pn_fa`
			`wt.pn_ma`
			`wt.tx_e`
			`wt.z_currency`

`CART_REMOVE_SUCCESS`	Published from `cart.js` when an item is removed from the cart view model. It is also published with a product commerce id.
	Published from `cart.js` when a place holder item is removed from the cart.
		• page URI
		• user-id
		• content-group name ("Cart")
		• step name ("Remove from Cart")
		• product id
		• SKU id
		• removed quantity
		• price
		• product type
		• brand
		• transaction event ("r")
		• currency
			`page-uri`
			`wt.dcsvid`
			`wt.cg_n`
			`wt.si_p`
			`wt.pn_sku`
			`wt.tx_u`
			`wt.tx_s`
			`wt.pn_fa`
			`wt.pn_ma`
			`wt.tx_e`
			`wt.z_currency`

Table 6-1 (Cont.) Commerce/Infinity parameter mappings

Event	Event Details	Data tracked	Infinity Parameters

`CART_ADD_SUCCESS`	Published from `cart.js` when a cart is updated and the last cart event is cart-add-item.
	
	The cart is updated when the REST call is made to fetch the current profile order and price information to refresh the cart data.
		• page URI
		• user-id
		• content-group name ("Cart")
		• step name ("Add to Cart")
		• product id
		• SKU id
		• quantity
		• price
		• product type
		• brand
		• transaction event ("a")
		• currency
			`page-uri`
			`wt.dcsvid`
			`wt.cg_n`
			`wt.si_p`
			`wt.pn_id`
			`wt.pn_sku`
			`wt.tx_u`
			`wt.tx_s`
			`wt.pn_fa`
			`wt.pn_ma`
			`wt.tx_e`

`ADD_TO_QUICK_ORDER`	Published from the product-add-to-quick-order element when adding to a quick order and the button is clicked.	• page URI
		• user-id
		• content-group name ("Quick Order")
		• step name ("Add to Quick Order")
		• product id
		• SKU id
		• quantity
		• price
		• product type
		• brand
		• transaction event ("q")
		• currency
			`page-uri`
			`wt.dcsvid`
			`wt.cg_n`
			`wt.si_p`
			`wt.pn_sku`
			`wt.tx_u`
			`wt.tx_s`
			`wt.pn_fa`
			`wt.pn_ma`
			`wt.tx_e`
			`wt.z_currency`

Keep in mind the following about the integration parameters:

- All methods are prototypes, so, if you want to add more parameters, you can extend them.
- If for some reason a data field is not provided in Infinity, you can create custom parameters in the following format so that : `wt.z_<yourParameterName>` the Oracle Infinity platform will start to record it.
- Some events are published from provided widgets which earlier did not publish any relevant data. These have since been changed to publish relevant information. If you are not using the provided widgets in any case, you must publish similar data in the events. These widgets include ORDER_SUBMISSION_SUCCESS (Split Payments widget), PRODUCT_ADDED_TO_PURCHASE_LIST_SUCCESS (Purchase List widget), USER_PROFILE_UPDATE_SUBMIT (Shopper Details, Address Book, and Update Password widget), and CART_REMOVE_SUCCESS. It is necessary, then, to take the latest changes accordingly. If you are publishing any of the events listed in the table and not publishing relevant data similar to the ones provided, you need to provide updates to correct this.

- The Infinity `viewmodel.js` depends on the `ORA_ANALYTICS_READY` event on DOM (published by the Infinity JS) to initialize the Infinity API. Refer to your Infinity Administrator to turn on the `READY EVENT` the tag for the same.

For more complete details on using Infinity and its capabilities, refer to the Oracle Infinity documentation.

7 Integrate with Oracle Order Management Cloud

Integration Oracle Order Management Cloud with Oracle Commerce.

Oracle Order Management Cloud can improve order handling when working with order-to-cash processes. You can configure fulfillment monitoring, global availability and enterprise-wide policies that increase your shopper's satisfaction.

Introduction

Oracle Commerce and Oracle Order Management Cloud can be combined through Oracle Integration Cloud to provide a robust architecture for capturing and fulfilling orders placed in your online store.

Oracle Commerce is an industry-leading commerce platform re-designed for the modern cloud. Oracle Order Management Cloud is tightly integrated with Oracle Global Order Promising Cloud. It can improve order handling across the order-to-cash process. Pre-integration, centrally-managed orchestration policies, global availability, and fulfillment monitoring can lead to increased customer satisfaction and order profitability.

This document describes how to integrate these two services.

Audience

This document is written for Commerce and Order Management Cloud administrators who are setting up and configuring the integration between these two systems.

Readers of this document should have experience with both Commerce and Order Management Cloud. See Additional documentation for links to comprehensive information on these services.

Features

The integration of Commerce with Order Management Cloud combines the capabilities of these two services into a single solution.

This solution provides support for the following:

- Pushing completed Commerce orders to Order Management Cloud for fulfillment.
- Retrieving and updating the status of the order from Oracle Order Management in Commerce real time for the orders created using Commerce.
- Furnishing returns and exchanges created on Commerce to Order Management Cloud for fulfillment.
- Retrieving the status of the returns from Order Management into Commerce.

The default integration assumes that the orders and returns are created using Commerce channels.

Architectural overview

The message flow of business entities from Commerce to Order Management Cloud happens through Oracle Integration Cloud.

Commerce includes an Oracle Integration Cloud adapter (the Commerce adapter) that is used for the integration. The Order Management Cloud part of the integration uses the generic SOAP adapter of Oracle Integration Cloud to manage the integration. Data from the Commerce store is transmitted using webhooks through Oracle Integration Cloud to reach Order Management Cloud. The data from Order Management Cloud reaches Commerce through Oracle Integration Cloud as well.

Additional documentation

For information about Commerce, and the Oracle Order Management Cloud, refer to the following Oracle Help Center pages:

Oracle Commerce documentation: First Steps.

Documentation of Oracle Order Management Cloud is available through Oracle Applications Help: Oracle Order Management Cloud documentation.

For information about Oracle Integration Cloud and the Commerce adapter, refer to Using the Oracle Commerce Adapter in the Oracle Help Center at this URL: Getting Started with the Oracle CX Commerce Adapter.

Prerequisites

Prerequisites for a successful integration include specific access rights to both systems. In addition, certain assumptions exist about the way each service is used.

This section provides details about the prerequisites.

Access rights

To configure this integration, you need administrator access to the following systems:

- The Oracle Commerce administration interface, which enables you to configure the webhook settings.
- Oracle Integration Cloud, which enables you to map the attributes between Commerce and Order Management Cloud. Note that in earlier releases, Oracle Integration Cloud is referred to as Integration Cloud Service (ICS). The instructions in this guide, as well as the integration itself, apply to both cases.
- Order Management Cloud, which enables you to set up the fulfillment and configure the products and the activities associated with fulfillment.

Assumptions

This integration makes the following functional assumptions:

- Orders can be created only on Commerce channels. Orders created on non-commerce channels are not within the scope of this integration.

- The Returns and exchange orders are initiated from Commerce. The cases where the returns and exchanges are created on Order Management Cloud are not synched back to Commerce.
- Commerce does not capture any credit card settlement information. It is assumed that the merchant updates the card settlement information about Commerce.
- Purchase Order status payment groups are updated as settled/No Pending Action after the payment processing is complete.
- The inventory and product SKU information is assumed to be in synch across both the systems.
- Commerce acts as the pricing and the promotions engine.
- The tax calculation for an order is performed in Commerce.
- The scope of this integration is restricted to simple products. Configurable products are out of its scope.

This integration is extensible: additional attributes can be mapped on both systems without interfering with the merchant's use cases.

Configure Oracle Commerce for Oracle Management Cloud

The integration is configured on the Commerce side by setting up specific web APIs to use Oracle Integration Cloud endpoints.

Web APIs enable you to subscribe to events for your products and orders by creating webhooks that push notifications to a specified URL. (For additional information about webhooks, see Use Webhooks.)

You update the Commerce web APIs through the administration interface to use the appropriate Oracle Integration Cloud endpoints. The web APIs send order information to Oracle Integration Cloud, and the orders go to Order Management for fulfillment.

To ensure compliance with PCI requirements, Commerce uses a webhook for order submission (Order Submit without Payment Details) that discards credit card information and sends the rest of the Order to external systems. Use of the Order Submit without Payment Details webhook is recommended in cases where the merchant wants to send the token details of the credit card.

The topics in this section describe how to make the necessary changes to the Commerce web APIs.

Submit Order webhook

Configuring the Order Submission webhook makes the order information created in Commerce available to the Oracle Integration Cloud system.

To configure the Order Submit for Non-PCI Compliant Target Systems Event API, follow these steps:

1. In the Commerce administration interface, display the **Settings** page.
2. Select **Web APIs** and display the **Webhook** tab.
3. Select the **Order Submit without Payment Details** event API.
4. Provide the Oracle Integration Cloud URL that receives the order message. Include the server name and port used.

5. Provide a user name and password for accessing the server.
6. Click **Save**.

Return Order webhook

Configuring the Return Request Update for Non-PCI Compliant Target Systems event API displays the returns workflow.

To configure the Return Request Update for Non-PCI Compliant Target Systems event API, follow these steps:

1. In the Commerce administration interface, display the **Settings** page.
2. Select **Web APIs** and display the **Webhook** tab.
3. Select the **Return Request Update without Payment Details** event API.
4. Provide the Oracle Integration Cloud URL that receives the order message. Include the server name and port used.
5. Provide a user name and password for accessing the server.
6. Click **Save**.

Configure the Oracle Integration Cloud Adapter

The integration mappings and the associated files you configure for Oracle Integration Cloud are available in the form of packages. These packages are ready for you to use as soon as they are imported into the Oracle Integration Cloud infrastructure.

The Commerce team already has a Commerce adapter available for Oracle Integration Cloud. The generic SOAP adapter is a generic adapter provided by Oracle Integration Cloud for processing the SOAP-based messages. This adapter is used when Order Management Cloud sends messages in SOAP format.

The integration package, `OracleCommerce-OrderManagementIntegration.par`, is available on My Oracle Support. The contents of this package are described in the sections that follow.

Connections

The integration package includes the required connections for the servers and events used by the integration.

- Oracle Commerce: The connection to the Commerce admin server
- Oracle Commerce Agent: The connection to Commerce agent server
- Oracle Order Management Cloud: The connection to Order Management Order synch service
- Oracle Order Management Cloud Events: The connection to Order Management events
- Oracle Order Management Cloud Order Information Service: The connection to the Order Management order retrieval service

Lookup configuration

The mappings for the return status, carrier information, and shipping information are updated as part of the lookup configuration.

You should modify these as required and as configured in your order management system. Refer to the integration settings to view the lookups used.

Integrations

The package includes integrations for order synch, return synch, and order and return status synch.

- Order synch from Commerceto Order Management
- Return synch from Commerce to Order Management
- Order and Return status synch from Order Management to Commerce

Unpack the `OracleCommerce-OrderManagementIntegration.par` package. It unpacks into three inventory archives (IARs). Unpack each IAR and update the constants as shown in the following sections. Update the following constants for order synch from Commerce to the Order Management flow and import using the Oracle Integration Cloud mapper in Oracle Integration Cloud.

XSL location

XSL location is described in the following section.

```
<Unpacked
        iar>/icspackage/project/RETU_SYNC_FROM_COMM_TO_ORDE_MGMT_01.00.0000/
resources/processor_52/resourcegroup_913
<xsl:variable name="SOURCE_SYSTEM" select="'ATG'"
        xml:id="id_2076"/>
<xsl:variable name="BUSINESS_UNIT_IDENTIFIER"
        select="'204'" xml:id="id_2077"/>
<xsl:variable name="LEGAL_UNIT_IDENTIFIER"
        select="'204'" xml:id="id_2078"/>
<xsl:variable name="FULFILLMENT_ORG_IDENTIFIER"
        select="'207'" xml:id="id_2079"/>
<xsl:variable name="INVENTORY_ORG_IDENTIFIER"
        select="'207'" xml:id="id_2080"/>
<xsl:variable name="UOM_CODE" select="'Ea'"
        xml:id="id_2081"/>
```

Update the following constant for the return synch from Commerce to the Order Management flow and import using the Oracle Integration Cloud mapper in Oracle Integration Cloud.

```
<Unpacked
        iar>/icspackage/project/RETU_SYNC_FROM_COMM_TO_ORDE_MGMT_01.00.0000/
resources/processor_52/resourcegroup_913
<xsl:variable name="SOURCE_SYSTEM" select="'ATG'"
```

```
        xml:id="id_2076"/>
<xsl:variable name="BUSINESS_UNIT_IDENTIFIER"
        select="'204'" xml:id="id_2077"/>
<xsl:variable name="LEGAL_UNIT_IDENTIFIER"
        select="'204'" xml:id="id_2078"/>
<xsl:variable name="FULFILLMENT_ORG_IDENTIFIER"
        select="'207'" xml:id="id_2079"/>
<xsl:variable name="INVENTORY_ORG_IDENTIFIER"
        select="'207'" xml:id="id_2080"/>
<xsl:variable name="UOM_CODE" select="'Ea'"
        xml:id="id_2081"/>
```

Note: The connection details that come as part of the integration package are empty. You provide the details of the connections and test them. You can then enable the integration.

Configure Order Management Cloud

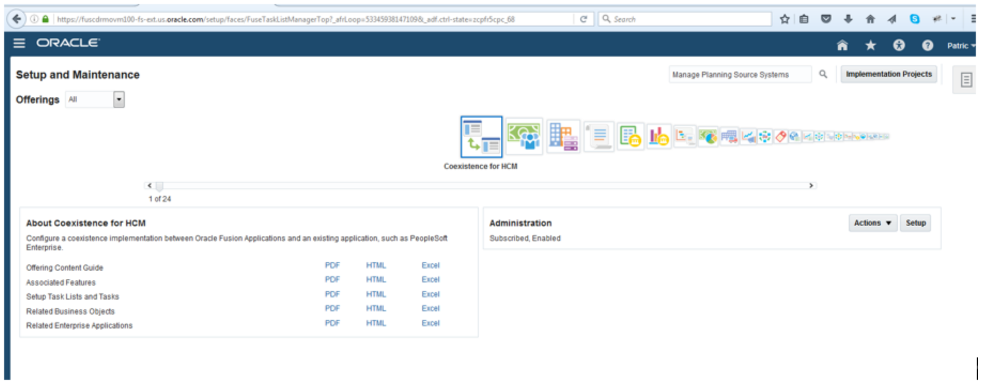
Configuring Order Management Cloud for the integration involves creating the source system, the default rules, the orchestration process, and the required connectors.

The topics in this section describe how to perform these Order Management Cloud configuration tasks.

Create the source system

This section describes the steps you follow to create a source system for the integration.

1. Go to the FSM link `https://example.com/setup/faces/TaskListManagerTop` and search for Manage Planning Source Systems.



https://fuscdrmovm100-fs-ext.us.oracle.com/setup/faces/FuseTaskListManagerTop?_afrLoop=53345938147109&_adf.ctrl-state=zcpfr5cpc_68

Setup and Maintenance

Manage Planning Source Systems

Offerings: All

Coexistence for HCM

1 of 24

About Coexistence for HCM

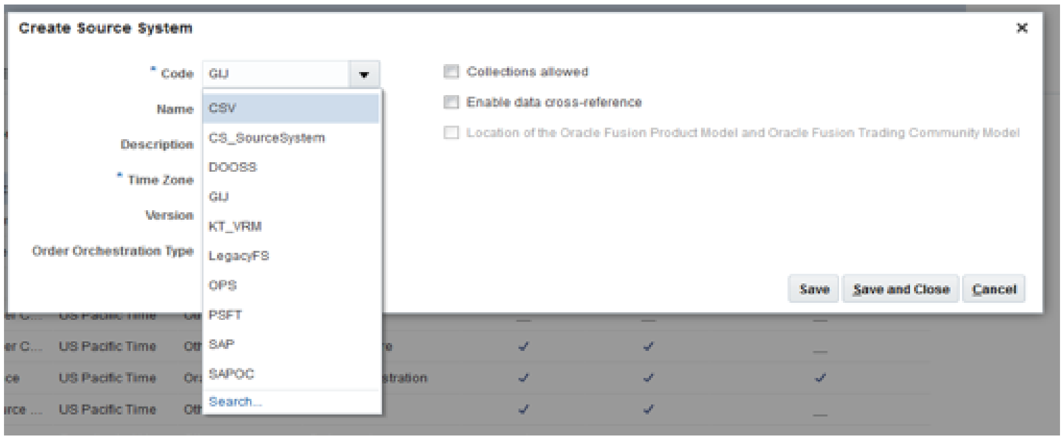
Configure a coexistence implementation between Oracle Fusion Applications and an existing application, such as PeopleSoft Enterprise.

Offering Content Guide	PDF	HTML	Excel
Associated Features	PDF	HTML	Excel
Setup Task Lists and Tasks	PDF	HTML	Excel
Related Business Objects	PDF	HTML	Excel
Related Enterprise Applications	PDF	HTML	Excel

Administration

Subscribed, Enabled

2. Display the **Actions** tab.



Create Source System

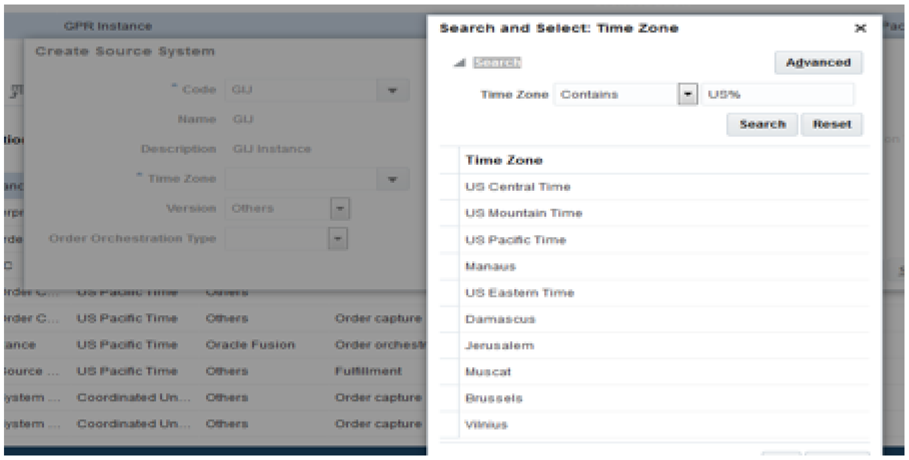
* Code	GIJ
Name	
Description	
* Time Zone	
Version	
Order Orchestration Type	

Code list: CSV, CS_SourceSystem, DOOSS, GIJ, KT_VRM, LegacyFS, OPS, PSFT, SAP, SAPOC, Search...

- ☐ Collections allowed
- ☐ Enable data cross-reference
- ☐ Location of the Oracle Fusion Product Model and Oracle Fusion Trading Community Model

3. Create a source system for Commerce. If necessary, create a new code entry for Commerce.

4. Select the time zone as required.



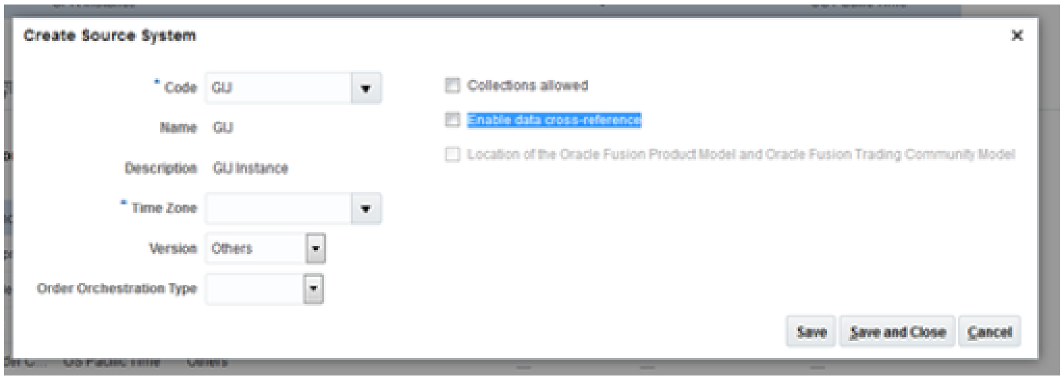
Search and Select: Time Zone

Search

Time Zone Contains US%

Time Zone
US Central Time
US Mountain Time
US Pacific Time
Manaus
US Eastern Time
Damascus
Jerusalem
Muscat
Brussels
Vilnius

5. Disable the cross-references between Commerce and Order Management as the cross-references are handled by Oracle Integration Cloud.

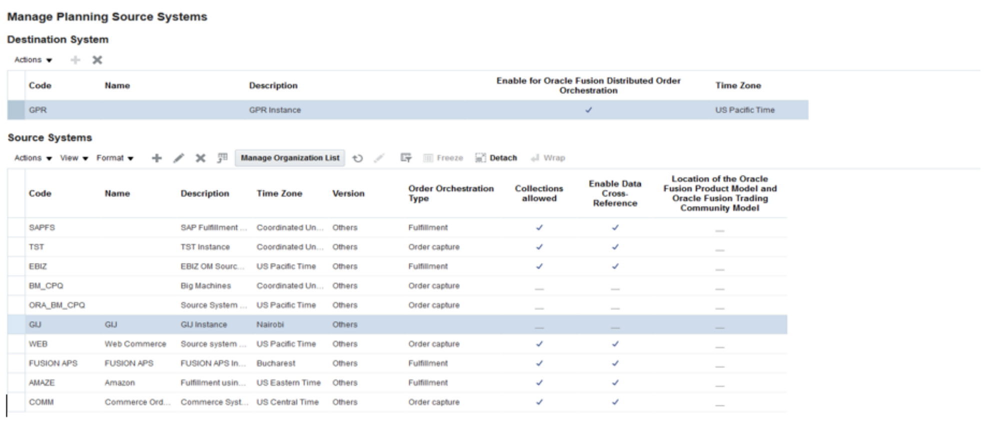


Create Source System

* Code	GIJ
Name	GIJ
Description	GIJ Instance
* Time Zone	
Version	Others
Order Orchestration Type	

- ☐ Collections allowed
- ☐ Enable data cross-reference
- ☐ Location of the Oracle Fusion Product Model and Oracle Fusion Trading Community Model

6. When you have finished entering the setup details, the final screen for the source system appears:



Manage Planning Source Systems

Destination System

Actions

Code	Name	Description	Enable for Oracle Fusion Distributed Order Orchestration	Time Zone
GPR		GPR Instance	✓	US Pacific Time

Source Systems

Actions View Format Manage Organization List Freeze Detach Wrap

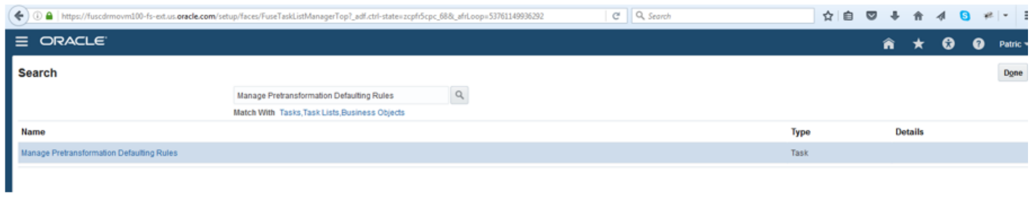
Code	Name	Description	Time Zone	Version	Order Orchestration Type	Collections allowed	Enable Data Cross-Reference	Location of the Oracle Fusion Product Model and Oracle Fusion Trading Community Model
SAPFS		SAP Fulfillment ...	Coordinated Un...	Others	Fulfillment	✓	✓	—
TST		TST Instance	Coordinated Un...	Others	Order capture	✓	✓	—
EBIZ		EBIZ OM Sourc...	US Pacific Time	Others	Fulfillment	✓	✓	—
BM_CPQ		Big Machines	Coordinated Un...	Others	Order capture	—	—	—
ORA_BM_CPQ		Source System ...	US Pacific Time	Others	Order capture	—	—	—
GIJ	GIJ	GIJ Instance	Nairobi	Others		—	—	—
WEB	Web Commerce	Source system ...	US Pacific Time	Others	Order capture	✓	✓	—
FUSION APS	FUSION APS	FUSION APS In...	Bucharest	Others	Fulfillment	✓	✓	—
AMAZE	Amazon	Fulfillment usin...	US Eastern Time	Others	Fulfillment	✓	✓	—
COMM	Commerce Ord...	Commerce Syst...	US Central Time	Others	Order capture	✓	✓	—

Create defaulting rules

Defaulting rules must be created for this integration in the Order Management system to ensure that the shipping and scheduling rules cover Commerce as a channel.

The following rules must be defined:

- Shipping Method Defaulting Rule1
- Scheduling Related Rule
- Shipping Method Defaulting Rule2



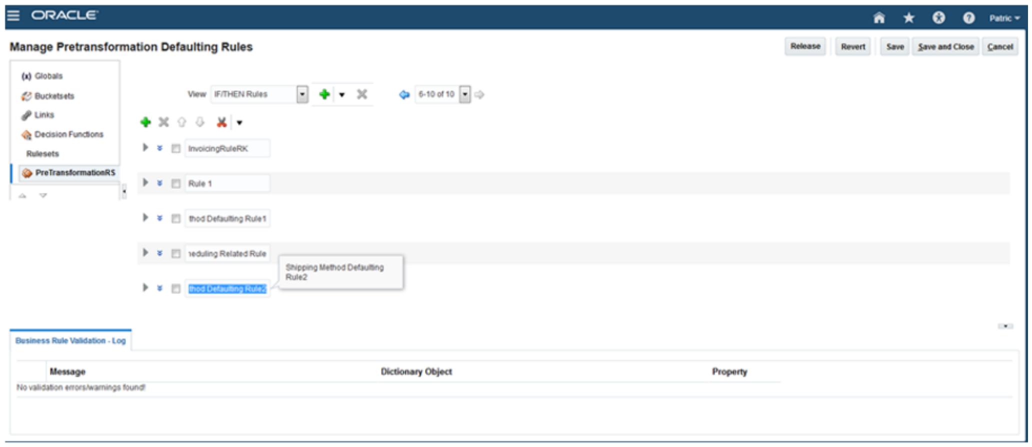
https://fuscdrmovm100-fs-ext.us.oracle.com/setup/faces/FuseTaskListManagerTop?_adf.ctrl-state=zcpfr5cpc_68&_afrLoop=53761149936292

Search

Manage Pretransformation Defaulting Rules

Match With Tasks,Task Lists,Business Objects

Name	Type	Details
Manage Pretransformation Defaulting Rules	Task	



Manage Pretransformation Defaulting Rules

Release Revert Save Save and Close Cancel

(x) Globals
Bucketsets
Links
Decision Functions
Rulesets
PreTransformationRS

View IF/THEN Rules   6-10 of 10

InvoicingRuleRK
Rule 1
thod Defaulting Rule1
heduling Related Rule
thod Defaulting Rule2

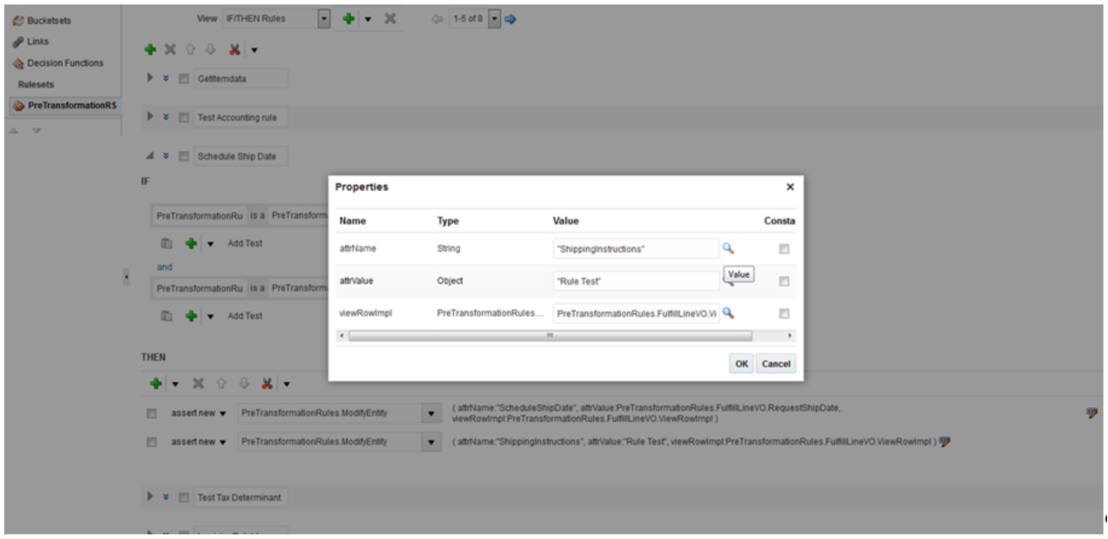
Shipping Method Defaulting Rule2

Business Rule Validation - Log

Message	Dictionary Object	Property
No validation errors/warnings found!		

Note: Ideas for rules to create are provided here; you can configure these as necessary for your business requirements.

This process ensures that the shipping and scheduling rules cover Commerce as a channel.



View IF/THEN Rules · 1-5 of 8

Bucketsets
Links
Decision Functions
Rulesets
PreTransformationRS

Getitemdata
Test Accounting rule
Schedule Ship Date

IF
PreTransformationRu is a PreTransform
Add Test
and
PreTransformationRu is a PreTransform
Add Test

Properties

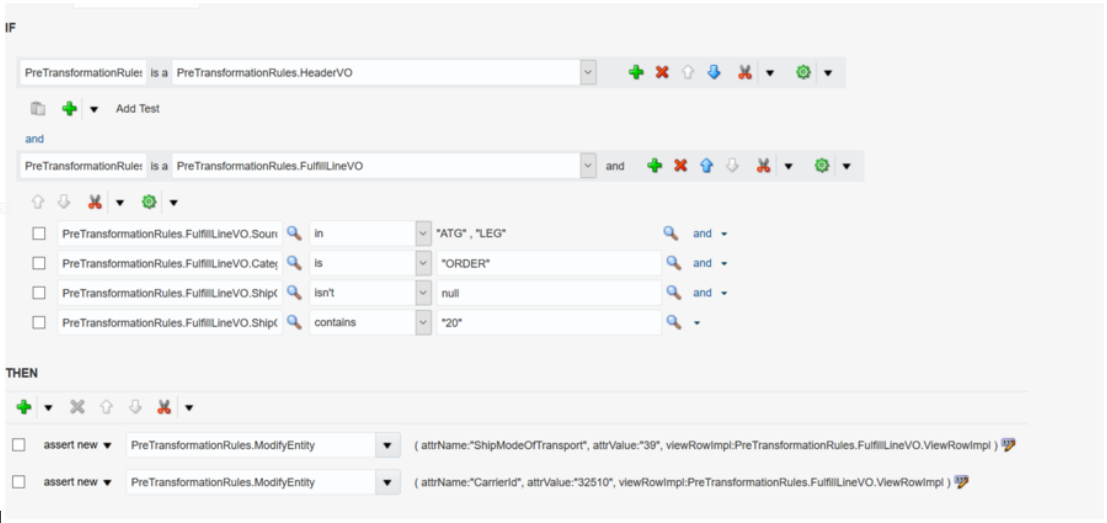
Name	Type	Value	Consta
attrName	String	"ShippingInstructions"	
attrValue	Object	"Rule Test"	
viewRowImpl	PreTransformationRules...	PreTransformationRules.FulfillLineVO.Vi	

OK Cancel

THEN
assert new PreTransformationRules.ModifyEntity (attrName:"ScheduleShipDate", attrValue:PreTransformationRules.FulfillLineVO.RequestShipDate, viewRowImpl:PreTransformationRules.FulfillLineVO.ViewRowImpl)
assert new PreTransformationRules.ModifyEntity (attrName:"ShippingInstructions", attrValue:"Rule Test", viewRowImpl:PreTransformationRules.FulfillLineVO.ViewRowImpl)

Test Tax Determinant

Shipping Method Defaulting Rule1

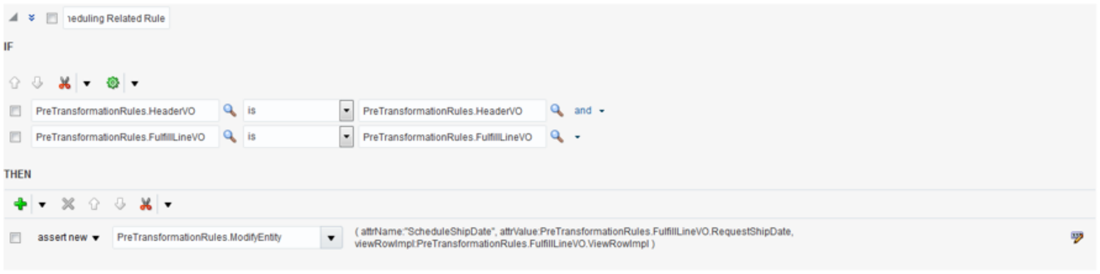


IF
PreTransformationRules is a PreTransformationRules.HeaderVO
Add Test
and
PreTransformationRules is a PreTransformationRules.FulfillLineVO and

PreTransformationRules.FulfillLineVO.Sour	in	"ATG" , "LEG"	and
PreTransformationRules.FulfillLineVO.Categ	is	"ORDER"	and
PreTransformationRules.FulfillLineVO.Ship(	isn't	null	and
PreTransformationRules.FulfillLineVO.Ship(	contains	"20"	

THEN
assert new PreTransformationRules.ModifyEntity (attrName:"ShipModeOfTransport", attrValue:"39", viewRowImpl:PreTransformationRules.FulfillLineVO.ViewRowImpl)
assert new PreTransformationRules.ModifyEntity (attrName:"CarrierId", attrValue:"32510", viewRowImpl:PreTransformationRules.FulfillLineVO.ViewRowImpl)

Scheduling Related Rule



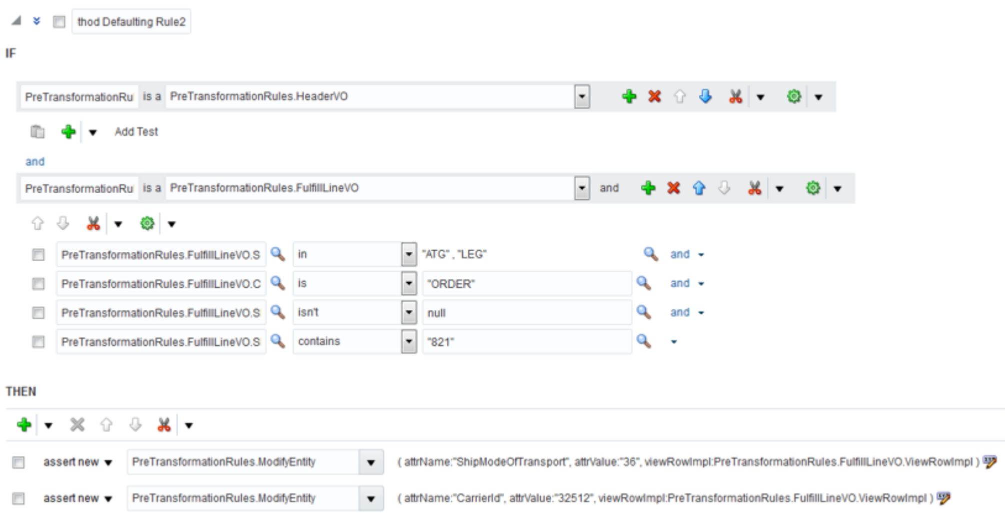
ıeduling Related Rule

IF

PreTransformationRules.HeaderVO	is	PreTransformationRules.HeaderVO	and
PreTransformationRules.FulfillLineVO	is	PreTransformationRules.FulfillLineVO	

THEN
assert new PreTransformationRules.ModifyEntity (attrName:"ScheduleShipDate", attrValue:PreTransformationRules.FulfillLineVO.RequestShipDate, viewRowImpl:PreTransformationRules.FulfillLineVO.ViewRowImpl)

Shipping Method Defaulting Rule2


thod Defaulting Rule2

IF

PreTransformationRu is a PreTransformationRules.HeaderVO

Add Test

and

PreTransformationRu is a PreTransformationRules.FulfillLineVO and

PreTransformationRules.FulfillLineVO.S	in	"ATG" , "LEG"	and
PreTransformationRules.FulfillLineVO.C	is	"ORDER"	and
PreTransformationRules.FulfillLineVO.S	isn't	null	and
PreTransformationRules.FulfillLineVO.S	contains	"821"	

THEN

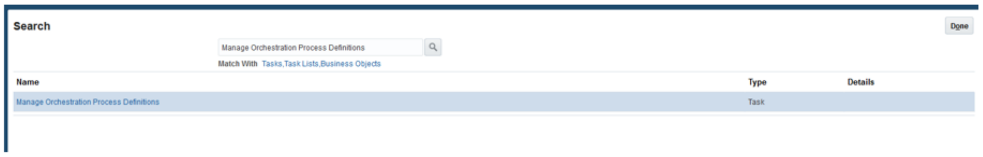
assert new PreTransformationRules.ModifyEntity (attrName:"ShipModeOfTransport", attrValue:"36", viewRowImpl:PreTransformationRules.FulfillLineVO.ViewRowImpl)

assert new PreTransformationRules.ModifyEntity (attrName:"CarrierId", attrValue:"32512", viewRowImpl:PreTransformationRules.FulfillLineVO.ViewRowImpl)

Create the orchestration process

This topic describes how to create the orchestration process for the integration.

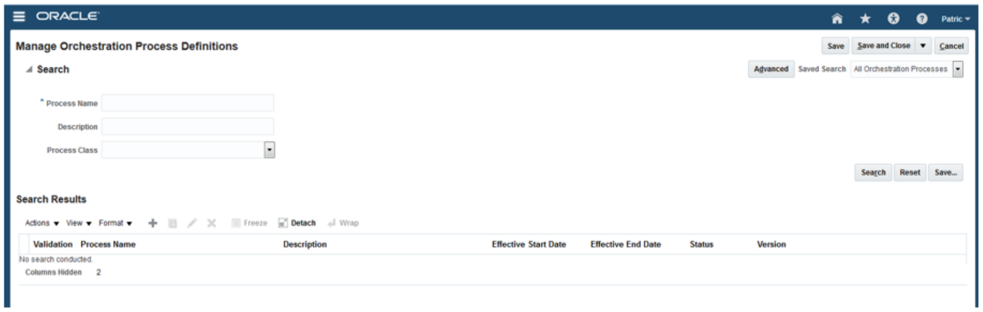
1. Go to the FSM link `https://example.com/setup/faces/TaskListManagerTop` and create the orchestration process:


Search

Manage Orchestration Process Definitions

Match With Tasks,Task Lists,Business Objects

Name	Type	Details
Manage Orchestration Process Definitions	Task	


Manage Orchestration Process Definitions

Search

Saved Search All Orchestration Processes

* Process Name

Description

Process Class

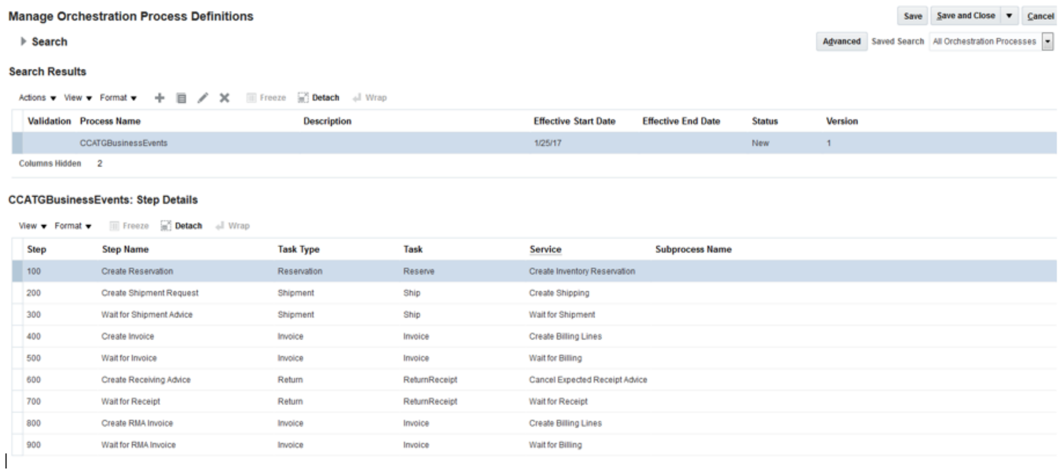
Search Results

Validation	Process Name	Description	Effective Start Date	Effective End Date	Status	Version

No search conducted.

Columns Hidden 2

2. Search for Commerce in the **Process Name** field.



Manage Orchestration Process Definitions

Search

Search Results

Validation	Process Name	Description	Effective Start Date	Effective End Date	Status	Version
	CCATGBusinessEvents		1/25/17		New	1

Columns Hidden 2

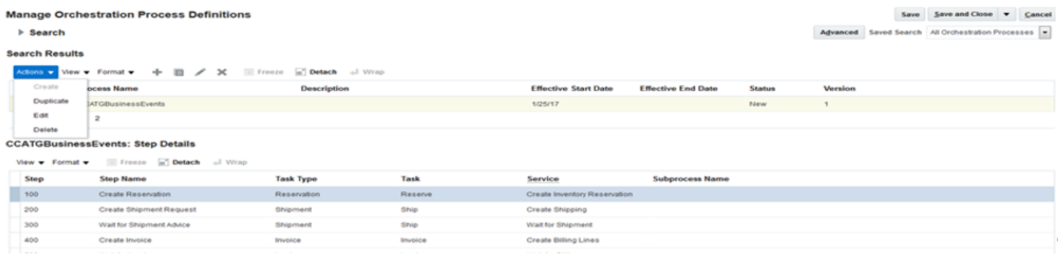
CCATGBusinessEvents: Step Details

Step	Step Name	Task Type	Task	Service	Subprocess Name
100	Create Reservation	Reservation	Reserve	Create Inventory Reservation	
200	Create Shipment Request	Shipment	Ship	Create Shipping	
300	Wait for Shipment Advice	Shipment	Ship	Wait for Shipment	
400	Create Invoice	Invoice	Invoice	Create Billing Lines	
500	Wait for Invoice	Invoice	Invoice	Wait for Billing	
600	Create Receiving Advice	Return	ReturnReceipt	Cancel Expected Receipt Advice	
700	Wait for Receipt	Return	ReturnReceipt	Wait for Receipt	
800	Create RMA Invoice	Invoice	Invoice	Create Billing Lines	
900	Wait for RMA Invoice	Invoice	Invoice	Wait for Billing	

A predefined process, `CCATGBusinessEvents`, appears.

The process can be created if necessary.

The following screens show the orchestration details:



Manage Orchestration Process Definitions

Search

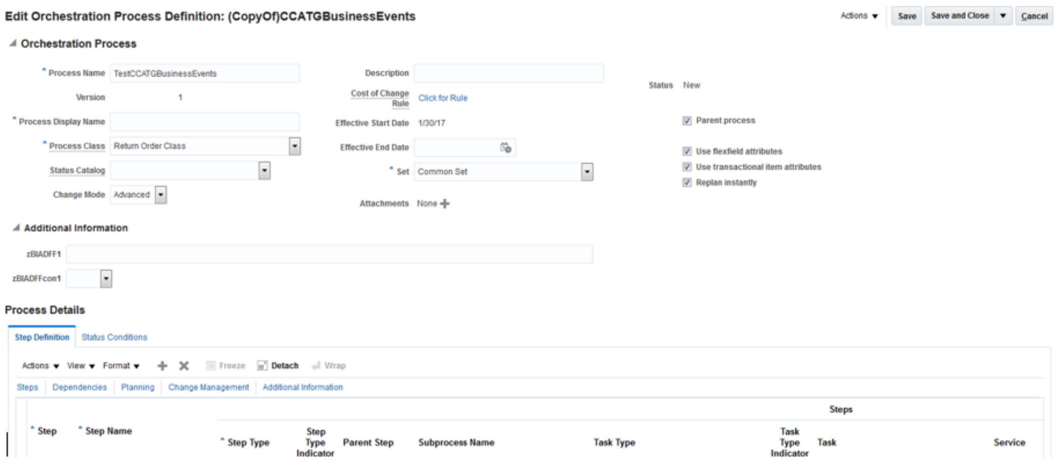
Search Results

Actions: Create, Duplicate, Edit, Delete

Process Name	Description	Effective Start Date	Effective End Date	Status	Version
CCATGBusinessEvents		1/25/17		New	1

CCATGBusinessEvents: Step Details

Step	Step Name	Task Type	Task	Service	Subprocess Name
100	Create Reservation	Reservation	Reserve	Create Inventory Reservation	
200	Create Shipment Request	Shipment	Ship	Create Shipping	
300	Wait for Shipment Advice	Shipment	Ship	Wait for Shipment	
400	Create Invoice	Invoice	Invoice	Create Billing Lines	



Edit Orchestration Process Definition: (CopyOf)CCATGBusinessEvents

Orchestration Process

* Process Name: TestCCATGBusinessEvents
Version: 1
* Process Display Name:
* Process Class: Return Order Class
Status Catalog:
Change Mode: Advanced
Description:
Cost of Change Rule: Click for Rule
Effective Start Date: 1/30/17
Effective End Date:
* Set: Common Set
Attachments: None
Status: New
✓ Parent process
✓ Use flexfield attributes
✓ Use transactional item attributes
✓ Replan instantly

Additional Information

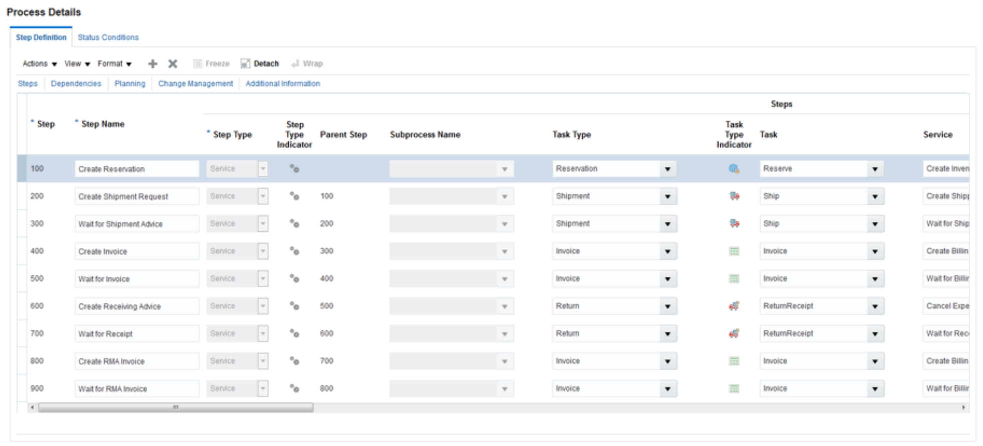
zBIADFF1:
zBIADFFcon1:

Process Details

Step Definition | Status Conditions

Steps | Dependencies | Planning | Change Management | Additional Information

		Steps						
* Step	* Step Name	* Step Type	Step Type Indicator	Parent Step	Subprocess Name	Task Type	Task Type Indicator	Task	Service



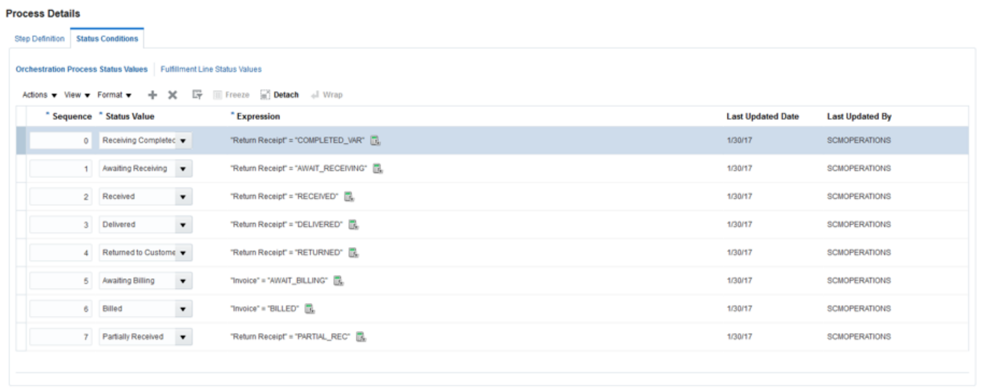
Process Details

Step Definition | Status Conditions

Actions ▾ View ▾ Format ▾ Freeze Detach Wrap

Steps | Dependencies | Planning | Change Management | Additional Information

* Step	* Step Name	* Step Type	Step Type Indicator	Parent Step	Subprocess Name	Task Type	Task Type Indicator	Task	Service
								Steps	
100	Create Reservation	Service				Reservation		Reserve	Create Inven
200	Create Shipment Request	Service		100		Shipment		Ship	Create Shipi
300	Wait for Shipment Advice	Service		200		Shipment		Ship	Wait for Ship
400	Create Invoice	Service		300		Invoice		Invoice	Create Billin
500	Wait for Invoice	Service		400		Invoice		Invoice	Wait for Billir
600	Create Receiving Advice	Service		500		Return		ReturnReceipt	Cancel Expe
700	Wait for Receipt	Service		600		Return		ReturnReceipt	Wait for Rec
800	Create RMA Invoice	Service		700		Invoice		Invoice	Create Billin
900	Wait for RMA Invoice	Service		800		Invoice		Invoice	Wait for Billir



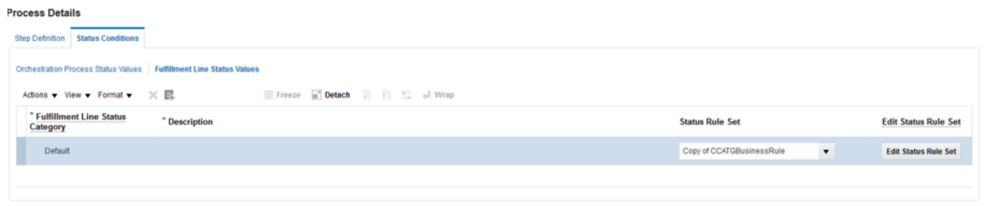
Process Details

Step Definition | Status Conditions

Orchestration Process Status Values | Fulfillment Line Status Values

Actions ▾ View ▾ Format ▾ Freeze Detach Wrap

* Sequence	* Status Value	* Expression	Last Updated Date	Last Updated By
0	Receiving Completec	"Return Receipt" = "COMPLETED_VAR"	1/30/17	SCMOPERATIONS
1	Awaiting Receiving	"Return Receipt" = "AWAIT_RECEIVING"	1/30/17	SCMOPERATIONS
2	Received	"Return Receipt" = "RECEIVED"	1/30/17	SCMOPERATIONS
3	Delivered	"Return Receipt" = "DELIVERED"	1/30/17	SCMOPERATIONS
4	Returned to Custome	"Return Receipt" = "RETURNED"	1/30/17	SCMOPERATIONS
5	Awaiting Billing	"Invoice" = "AWAIT_BILLING"	1/30/17	SCMOPERATIONS
6	Billed	"Invoice" = "BILLED"	1/30/17	SCMOPERATIONS
7	Partially Received	"Return Receipt" = "PARTIAL_REC"	1/30/17	SCMOPERATIONS



Process Details

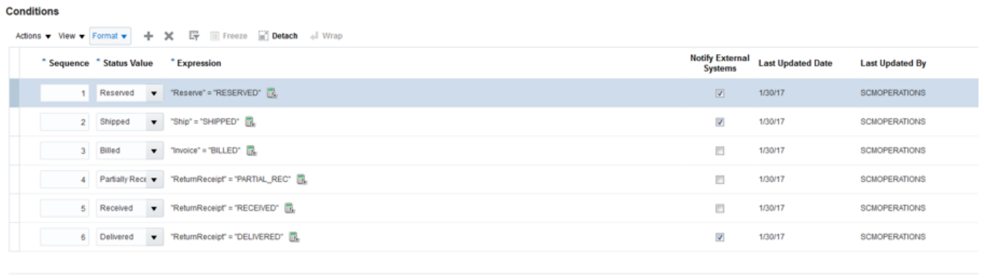
Step Definition | Status Conditions

Orchestration Process Status Values | Fulfillment Line Status Values

Actions ▾ View ▾ Format ▾ Freeze Detach Wrap

* Fulfillment Line Status Category	* Description	Status Rule Set	Edit Status Rule Set
Default		Copy of CCATGBusinessRule	Edit Status Rule Set

3. Click **Edit Status Rule Set**.



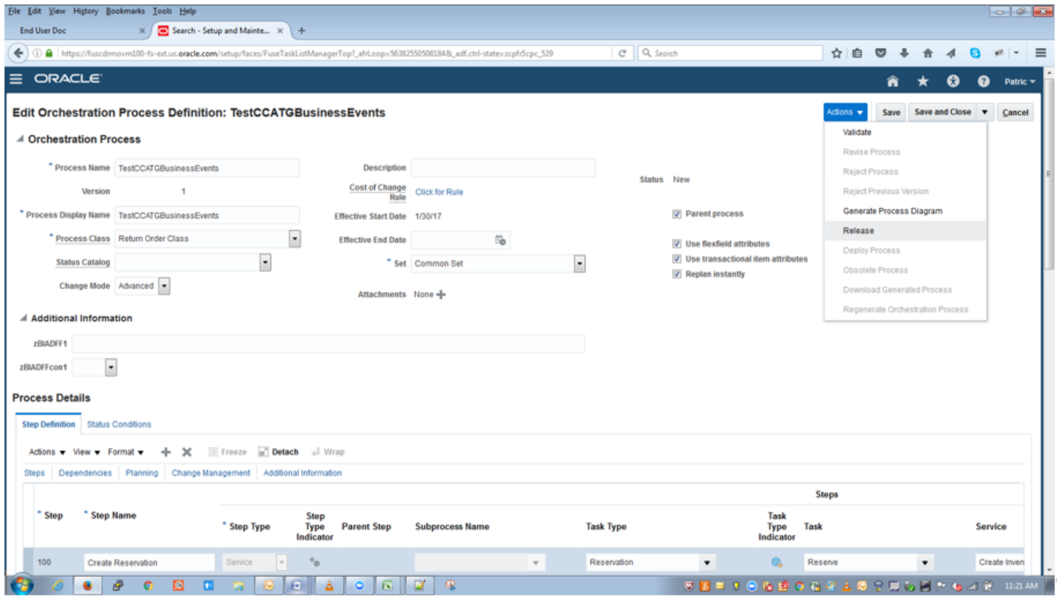
Conditions

Actions ▾ View ▾ Format ▾ Freeze Detach Wrap

* Sequence	* Status Value	* Expression	Notify External Systems	Last Updated Date	Last Updated By
1	Reserved	"Reserve" = "RESERVED"	✓	1/30/17	SCMOPERATIONS
2	Shipped	"Ship" = "SHIPPED"	✓	1/30/17	SCMOPERATIONS
3	Billed	"Invoice" = "BILLED"	☐	1/30/17	SCMOPERATIONS
4	Partially Rece	"ReturnReceipt" = "PARTIAL_REC"	☐	1/30/17	SCMOPERATIONS
5	Received	"ReturnReceipt" = "RECEIVED"	☐	1/30/17	SCMOPERATIONS
6	Delivered	"ReturnReceipt" = "DELIVERED"	✓	1/30/17	SCMOPERATIONS

4. Save the details.

5. On the **Actions** menu, click **Release**.



Edit Orchestration Process Definition: TestCCATGBusinessEvents

Orchestration Process

* Process Name	TestCCATGBusinessEvents	Description	
Version	1	Cost of Change Rule	Click for Rule
* Process Display Name	TestCCATGBusinessEvents	Effective Start Date	1/30/17
* Process Class	Return Order Class	Effective End Date	
Status Catalog		* Set	Common Set
Change Mode	Advanced	Attachments	None

Status New

- [✓] Parent process
- [✓] Use flexfield attributes
- [✓] Use transactional item attributes
- [✓] Replan instantly

Additional Information

zBIADFF1	
zBIADFFcon1	

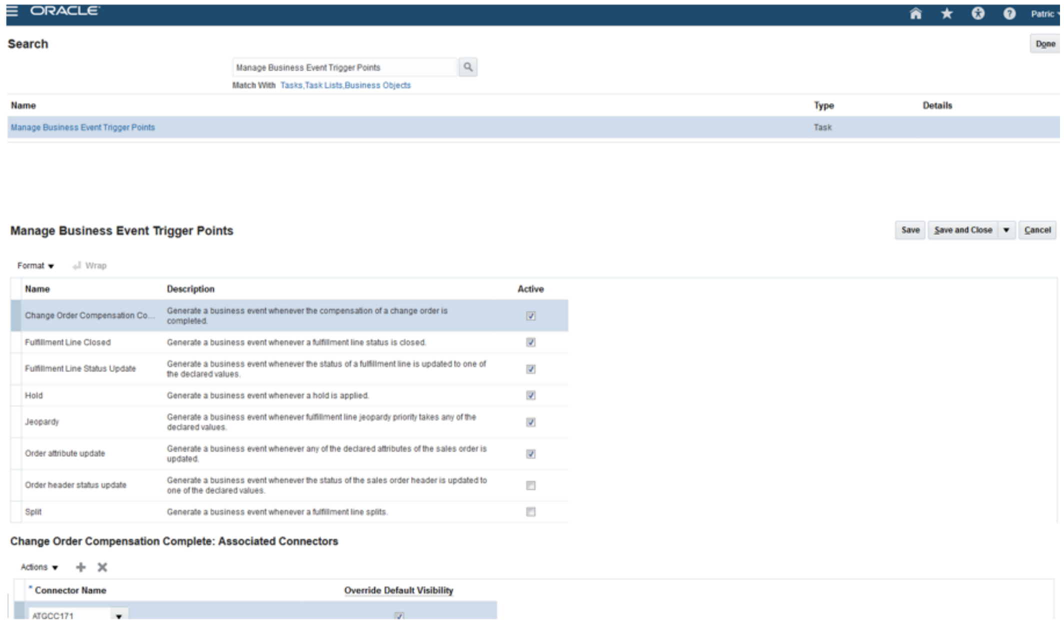
Process Details

				Steps					
* Step	* Step Name	* Step Type	Step Type Indicator	Parent Step	Subprocess Name	Task Type	Task Type Indicator	Task	Service
100	Create Reservation	Service				Reservation		Reserve	Create Inven

Create the connector

Creating a connector with the Oracle Integration Cloud URL is a required step for the integration.

1. Go to the FSM link `https://example.com/setup/faces/TaskListManagerTop` and search for Manage Business Event Trigger Points.



Search

Manage Business Event Trigger Points

Match With Tasks,Task Lists,Business Objects

Name	Type	Details
Manage Business Event Trigger Points	Task	

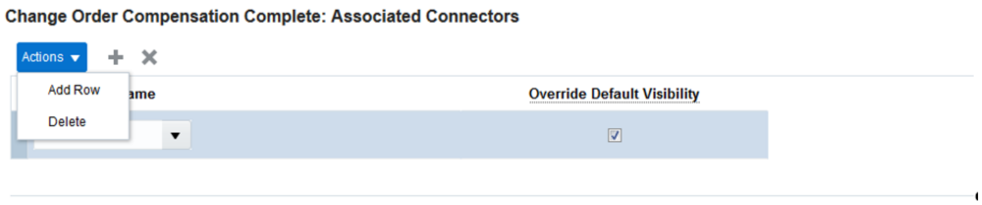
Manage Business Event Trigger Points

Name	Description	Active
Change Order Compensation Co...	Generate a business event whenever the compensation of a change order is completed.	✓
Fulfillment Line Closed	Generate a business event whenever a fulfillment line status is closed.	✓
Fulfillment Line Status Update	Generate a business event whenever the status of a fulfillment line is updated to one of the declared values.	✓
Hold	Generate a business event whenever a hold is applied.	✓
Jeopardy	Generate a business event whenever fulfillment line jeopardy priority takes any of the declared values.	✓
Order attribute update	Generate a business event whenever any of the declared attributes of the sales order is updated.	✓
Order header status update	Generate a business event whenever the status of the sales order header is updated to one of the declared values.	☐
Split	Generate a business event whenever a fulfillment line splits.	☐

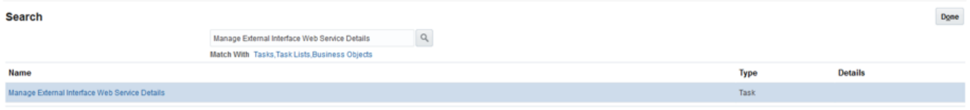
Change Order Compensation Complete: Associated Connectors

* Connector Name	Override Default Visibility
ATGCC171	✓

2. Ensure that the fulfillment tabs are enabled.


Change Order Compensation Complete: Associated Connectors
Actions
Add Row
Delete
ame
Override Default Visibility

3. Pick the connector to enable.



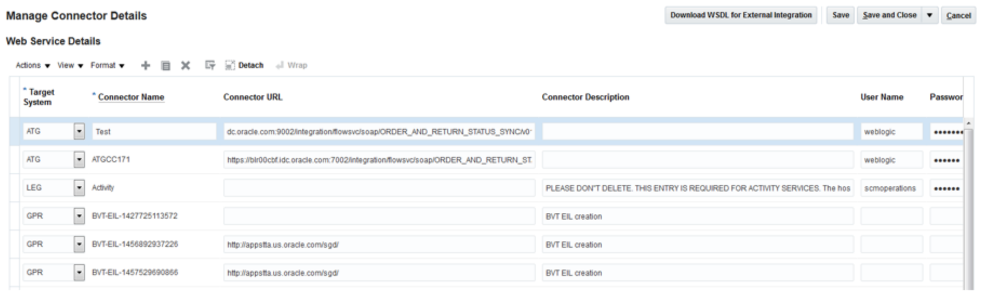
Search

Manage External Interface Web Service Details

Match With Tasks,Task Lists,Business Objects

Name	Type	Details
Manage External Interface Web Service Details	Task	

4. Create a name for the connector and provide the Oracle Integration Cloud integration details in the connector URL.

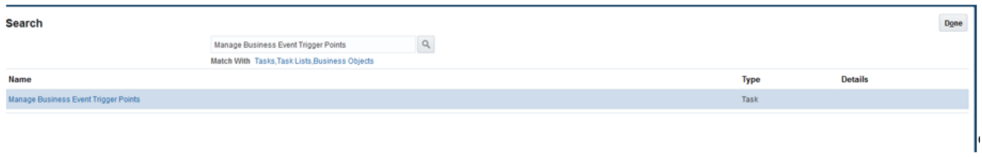


Manage Connector Details

Web Service Details

* Target System	* Connector Name	Connector URL	Connector Description	User Name	Passwor
ATG	Test	dc.oracle.com:9002/integration/flowsvc/soap/ORDER_AND_RETURN_STATUS_SYNC/v0		weblogic	•••••••
ATG	ATGCC171	https://blr00cbf.idc.oracle.com:7002/integration/flowsvc/soap/ORDER_AND_RETURN_ST		weblogic	••••••
LEG	Activity		PLEASE DON'T DELETE. THIS ENTRY IS REQUIRED FOR ACTIVITY SERVICES. The hos	scmoperations	••••••
GPR	BVT-EIL-1427725113572		BVT EIL creation		
GPR	BVT-EIL-1456892937226	http://appstta.us.oracle.com/sgd/	BVT EIL creation		
GPR	BVT-EIL-1457529690866	http://appstta.us.oracle.com/sgd/	BVT EIL creation		

5. Verify.



Search

Manage Business Event Trigger Points

Match With Tasks,Task Lists,Business Objects

Name	Type	Details
Manage Business Event Trigger Points	Task	

Order creation

Order creation in Commerce requires no change in the functional flow with respect to this integration. In general, orders created in Commerce are sent to Order Management Cloud for fulfillment. Several special cases are listed in this section.

The mechanism used to send the orders created in Commerce is webhooks.

Note: Commerce provides a remorse period during which shoppers can edit their orders. The orders are sent to Order Management Cloud for fulfillment as soon as the remorse period ends.

SKUs

Commerce supports simple SKUs. It can also support configurable SKUs through an integration with Oracle CPQ.

As part of this integration, simple SKUs are sent to the fulfillment system. Configurable SKUs are not sent to the integration workflows. However, configurable SKUs are available to the webhook and can map to Order Management Cloud attributes if required. A separate integration exists for Oracle CPQ and Order Management Cloud.

Payment

Order pricing and payment authorization happen in Oracle Commerce. The payment attributes are not mapped to any attribute in Order Management Cloud because the order management system does not need the payment information for fulfillment. With account-based (B2B) shopping, the order does have a purchase order number that is passed on to the order management system.

Note: The scope of this integration is restricted to Order Management Cloud. Other Oracle services can perform additional order processing. For example, Order Management Cloud can integrate with Account Receivables post fulfillment; this functionality is outside the scope of this integration.

Commerce currently expects the payment capture information to be updated by an external system. This assumption holds true for this integration as well. The order payment is authorized by Commerce via the payment gateways before the order is sent to Order Management for fulfillment.

It is recommended that you verify that the order is ready to be shipped in Order Management and capture the money against the authorization. This can be done by integrating with the payment gateways and sending the updated status back to Commerce as captured/settled. The payment group status would have to be updated once the transaction is captured in the Commerce repository. This status unavailability hinders return processing against an order.

Order types

Orders in Commerce can be placed for consumer-based (B2C) and account-based (B2B) shopping.

These orders, when triggered from Commerce, need to be fed to the order management system. This integration handles both types of order and maps the attributes of Commerce and Order Management Cloud accordingly.

Pricing and tax

Pricing and tax calculations are done in Commerce and the order is passed to the order management system. The orders created in Order Management Cloud are not synched into Commerce.

The order creation integration workflow can pass the order created in Commerce to Order Management Cloud for fulfillment using the Order Submit without Payment Details webhook, which is triggered after the remorse period in Commerce.

Shipping methods

Commerce shipping methods are mapped to Oracle Order Management Cloud shipping methods. The mappings are made using the Lookup service of Oracle Integration Cloud.

The shipping methods are sent to Order Management Cloud as part of order creation. If you create a new shipping method in Commerce, map it to a shipping method in Order Management Cloud. These mappings ensure that the orders created in Oracle Commerce always use the shipping method chosen by the customer.

Shipping methods are creating in the Shipping Methods tab, which is described in Work with accounts. For information about Order Management Cloud shipping methods, refer to the Oracle Order Management Cloud documentation.

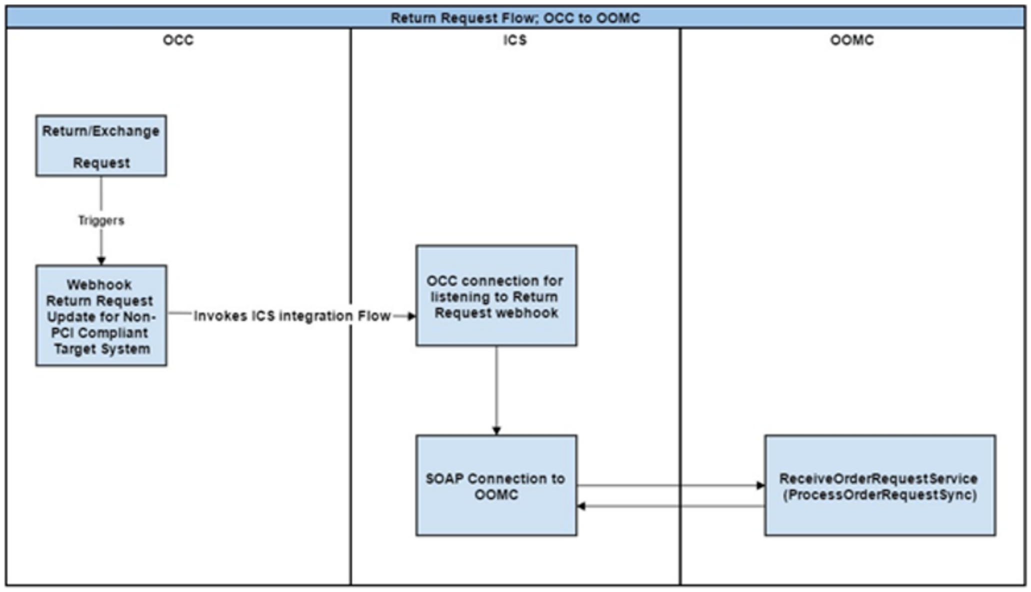
Mapping of attributes

The attribute mapping of Commerce and Order Management Cloud is done through Oracle Integration Cloud. There could be some additional processing that is needed on the attributes before pushing the order to Order Management Cloud.

This additional processing of attributes can be can be built into the integration layer without a need to modify Commerce. You can build an XSLT which acts on top of the Oracle Integration Cloud UI and provides additional flexibility. A sample XSLT is also provided as part of the integration package. The XSLT can be uploaded into any integration in Oracle Integration Cloud as soon as the integration has been deactivated.

The location of the XSLT for order creation is `ICSpackage9a47f2ae-5f90-4cf7-a87f-cb612dda70d1.iar/icspackage/project/ORDE_SYNC_FROM_COMM_TO_ORDE_MGMT_01.00.0000/resources/processor_859/resourcegroup_755/req_19fd482efc0a4728b838273aec72b649.xsl`.

The following figure illustrates the order creation workflow from Commerce to Order Management Cloud:


Return Request Flow; OCC to OOMC
OCC
ICS
OOMC
Return/Exchange Request
Triggers
Webhook Return Request Update for Non-PCI Compliant Target System
Invokes ICS integration Flow
OCC connection for listening to Return Request webhook
SOAP Connection to OOMC
ReceiveOrderRequestService (ProcessOrderRequestSync)

Order Status

The status of an order is updated in Commerce by Order Management Cloud. Order Management Cloud provides notifications on the status changes of the order.

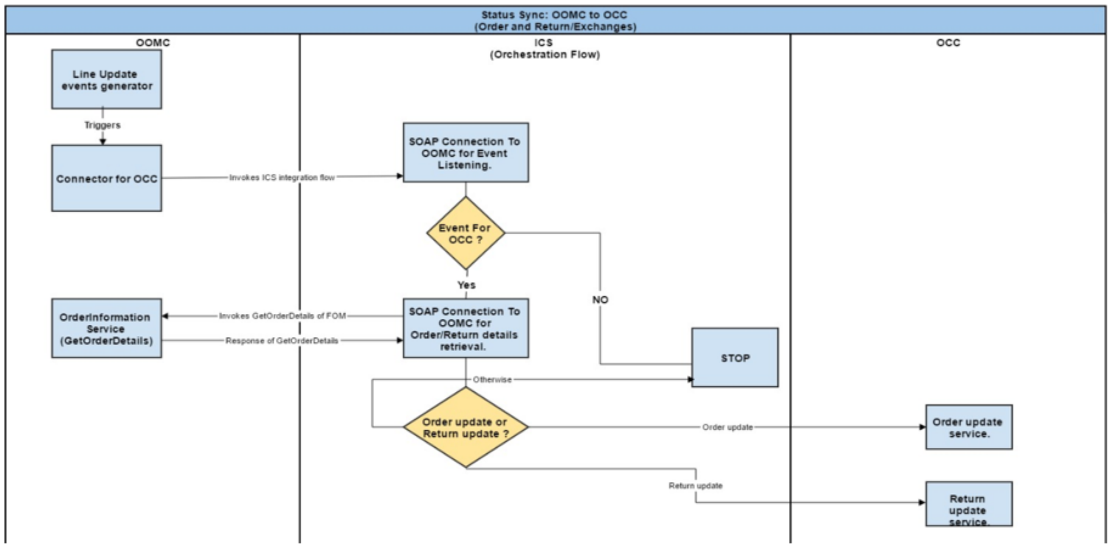
As part of this integration, the shipment group status is updated for the order as No Pending Action as soon as the shipment is closed in Order Management. The tracking details of the order are also updated in Commerce as and when they are available in Order Management. The payment group status for the order is set to Authorized.

The order in Commerce is set to Submitted for fulfillment. You should update the payment group status details based on your integrations with payment gateways or using the capabilities of products such as Oracle Fusion Payments.

Map attributes for order status

The following is the location of the XSL transformation:
`icspackage58e6510b-1e7f-491e-903c-bc59e1062bfc.iar/icspackage/project/ORDER_AND_RETURN_STATUS_SYNC_01.00.0000/resources/processor_837/resourcegroup_173/req_f7a1d4886a4b4ca3935231d5421b27ec.xsl`

The following diagram illustrates the status workflow:


Status Sync: OOMC to OCC
(Order and Return/Exchanges)
OOMC
ICS
(Orchestration Flow)
OCC
Line Update events generator
Triggers
Connector for OCC
Invokes ICS integration flow
SOAP Connection To OOMC for Event Listening.
Event For OCC ?
Yes
NO
OrderInformation Service (GetOrderDetails)
Invokes GetOrderDetails of FOM
Response of GetOrderDetails
SOAP Connection To OOMC for Order/Return details retrieval.
STOP
Otherwise
Order update or Return update ?
Order update
Order update service.
Return update
Return update service.

Returns

The returns created in Commerce are passed to Order Management Cloud for fulfillment.

Note: All the prerequisites needed for submitting a return on Commerce are valid for this integration.

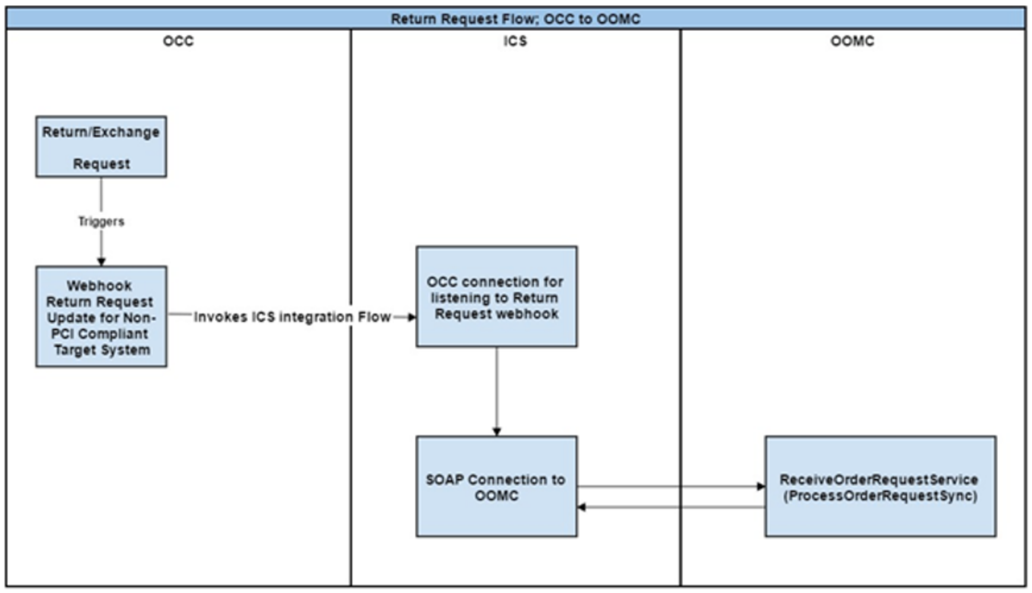
As soon as a return is created by Commerce, a new endpoint is created for processing the return and submitting the return order to Order Management Cloud. Because the reason code is mandatory when the return is created in Order Management Cloud, as part of the return submission process, the disposition code of Commerce is mapped to Return with Refund.

As soon as the return is processed by the Order Management Cloud, the disposition code is updated with the correct value as sent by Order Management Cloud.

Map attributes for returns

The following is the location of the XSL transformation for order returns:
`icspackage58e6510b-1e7f-491e-903c-bc59e1062bfc.iar/icspackage/project/ORDER_AND_RETURN_STATUS_SYNC_01.00.0000/resources/processor_943/resourcegroup_124/req_52e0e60451ef4c66931157e8ea6172ad.xsl`

The following diagram illustrates the workflow for returns:


Return Request Flow: OCC to OOMC
OCC
ICS
OOMC
Return/Exchange Request
Triggers
Webhook Return Request Update for Non-PCI Compliant Target System
Invokes ICS integration Flow
OCC connection for listening to Return Request webhook
SOAP Connection to OOMC
ReceiveOrderRequestService (ProcessOrderRequestSync)

Exchanges

This topic describes exchange processing for this integration.

- The exchange request is created along with a new order for the exchange.
- The exchange and the order are submitted to Order Management Cloud as part of this integration.
- The exchange request is submitted to Order Management Cloud for receiving the goods.
- When the status of the exchange is set to Received, the new order created as part of the exchange is submitted to Order Management Cloud for fulfillment.

The flows for the return and the order creation are the same as described in the previous sections. The order is submitted to Order Management only when the return information is available in Commerce. The flows for the exchange are the same as previously described.

8

Manage Page Tags

Oracle Commerce lets you create script tags and embed them in each page on a site. Business users can create, manage, and place tags without the need for a developer to make code changes to the storefront.

Page tags are HTML script tags that are embedded in every storefront page. They let you add both Oracle and third-party plug-in services (such as analytics, chat services, or A/B testing) to your storefront.

Understand page tags

Oracle Commerce lets you create page tags and embed them in each page on a site. Business users can create, manage, and decide where to place tags without the need for a developer to make code changes to the storefront.

Page tags are HTML `<script>` tags that are embedded in every storefront page. They let you add both Oracle and third-party plug-in services to your storefront. For example, you can use page tags to add analytics services, like Google Analytics.

Using page tags to embed content in storefront pages can improve performance vs. adding the content with global widgets.

Note that there is no need to configure CORS support in Commerce for page tags. Page tags allow the shopper's browser to make calls to other domains like Google Analytics when the browser is displaying content loaded from a Commerce domain. You only need to configure CORS support in Commerce when the shopper's browser accesses your Commerce site after displaying a page loaded from the external domain. You may, however, need to configure CORS support in the service whose plug-ins you are adding to your Commerce site.

To learn how to use the administration interface to add and manage the page tags for a site, see Create and edit page tags.

Create and edit page tags

Use the administration interface to create, edit, delete, and enable page tags.

You can access the Page Tags page only if your profile has privileges assigned that let you access Settings pages in the administration interface.

Create a page tag

To create a page tag in the administration interface:

1. Click the **Settings** icon, then select **Page Tags**.
2. Click the **New Script Tag** button for the section where you want to embed the tag on each page (**Head**, **Beginning of Body**, or **End of Body**).
3. Enter a name for the tag.
   The name is used internally by Commerce and is never displayed on the storefront.

The name must be unique and can be up to 50 characters long.

4. Enter the script content. You will usually get this content from the application you are integrating with and simply paste it in here. Keep the following in mind when you add the script:
   The script can be up to 3,000 characters long.

   The Script Tag Editor validates the structure of a tag but it does not validate the actual content of the tag.
5. Tags are not enabled by default. To enable the tag, click the **Enable** checkbox. You do not have to publish tags.
6. Click **Save**.
7. Once a tag is successfully created, it appears in a list below the section where you created it. You can edit a tag by clicking its name in the list. You can change its name and content, and enable or disable it. Click **Save** when you finish your changes.

Reorder tags in a section

You can reorder tags within each section (head, beginning of body, and end of body) to control the order of insertion of the tag content. To move a tag to the top of a section's list, click the tag's **Move to Top** arrow. To move a tag, click its **Drag and Drop** icon and drag it to a new position in the section.

Delete a tag

Deleting a tag removes it from the system. To keep a tag but stop using it, simply edit the tag and uncheck its **Enable** checkbox. To delete a tag, click its **Delete** icon and then confirm that you want to delete the tag.

9

Integrate with Oracle Product Hub Cloud

Oracle Commerce provides an integration with Oracle Product Hub Cloud (PHC) that you can use to provide a robust commerce architecture for order capture and product management.

Oracle Cloud Product Hub, part of Oracle Cloud PLM suite, is an enterprise-class product master data management (PMDM) software, delivered via cloud for lower cost and faster deployment.

Understand the Product Hub integration

Read this section to learn concepts that are important to know before you configure and use the integration between Oracle Commerce and Oracle Product Hub Cloud.

This section describes concepts to know before you configure and use the integration between Commerce and Oracle Product Hub Cloud. It includes the following topics:

- Audience
- Overview
- Prerequisites
- Assumptions

Audience

This section is written for Commerce and Product Hub administrators who want to set up and configure the integration between these two systems. To use this documentation, you should have experience with Commerce, Oracle Integration Cloud (OIC), and Product Hub. This section does not provide any instructions for configuring any other aspects of these systems beyond those required for the integration. For information on other configurations, refer to each product's documentation, available on the Oracle Help Center.

Overview

This integration provides the following features:

- Syncs products created in Product Hub to Commerce
- Associate collections created in Commerce to the products
- Optionally trigger a publish event in Commerce when import is complete
- Upload images from Product Hub to Commerce and associate them with products

Sync products and SKUs: This integration assumes that products and SKUs are created and maintained in Product Hub. The integration syncs them to Commerce and make them available on the storefront, as described by the following process:

- The item publication job can be scheduled in Product Hub with a required frequency, for example once per day or once every six hours. It exports new and updated items based on the defined filter criteria. The job runs as specified by the schedule, and on completion, it posts the exported files to Oracle Universal Content Management (UCM) and triggers the item-publication job event.

- The item publication job event triggers the OIC flow. It checks if a publish event is currently running in Commerce. If Commerce is publishing or if another import or export job is active, then it cannot accept new requests for import. In these cases, the integration execution stops and is moved to a queue to be retried at a later time. When publishing completes, the integration is resumed from the queue.
- The integration downloads the exported file from UCM. The archive file may contain multiple XML files, which are transformed into Commerce JSON format. The integration archives the JSON files and uploads them to Commerce. It then triggers the bulk product import process to load the data into the Commerce Admin server.

Publish imported items: After the import is successful, the integration flow checks if auto-publish is enabled in the OIC lookup. If enabled, it gets the total number of records pending publishing in Commerce for the user/application configured in the OIC Commerce connection. If it finds any such record and the number of such records is less than the threshold configured in OIC lookup, it publishes those records. Note that the publish operation includes all the records for the user, not only the ones imported as part of the current integration flow. If the threshold exceeds 10 MB, it sends an email notification to the administrator to inform that publish was not initiated automatically. A Commerce admin user can manually start a publish in this case.

Upload images: If Media Sync is enabled (that is, if `CXCommerceMediaSyncEnabled` is set to `true` in OIC lookups), then media items are also synced along with the items/ products from Product Hub to Commerce. Supporting formats for images to be linked with products in Commerce are JPG, JPEG, PNG, and GIF. Separate ZIP file will be created with all the images of supported formats and then uploaded to Commerce. For more information about the rules Commerce enforces for images, see Manage Media for Your Store in *Using Oracle Commerce*.

Prerequisites

Configuring and using this integration requires the following. If you require one or more of these, please contact an Oracle sales representative.

- A Commerce account and access to Commerce 20D or later.
- An Oracle Fusion Product Hub account and access to Product Hub Cloud 20B or later.
- An Oracle Integration Cloud account and access to Oracle Integration Cloud Service.

Assumptions

This integration makes the following functional assumptions. These assumptions require a functional understanding of both Commerce and Product Hub.

- Catalogs, collections, and product types are created in Commerce before the products are imported into Commerce by the integration.
- Products and SKUs data is always managed in Oracle Product Hub and is imported into Commerce. This means that Product Hub is the only source of products and SKUs for Commerce.
- By default, Commerce list prices are mapped to Product Hub's purchase list prices. It's recommended to create an Extensible Flexfield (EFF) attribute for sale list price and override the mapping.
- All changes are published on Commerce, once the products are imported.

- Configurable SKUs, add-on products, related SKUs, services and subscriptions are not supported by default in this integration. You will need to extend the integration if you wish to support these components.
- PDH Item Number is mapped to Commerce Product ID and SKU ID. While Item Number can include spaces and have a maximum length of 256 characters, Commerce IDs do not support spaces and can have a maximum length of 165 characters. Therefore, appropriate rules must be set in PDH to conform data to Commerce requirements. Also, the Item Number must not be modified in PDH.
- Translations to secondary locales are not supported by default in this integration. If your Commerce environment requires translations, this requires multiple exports to be triggered in PDH in different locales, and any extension to this integration must handle them accordingly in the integration flow.
- You can customize this integration to link Images to SKUs and to map an image's path and name under SKUs. Before you can do this, make sure that the `productType` is mapped to a Commerce product type for which Allow product images at the variant property value level is enabled in the Variant properties.

Configure Oracle Commerce with Oracle Product Hub

This section describes tasks you must perform to configure Commerce for the integration.

You perform these tasks in the Commerce administration interface and with the Commerce REST APIs.

Register the application and generate a security token

This integration uses the Commerce REST APIs to access Commerce data. You must register the integration within Commerce and generate a security token in order for the integration to be granted access to the data.

To generate a security token:

1. Log into the Commerce administration interface.
2. Click the Settings menu and select Web APIs.
3. Click Registered Applications from the Web APIs panel.
4. Click the **Register Application** button.
5. Enter a name for the integration application. Create a meaningful name that reflects the purpose of the application.
6. Click **Save**. The Application ID and Application Key are automatically generated and the application is added to the Registered Applications page.
7. Click on the name of the application you created.
8. Click to reveal link to display the application key. You can copy the application key to use as the security token for the Oracle Commerce connection.

For more information on managing an application within Commerce, see Register applications.

Configure the Commerce webhooks

You must configure the Publish Complete and Import Complete webhooks.. Follow these steps to configure the webhooks in the Commerce administration interface:

1. Log into the Commerce administration interface.
2. Click the **Settings** icon.
3. Click Web APIs and then click the Webhook tab.
4. Click the Publish Complete (Production) webhook. Enter the integration (Oracle Commerce OIC ProductHubInt Resubmit Webhook) endpoint URL in the URL box and enter the OIC username and password, under Basic Authorization.
5. Click the Import Complete (Production) webhook. Enter the integration (Oracle Commerce OIC ImportComplete Post Processing) endpoint URL in the URL box and enter the OIC username and password under Basic Authorization.
6. Click **Save**.

Attribute mappings

The following table shows the relationships between Product Hub product Item properties and Commerce product properties.

Commerce product property	Product Hub product item field
`id`	`ItemNumber`
`displayName`	`ItemDescription`
`nonreturnable`	`ReturnableFlag`
`orderLimit`	`MaximumOrderQuantity`
`description`	`LongDescription`
`shippable`	`ShippableFlag`
`taxCode`	`OutputTaxClassificationCodeValue`
`active`	`ItemStatusValue`
`listPrice`	`ListPrice`

The following table shows the relationships between Product Hub product Item properties and Commerce SKU properties.

Commerce SKU property	Product Hub product item field
`id`	`ItemNumber`
`productId`	`ItemNumber` of parent style Item
`displayName`	`ItemDescription`
`nonreturnable`	`ReturnableFlag`
`active`	`ItemStatusValue`
`listPrice`	`ListPrice`

Configure Oracle Product Hub

This chapter describes tasks you perform in Oracle Fusion Product Hub to support the integration

Before you configure the integration, you should also plan the following tasks in Product Hub:

- Create a Spoke system in Product Hub for which item-publication jobs can be scheduled.

- Configure the item-publication criteria, select entities Attributes and Item Category Assignments. You can select a date filter in criteria to filter out items, for example, items updated within the past day, if the publication job is scheduled for a daily frequency.
- Configure the size of exported zip file. Remember that OIC will not accept ZIP files larger than 1GB.
  The exported zip files contains `ITEM_*.xml` files. Configure the size of XML by providing an optimal number of records an XML file can have, so that the size of the XML doesn't exceed 10 MB. This is to match OIC restrictions in place for performance considerations.
- Create a schedule for the publication job.

Install and Configure the Integration in OIC

This section describes how to install the integration package in Oracle Integration Cloud (OIC).

The OIC Home Page is the starting point for these tasks.

Install the recipe and configure the connections

This integration is provided as a recipe, which is a pre-assembled solutions to help jump-start your integration development.

First, log into an Oracle Integration instance to display the OIC Home Page. Find the integration recipe Oracle Product Hub — Oracle CX Commerce | Product Sync from the OIC home page. and install the recipe by hovering over the card and clicking the + sign icon.

Once the recipe is installed, hover over the card again to display options to configure, activate, and delete. Select Configure and then configure the connections used by the integration:

1. Log in to OIC as an admin user.
2. Select **Integration**->**Connections**.
3. Select **Oracle CX Commerce**. The Connection Properties dialog appears.
   - Enter the URL to connect to Commerce as the value of the Connection Base URL property (`https://<hostname>/ccadmin/v1`).
   - Enter the security token value, which you can find in the Commerce administration settings and click **OK**. The security token is the application key in the OracleCommerce Interface found under **Registered Applications Settings**. Contact your Commerce Administrator to get this application key.
4. Select **Oracle Integration Connection**. The Connection Properties dialog appears.
   - Select the connection type (REST API Base URL) and enter the OIC connection URL (`http://{OIC-server}/ic/api/integration/v1`).
   - Under **Security**, select **Basic Authentication** and provide login credentials to access the endpoint.
5. Select **Oracle Product Hub Connection**. The Connection Properties dialog appears.
   - Enter the ERP Services Catalog WSDL URL. For example: `https://{productHub-server}/fscmService/ServiceCatalogService?WSDL`
   - Enter the ERP Events Catalog URL. For example: `https://{productHub-server}/soa-infra`

- Under **Security Policy**, select **Username Password Token** and enter the login credentials to access the endpoint.

6. Select **Oracle UCM Connection**, The Connection Properties dialog appears.
   - Enter the WSDL URL. For example: `https://{productHub-server}/idcws/GenericSoapPort?wsdl`
   - Under **Security Policy**, select **Basic Authentication** and enter the login credentials to access the endpoint.

Update mapping for prices for products and SKUs

Update the mapping in the integration flow Oracle PRODUCT HUB COMMERCE `ItemToProductSync` to map the EFF attribute representing List Price in Product Hub to List Price in Commerce. By default Product Hub's Purchase List Price is mapped, but this must be updated to be mapped to the correct EFF attribute. Additionally, since List Price is required for creating a product in Commerce, the EFF attribute must be configured as required in Product Hub.

Customize mapping for additional custom fields

If you include additional custom fields in either Commerce products and SKUs or Product Hub Items, you will need to update the mapping in the integration flow Oracle PRODUCTHUB COMMERCE `ItemToProductSync`.

To include additional fields in Product Hub Items:

1. Once you have added new fields, generate the XSD in Product Hub using Item Publication job. The XSD will include any new fields you added.
2. Open the integration flow Oracle PRODUCTHUB COMMERCE `ItemToProductSync` to edit.
3. Navigate to **DownloadItemsAndProcess (Scope)** > **readItemFile (stage operation)**.
   Edit the stage operation node.

   Upload the latest XSD ZIP file.

   Select the definition for publication items (`.../commmon/publicationService/}Items`)
4. Click **Next**, then click **Done**.
5. The mapper should show the new fields for Product Hub item.

To include additional fields in Commerce Products and SKUs

1. Once you have added new fields, export a product from Commerce. The exported data will contain the new fields.
2. Open the integration flow Oracle PRODUCTHUB COMMERCE `ItemToProductSync` to edit.
3. Navigate to **DownloadItemsAndProcess (Scope)** > **writeCommerceJsonFile (stage operation)**.
   - Edit the stage operation.
   - Upload the exported sample JSON.
4. Click **Next**, then click **Done**.

5. The mapper should show the new fields for Commerce product and SKU.

Configure lookups

You must update the lookup values in the lookup table Oracle-PDH_Comm_Int_Settings:

- `PDHSpokeSystem` : Spoke system created in Product Hub for performing item-publication job. The integration will process the exported file for this spoke system.
- `PDHCatalogsForCategory-ProductsLinks` : Comma-separated list of catalogs. Configure if the product in Commerce will be associated with specific collections and catalogs. If this is not provided, the integration flow will link the product to all the collections specified by the exported data received from Product Hub. If no collection or catalog associations are required then configure this variable as NULL.
- `ToEmailAddresses`: Recipient email addresses to send error notifications. If an import fails in Commerce, an email message containing a link to failed records is sent to this ID.
- `FromEmailAddress`: Sender email address for the error notifications.
- `OICMaximumRetryCount`: Maximum number of retry counts for resubmitting failed instances. There are a number of reasons for integration instance failure, for example, when publishing is running in Commerce, or if there is a network failure.
- `CXCommerceAutoPublishEnabled` – Set this to `true` if all the records imported to Commerce through specific registered App ID (`CXCommerceRegisteredAppId`) needs to be published automatically; otherwise, configure this variable as `false`.
- `CXCommerceRegisteredAppId` - Registered Commerce Application ID of the user for which the all the data in the publishing queue will be published, if automatic publishing (`CXCommerceAutoPublishEnabled`) is enabled.
- `CXCommerceAutoPublishThreshold` - Maximum number of records for which automatic publishing will be triggered. Otherwise, the integration will send email to the administrator so they can manually publish.
- `CXCommerceMediaSyncEnabled` – Set this to `true` if media items from Product Hub need to be synced with Commerce; otherwise set to `false`.

The following example shows sample lookup values in the lookup table Oracle-PDH_Comm_Int_Settings:

```
FromEmailAddress - emailIdSource@example.com
PDHCatalogsForCategory-ProductsLinks – NULL
PDHSpokeSystem – CXCommerce
OICMaximumRetryCount – 5
CXCommerceRegisteredAppId – EnterYourRegisteredAppId
CXCommerceAutoPublishEnabled - false
CXCommerceAutoPublishThreshold – 1000000
CXCommerceMediaSyncEnabled - true
```

Configure email notifications

The integration includes the ability to send emails that notify administrators of the following issues:

- Record imports fail
- The number of items to automatically publish exceeds the specified threshold

- Exported XML/image size exceeds 10 MB

You specify the email address that notifications are sent to with the `ToEmailAddresses` lookup value and the email address that notifications are sent from with the `FromEmailAddress` lookup value. See Configure lookups for more information.

To learn how to customize and send email with OIC, see the following Oracle blog posts:

- An Advanced Guide to OIC Notification via Emails
- How to send email with attachments in OIC

Activate the integration flows

After you configure the Oracle Product Hub and Commerce connections, you must activate the integrations that were created when the integration package was imported to Oracle Integration Cloud. To do this, follow these steps:

1. Log in to Oracle Integration Cloud (OIC) as an admin user.
2. Click the **Integrations** icon to display the Integrations list.
3. Click the **Activate** button for each of the following integrations:
    - Oracle Commerce Product Import
    - Oracle Product Hub Oracle CX Commerce Products Int
    - Oracle Commerce Integration Resubmit
    - CX Commerce Product ImportComplete Post Processing
    - Oracle Commerce Product Hub Int Resubmit Publish
    - Oracle Commerce Product Hub Int Resubmit Schedule

    OIC displays a message to indicate that the integration flow was successfully activated.

Understand the integration flows

This section describes the out-of-the-box integration flows and includes a diagram that illustrates the overall integration flow.

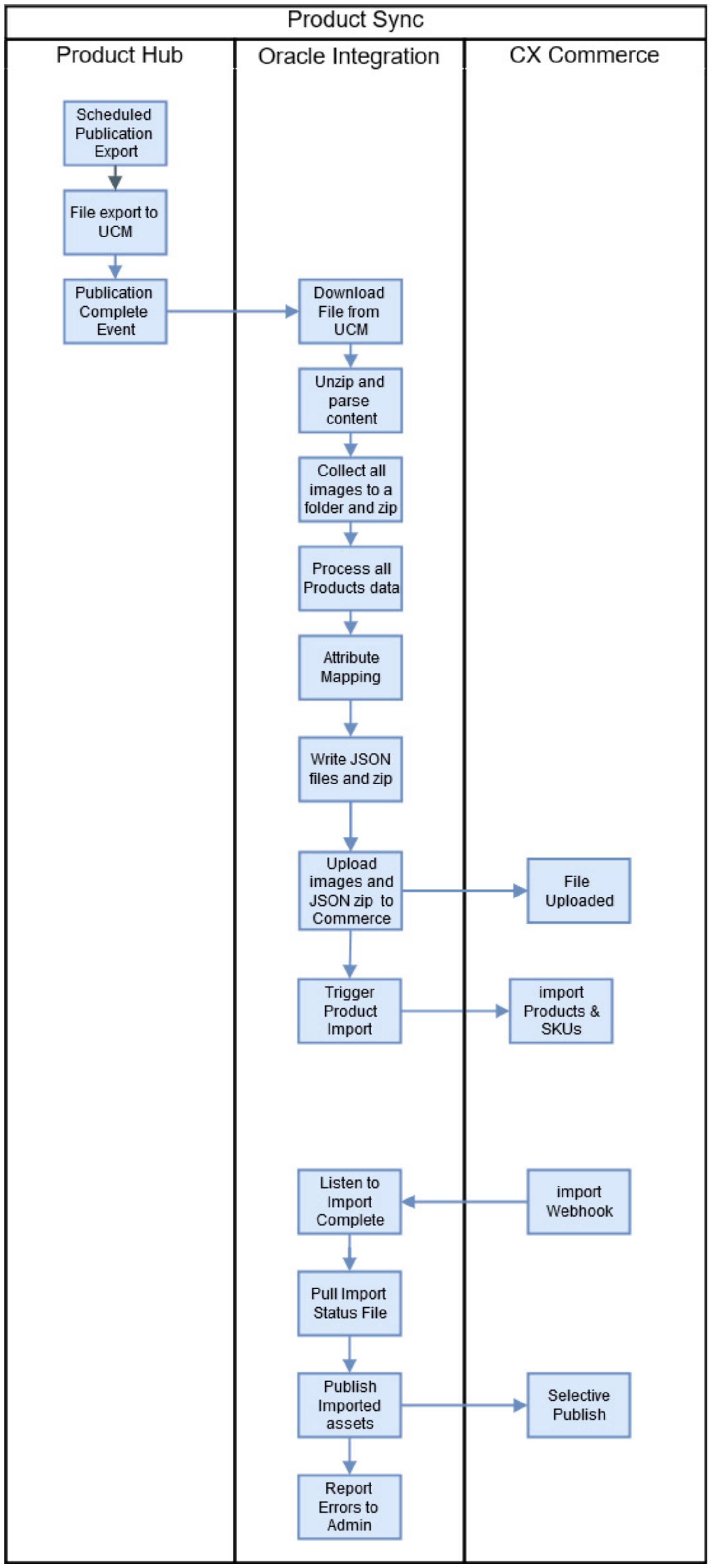
The Product Hub integration includes the following flows:

- `Oracle PRODUCT HUB COMMERCE ItemToProductSync`
  Syncs the products, SKUs, and images (if images are enabled in lookup) data from Product Hub to Commerce. This flow is triggered by the Product Hub item-publication event.
- `Oracle OIC COMMERCE ImportStart`
  Issues the command to start the import on a file to Commerce.
- `Oracle Commerce OIC ImportComplete Post Processing`
  Includes the flow for automatically publishing the records in Commerce. If the number of imported records exceeds the limit, sends a notification so publishing can be manually started in Commerce. This flow is triggered by the Commerce Import Complete Webhook .It sends a notification to the configured email address if the import has any failed records.
- `Commerce OIC Product HubInt Resubmit Webhook`

Listens for the Commerce Publish Complete webhook's POST request and re submits the failed integrations run of `Oracle PRODUCT HUB COMMERCE ItemToProductSync` which fails because of publishing is running in Commerce.

- `Oracle OIC OICREST RESUBMITERRORRUN`
  Fetches the failed integrations for the input integration name and resubmit the first entry in the failed integration.
- `Oracle SCHEDULE OIC Product HubInt Resubmit`
  The schedule integration, which resubmits the failed integrations run of failed `Oracle PRODUCT HUB COMMERCE ItemToProductSync`. Failures may occur because there was a processing error with the exported file in the integration or upload and import of the products file to Commerce was not successful.

The following diagram shows an overview of the integration flows:


Product Sync
Product Hub
Oracle Integration
CX Commerce
Scheduled Publication Export
File export to UCM
Publication Complete Event
Download File from UCM
Unzip and parse content
Collect all images to a folder and zip
Process all Products data
Attribute Mapping
Write JSON files and zip
Upload images and JSON zip to Commerce
File Uploaded
Trigger Product Import
import Products & SKUs
Listen to Import Complete
import Webhook
Pull Import Status File
Publish Imported assets
Selective Publish
Report Errors to Admin

10 Integrate with Oracle Responsys

Integrating Oracle Responsys and Oracle Commerce.

When you integrate Commerce and Oracle Responsys you create a unified solution for your customers as well as the ability to communicate with those customers in a relevant and structured dialogue based on their commerce activity.

Understand the Oracle Responsys Integration

This information provides instructions on how to implement the integration between Oracle Commerce and Oracle Responsys.

Oracle Responsys is an application within the Oracle Marketing Cloud suite of products that empowers data-driven marketing teams with the tools to deliver the relevant, engaging experiences their customers demand across devices, channels, and lifecycles. It is easy to make data from disparate sources useful, create precisely targeted audiences, and then empower customers to determine their own next experience by interacting with them in near real-time.

Commerce provides the overall customer experience allowing merchants to provide the most relevant content to shoppers at all times and, by integrating Commerce and Responsys, retailers can connect online behaviors in near real time to immediately respond to customer's actions and trigger personalized communications.

In addition, Commerce data can be used to create a more complete user profile which allows retailers to deliver a more personalized and engaging experiences to drive conversions and revenue.

Objective

The integration of Oracle Commerce and Oracle Responsys targets support for the following customer commerce activity:

- Welcome: Communicate with newly registered customers to enhance their relationship with the merchant.
- Win Back: Communicate with customers who have not created any new orders in a significant period of time. The message to the customer can be tailored to include a Commerce promotion to encourage them to return to the website.
- Milestone: Using Commerce profile data, communicate with customers based on personal milestones such as their birthday or the anniversary of their registration date.
- Abandoned Cart: Communicate with customers who added items to their shopping cart and then left the website without purchasing the items they placed in the cart. The message to the customer can be tailored to include aCommerce promotion to encourage them to return to the website.

This document provides instructions on how to set up an integration between Commerce and Responsys so that Commerce information is automatically passed to Responsys ensuring that the supported marketing campaigns are always based on current shopper activity. This document provides instructions on the following topics:

- How to download the Oracle Integration Cloud Integration Flows.
- How to configure and set up the Oracle Integration Cloud Integration Flows.
- How to setup the connection to Responsys.
- How to set up the required data tables in Responsys.
- How to setup the connection to Oracle Commerce.
- How to configure the webhooks to trigger the integration flows.
- How to use the abandoned cart functionality supported by the integration.

Audience

This document is written for Oracle Commerce and Oracle Responsys administrators who need to set up and configure the integration between these two systems.

Readers of this document should have experience with Commerce, Responsys, and Oracle Integration Cloud administration.

Prerequisites

For the purposes of this document, it is assumed that you already have:

- An Oracle Commerce account and access to Oracle Commerce 16.6 or later.
- An Oracle Responsys account and access to Oracle Responsys 6.31 or later.
- An Oracle Integration Cloud account and access to Oracle Integration Cloud Service 16.4.5 or later.

If you do not have one or more of these, please contact an Oracle sales representative for information on how to acquire one: http://www.oracle.com/us/corporate/contact/index.html.

Configuring the Integration

There are four stages to configuring the integration that are covered in this chapter.

This chapter will provide information on each of the stages to assist you in configuring your integration.

Configure the Integration Package

This section provides detail about where the integration package can be downloaded and how to import the integration package.

Importing the integration package in Oracle Integration Cloud creates connections to Oracle Commerce and Oracle Responsys in Oracle Integration Cloud. It also creates an integration between Commerce and Responsys with some default mappings in place.

Download the integration package

Follow these steps to download the integration package:

1. Log on to My Oracle Support at https://support.oracle.com
2. Search for OCCS-OMC_Integration.
3. Download the `OCCS-OMC_Integration_3.0.par` file. You should save it to a location where you can access it from Oracle Integration Cloud.

Import the integration package

Follow these steps to import the integration package on Oracle Integration Cloud.

1. Log on to Oracle Integration Cloud as an administrator.
2. Click the **Packages** icon.
3. Click the **Import Package** button.
4. Click **Browse** to open a navigation pane.
5. Browse for and select the packages archive (PAR) file when prompted.
6. Click **Import**. The package should be added to the Packages list.

Clicking on the name of the package in the Package list displays the integrations that are included in the package. This package includes three integrations. These are:

- OCCS-OMC Integration Order
- OCCS-OMC Integration Profile
- OCCS-OMC Idle Cart.

The OCCS-OMC Integration Order integration flow is responsible for sending information about submitted orders from Commerce to Responsys.

The OCCS-OMC Integration Profile integration flow is responsible for sending information about customer profiles from Commerce to Responsys.

The OCCS-OMC Idle Cart integration flow is responsible for sending information about a cart that has been abandoned and adds the visitor to the OMC Abandoned Cart orchestration program.

You can now configure connections for these integrations.

Configure the Oracle Responsys Connection

This section provides instructions on configuring the connection from the integrations to Oracle Responsys. Follow these instructions to configure the Responsys connection:

1. Log on to Oracle Integration Cloud as an administrator.
2. Click the **Connections** icon.
3. Click the **Oracle Marketing Cloud** connection.
4. Click the **Configure Connectivity** button.
5. Add the Responsys Login URL.

Note: This is not the URL you use to connect to Responsys. You can get the URL required here from your Responsys administrator.

6. Click **OK**.
7. Click on the **Configure Security** button.
8. Select **Custom Security Policy** in the Security policy list.
9. Complete the **Username**, **Password**, and **Confirm Password** fields. These are the credentials required to access your Responsys account.
10. Click **OK**.
11. Click **Test** to test your connection.
12. Click **Save**.

Your Responsys connection is now configured for the integration.

Configure the Oracle Responsys Connection

This section provides instructions on configuring the connection from the integrations to Oracle Responsys.

Follow these instructions to configure the Responsys connection:

1. Log on to Oracle Integration Cloud as an admin user.
2. Click the **Connections** icon.
3. Click the **Oracle Marketing Cloud** connection.
4. Click the **Configure Connectivity** button.
5. Add the Responsys Login URL.

   Note: This is not the URL you use to connect to Responsys. You can get the URL required here from your Responsys administrator.
6. Click **OK**.
7. Click on the **Configure Security** button.
8. Select **Custom Security Policy** in the Security policy list.
9. Complete the **Username**, **Password**, and **Confirm Password** fields. These are the credentials required to access your Responsys account.
10. Click **OK**.
11. Click **Test** to test your connection.
12. Click **Save**.

Your Responsys connection is now configured for the integration.

Configure the Oracle Responsys Database Tables

Once you have configured the Oracle Responsys connection, you need to create two tables to store the information created by the integration. These tables are

`CC_Master_User_List` and `CC_Submit_Order`. These tables should be created in a folder called CC in Responsys.

Note: These are the default names of the tables that are populated by this integration. If you create tables with different names or in a different folder then you must also modify the adapter configuration to point to the different tables.

This section shows the structure for each of these tables.

CC_Master_User_List Table

The following is a list of fields that must be included in a table called `CC_Master_User_List` created in a folder called `/CC` in Responsys. The first list is the system fields for the table, and the second list is the custom fields for the table.

Field Name	Field Type
RIID_	Integer Field
CREATED_SOURCE_IP_	Text Field (to 255 chars)
CUSTOMER_ID_	Text Field (to 255 chars)
EMAIL_ADDRESS_	Text Field (to 500 chars)
EMAIL_DOMAIN_	Text Field (to 255 chars)
EMAIL_ISP_	Text Field (to 255 chars)
EMAIL_FORMAT_	Single character field
EMAIL_PERMISSION_STATUS_	Single character field
EMAIL_DELIVERABILITY_STATUS_	Single character field
EMAIL_PERMISSION_REASON_	Text Field (to 255 chars)
EMAIL_MD5_HASH_	Text Field (to 50 chars)
EMAIL_SHA256_HASH_	Text Field (to 100 chars)
MOBILE_NUMBER_	Text Field (to 50 chars)
MOBILE_COUNTRY_	Text Field (to 25 chars)
MOBILE_PERMISSION_STATUS_	Single character field
MOBILE_DELIVERABILITY_STATUS_	Single character field
MOBILE_PERMISSION_REASON_	Text Field (to 255 chars)
POSTAL_STREET_1_	Text Field (to 255 chars)
POSTAL_STREET_2_	Text Field (to 255 chars)
CITY_	Text Field (to 50 chars)
STATE_	Text Field (to 50 chars)
POSTAL_CODE_	Text Field (to 25 chars)
COUNTRY_	Text Field (to 50 chars)
POSTAL_PERMISSION_STATUS_	Single character field
POSTAL_DELIVERABILITY_STATUS_	Single character field
POSTAL_PERMISSION_REASON_	Text Field (to 255 chars)
CREATED_DATE_	Time Stamp Field (date + time)
MODIFIED_DATE_	Time Stamp Field (date + time)
LOCALE	Text Field (to 25 chars)
FIRST_NAME	Text Field (to 100 chars)
LAST_NAME	Text Field (to 100 chars)
COMMERCE_REGISTRATION_DATE	Time Stamp Field (date + time)
COMMERCE_LAST_ACTIVITY	Time Stamp Field (date + time)

Field Name	Field Type
AGE	Number Field
PROFILEATTRIBUTE	Text Field (to 100 chars)

CC_Submit_Order Table

The following is a list of fields that must be included in a list extension table called `CC_Submit_Order` created in a folder called CC in Responsys.

Field Name	Field Type
RIID_	Integer Field
ORDER_ID	Text Field (to 25 chars)
NUMBER_OF_ITEMS	Number Field
ORDER_DETAILS_URL	Text Field (to 100 chars)
ORDER_CURRENCY	Text Field (to 25 chars)
ORDER_SUBMIT_TIME	Time Stamp Field (date + time)
ORDER_SHIPPING_ADDRESS1	Text Field (to 100 chars)
ORDER_SHIPPING_ADDRESS2	Text Field (to 100 chars)
ORDER_SHIPPING_CITY	Text Field (to 25 chars)
ORDER_SHIPPING_STATE	Text Field (to 25 chars)
ORDER_SHIPPING_COUNTRY	Text Field (to 25 chars)
ORDER_SHIPPING_POSTAL	Text Field (to 25 chars)
ORDER_TOTAL	Text Field (to 25 chars)
ORDER_SHIPPING_MOBILE	Text Field (to 25 chars)
ORDER_ITEM_NAMES	Text Field (to 50 chars)
CREATED_DATE_	Time Stamp Field (date + time)
MODIFIED_DATE_	Time Stamp Field (date + time)
SITE_ID	Text Field (to 25 chars)
SITE_NAME	Text Field (to 500 chars)

Configure the Oracle Commerce Connection

This section provides instructions on configuring the connection from the integrations to Oracle Commerce.

Follow these instructions to configure the Oracle Commerce connection:

1. Log on to Oracle Integration Cloud as an administrator.
2. Click the **Connections** icon.
3. Click the Oracle Commerce connection.
4. Click the **Configure Connectivity** button.
5. Enter the Connection base URL. The Connection base URL is derived using the following structure:

   Connection base `URL: https://<siteURL>/ccadmin/v1`

   where `<siteURL>` is the base URL of the Commerce site that integrates with Oracle Integration Cloud.

6. Click the **Configure Security** button.
7. The Commerce connection uses the OAuth security policy, so you must enter a Security token for the connection. This token is generated in Commerce. Instructions on generating the token can be found in the Generate a Security Token section of this document.
8. Click **OK**.
9. Click **Test** to test that the connection is working.
10. Click **Save**.

Your Commerce connection is now configured for the integration.

Generate a Security Token

This integration uses the Oracle Commerce REST web services APIs to access Commerce data. You must register the integration within Commerce and generate a security token in order for the integration to be granted access to the data.

Follow these instructions in order to generate a security token:

1. Log onto Commerce.
2. Click the **Settings** icon.
3. Click **Web APIs** from the sidebar menu.
4. Click **Registered Applications** from the Web APIs panel.
5. Click the **Register Application** button.
6. Enter a name for the integration. The application you are registering is Oracle Integration Cloud, so you should choose a name that is meaningful and reflects this.
7. Click **Save**.

   The Application ID and Application Key are automatically generated and the application is added to the Registered Applications page.
8. Click on the name of the application you created.
9. Click on **Click to reveal** to display the application key. You can copy the application key to use as the security token for the Commerce connection.

For more information on managing an application within Commerce, please refer to the Register Applications section of the Using Oracle Commerce document.

Activate the Integration

Once you have configured the Oracle Responsys and Oracle Commerce connections you can activate the integrations that were created when the integration package was imported to Oracle Integration Cloud.

Follow these instructions to activate the integrations:

1. Log on to Oracle Integration Cloud as an admin user.
2. Click on the **Integrations** icon to display the Integrations list.
3. Click on the **Activate** button for the integration you wish to activate.

4. You can choose here whether to switch on detailed tracing. Detailed tracing collects information about messages processed by the integration flow. This may assist in troubleshooting issues with the integration flow, but it may impact performance.

   To switch on detailed tracing, check the **Enable detailed tracing** checkbox.

   Note: Once an integration flow is active you must deactivate it and activate it again to switch detailed tracing on or off.

5. Click **Activate**.

A message should be displayed to indicate that the integration flow has been successfully activated.

Configure the Oracle Commerce Webhooks

When the integration flows have been activated you must configure the webhooks in Oracle Commerce. These webhooks push a JSON notification to a URL that you provide to the webhook. The URL you provide maps to the specific integration workflow set up in Oracle Integration Cloud.

For the integration flows covered by this document there are four webhooks that must be configured. These are:

- Shopper Registration: This sends a notification when a new user account is created by a visitor to your website. This webhook pushes notifications using the OCCS-OMS Integration Profile integration flow.
- ShopperAccount Update: This sends a notification when an already existing user account is modified by a visitor to your website. This webhook pushes notifications using the OCCS-OMS Integration Profile integration flow.
- Order Submit: This sends a notification when a registered shopper submits an order on your website. This webhook pushes notifications using the OCCS-OMS Integration Order integration flow.
- Cart Idle: This sends a notification when a registered shopper leaves your site without purchasing items that were added to their cart. This webhook pushes notifications using the OCCS-OMS Idle Cart integration flow.

You must configure the Production and Preview version of these webhooks to ensure that they work in all environments. The Production webhooks send information from your live store to production environments of your live systems, while preview webhooks send information from your preview environment to the test or sandbox environments of your external systems.

You can configure these webhooks through Commerce. Follow these instructions to configure a webhook:

1. Log on to Oracle Integration Cloud as an admin user.
2. Click on the **Integrations** icon.
3. Click on the **Integration Details** icon to display information about the integration flow.

   If you are configuring the Shopper Registration or Shopper Account Update webhooks then you should display information for the OCCS-OMC Integration Profile integration.

If you are configuring the Order Submit webhook then you should display information for the OCCS-OMC Integration Order integration.

If you are configuring the Cart Idle webhook then you should display information for the OCCS-OMC Idle Cart integration.

4. Copy the Endpoint URL for the integration.
5. Log on to Oracle Commerce.
6. Click on the **Settings** icon.
7. Select **Web APIs** from the sidebar menu.
8. Click on the webhook you wish to configure.
9. Paste the Endpoint URL you copied into the URL field for the webhook.
10. Remove the “metadata” text from the end of the URL.
11. Enter the Username and Password for your Oracle Integration Cloud account.
12. Click the **Save** button.

The webhook is now configured and is triggered each time the relevant event occurs, which in turn triggers the relevant integration flow.

For more information on Commerce webhooks, please refer to the Configure Webhooks chapter of the Using Oracle Commerce document.

Using the integration

This chapter provides instruction on how to use the functionality supported by the integration.

This chapter includes information on creating new custom events and Oracle Responsys programs,

Create an Abandoned Cart Program

Oracle Commerce monitors the shopping cart activities of visitors to your website and can detect if a shopper has added items to a cart and then abandoned the cart.

When Commerce detects an abandoned cart it triggers a program that is defined in your Oracle Responsys account. Commerce passes information about the items in the abandoned cart to Responsys.

This section provides instruction on how to create a new custom event and how to create a new program that runs when the new custom event occurs.

Create a New Custom Event

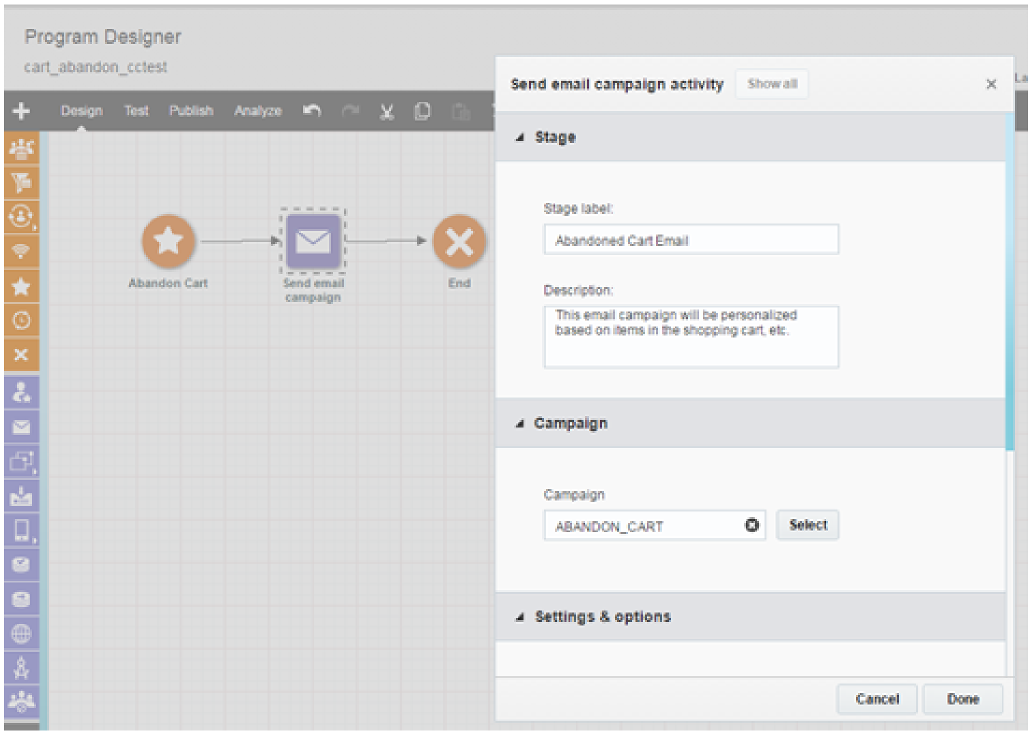
Follow these steps to create a new custom event on Oracle Responsys:

1. Log on to Oracle Responsys as an account administrator.
2. Select the **Account** icon.
3. Click on the **Define custom event types** link. This link is in the Account Customization section, under the Global Settings heading.
4. Click on the **Add new type** link, which can be found under the table of custom event types.

5. Enter “CC_Abandoned_Cart” as the Custom Event Type Name, and enter a meaningful description in the Description field.
6. Click on the **Save** button.
7. Click on the **Done** button.

Create a New Program

You can create a new Oracle Responsys program by selecting the profile list used for the Oracle Commerce integration. This program must start with a custom event followed by an email campaign that leads to the end of the program. You can see how the program should be configured in this illustration:



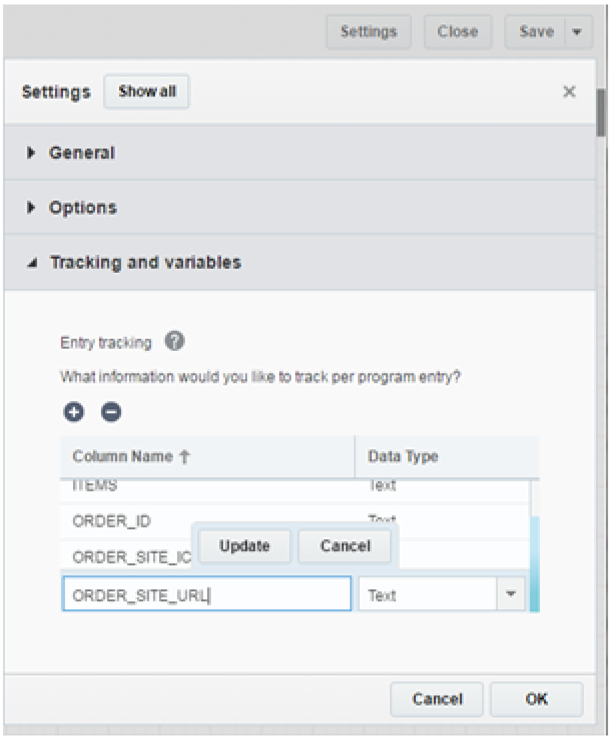
Stage label: Abandoned Cart Email

Description: This email campaign will be personalized based on items in the shopping cart, etc.

Campaign: ABANDON_CART

The abandon custom event must listen to a specific event name that is passed by Commerce in the API call that triggers this program. You must select `CC_Abandoned_Cart` from the Listen for custom event type dropdown menu.

Commerce passes a set of dynamic variables to the program through the API call. These dynamic variables must be specified for the program using the Settings configuration options.

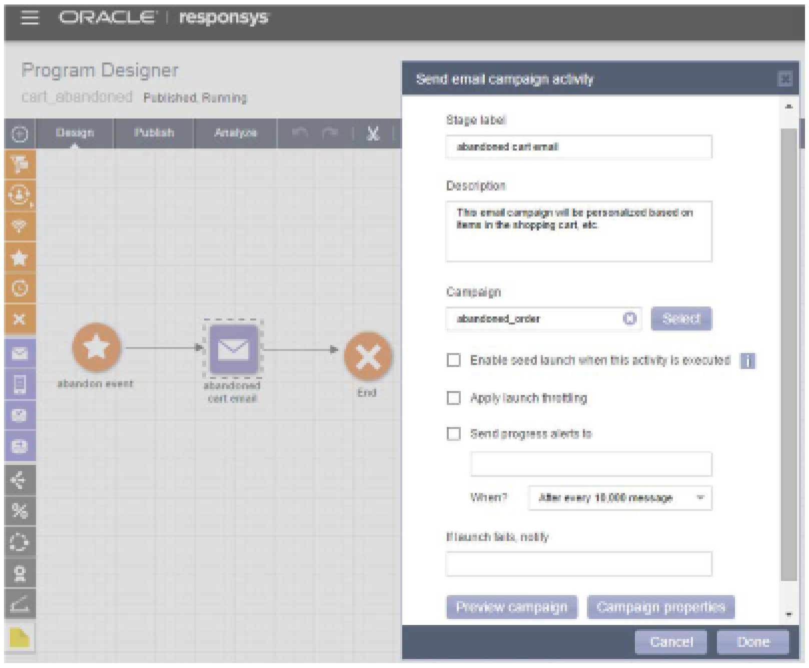


Entry tracking

What information would you like to track per program entry?

Column Name	Data Type
ITEMS	Text
ORDER_ID	Text
ORDER_SITE_ID	
ORDER_SITE_URL	Text

You must then associate a specific email campaign with the Send email campaign activity.



Stage label: abandoned cart email

Description: This email campaign will be personalized based on items in the shopping cart, etc.

Campaign: abandoned_order

☐ Enable seed launch when this activity is executed

☐ Apply launch throttling

☐ Send progress alerts to

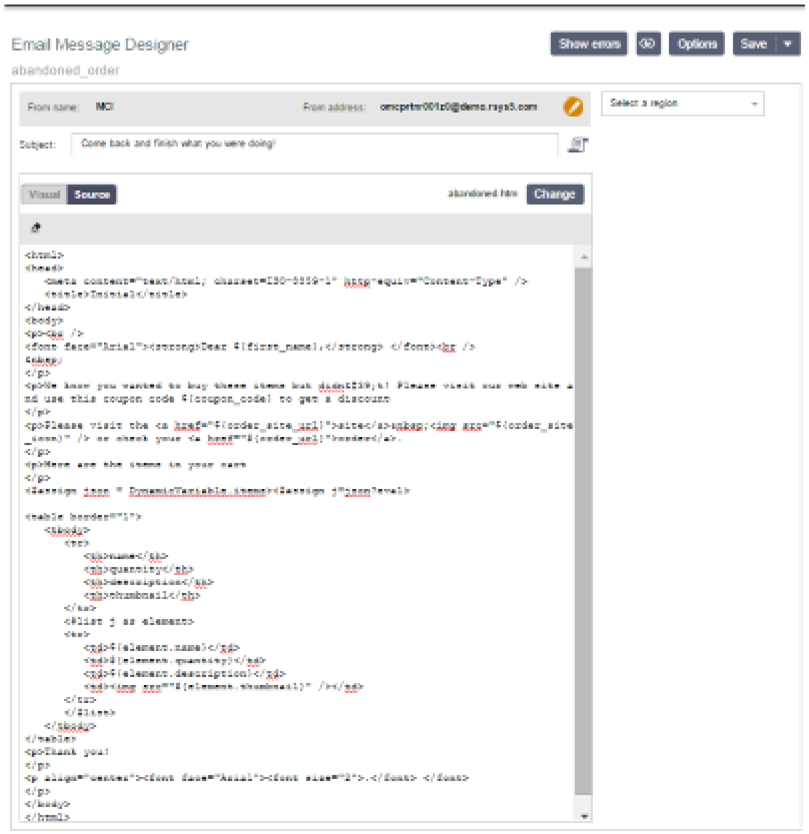
When? After every 10,000 message

If launch fails, notify

You should use the Email Message Designer to specify the personalization rules used by the email campaign associated with the email widget. You can do this using Responsys Personalization Language (RPL).

The following figure shows a sample HTML code using RPL to personalize the email message based on the dynamic variables passed by Commerce to the abandoned cart program.

Note: This sample code is provided for guidance only and should not be directly copied as it will not work with your integration.



From name: MCI

From address: omcprtnr001z0@demo.rsys5.com

Subject: Come back and finish what you were doing!

```
<html>
<head>
    <meta content="text/html; charset=ISO-8859-1" http-equiv="Content-Type" />
    <title>Initial</title>
</head>
<body>
<p><br />
<font face="Arial"><strong>Dear ${first_name},</strong> </font><br />
&nbsp;
</p>
<p>We know you wanted to buy these items but didn&#39;t! Please visit our web site a
nd use this coupon code ${coupon_code} to get a discount
</p>
<p>Please visit the <a href="${order_site_url}">site</a>&nbsp;<img src="${order_site
_icon}" /> or check your <a href="${order_url}">order</a>.
</p>
<p>Here are the items in your cart
</p>
<#assign json = DynamicVariable.items><#assign j=json?eval>

<table border="1">
    <tbody>
        <tr>
            <th>name</th>
            <th>quantity</th>
            <th>description</th>
            <th>thumbnail</th>
        </tr>
        <#list j as element>
        <tr>
            <td>${element.name}</td>
            <td>${element.quantity}</td>
            <td>${element.description}</td>
            <td><img src="${element.thumbnail}" /></td>
        </tr>
        </#list>
    </tbody>
</table>
<p>Thank you!
</p>
<p align="center"><font face="Arial"><font size="2">.</font> </font>
</p>
</body>
</html>
```

When you create the email using RPL you can see a preview of the email message rendered with some default personalization values.

For more information on using the Email Message Designer functionality, please refer to your Oracle Responsys documentation.

11 Integrate with Oracle Retail Order Management System

Integrate Oracle Retail Order Management System with Oracle Commerce.

Oracle Order Management Cloud is integrated with the Oracle Global Order Promising Cloud. Together with Commerce they allow you to provide fulfillment monitoring and global availability for your customers.

Introduction

Oracle Retail Order Management System Cloud Service (OROMS) is an order management system that supports retail transactions, including fulfillment, warehousing/inventory control, customer service, merchandising, marketing and finance.

Commerce and Oracle Retail Order Management can be used together to provide a robust commerce architecture.

Audience

This document is written for Commerce and Oracle Retail Order Management administrators who need to set up and configure the integration between these two systems.

Readers of this document should have experience with both Commerce and Oracle Retail Order Management administration. This document does not provide instructions on configuring aspects other than integration for Commerce and Oracle Retail Order Management. For that information, refer to the product documentation.

Features

The integration between Commerceand Oracle Retail Order Management provides a solution that combines the capabilities of these two products.

This integration provides the following features:

- Turning integration on and off using the Commerce administration interface
- Providing configuration based on your environment
- Pushing completed Commerce orders to Oracle Retail Order Management for fulfillment
- Retrieving and displaying Oracle Retail Order Management order status in Commerce

This integration provides retailers with an opportunity to manage order and fulfillment information.

Architectural Overview

Shoppers use the Commerce storefront to place an order.

When a retailer enables integration, the order created in the storefront is sent to Oracle Retail Order Management where it is fulfilled. Commerce can obtain the details of the order from Oracle Retail Order Management and display the status in the customer storefront or the Agent Console of Commerce.

Commerce manages the promotions and discounts for the order and passes the final price of the order to Oracle Retail Order Management. The shipping methods in both systems are synchronized, ensuring that the customer is choosing from shipping methods that are available from Oracle Retail Order Management.

To perform data synchronization, Commerce communicates with Oracle Retail Order Management using REST. Commerce provides data output in JSON, while Oracle Retail Order Management exposes REST services that accept XML. Integration services take the JSON data from Commerce, convert it to XML using XSLT transformers, and send it to all Oracle Retail Order Management systems. In turn, XML data sent from Oracle Retail Order Management is read by the integration service and converted to JSON. The webhook target that communicates with Oracle Retail Order Management is set using the administration interface. If the integration is disabled, the transformation logic is skipped and the webhook behaves as a standard webhook.

Webhooks submit JSON, however, Serenade accepts only XML. When an order is submit, it is converted from JSON to XML. Webhooks support multiple target orders, and if the target contains

Integration services are configured using the Commerce administration interface.

When an order or order detail is queried in Commerce, a service call is made to Oracle Retail Order Management to get the latest status for that order. Orders that are created in Commerce are synchronized to Oracle Retail Order Management, and order status updates regarding fulfillment from Oracle Retail Order Management are requested by Commerce on demand.

Additional Documentation

For additional information on Commerce:

For additional information on using Commerce, see the Oracle Commerce documentation.

Prerequisites

This section contains information on prerequisites needed before configuring the integration.

Before you begin the integration process, you must ensure that you have met the following conditions.

Access Rights

To configure integration, you need to have administrator access to Commerce.

This allows you to configure the integration settings using the administration interface.

Data Configuration

The product and SKU information should be in sync on both the Commerce and Oracle Retail Order Management servers.

The following describes the Item and SKU fields and the mapping between them:

Item Configuration

Oracle Retail Order Management allows the `ITEM_ID` field to contain a maximum of 12 alphanumeric characters. Therefore, items created with Commerce should contain no more than 12 alphanumeric characters, for example, `Item001`, `wallet`, etc.

SKU ID Configuration

The Oracle Retail Order Management system generated `short_sku_number` number is unique for a company as well as a site. As such, the Commerce SKU ID should also have a unique ID for the site. Note that the SKU ID must be numeric and limited to 7 characters.

Variant Configuration

Oracle Retail Order Management allows a maximum of three SKU attributes, for example, `color`, `size` and `collar`. Note that all items will have the same elements. For example, both the shirts and the shoe SKUs will contain the same variants, for example, `color`, `size`, and collar.

Commerce allows more than three variants, for example, `color`, `size`, `collar`, and `sleeve`. However, Oracle Retail Order Management cannot recognize more than three variants. If the SKU attribute in Oracle Retail Order Management and the SKU variant in Commerce both contain fewer than three entries, the SKU attribute will be the same.

While creating items and SKUs in Commerce and Oracle Retail Order Management, consider the following:

- When creating SKUs, limit variants to less than three
- Create SKUS in Oracle Retail Order Management. The same SKUs with the `short_sku_number` field can be manually exported and created in Commerce
- Create variants that are as general as possible so that they can be shared across items
- Before the integration process begins, the Commerce and Oracle Retail Order Management inventories should be synchronized

For detailed information on configuring and managing catalogs and SKUs, refer to the Oracle Commerce documentation.

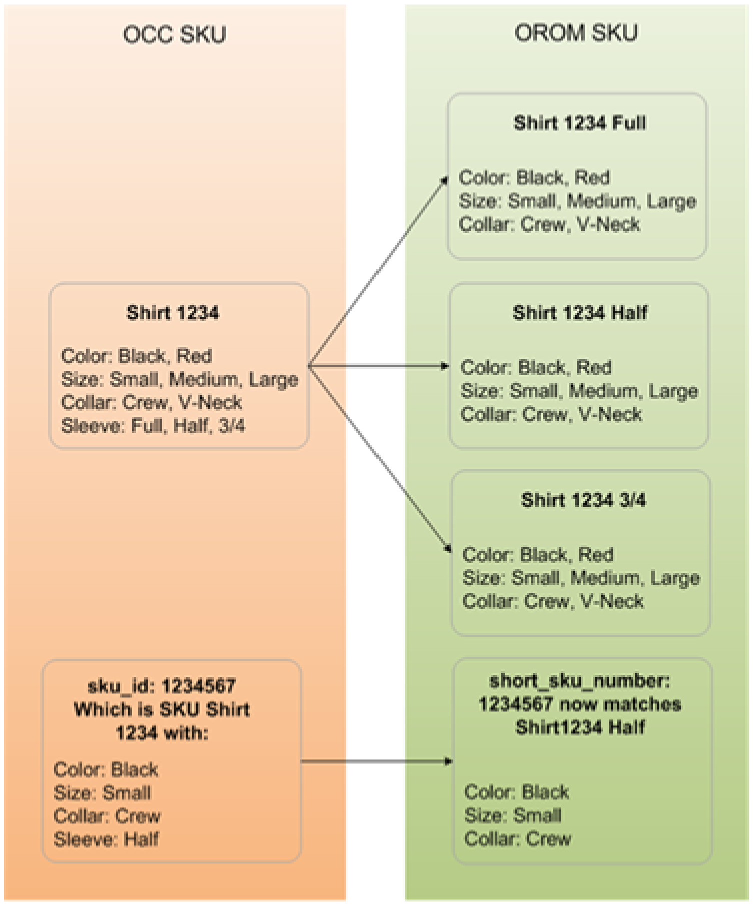
Working with More Than Three Variants

Commerce allows you to configure SKUs so that they contain variants, such as `color`, `size`, `collar`, etc. However, Oracle Retail Order Management can accept only up to three of these variants. In the following example, the SKU for a shirt, as it is defined in Commerce, contains the variants for `color`, `size`, `sleeve` and `collar`. Because Oracle Retail Order Management can only accept three of these variants to be part of the SKU, the SKU is could be modified.

Note: If a Commerce SKU contains three or less variants, it is converted into the Oracle Retail Order Management system without any changes.

In the following example, the Commerce Shirt 1234 item contains the following variants: `color`, `size`, `collar`, and `sleeve`. A SKU with the `sku_id` 1234567 is a Shirt1234 item that has the variants values of `black`, `small`, `crew`, half.

Because the Commerce SKU has more variants than Oracle Retail Order Management can accept, you might configure your SKU so that the last variant, `sleeve`, is removed, creating three separate items: Shirt1234 Full, Shirt 1234 Half and Shirt 1234 3/4. The Commerce SKU now matches the Oracle Retail Order Management Shirt1234 Half item. The SKU with the Commerce `sku_id` 1234567 receives the system-generated `short_sku_number` 1234567.



The above diagram shows how the Commerce item Shirt 1234 can be converted into three separate Oracle Retail Order Management items based on the fourth variant, `sleeve`. The diagram also shows how the Commerce SKU Shirt1234 relates to the Oracle Retail Order Management SKU 1234567.

Setting Up the Integration

The following section provides information on configuring and accessing integration.

This includes configuring both the Oracle Commerce and the Production Integration.

Commerce Configuration

Before you set up the Oracle Retail Order Management integration with Oracle Commerce ensure that the necessary integration software is running on your server.

Configuring Webhooks

As described in the Architectural Overview section, the integration service is based on the Web API settings in Commerce.

Web APIs allow you to subscribe to events for your products and orders by creating webhooks that push JSON notifications to a URL you specify. For additional information on webhooks, refer to Use Webhooks.

Before you can configure the integration settings, the webhook must be configured. To do this, ensure that the Order Submit Event API has been configured with the URL necessary to connect to the Oracle Retail Order Management server.

To Configure the Order Submit Event API

1. From the Commerce administration interface, select the **Settings** menu.
2. Open the Web API page and select the **Webhook** tab.
3. Open the Order Submit Event API.
4. Provide the URL of the server that will be accepting the order, including the server name and port used for the CWOrderIn service. For example: `https://my.example.com:8443/SerenadeSeam/sxrs/SerenadeREST/CWOrderIn`
5. Provide a user name and password for accessing the server.
6. If you are using HMAC authentication, you can view or reset the key.
7. Click **Save** to save your changes.

Accessing the Oracle Integrations Console

The Oracle Integrations page is accessed through the Commerce administration interface.

1. In the Commerce administration interface, click the **Settings** icon.
2. Click **Oracle Integrations** in the left navigation pane.
3. Select OROMS (Oracle Retail Order Management) from the Oracle Integrations menu.
4. To enable the integration, select the **Enable integration** checkbox.

Configuring the Integration

The following sections describe how to configure components of the integration procedure.

Production Configuration

The Production Configuration section configures a number of details as well as the mappings for payment types and shipping methods. These values are passed to Oracle Retail Order Management allowing the orders to be identified.

Note: These settings are duplicated in the other Oracle Retail Order Management Integration setting, Preview Configuration.

The following properties are set in the fields provided by the Production Configuration page. Note that all fields are required unless otherwise noted.

Field	Description
XSLT Path	Name of the custom XSLT file to load.
Source	Identifies the source of the XML message. The source should default to IDC.
Target	Identifies the target of the XML message. The target should default to RDC. If multiple webhooks are specified in the Target field, all of the systems receive the same data. Transformation for specific URLs can be performed by adding the URLs to both the webhook and integration settings.
Company Code	Identifies the company for the order. The company code is validated against the Company table.
Source Code	Updates the source code field in the Order Header table. The source code provides information about the currency code. You must provide the Commerce currency code to the Oracle Retail Order Management System source code mapping using the Currency Code mapping table.
Order Type	Updates the order type field in the Order Header table.

URL	The URL path to the Oracle Retail Order Management Web Service CWMessageIn. For example:
	`https://my.example.com:8443/SerenadeSeam/sxrs/SerenadeREST/`
	`CWMessageIn`
	Note: Use the same hostname and port number that you provided for the Order Submit URL on the Web APIs Webhook page.

User Name	The name of the user who should have access to the Web Service.
Password	The password associated with the user name. To see the password, click the Reveal button to display the password.

Orders and multiple site configurations

The Commerce currency codes are mapped to the Oracle Retail Order Management Source Code. For example:

Commerce Currency Code	Oracle Retail Order Management Source Code
USD	CC_USD
EUR	CC_EUR

In an environment with multiple sites, you could configure the settings like this:

Site	Commerce Currency Code	Oracle Retail Order Management Source Code
USA	USD	CC_USD
UK	GBP	CC-UK_GBP
UK	EUR	CC_UK_EUR
Germany	EUR	CC_DE_EUR

Select the site with the site picker and select the appropriate Oracle Retail Order Management source code. Each source code points to a single offer record, which contains the currency code. This is how the offer record can provide the currency representation for an order.

Using XSLT

You can use the customized XSLT capability to extend the default integration. The system contains an internal XSLT file that maps all attributes. The merchant does not have access to this XSLT file; however, you can provide a path to the XSLT. The XSLT should contain the logic that customizes the Oracle Retail Order Management `createOrder` payload so that it includes dynamic properties or makes mapping changes.

This XSLT workflow is active only when the Oracle Retail Order Management integration is enabled. The merchant must upload the XSLT using Oracle Commerce's file upload endpoints. If no customized XSLT file is uploaded, the system uses the default XSLT file to pass orders, ignoring any custom attributes.

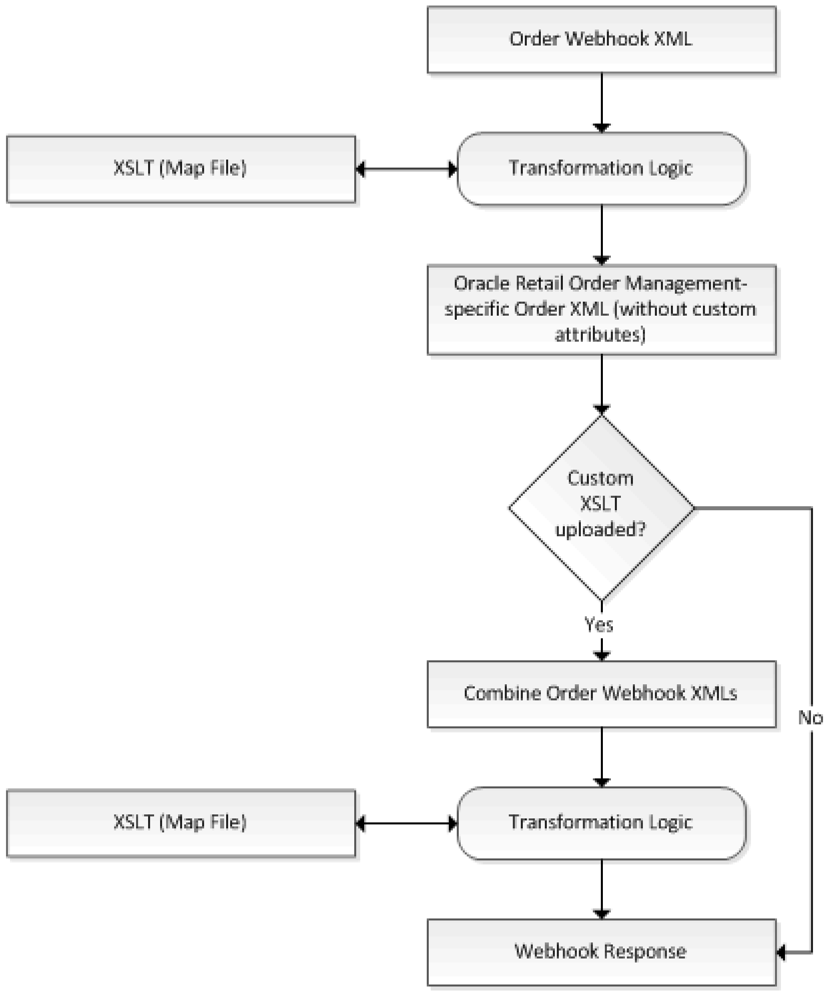
When creating a custom XSLT file:

- The default mapping can be overridden and mapped to other attributes as necessary.
- The merchant can use dynamic attributes created with Oracle Commerce to map attributes of the Oracle Retail Order Management.
- Because mapping is configured in the customized XSLT file, the merchant can map custom attributes that were created for an Order header level in Commerce to an order header or order line status in Oracle Retail Order Management.

To use a customized XSLT file, do the following:

1. Create a new XSLT file mapping.
2. Upload the XSLT file to Commerce using the file endpoints.
3. Use the Oracle Retail Order Management Integration Settings to provide the name of the uploaded XSLT.

The following illustration defines the way that the merchant's transformation is applied when the XSLT path is provided.



XSLT Example

The following is an example of an XSLT file:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="2.0"
    xmlns:str="http://exslt.org/strings" xmlns:exsl="http://exslt.org/
common"
    exclude-result-prefixes="str exsl">
  <xsl:output indent="yes" method="xml" omit-xml-declaration="yes" />
    <xsl:template match="/">
      <xsl:variable name="vOrder" select="request/atgResponse/order" />
      <Message>
        <xsl:copy-of select="request/Message/@*" />
        <Header>
          <xsl:copy-of select="request/Message/Header/@*" />
        <Payments>
          <xsl:for-each select="request/Message/Header/Payments/
Payment">
            <Payment>
              <xsl:copy-of select="@*" />
            </Payment>
          </xsl:for-each>
        </Payments>
        <ShipTos>
          <xsl:for-each select="request/Message/Header/ShipTos/ShipTo">
            <ShipTo>
              <xsl:copy-of select="@*" />
                <Items>
                  <xsl:for-each select="Items/Item">
                    <Item>
                      <xsl:copy-of select="@*" />
<!-- **************************************************
      Updating an existing mapping
******************************************************** -->
<!-- To change the mapping of an existing OROMS attribute, comment the line
(<xsl:copy-of select="@*" />) and uncomment below line and replace
<oroms_attribute> with the attribute name required in output and
<occ_attribute>
with attribute name in occ XML map an attribute in oroms XML to a different
value,
comment the above line and uncomment below line and replace
<oroms_attribute> with
the attribute name required in output and the <occ_attribute> with attribute
name
in occ XML -->

<!-- <xsl:copy-of select="@*[name()!='<oroms_attribute>']" /> <xsl:attribute
name="tax_override"> <xsl:value-of select="//request/atgResponse/order/
<occ_attribute>" /> </xsl:attribute> -->

<!-- *********************************************************
Mapping a dynamic attribute of OCC to a new attribute in OROMS
*********************************************************** -->
<!-- To add a new attribute "category" at item level in oroms XML, which
reads the
data from the dynamic attribute shopperCategory. Replace <occ_attribute>
with the
dynamic attribute name in occ. -->

<!-- <xsl:attribute name="category"> <xsl:value-of select="//request/
atgResponse/
order/<occ_attribute>" /> </xsl:attribute> -->

<!-- *******************************************************
Mapping a dynamic attribute of OCC with comma separated item level
data to a new attribute in OROMS
************************************************************* -->
<!-- To map a dynamic attribute in occ in format skuId1-value1,skuId2-value2.
Replace <occ_attribute> with the dynamic attribute name in occ and
<oroms_attribute> with oroms attribute name. -->

<!-- <xsl:variable name="vOromsAttribute" select="@short_sku_number"/>

<xsl:for-each select="str:tokenize($vOrder/<occ_attribute>,',')">
<xsl:variable name="temp" select="str:tokenize(.,'-')"/>
<xsl:if test="$temp[1]=$vOromsAttribute">
<xsl:attribute name="<oroms_attribute>"> <xsl:value-of select="$temp[2]" />
```

```
</xsl:attribute>
</xsl:if>

</xsl:for-each> -->

<!--
*******************************************************************
Mapping a dynamic attribute of OCC in JSON with a new attribute in
OROMS
******************************************************************** --
>
<!-- To map a dynamic attribute in occ in json format (sample json
format is given
below ). Replace <occ_attribute> with the dynamic attribute name in
occ,
<oroms_attribute> with oroms attribute name and
<dynamicAttributeFieldName> with
specific field name in the dynamic attribute json -->

<!--  Sample JSON:
  [
    {
      "giftwraplineId":"gift-wrap-item-gwprod1001-834215",
      "giftWrapSkuId":"gwprod1001",
      "giftWrapSkuDescription":"Gift Wrap Product",
      "giftWrapSkuPrice":5,
      "skuId":"834215",
      "skuDescription":"Opal Innocence Silver 8\" Salad Plate",
      "quantity":1
    },
    {
      "giftwraplineId":"gift-wrap-item-gwprod1001-845353",
      "giftWrapSkuId":"gwprod1001",
      "giftWrapSkuDescription":"Gift Wrap Product",
      "giftWrapSkuPrice":5,
      "skuId":"845353",
      "skuDescription":"Bald Eagle Figurine",
      "quantity":1
    }
  ]
 -->

<!-- <xsl:variable name="vOromsAttribute" select="@short_sku_number" />
<xsl:attribute name="<oroms_attribute>">
  <xsl:call-template name="readCustomProperty">
    <xsl:with-param name="json" select="$vOrder/<occ_attribute>" />
    <xsl:with-param name="skuId" select="$vOromsAttribute" />
    <xsl:with-param name="dynamicAttributeName" select="
        <dynamicAttributeFieldName>" />
  </xsl:call-template>
</xsl:attribute>
 -->
                    </Item>
                  </xsl:for-each>
                </Items>
```

```
          </ShipTo>
        </xsl:for-each>
    </ShipTos>
    </Header>
  </Message>
</xsl:template>

<xsl:variable name="quot" select="'&quot;'" />
<xsl:variable name="skuIdWithQuots" select="concat('skuId\', $quot, ':\'
    $quot)" />

<xsl:template name="readCustomProperty">
  <xsl:param name="json" />
  <xsl:param name="skuId" />
  <xsl:param name="dynamicAttributeName" />

  <xsl:variable name="temp" select="normalize-space
      (substring-after($json,'{'))" />
  <xsl:variable name="fullSkuJson" select="normalize-space
      (substring-before($temp,'}'))" />
  <xsl:variable name="remainingJson" select="normalize-space
      (substring-after($temp,'}'))" />

  <xsl:call-template name="readCustomPropertyBySkuId">
    <xsl:with-param name="fullSkuJson" select="$fullSkuJson" />
    <xsl:with-param name="remainingJson" select="$remainingJson" />
    <xsl:with-param name="skuId" select="$skuId" />
    <xsl:with-param name="dynamicAttributeName" select=
      "$dynamicAttributeName" />
  </xsl:call-template>
</xsl:template>

<xsl:template name="readCustomPropertyBySkuId">
  <xsl:param name="fullSkuJson" />
  <xsl:param name="remainingJson" />
  <xsl:param name="dynamicAttributeName" />
  <xsl:param name="skuId" />
  <xsl:variable name="temp" select="normalize-space
      (substring-after($fullSkuJson,$skuIdWithQuots))" />
  <xsl:variable name="skuIdValue" select="normalize-space
    (substring-before($temp,'\'))" />

  <xsl:if test='$remainingJson'>
    <xsl:variable name="temp1" select="normalize-space
      (substring-after($remainingJson,'{'))" />
    <xsl:variable name="fullSkuJson1" select="normalize-space
      (substring-before($temp1,'}'))" />
    <xsl:variable name="remainingJson1" select="normalize-space
      (substring-after($temp1,'}'))" />
    <xsl:call-template name="readCustomPropertyBySkuId">
      <xsl:with-param name="fullSkuJson" select="$fullSkuJson1" />
      <xsl:with-param name="remainingJson" select="$remainingJson1" />
      <xsl:with-param name="skuId" select="$skuId" />
      <xsl:with-param name="dynamicAttributeName"
        select="$dynamicAttributeName" />
```

```
          </xsl:call-template>
        </xsl:if>

        <xsl:if test='$skuId = $skuIdValue'>
          <xsl:variable name="attributeNameWithQuots"
              select="concat($dynamicAttributeName, '\', $quot,
':\' , $quot)" />
          <xsl:variable name="temp2" select="normalize-space
              (substring-after($fullSkuJson,$attributeNameWithQuots))" />
          <xsl:variable name="attributeValue" select="normalize-space
              (substring-before($temp2,'\'))" />
          <xsl:value-of select="$attributeValue" />
        </xsl:if>
  </xsl:template>
</xsl:stylesheet>
```

Order Creation

Orders created in Commerce are sent to Oracle Retail Order Management using the Submit Order webhook.

Note that the Oracle Retail Order Management webhook overrides the Submit Order Webhook to send XML messages that contain the entire XML payload. If the webhook is configured to send to multiple destinations, all of the destinations will receive this XML payload.

The following is an XML example of an order creation message received by Oracle Retail Order Management.

Sample Order Creation Message

```
<Message source="IDC" target="RDC" type="CWOrderIn">
  <Header order_number="o60412" order_type="Y" company_code="51"
order_channel="I"
    source_code="A123_USD" payment_only="N" response_type="E"
        order_date="08222016"
    sold_to_prefix=""sold_to_fname="Kim" sold_to_lname="Anderson"
sold_to_suffix=""sold_to_busres="R" sold_to_address1="21 Cedar Ave"
    sold_to_address2=""sold_to_address3="" sold_to_address4=""
    sold_to_city="Syracuse" sold_to_state="NY" sold_to_zip="13202"
    sold_to_country="US"sold_to_email_update="N"
sold_to_day_phone="212-555-977"
    sold_to_eve_phone="212-555-1977"sold_to_address_update="Y"
pay_incl="Y"
    bill_to_prefix=""bill_to_fname="Kim" bill_to_lname="Anderson"
    bill_to_suffix=""bill_to_address1="21 Cedar Ave"
bill_to_address2=""
    bill_to_address3=""bill_to_address4="" bill_to_city="Syracuse"
    bill_to_state="NY"bill_to_zip="13202" bill_to_country="US"
    bill_to_day_phone="212-555-1977" bill_to_eve_phone="212-555-1977"
    bill_to_fax_phone=""bill_to_email="kim@example.com"
bill_to_company_name=""
    ind_number=""order_email="kim@example.com"
alternate_sold_to_id="se-570031">
  <Payments>
    <Payment payment_type=""cc_number="9997000108950573"
```

```
cc_exp_month="03"
      cc_exp_year="2018" auth_amount="75.88" auth_date="01011970"
      ecommerce_indicator="Y" already_tokenized="Y"
      transaction_id="1ni4eg211lj6iqt6097hopidv7" vendor_response="100"/>
  </Payments>
  <ShipTos>
    <ShipTo shipping_method="01" freight_tax_override="Y"
      freight_tax_amount="4.22" calc_frt="N" ship_to_fax_phone=""
      ship_to_evening_phone="212-555-1977" ship_to_day_phone="212-555-1977"
      ship_to_email=kim@example.com ship_to_zip="13202" ship_to_state="NY"
      ship_to_country="US" ship_to_city="Syracuse" ship_to_address3=""
      ship_to_address2="" ship_to_address1="21 Cedar Ave" ship_to_company=""
      ship_to_suffix=""ship_to_lname="Anderson" ship_to_fname="Kim"
      ship_to_prefix="" freight="50"contact_name="KimAnderson">
        <Items>
          <Item tax_override="Y" price_override="Y" short_sku_number="130"
            item_id="LAPTOP" actual_price="6.67" tax_amount="1.11"
quantity="2"/>
          <Item tax_override="Y" price_override="Y" short_sku_number="130"
            item_id="LAPTOP" actual_price="6.66" tax_amount="0.55"
quantity="1"/>
        </Items>
      <ShipTo>
    <ShipTos>
  </Header>
</Message>
```

Order Status Updates

Commerce retrieves the order status and tracking information from the Oracle Retail Order Management System to display in the client. The status, which is obtained when an order detail is queried, will not be persisted or updated in the repository.

Promotions

Promotions and offers are handled by Commerce with the corresponding discount price sent as part of the order.

Returns and Exchanges

By default, the Oracle Retail Order Management integration does not support returns and exchanges.

Configurable Product Support

By default, the Oracle Retail Order Management integration does not support configurable products.

Payment Methods

Oracle Retail Order Management uses payment methods that are identified with unique integers. For integration, Commerce Payment Methods are mapped to Oracle Retail Order Management numeric payment methods.

You can obtain a list of the payment methods available by calling the following endpoint:

```
GET /ccadmin/v1/merchant/paymentGateways
```

This returns a list of available payment methods. The response also includes the ID of the merchant, whether the method has been enabled, and the repository ID of the payment method. The repository ID identifies the payment method code to use. The following example displays a partial response that identifies a CyberSource payment method:

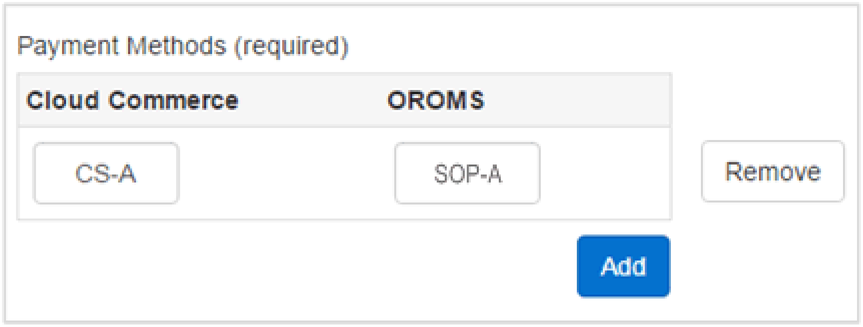
```
{
  "paymentGateways": [
  {
    "sopCredentials": {
      "storefront": {
      "sopURL": "http://10.101.101.101:8080/ccstore/v1/PM/
cybersourceSOP",
      "expirationDate": "2019-01-28T11:54:30.207Z",
      "profileId": "Admin",
      "applicationName": "storefront",
      "hasSecretKey": true,
      "hasAccessKey": true,
      "repositoryId": "SOP-A"
  },
```

In an environment with multiple sites, you need to pass the `siteId` value in header to get payment gateway details for the site. To fetch payment gateways specific to a site, you need to pass `x-ccsite` header. For example, to fetch payment gateways for `siteUS` site, the header should include `x-ccsite: siteUS`.

You can find information about individual endpoints in the REST API documentation that is available through the Oracle Help Center.

Note that this documentation reflects the most recent version of Commerce. If you are currently using an earlier version of Commerce, the API documentation on the Oracle Help Center may include endpoints that are not available on your version.

Once you have obtained the repository ID, use it to map the payment method as follows:


Payment Methods (required)

Cloud Commerce	OROMS
CS-A	SOP-A

As orders are created in Commerce and passed to Oracle Retail Order Management, the payment mappings are passed to Oracle Retail Order Management for fulfillment and settlement with the required payment gateways.

Note: To ensure PCI compliance, no credit card or gift card numbers are sent to Oracle Retail Order Management. The transaction reference and the authorization ID are sent with the order.

Payment Gateways

Commerce has preconfigured integrations with Chase and CyberSource payment gateways. By default, the CyberSource payment gateway is selected for the integration. The Oracle Retail Order Management System requires a credit card to be provided when using the Chase gateway, however, Commerce does not store credit cards and cannot provide them. As such, Chase credit cards are not supported with this integration. Oracle Retail Order Management supports CyberSource and PayPal gateways by default.

Note: When using the CyberSource payment gateway, you must provide the AuthNumber for reauthorization and settlement. Because Commerce does not store the AuthNumber, the field is set to NA by default. You must update this field with the AuthNumber values for the CyberSource gateway.

To configure integrations with other order processing systems, use the Generic Payment Framework, as described in the Oracle Commerce documentation. Generic payment is not supported by default; however, Oracle Retail Order Management APIs can support sending payment details separately for an order. Whenever a generic payment is used in Commerce, payment details are not sent and the order is put into an error state. Once you update the payment details for the order, the Oracle Retail Order Management System changes the order to an open state.

External Pricing

Note that this integration does not support external pricing. If you are using a combination of external prices and sale prices from Commerce, this integration will average the prices of items of the same SKU. For example, a customer qualifies for a promotion in Commerce that enables him to purchase 5 hats for the sales price of $5 each, instead of the list price of $6. If the customer wants to buy 10 hats, in Commerce, 5 of the hats would be sold at $5 and 5 of the hats would be sold at $6 for a total of $55. Oracle Retail Order Management does not differentiate between the price lists. Instead, it averages the final price between all of the hats, therefore all 10 hats are priced at $5.50 each for a total of $55. This may impact return processes, as in the above example, the return price of a hat would be $5.50 and not $5 or $6.

Payment Types

Payment Types are set up and configured using the Payment Processing setting and the Payment Types tab, which is described in the Oracle Commerce documentation. For information on Oracle Retail Order Management payment types, refer to the Oracle Retail Order Management documentation.

Shipping Methods

Oracle Retail Order Management uses shipping methods that are identified by unique integers. You can configure multiple shipping methods. For integration, Commerce shipping methods are mapped to Oracle Retail Order Management numeric shipping methods.

The shipping methods are sent to Oracle Retail Order Management during fulfillment. Commerce and Oracle Retail Order Management shipping methods should be synchronized. These mapping ensure that the orders created on Commerce refer to and use similar shipping method chosen by the customer for shipping.

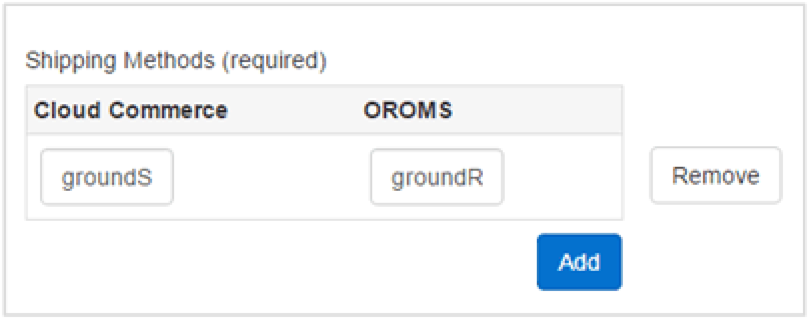
You can obtain a list of the shipping methods available by calling the following endpoint:

```
GET /ccadmin/v1/merchant/shippingMethods
```

This returns a list of available shipping methods. The response also includes information on each shipping method, and the repository ID of the shipping method. The repository ID identifies the shipping method code to use. The following example displays a partial response that identifies a ground shipping method:

```
"enabled": true,
"displaySequence": 0,
"eligibleForProductWithSurcharges": false,
"ranges": [
  {
    "amount": 4.75,
    "high": 14.99,
    "low": 0,
    "repositoryId": "groundRange1"
  }
```

Once you have obtained the repository ID, use it to map the shipping method:



Shipping Methods (required)

Cloud Commerce	OROMS
groundS	groundR

Shipping methods are set up and configured using the Shipping Methods setting, which is described in the Oracle Commerce documentation. For information for integrating with external shipping systems, refer to the Oracle Commerce documentation. For information on Oracle Retail Order Management shipping methods, refer to the Oracle Retail Order Management documentation.

External Shipping Methods

Commerce supports external shipping methods. However, these shipping methods may not be available in the list of shipping methods displayed in the administration interface when configuring your Oracle Retail Order Management integration. To use the shipping methods for your external system you must add them to the Shipping Methods mapping table.

Once all of the integration parameters have been configured and saved, the integration process can occur.

12 Integrate with Oracle Subscription Management

Integrating with Oracle Subscription Management allows shoppers on your Oracle Commerce site to purchase and manage their subscriptions.

Subscriptions are an increasingly popular way that allows your shoppers to buy products and services online. When you offer shopper subscriptions, you help strengthen customer relationships and take advantage of a solution that supports complex products and services.

Understand the Subscription Management integration

This integration between Oracle Commerce and Oracle Subscription Management lets shoppers on a Commerce site purchase and manage subscriptions.

Subscriptions are an increasingly popular way to buy products and services online. Offering subscriptions can help strengthen customer relationships with merchants and evolve their strategies by taking advantage of a solution that also supports complex products and services.

This integration offers the following benefits:

- Self-service for managing the entire life cycle of subscription management.
- Self-service capability for subscription management of complex configurable services
- Subscriptions can be created and modified seamlessly across Commerce, CPQ and Oracle Subscription Management.
- Allows merchants to offer subscriptions of configurable services to shoppers.

Prerequisites

The following software, account, and data prerequisites are required before you can install and configure the subscription.

- An Oracle Commerce account and access to the Oracle Commerce 20.3.12 or later.
- An Oracle CPQ account and access to Oracle CPQ 20D or later. You must have already configured the integration between Oracle CPQ and Commerce. See Configure the integration for more information.
- An Oracle Subscription Management Cloud account and access to Subscription Management Cloud 20C or later.
- Oracle Financials Account Receivables (AR)
- Oracle Customer Data Management (CDM)
- An Oracle Integration Cloud account and access to Oracle Integration Cloud Service.
- Oracle Product Master Data Management
- The Oracle CX Commerce to CPQ Get Assets integration - version 7. (Downloadable as OCCS_CPQ_ASSET_INTEGRATION_7.0.par).

Assumptions

This integration makes the following functional assumptions:

- Profile and Account are in sync with Oracle Customer Data Management.
- Products are in sync between Oracle Product Hub, Oracle CPQ, and Oracle Subscription Management Cloud.
- Only configurable products from CPQ are eligible for the integration and all subscription-based products need to be configured in CPQ with recurring prices.
- Asset Based Ordering must be enabled in CPQ, so that Asset Keys get generated as part of the configuration. These Keys are used to track the relationship between configuration details in Commerce items, Subscription products and CPQ Assets. For Commerce to store Asset Keys at Commerce item level, products in Commerce Catalog must have their `assetable` property set to `TRUE`.

Roadmap items and limitations in this release

This section describes limitations in the current release you should keep in mind before you begin working with this integration.

- An 0rder with modify action (or any other action) on asset includes price for the new asset/configuration, instead of the change in price.
- Cancellation charges are not supported by this integration.
- CPQ charge structure is not supported in this integration flow. For this flow to work in conjunction with the CPQ-Subscription flow, do not use the charge structure in the CPQ-Subscription flow.
- The Commerce-to-CPQ Quotation flow does not support recurring prices.
- A subscription order can have only one shipping group and one payment group.
- A subscription order can have only one shipping group and one payment group. When you design the checkout widgets on your storefront, make sure that shoppers cannot select split shipping or split payments for subscription orders.

Configure Oracle Commerce with Oracle Subscription Management

The first step in the subscriptions integration is configuring Oracle Commerce.

This section describes tasks you must perform to configure Commerce for the integration. You perform these tasks in the Commerce administration interface and with the Commerce REST APIs.

Register the application and generate a security token

This integration uses the Commerce REST APIs to access Commerce data. You must register the integration within Commerce and generate a security token in order for the integration to be granted access to the data.

To generate a security token:

1. Log into the Oracle Commerce administration interface.

2. Click the Settings menu and select Web APIs.
3. Click Registered Applications from the Web APIs panel.
4. Click the Register Application button.
5. Enter a name for the integration application. Create a meaningful name that reflects the purpose of the application.
6. Click Save. The Application ID and Application Key are automatically generated and the application is added to the Registered Applications page.
7. Click on the name of the application you created.
8. Click the Click to reveal link to display the application key. You can copy the application key to use as the security token for the Oracle Commerce Cloud connection.

For more information on managing an application within Oracle Commerce Cloud, see Register applications in *Extending Oracle Commerce*.

Configure the Oracle CPQ Configuration integration

Configure and enable the Oracle CPQ Configuration integration as described in Use Oracle CPQ Cloud Features.

The configuration of Commerce items and corresponding pricing rules must be defined in Oracle CPQ. External pricing details and recurring charge details for configured items are retrieved from CPQ during order submission flow.

See Entity mappings for relationships between item properties across different systems in the integration, including Oracle CPQ.

Configure the Commerce webhooks

You must configure the Order Submit and Order Validation webhooks as follows:

- The Order Submit webhook must point to the OIC Subscription Cloud integration URL.
- The Order Validation webhook must point to the Subscriptions validation SSE URL `https://<host>/ccstorex/custom/v1/validateSubscriptionOrder`.

Follow these steps to configure the webhooks in the Commerce administration interface:

1. Log into the Commerce administration interface.
2. Click the Settings icon.
3. Click Web APIs and then click the Webhook tab.
4. Click the Order Submit (Production) webhook. Enter the OIC Subscription Cloud integration URL in the URL box and enter the OIC username and password, under Basic Authorization.
5. Click the Order Validation (Production) webhook. Enter `https://<host>/ccstorex/custom/v1/validateSubscriptionOrder` in the URL box.
6. Click Save.

Create custom properties

This integration requires a number of custom properties that you create with the Commerce REST APIs. This section lists the custom properties you must create for accounts, profiles, addresses, orders, and commerce items.

Account custom properties

You must create the following custom property for accounts. For details, see Create custom properties for accounts in *Extending Oracle Commerce*.

`occ_partyId`: The party ID of the account in CDM.

Profile custom properties

You must create the following custom property for profiles. For details, see Manage Shopper Profiles in *Extending Oracle Commerce*.

`occ_partyId`: The party ID of the profile in CDM.

Address custom properties

You must create the following custom property for addresses. For details, see Work with address types in *Extending Oracle Commerce*.

This property will be used at order level addresses, such as shipping and billing addresses. Widgets must be customized to populate these properties on the storefront. The value needs to be populated from Account's or Contact's address `ExternalAddressId` property, which holds CDM's Address Number value.

`occ_AddressNumber`: The Address Number of the address in CDM.

Orders custom properties

You must create the following custom properties for orders. For details, see Create custom properties for orders in *Extending Oracle Commerce*. Widget changes are required for populating these properties while a shopper checks out the cart or submits the order.

- `occ_accountPartyId`: Used to map the primary party ID value of a subscription for account-based shoppers.
- `occ_contactPartyId`: For B2C subscriptions, used to map the primary party ID value of a subscription. For account-based subscriptions, used to map the contact value of a subscription.

Commerce item custom properties

You must create the following custom properties for commerce items. For details, see Create custom properties for line items in *Extending Oracle Commerce*. Widget changes are required for populating these properties while a shopper adds the item to the cart.

- `occ_assetActionReason`: Provides a custom action reason for close, suspend and resume operations.
- `occ_assetActionDate`: Provides a custom action date for close, suspend and resume operations.

Entity mappings

This section shows the relationships between entities across different systems in the integration. Not all entities are mapped to all systems in the integration.

B2B Account

- CDM: PartyNumber
  Commerce:externalOrganizationId

- CDM: PartyId
  Commerce: occ_partyId (custom)

B2B Contact

- CDM: PartyNumber
  customerContactId
- CDM: PartyId
  Commerce: occ_partyId (custom)

B2C Contact

- CDM: PartyNumber
  customerContactId
- CDM: PartyId
  Commerce: occ_partyId (custom)

Account/Contact Address

- CDM: AddressNumber
  Commerce: externalAddressId
- CDM: AddressId

B2B Order Account

- CDM: PartyId
  Commerce: occ_accountPartyId (custom)

  CPQ: Customer

  SMC: PrimaryPartyId

B2B Order Contact

- CDM: PartyId
  Commerce: occ_contactPartyId (custom)

  CPQ: _asset_custom_contactPartyId (custom)

  SMC: QuoteToContactId

B2B Order Address

- CDM: AddressNumber
  Commerce: occ_AddressNumber (custom)
- AR: CustomerAccountId
  CPQ: accountNumber_t

  SMC: BillToAccountId
- AR: CustomerAccountSite > CustomerAccountSiteUse > SiteUseId
  CPQ: billToSiteUseId_t

  SMC: BillToSiteId

B2C Order Contact

- CDM: PartyId
  Commerce: occ_contactPartyId (custom)

  CPQ: customer

  SMC: PrimaryPartyId

B2C Order Address

- CDM: AddressNumber
  Commerce: occ_AddressNumber (custom)

The following table shows the relationships between products and SKUs across systems.

Product Hub	Commerce	CPQ	Subscription Management Cloud
ItemId	N/A	PartnerPartId	InventoryItemId
ItemNumber	SkuId	PartNumber	ProductName
ItemDescription	DisplayName	Description	N/A

Configure the server-side extensions

The subscription integration functionality is provided through server-side extensions that run on the Node.js server associated with your Commerce environment. Download each extension from the Commerce administration server, then configure the extension and upload it to your Node.js server. The server-side extensions implement custom REST endpoints, which have the prefix `/ccstorex/custom` for the Commerce storefront and `/ccagentx/custom` for Oracle Commerce Agent.

For the Agent extensions, `shopperProfileId` should be included in the `X-CCAgentContext` header. For account-based shoppers, the `X-CCOrganization` header should also be present.

This section describes the server-side extensions that are included with the integration. For details about server-side extensions and how to develop them for use with Commerce, see Develop server-side extensions in *Extending Oracle Commerce*.

- `subscriptions-app-store.zip` Provides endpoints that fetch subscriptions and subscription products for the Commerce storefront. Also provides an endpoint that validates the order for both the storefront and the Agent console.
- `subscriptions-app-agent.zip` Provides endpoints that fetch subscriptions and subscription products for the Agent.
- `subscriptions-assets-store.zip` Provides endpoints that use Oracle CPQ to view and perform asset-based ordering actions on subscription assets. This extension is for the Commerce storefront.
- `subscriptions-assets-agent.zip` Provides endpoints that use Oracle CPQ to view and perform asset-based ordering actions on subscription assets. This extension is for the Agent.

Each ZIP file includes `readme.md` files that describe classes and endpoints and include information about how to install and extend the extensions.

Create environment variables

This section describes how to set environment variables required by the server-side extensions. The integration requires the following environment variables:

- `OIC_USERNAME`: Specifies the basic authentication username of OIC.
- `OIC_PASSWORD`: Specifies the basic authentication password of OIC.

- `CPQ_USERNAME`: Specifies the basic authentication username for requests that go directly to Oracle CPQ.
- `CPQ_PASSWORD`: Specifies the basic authentication password for requests that go directly to Oracle CPQ.
- `OSS_USERNAME`: Specifies the basic authentication username for requests that go directly to Oracle Subscription Management.
- `OSS_PASSWORD`: Specifies the basic authentication password for requests that go directly to Oracle Subscription Management.

The following example issues a POST request to the `doCreateExtensionVariable` endpoint that sets the `OSS_USERNAME`:

```
/ccadmin/v1/extensionEnvironmentVariables   POST
 {
    "name": "OSS_USERNAME",
    "value": "conmgr"
}
```

Work with the SSE endpoints

This section describes the storefront and Agent endpoints included in the server-side extensions. All the endpoints are authenticated URLs and all requests must be sent via HTTPS.

- /ccstorex/custom/v1/subscriptions?q={query param}
  /ccagentx/custom/v1/subscriptions?q={query param}

  Issue a GET request to return the details about a particular shopper’s subscriptions.
- /ccstorex/custom/v1/subscriptions/{subscriptionNumber}
  /ccagetx/custom/v1/subscriptions/{subscriptionNumber}

  Issue a GET request to return the details of a subscription.
- /ccstorex/custom/v1/subscriptions/{subscriptionNumber}/products?q={query param}
  /ccagentx/custom/v1/subscriptions/{subscriptionNumber}/products?q={query param}

  Issue a GET request to return the line items of a subscription.
- /ccstorex/custom/v1/subscriptions/{subscriptionNumber}/products/{subscriptionProductPuid}
  /ccagentx/custom/v1/subscriptions/{subscriptionNumber}/products/{subscriptionProductPuid}

  Issue a GET request to return the details of a subscription order line item.
- /ccstorex/custom/v1/validateSubscriptionOrder
  Issue a POST request to validate a subscription order. This endpoint can be used for both storefront and Agent orders.
- /ccstorex/custom/v1/assets
  /ccagentx/custom/v1/assets

  Issue a GET request to get all assets for a particular shopper.
- /ccstorex/custom/v1/assets/{assetId}
  /ccagentx/custom/v1/assets/{assetId}

  Issue a GET request to get an asset based on asset ID.

- /ccstorex/custom/v1/assets/{assetId}/modify
  /ccagentx/custom/v1/assets/{assetId}/modify

  Issue a POST request to modify an asset(with CPQ punch-in).
- /ccstorex/custom/v2/assets/{assetId}/modify
  /ccagentx/custom/v2/assets/{assetId}/modify

  Issue a POST request to modify an asset (with CPQ configurator API).
- /ccstorex/custom/v1/assets/{assetId}/renew
  /ccagentx/custom/v1/assets/{assetId}/renew

  Issue a POST request to renew an asset.
- /ccstorex/custom/v1/assets/{assetId}/resume
  /ccagentx/custom/v1/assets/{assetId}/resume

  Issue a POST request to resume an asset.
- /ccstorex/custom/v1/assets/{assetId}/upgrade
  /ccagentx/custom/v1/assets/{assetId}/upgrade

  Issue a POST request to upgrade an asset. (with CPQ punchin).
- /ccstorex/custom/v2/assets/{assetId}/upgrade
  /ccagentx/custom/v2/assets/{assetId}/upgrade

  Issue a POST request to upgrade an asset. (with CPQ configurator API).
- /ccstorex/custom/v1/assets/{assetId}/terminate
  /ccagentx/custom/v1/assets/{assetId}/terminate

  Issue a POST request to terminate an asset.
- /ccstorex/custom/v1/assets/{assetId}/suspend
  /ccagentx/custom/v1/assets/{assetId}/suspend

  Issue a POST request to suspend an asset.

Create SSE routes to validate billing addresses

Oracle Subscription Management expects an account level address for a business account subscription. If an address which is linked to a contact is passed, Subscription Management rejects it with an appropriate error.

You should design your storefront in a way that allows contacts to select only account addresses when placing subscription orders. In the case where a contact is allowed to enter a contact-level address instead, it is a good idea to validate the billing address before the order is submitted.

If a contact enters their billing address instead of the account's billing address when creating a subscription order, the integration fails and the following error is logged: `The value of the attribute Billing Account is invalid. (OKC-195787).`

Since the billing details can be updated even after order validation, this validation can't be handled as part of the order validation logic and hence this validation is not available by default. You can avoid this error by adding validation code to the payment webhook (`genericCardPayment` webhook for credit card payments and `genericPayment` wehook for invoice payments) and creating a new SSE route (`/v1/validateSubscriptionOrderBillingAddress`) to perform the validation. Point the webhook to the newly created route.

For details about account and contact addresses, see Manage an Account-based Storefront in *Using Oracle Commerce*.

Configure payments

In this release of the integration, Commerce shoppers can pay for subscription orders with either a credit card or an invoice. For credit cards, create a custom CyberSource integration using the Generic Payment webhook. A payment gateway configured with the Generic Payment webhook handles stored credit cards and tokens, which , which are required for this integration. See Create a Generic Payment Gateway Integration in *Extending Oracle Commerce* for more information.

Credit card payments

For credit card payments to work properly, Commerce must send a multi- use token to Oracle Subscription Management.

Commerce does not store the complete credit card data. Instead, when a shopper stores a credit card, the payment processor associated with the payment gateway sends back a token that represents the card number. The token is used for each transaction associated with a subscription payment.

When the shopper submits a subscription order, Commerce, through the gateway, authorizes a one-time payment against the card. When you implement this integration, you should populate the muilti use token for the credit card used to pay for the order in `statusProps`.

After the initial one-time charge, further subscription and billing is not authorized by Commerce. Instead, Commerce will send the multi-use token to Oracle Subscription Management, which, in turn, sends the token to the Oracle Accounts Receivable module.

As part of the payment integration, you need to return back a reusable credit card token to the generic payment webhook response in the property `token` under `statusProps` under `authorizationStatus`. This value will be sent to Oracle Subscription Management for the periodic billing.

```
"paymentGroups": [
  {
    "authorizationStatus": [
      {
        "amount": 50,
        "statusProps": {
          "token": "12345678"
        }
      }
    ]
  }
]
```

Install and Configure the Integration in OIC

This section describes how to install the integration package in Oracle Integration Cloud (OIC).

You must download and import the integration packages and then configure settings in OIC.

Download and import the integration

The integration package includes two integrations:

- `Oracle_CXCommerce_SubscriptionCloud_Integration` is the main integration. It contains all the subscription flows, such as create, modify, terminate, suspend, resume, and renew.
- `Oracle Financials Bill To Account` is the integration flow that is called from the main Subscription flow.

To download the integration package, find the `Oracle_CXCommerce_SubscriptionCloud_Integration.par` on the Oracle Cloud Marketplace and download it to your local system.

To import the OIC Integration Package:

1. Log on to OIC as an admin user.
2. Click the Packages icon.
3. Click the Import button.
4. Click Browse to open a navigation pane.
5. Select the integration package archive (PAR) file you want to import.
6. Click Import.
   The package is added to the Packages list.

Configure the connections

After you install the `Oracle_CXCommerce_SubscriptionCloud_Integration.par` package, you must configure the connections used by the integration.

1. Log in to OIC as an admin user.
2. Select Integration->Connections.
3. Select Oracle Commerce. The Connection Properties dialog appears.
   - Enter the URL to connect to Oracle Commerce as the value of the Connection Base URL property (`https://<hostname>/ccadmin/v1`).
   - Enter the security token value, which you can find in the Oracle Commerce administration settings and click OK. The security token is the application key in the Oracle Commerce administration interface, under Registered Applications Settings. Contact your Oracle Commerce administrator to get this application key.
4. Select Sample Sales Cloud. This is connection to Oracle Subscription Cloud.
   - Enter the Subscription Cloud Services Catalog WSDL URL, Interface Catalog URL, and the Security (Username Password Token). For example:
     OSC Services Catalog WSDL URL: `https://host/fscmService/ServiceCatalogService?wsdl`

     Interface Catalog URL: `https://host/helpPortalApi/otherResources/latest/interfaceCatalogs`
5. Select Oracle Engagement Cloud. This is connection to Oracle Customer Data Management.

Enter the OSC Services Catalog WSDL URL, Interface Catalog URL, and Security (Username Password Token). For example:

OSC Services Catalog WSDL URL: `https://host/fscmService/ServiceCatalogService?wsdl`

Interface Catalog URL: `https://host/helpPortalApi/otherResources/latest/interfaceCatalogs`

6. Select Oracle REST Trigger Employee Service. This is the trigger connection for the child integration Oracle Financials Bill To Account Integration. No configuration is necessary; just test and save the trigger.
7. Select CDM_AR. This is connection to Oracle Financials Accounts Receivable.
Enter the WSDL URL and Security (Username Password Token). For example: `https://host/foundationParties/CustomerAccountService?WSDL`

Assign values to constants in OIC integration flows

The following table describes the default constants values in the integration flows. You must change these default values so they match your system values.

Constant	Value
subscriptionProfile	zOSS_SP_ServiceStartFixed_Advance_Month
InternalApproval	NOTREQUIRED
SourceSystem	CX_CLOUD_COMMERCE
GenerateBillingSchedule	Y
GenerateBillingScheduleForfullperiod	Y
ChargeDefinition	QP_RECURRING_SALE_PRICE, QP_SALE_PRICE
ExternalKey	Recurring fee, Activation fee
ExternalParentKey	Mapped to catalogRefId in order payload
PriceSystem	CPQ
name	Activate , resume, suspend and close
RelationShipTypeCode	RENEW, AMEND
PriceType	ORA_ONE_TIME, ORA_RECURRING

The constants are configured in mappers of different action nodes, such as `createSubscriptions_OSS`, `SuspendProduct`, and `ResumeProduct`. Follow these steps if you need to update any of the default values in the mapper:

1. Log into Oracle Integration Cloud (OIC).
2. Go to the integration Oracle CX Commerce To Subscription Integration.
3. Edit the mapper of the action node for which you want to update the constant. Enter the new value in respective mapping.
4. Save and close.

Configure lookups

You must update the following lookup tables to map the following relationships:

- Site Merchant Details – Provides a mapping relationship between Site id, Definition Organization ID, Business Unit ID and Legal entity ID.

- Payment Methods – Provides a mapping relationship between the payment method terms in Commerce and Oracle Subscription Management Cloud.
- Reason_CreditMethod_Lookup – Provides a mapping relationship between the action code of items and their default reason for some asset-based-order actions and their corresponding credit methods.
- Duration_OCC_OSS – Provides the mapping relationship between the duration terms in Commerce and their corresponding time code units in Oracle Subscription Management Cloud.

Activate the integration flows

After you configure the Oracle Subscription Management Cloud and Oracle Commerce connections, you must activate the integrations that were created when the integration package was imported to Oracle Integration Cloud. To do this, follow these steps:

1. Log in to Oracle Integration Cloud (OIC) as an admin user.
2. Click Integrations.
3. Select each of the following integrations and click its Activate button.
   Oracle CX Commerce To Subscription Integration

   Oracle Financials Bill To Account Integration

OIC displays a message to indicate that the integration flow was successfully activated.

Customize Storefront Widgets

You must customize two widgets and add them to your storefront layouts so that shoppers can view and work with subscriptions assigned to their accounts.

This chapter describes how to customize the following widgets:

- The customized Assets widget lets shoppers view all their subscriptions and each subscription’s line items. This widget uses the Subscriptions-app SSE module to fetch the details from Oracle Subscription Cloud.
- The customized Asset Details widget lets a shopper modify, renew, cancel, or change a subscription. This widget uses the Subscription-assets-store SSE module to perform some of the predefined asset actions, such as renew, modify, terminate, suspend, or resume.

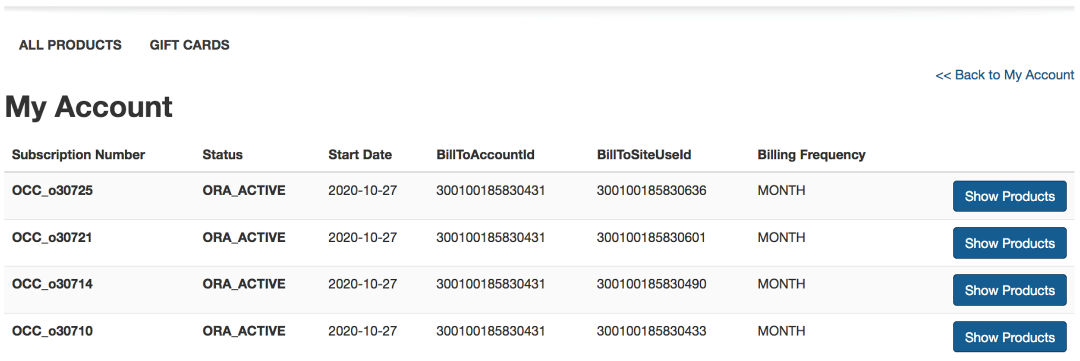
This chapter describes the changes you need to make to the Assets and Asset Details widgets so they can display subscriptions. For information about how to access the code for default widgets so you can edit it, see Customize layout components in *Using Oracle Commerce*.

Note: The SSEs must be installed and configured before you can customize the widgets and use them on the storefront.

Customize the Assets widget

The Assets widget lets shoppers view a list of services associated with their account. You can customize this widget so that it displays a list of subscriptions for the logged in user.. This widget uses the Subscriptions-app server-side extension module to fetch the details from Oracle Subscription Cloud. The Assets widget appears on the Assets layout.

The following illustration shows a shopper's subscriptions, displayed in the widget.



My Account

Subscription Number	Status	Start Date	BillToAccountId	BillToSiteUseId	Billing Frequency	
OCC_o30725	ORA_ACTIVE	2020-10-27	300100185830431	300100185830636	MONTH	Show Products
OCC_o30721	ORA_ACTIVE	2020-10-27	300100185830431	300100185830601	MONTH	Show Products
OCC_o30714	ORA_ACTIVE	2020-10-27	300100185830431	300100185830490	MONTH	Show Products
OCC_o30710	ORA_ACTIVE	2020-10-27	300100185830431	300100185830433	MONTH	Show Products

When the shopper clicks the Show Products button, the Subscription Asset Details widget displays in the Asset Details layout.

Customize the Assets widget code

By default, the Assets widget shows the root level assets on the assets page and clicking on the Details link redirects the shopper to the asset details. For this integration, you will customize the widget to show the subscriptions on the page first, then give the shopper the option to view subscription products. By checking details of a particular subscription product, the shopper will be directed to the root asset details associated with the product.

To make these changes to the Assets widget, update the widget's `.js` file.

First, add/update the following constants:

```
  var GET_ALL_SUBSCRIPTIONS = "getAllSubscription";
  var GET_ALL_SUBSCRIPTION_PRODUCTS = "getAllSubscriptionProducts";
  var ENDPOINT_VIEW_ACCOUNT_ASSET =    "getServices";
```

Next, add the following observables:

```
currentSubscriptionNumber: ko.observable(), productPageSize:
      ko.observableArray([]), productsPageSize:
      ko.observable(ccConstants,DEFAULT_SEARCH_RECORDS_PER_PAGE || 12),
productsOffset:
      ko.observable(0) productshasMore: ko.observable(false),
productsTotalResults:
      ko.observable(0), showProductsFlag: ko.observable(false),
```

Next, update the `beforeAppear` function:

```
beforeAppear: function (page) {
      var widget = this;

      if ( !this.user().loggedIn() ) {
        navigation.doLogin(navigation.getPath(), this.links().home.route);
      }
      widget.showProductsFlag(false);
    }
```

Finally, add/update the following methods inside the `onload` function:

```
        onLoad: function (widget) {
      // Add the Services SSE endpoints to the ccRestClient endpoint
registry.
      // Update the settings below if the Services SSE has been
customized.
      // ENSURE THAT THE SERVICES SSE IS INSTALLED, CONFIGURED AND
AVAILABLE
      ccRestClient.registerInitCallback(function(){

        ccRestClient.endpointRegistry[ENDPOINT_VIEW_ACCOUNT_ASSET] = {
          "authRequired": true,
          "cachingEnabled": false,
          "hasDoc": false,
          "hasPathParams": true,
          "httpsRequired": false,
          "id": ENDPOINT_VIEW_ACCOUNT_ASSET,
          "localeHint": "assetLanguageOptional",
          "method": "GET",
          "requestType": "application/json",
          "responseType": "application/json",
          "singular": false,
          "url": "/ccstorex/custom/v1/assets",
          "useOptimisticLock": false
        };

        ccRestClient.endpointRegistry.getAllSubscription = {
          "authRequired": true,
          "cachingEnabled": false,
          "hasDoc": false,
          "hasPathParams": false,
          "httpsRequired": false,
          "id": GET_ALL_SUBSCRIPTIONS,
          "localeHint": "assetLanguageOptional",
          "method": "GET",
          "requestType": "application/json",
          "responseType": "application/json",
          "singular": false,
          "url": "/ccstorex/custom/v1/oss/subscription/getAll",
          "useOptimisticLock": false
        }

        ccRestClient.endpointRegistry.getAllSubscriptionProducts = {
          "authRequired": true,
          "cachingEnabled": false,
          "hasDoc": false,
          "hasPathParams": false,
          "httpsRequired": false,
          "id": GET_ALL_SUBSCRIPTION_PRODUCTS,
          "localeHint": "assetLanguageOptional",
          "method": "GET",
          "requestType": "application/json",
          "responseType": "application/json",
          "singular": false,
          "url": "/ccstorex/custom/v1/oss/subscription/products",
          "useOptimisticLock": false
        }

      });
            // Use the widget's asetsPerPage config
      // option value if it has been set
      if ( widget.assetsPerPage && !isNaN(widget.assetsPerPage()) ) {
        widget.pageSize(10);
      }

      // Computeds for paging control
      widget.totalPages =  ko.pureComputed(function() {
        var returnValue = Math.ceil( widget.totalResults() /
widget.pageSize() );
        return returnValue;
      });

      widget.currentPage = ko.pureComputed(function() {
        var returnValue = Math.ceil( ( widget.offset() +
widget.pageSize() ) / widget.pageSize() );
        return returnValue;
      });

      widget.previousPage = ko.pureComputed(function() {
        var calculatedPreviousPage = widget.currentPage() - 1;
        var returnValue = ( ( calculatedPreviousPage < 1 ) ? 1 :
calculatedPreviousPage );
        return returnValue;
      });

      widget.nextPage = ko.pureComputed(function() {
        var calculatedNextPage = widget.currentPage() + 1;
        var returnValue = ( ( calculatedNextPage > widget.totalPages ) ?
widget.totalPages : calculatedNextPage );
        return returnValue;
      });

      widget.onFirstPage = ko.pureComputed(function() {
        return ( widget.currentPage() === 1 );
      });

      widget.onLastPage = ko.pureComputed(function() {
        var returnValue = false;

        if ( widget.totalPages() > 1 )  {
          if ( widget.currentPage() === widget.totalPages() ) {
            returnValue = true;
          }
        }
        else if ( !widget.onFirstPage() && !widget.hasMore() ) {
          returnValue = true;
        }

        return returnValue;
```

```
    });

    widget.pageLinks = ko.pureComputed(function() {
      // This would be a good place to do something more
      // sensible with the individual page links that
      // are displayed when there are a large number of
      // results e.g. could display just the 5 pages either
      // side of the current page. For now display every
      // individual page.
      var links = [];

      for (var i = 1; i <= widget.totalPages(); i++) {
        links.push({
          pageNumber : i,
          active : i === widget.currentPage()
        });
      }

      return links;
    });

widget.shouldShowGoToLastPage = ko.pureComputed(function() {
      return ( widget.totalPages() > 1 );
    });

    widget.isPagingRequired = ko.pureComputed(function() {
      var returnValue = false;

      if ( widget.totalPages() > 1 ) {
        returnValue = true;
      }
      else if ( widget.hasMore() ) {
        returnValue = true;
      }
      else if ( !widget.onFirstPage() ) {
        returnValue = true;
      }

      return returnValue;
    });

    // The goToPage function handles click events from the
    // template's paging links. It takes a single input
    // parameter - pageNumber - which indicates the page
    // of assets to load. If the REST call is successful
    // the assets observable is updated with the returned
    // data and the three paging observables (offset, hasMore
    // and totalResults) are updated with the respective
    // values returned from the REST call.
    widget.goToPage = function (pageNumber) {

      function success (data) {
        var productQuery = '';
        var productIdsSet =  new Set();
```

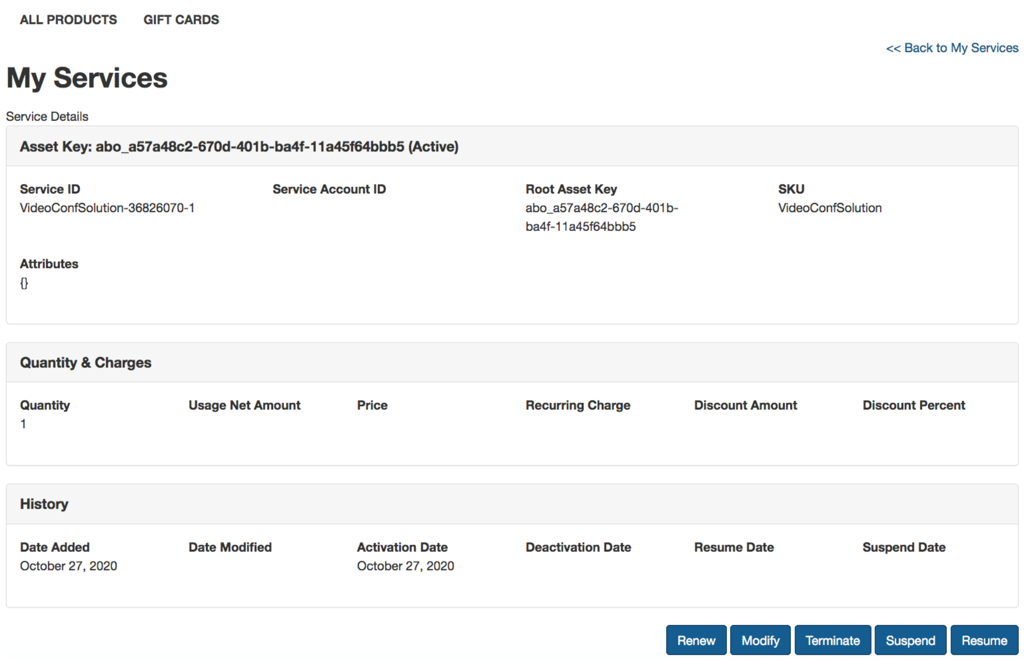
```
              widget.assets(data.items);
              widget.offset(data.offset);
              widget.hasMore(data.hasMore);
              // if totalPages is not returned (i.e. is null)
              // then set to -1; in this scenario only simple
              // paging will be available (i.e. go to first page,
              // go to previous page and go to next page) and
              // the hasMore value will be used to control
              widget.totalResults(data.totalResults || -1);

              spinner.destroyWithoutDelay(widget.spinnerOptions.parent);
           // }

          }

          function error (data) {
            if (data.status == ccConstants.HTTP_UNAUTHORIZED_ERROR) {
              widget.user().handleSessionExpired();

              navigation.doLogin(navigation.getPath,
widget.links().home.route);

            } else {
              navigation.goTo(widget.links().profile.route);
            }

            spinner.destroyWithoutDelay(widget.spinnerOptions.parent);
          }

          if ( widget.user().loggedIn() ) {
            var calculatedOffset = ( pageNumber - 1 ) * widget.pageSize();

            spinner.create(widget.spinnerOptions);
            var queryString = 'Status=ORA_ACTIVE';
            var payload = {
              limit: 10,
              offset:calculatedOffset,
              q: queryString,
              orderBy:"CreationDate:desc"
            };

            ccRestClient.request(
             GET_ALL_SUBSCRIPTIONS,
             payload,
             success,
             error
            );
          }
        };

        widget.handleQuickViewClick = function(pIsModal) {
          var popup;
          if(pIsModal === true && this.productDetails) {
            widget.productDetails(this.productDetails);
            popup = $("#cc-upgrade-asset-display");
            popup.modal('show');
          }
        };

        //---------------------------------------------------//
          //Function Specific to subscription products table
          widget.productTotalPages = ko.pureComputed(function(){
            var returnValue = Math.ceil(widget.productsTotalResults()/
widget.productPageSize())
            return returnValue;
          });

          widget.productCurrentPage = ko.pureComputed(function(){
            var returnValue = Math.ceil((widget.productsOffset()
+widget.pageSize())/widget.pageSize());
            return returnValue;
          });

          widget.previousProductPage = ko.pureComputed(function() {
            var calculatedPreviousPage = widget.productCurrentPage()-1;
            var  returnValue = ( ( calculatedPreviousPage < 1 ) ? 1 :
calculatedPreviousPage );
            return returnValue;
          });

          widget.nextProductPage =  ko.pureComputed(function() {
            var calculatedNextPage = widget.productCurrentPage() + 1;
            var returnValue = ( ( calculatedNextPage >
widget.productTotalPages ) ? widget.productTotalPages :
calculatedNextPage );
            return returnValue;
          });

          widget.onProductFirstPage = ko.pureComputed(function() {
            return ( widget.productCurrentPage() === 1 );
          });

          widget.onProductLastPage = ko.pureComputed(function() {
            var returnValue = false;

            if ( widget.productTotalPages() > 1 )  {
              if ( widget.productCurrentPage() ===
widget.productTotalPages() ) {
              returnValue = true;
              }
            }
            else if ( !widget.onProductFirstPage() && !
widget.productshasMore() ) {
              returnValue = true;
            }

            return returnValue;
          });

          widget.shouldShowGoToProductsLastPage =
ko.pureComputed(function() {
            return ( widget.productTotalPages() > 1 );
          });

          widget.isProductPagingRequired = ko.pureComputed(function() {
            var returnValue = false;

            if ( widget.productTotalPages() > 1 ) {
              returnValue = true;
            }
            else if ( widget.productshasMore() ) {
              returnValue = true;
            }
            else if ( !widget.onFirstPage() ) {
              returnValue = true;
            }

            return returnValue;
          });

          widget.productPageLinks = ko.pureComputed(function() {
            var links = [];

            for (var i = 1; i <= widget.productTotalPages(); i++) {
              links.push({
              pageNumber : i,
              active : i === widget.productCurrentPage()
              });
            }

            return links;
          });

           widget.goToProductPage = function(pageNumber) {
            function showProductSuccess(data) {
              widget.subscriptionProducts(data.items);
              widget.productsOffset(data.offset);
              widget.productshasMore(data.hasMore);
              widget.productsTotalResults(data.totalResults || -1);
              spinner.destroyWithoutDelay(widget.spinnerOptions.parent);
            }

            function showProductFailure(err) {
              if (data.status == ccConstants.HTTP_UNAUTHORIZED_ERROR) {
              widget.user().handleSessionExpired();

              navigation.doLogin(navigation.getPath,
widget.links().home.route);

              } else {
              navigation.goTo(widget.links().profile.route);
              }

              spinner.destroyWithoutDelay(widget.spinnerOptions.parent);
            }
          if (widget.user().loggedIn()){
            var calculatedOffset = ( pageNumber - 1 ) *
widget.pageSize();
            //let queryString = `SubscriptionNumber=$
{widget.currentSubscriptionNumber()};Status=ORA_ACTIVE;ExternalParentAs
setKey is NULL`;
            //var queryString = "SubscriptionNumber=" +
widget.currentSubscriptionNumber();
           var queryString = "SubscriptionNumber=" +
widget.currentSubscriptionNumber();
            var payload = {
              limit: 25,
              offset:calculatedOffset,
              q: queryString,
              orderBy:"StartDate:desc"
            };
            ccRestClient.request(
              GET_ALL_SUBSCRIPTION_PRODUCTS,
              payload,
              showProductSuccess,
              showProductFailure
            );
          }
        };

        widget.onShowProductsClicked = function(p1, p2) {
          widget.subscriptionProducts([]);
          widget.currentSubscriptionNumber(p1.SubscriptionNumber);
          widget.showProductsFlag(true);
          spinner.create(widget.spinnerOptions);
          widget.goToProductPage(1);

        }

        widget.backToSubscriptionTable = function(p1, p2) {
          widget.showProductsFlag(false);
        }

        widget.redirectToAssetDetailsPage = function(p1, p2) {
          //let assetKeys = p1.ExternalAssetKey;
          var assetKeys = null;
          if(p1.ExternalRootAssetKey) {
              assetKeys= p1.ExternalRootAssetKey;
          } else {
            assetKeys = p1.ExternalAssetKey;
          }
          var payload = {
           // q: queryString,
            limit: 10,
            offset:0,
            assetKeys:assetKeys
          };

          // Call WAPI.
          ccRestClient.request(
           ENDPOINT_VIEW_ACCOUNT_ASSET,
           payload,
           widget.getAssetDetailsSuccess,
           widget.getAssetDetailsError
          );
        };

        widget.getAssetDetailsSuccess = function(data) {
          /* if(data && data.items) {
              data.items.forEach(function(item){
                $.extend(item, {
                  route:
                  `${widget.links().assetDetails.route}/${item.assetId}`
                })
              })
          } */
          if(data.items.length>0)
            navigation.goTo(widget.links().assetDetails.route  + "/" +
data.items[0].assetId);
          else
            navigation.goTo(widget.links().profile.route);
        };

        widget.getAssetDetailsError = function(data) {
          if (data.status == ccConstants.HTTP_UNAUTHORIZED_ERROR ||
data.status == ccConstants.BAD_REQUEST) {
            widget.user().handleSessionExpired();
            navigation.doLogin(navigation.getPath,
widget.links().home.route);
          }
          else {
            navigation.goTo(widget.links().profile.route);
          }
          spinner.destroyWithoutDelay(widget.spinnerOptions.parent);
        };
```

Customize the Asset Details widget

The Asset Details widget lets an account-based contact perform asset-based ordering actions on asset. This customized widget uses the Subscription-assets-store server-side extension module to perform some of the predefined asset actions. This widget appears on the Asset Details layout.

- The widget’s `display.template` file includes predefined action buttons that let shoppers perform subscription upgrades, renewals, modifications, cancellations, and suspends/ resumes. You can enable or disable these options, based on the asset status or rules you have defined in Oracle CPQ. The widget uses the subscription-service-assets-details SSE to perform these actions.
- Once a shopper clicks one of the action buttons, the asset is added to their cart with the required action code or a configurator window opens with asset details.
- The actions are performed based on the asset’s ID, which the SSE endpoints fetch from CPQ. The following illustration shows the details of a subscription, displayed in the widget.

The following illustration shows the details of a subscription, displayed in the widget.



My Services

Service Details

Asset Key: abo_a57a48c2-670d-401b-ba4f-11a45f64bbb5 (Active)

Service ID	Service Account ID	Root Asset Key	SKU
VideoConfSolution-36826070-1		abo_a57a48c2-670d-401b-ba4f-11a45f64bbb5	VideoConfSolution

Attributes
{}

Quantity & Charges

Quantity	Usage Net Amount	Price	Recurring Charge	Discount Amount	Discount Percent
1					

History

Date Added	Date Modified	Activation Date	Deactivation Date	Resume Date	Suspend Date
October 27, 2020		October 27, 2020			

Customize the Asset Details widget

To customize the Asset Details widget, you must update the endpoint references. All the REST API paths must be changed from `/ccstorex/custom/v1/services*` to `/ccstorex/custom/v1/assets*`.

A total of seven endpoints must be updated. In the widget's `asset_details.js` file. replace the call to `ccRestClient.registerInitCallback()` in the `onLoad()` function with the following:

```
    onLoad: function (widget) {

      // Add the Subscription-assets-store SSE endpoints to the
ccRestClient endpoint registry.
      // Update the settings below if the SSE has been customized.
      // ENSURE THAT THE SSE IS INSTALLED, CONFIGURED AND AVAILABLE

      ccRestClient.registerInitCallback(function(){

        ccRestClient.endpointRegistry[ENDPOINT_VIEW_ACCOUNT_ASSET] = {
          "authRequired": true,
          "cachingEnabled": false,
          "hasDoc": false,
          "hasPathParams": true,
          "httpsRequired": false,
          "id": ENDPOINT_VIEW_ACCOUNT_ASSET,
          "localeHint": "assetLanguageOptional",
          "method": "GET",
          "requestType": "application/json",
          "responseType": "application/json",
          "singular": false,
          "url": "/ccstorex/custom/v1/assets/{}",
          "useOptimisticLock": false
```

```
};

ccRestClient.endpointRegistry[ENDPOINT_RENEW_ACCOUNT_ASSET] = {
  "authRequired": true,
  "cachingEnabled": false,
  "hasDoc": false,
  "hasPathParams": true,
  "httpsRequired": false,
  "id": ENDPOINT_RENEW_ACCOUNT_ASSET,
  "localeHint": "assetLanguageOptional",
  "method": "POST",
  "requestType": "application/json",
  "responseType": "application/json",
  "singular": false,
  "url": "/ccstorex/custom/v1/assets/{}/renew",
  "useOptimisticLock": false
};

ccRestClient.endpointRegistry[ENDPOINT_MODIFY_ACCOUNT_ASSET] = {
  "authRequired": true,
  "cachingEnabled": false,
  "hasDoc": false,
  "hasPathParams": true,
  "httpsRequired": false,
  "id": ENDPOINT_MODIFY_ACCOUNT_ASSET,
  "localeHint": "assetLanguageOptional",
  "method": "POST",
  "requestType": "application/json",
  "responseType": "application/json",
  "singular": false,
  "url": "/ccstorex/custom/v1/assets/{}/modify",
  "useOptimisticLock": false
};

ccRestClient.endpointRegistry[ENDPOINT_TERMINATE_ACCOUNT_ASSET] = {
  "authRequired": true,
  "cachingEnabled": false,
  "hasDoc": false,
  "hasPathParams": true,
  "httpsRequired": false,
  "id": ENDPOINT_TERMINATE_ACCOUNT_ASSET,
  "localeHint": "assetLanguageOptional",
  "method": "POST",
  "requestType": "application/json",
  "responseType": "application/json",
  "singular": false,
  "url": "/ccstorex/custom/v1/assets/{}/terminate",
  "useOptimisticLock": false
};

ccRestClient.endpointRegistry[ENDPOINT_SUSPEND_ACCOUNT_ASSET] = {
  "authRequired": true,
  "cachingEnabled": false,
  "hasDoc": false,
  "hasPathParams": true,
          "httpsRequired": false,
          "id": ENDPOINT_SUSPEND_ACCOUNT_ASSET,
          "localeHint": "assetLanguageOptional",
          "method": "POST",
          "requestType": "application/json",
          "responseType": "application/json",
          "singular": false,
          "url": "/ccstorex/custom/v1/assets/{}/suspend",
          "useOptimisticLock": false
        };

        ccRestClient.endpointRegistry[ENDPOINT_RESUME_ACCOUNT_ASSET] =
{
          "authRequired": true,
          "cachingEnabled": false,
          "hasDoc": false,
          "hasPathParams": true,
          "httpsRequired": false,
          "id": ENDPOINT_RESUME_ACCOUNT_ASSET,
          "localeHint": "assetLanguageOptional",
          "method": "POST",
          "requestType": "application/json",
          "responseType": "application/json",
          "singular": false,
          "url": "/ccstorex/custom/v1/assets/{}/resume",
          "useOptimisticLock": false
        };

        ccRestClient.endpointRegistry[ENDPOINT_UPGRADE_ACCOUNT_ASSET]
= {
                "authRequired": true,
                "cachingEnabled": false,
                "hasDoc": false,
                "hasPathParams": true,
                "httpsRequired": false,
                "id": ENDPOINT_UPGRADE_ACCOUNT_ASSET,
                "localeHint": "assetLanguageOptional",
                "method": "POST",
                "requestType": "application/json",
                "responseType": "application/json",
                "singular": false,
                "url": "/ccstorex/custom/v1/assets/{}/upgrade",
                "useOptimisticLock": false
              };

      });
```

Customize text in the widgets

In addition to updating the widget's code, you can optionally modify its display text so that shoppers understand that the widgets specifically show details about subscriptions. For example, you could change the word services to subscriptions wherever it appears in widget text. To learn how to modify the text a widget displays, see Modify a component's code in *Using Oracle Commerce*.

Integration Flows

This integration includes six process flows.

This integration includes the following process flows:

- Create/View Subscription
- Modify Subscription
- Renew Subscription
- Terminate Subscription
- Suspend Subscription
- Resume Subscription

To learn more about the product abbreviations used in these flows, see Understand the Subscription Management Integration.

Create / View Subscription

The integration creates subscriptions of complex configurable services with recurring prices and flexible durations as defined in CPQ.

1. CDM: Get AddressId using AddressNumbers (shipping and billing addresses) for Account
2. CDM: If Account addresses are not found, get AddressId using AddressNumbers (shipping and billing addresses) for Contact
3. CDM: If Contact addresses are not found, create the addresses
4. AR: Check if a site exists for the addresses
5. AR: If not, then create a site
6. Subscription: Create draft subscription
7. Subscription: Add contact for account-based subscription
8. Subscription: Activate subscription

Modify Subscription

The integration modifies configurations, changes quantities, upgrades or downgrades the service as defined in the CPQ configuration.

1. CDM: Get AddressId using AddressNumbers (shipping and billing addresses) for Account
2. CDM: If Account addresses are not found, get AddressId using AddressNumbers (shipping and billing addresses) for Contact
3. CDM: If Contact addresses are not found, create the addresses
4. AR: Check if a site exists for the addresses
5. AR: If not, then create a site
6. Subscription: Get subscription lines details for each line in a loop
7. Subscription: Add/Update/Delete subscription lines for each line in a loop
8. Subscription: Activate subscription

Suspend Subscription

The integration suspends, but does not terminate, an active subscription.

1. CDM: Get AddressId using AddressNumbers (shipping and billing addresses) for Account
2. CDM: If Account addresses are not found, get AddressId using AddressNumbers (shipping and billing addresses) for Contact
3. CDM: If Contact addresses are not found, create the addresses
4. AR: Check if a site exists for the addresses
5. AR: If not, then create a site
6. Subscription: Get subscription lines details for each line in a loop
7. Subscription: Update subscription lines for each line in a loop

Resume Subscripiton

The integration resumes a suspended subscription.

1. CDM: Get AddressId using AddressNumbers (shipping and billing addresses) for Account
2. CDM: If Account addresses are not found, get AddressId using AddressNumbers (shipping and billing addresses) for Contact
3. CDM: If Contact addresses are not found, create the addresses
4. AR: Check if a site exists for the addresses
5. AR: If not, then create a site
6. Subscription: Get subscription lines details for each line in a loop
7. Subscription: Update subscription lines (Resume SubscriptionProducts) for each line in a loop
8. Subscription: Get updated subscription lines details for each line in a loop
9. Subscription: Update subscription lines (with asset key) for each line in a loop
10. Subscription: Activate subscription

Terminate Subscription

The integration deletes a subscription.

1. CDM: Get AddressId using AddressNumber for Account
2. CDM: If Account addresses are not found, get AddressId using AddressNumber for Contact
3. CDM: If Contact addresses are not found, create address
4. AR: Check if a site exists for the address
5. AR: If not, then create site
6. Subscription: Get subscription lines details for each line in a loop
7. Subscription: Update subscription lines (Close) for each line in a loop

Renew Subscription

The integration renews subscriptions at the end of period of their original subscription date.

Note that a shopper cannot renew a subscription unless a renewal draft subscription has been created in Oracle Subscription Management via the ESS job. The Subscription Management `GET` subscription product endpoint returns `renew` as a valid action if a draft subscription is present for the current subscription.

1. CDM: Get AddressId using AddressNumber for Account
2. CDM: If Account addresses are not found, get AddressId using AddressNumber for Contact
3. CDM: If Contact addresses are not found, create address
4. AR: Check if a site exists for the address
5. AR: If not, then create site
6. Subscription: Get draft renew subscription
7. Subscription: Get original subscription lines
8. Subscription: Create new subscription
9. Subscription: Add contact for account-based subscription
10. Subscription: Activate subscription
11. Subscription: Update original subscription lines to do-not-renew
12. Subscription: Delete draft subscription lines for each line in a loop

13 Integrate with Oracle Unity

You can integrate Oracle Commerce with Oracle Unity. Oracle Unity allows you to combine customer data from various sources to create a single view of each customer within your Commerce instance.

Once you have integrated Oracle Unity with Commerce, you can personalize and customize your shopper's experience on your web site by using Commerce Audience rules. You can propagate data from Unity's unified customer and account models to attributes of Commerce profile and account objects. Once you have the attributes, you can configure Audience rules to present your shopper with personalized content.

For detailed information on working with Oracle Unity, refer to https://docs.oracle.com/en/cloud/saas/cx-unity/cx-unity-user/index.htm. For detailed information on developer-based tasks, refer to https://docs.oracle.com/en/cloud/saas/cx-unity/cx-unity-develop/index.htm. For detailed information on working with Commerce Audiences, refer to the Define Audiences section.

Configure Oracle Unity for Oracle Commerce

Before you can integrate Commercewith Oracle Unity, you need to configure both Oracle Unity and Oracle Commerce.

To obtain detailed information on the tasks described here, refer to the Oracle Unity documentation at https://docs.oracle.com/en/cloud/saas/cx-unity/books.html.

Integration Overview

Unity imports data from multiple systems into raw import tables. This raw data is moved into source data tables within the data warehouse, meaning that records from multiple sources are combined into a single source data table. Then a pipeline imports data from multiple sources into a master data table according to the rules that you have configured. By default, Unity uses the email field as the cluster key, although you can define any field, or set of field, as the key. Each cluster is linked to a single master record in a master entity and fields included in that master record are merged from the clustered source records. You can use the Customer 360 API to directly query source and the master tables.

You can then use the Commerce Admin API to map Commerce profile properties to the attributes of the Unity entities such as `MasterCustomer` and `MasterAccount`.

Note that there are two ways that your Commerce profile and account data are shared with Unity. When you initially set up your configuration, you should manually export all of the profile and account data form Commerce and import it into Unity. Once the integration is enabled and configured, an automated process will periodically export all Commerce profiles and accounts that have been updated and upload the export file to OOS.

Configure Unity components

The following describes how to configure Unity components so that they work with Oracle Commerce. These instructions describe exporting profiles and accounts from your Commerce instance and uploading those files to the Oracle Object Store (OOS). They also describe configuring a Unity data source to connect to the OOS, and then configuring a Unity ingest job to read files from the OOS. You will also need to incorporate Commerce Customer and Account records into the Unity mastering process and then run an ingestion pipeline.

Configure synchronization

Before you configure Unity components, it is recommended that you export your Profiles and Accounts to CSV files and upload the files to the Oracle Object Store. This allows you to configure an Ingest Job using your data. To do this use the Commerce Admin API and the `exportProcess` endpoint.

For detailed information on exporting data endpoints, refer to Export data endpoints.

When dynamically synchronizing profile and account data from Commerce to Unity, the OOS works as a data transfer repository. The integrator configures an OOS bucket that is accessible to both Commerce and Unity. It is recommended that you use a dedicated OOS bucket to transfer data between Commerce and Unity. Only a single bucket is required, and you can use the same bucket between manual bulk uploads, or for automated incremental uploads that the integration feature performs.

You should also create or identify a user that has permissions to manage objects in the OOS.

Next, you must upload the exported profile and account CSV files to the Oracle Object Storage (OOS). It is recommended that you use the same file name prefixes as used in the automatic incremental profile and account export, which are `occs-profiles` and `occs-accounts`. This allows you to use the same Unity data source and Ingest Job definition for both batch and any incremental imports.

Note: Because Unity does not provide a mechanism for removing files after ingestion, it is recommended that you implement an appropriate Object Lifecycle Management policy to eventually archive and/or delete these files. You must grant service permissions to the Object Storage Service so that OOS may archive and delete objects. For information on Object Lifecycle Management, refer to the Using Object Lifecycle Management section in the Unity documentation.

Commerce uploads incremental export files to the OOS through a pre-authenticated request (PAR) URL, which is configured by the integrator. For information on creating this URL, refer to Using pre-authenticated requests. You must configure the PAR to be write-only, with access to the appropriate bucket. OOS requires that you set an expiration date on the PAR. Note that when the PAR expires, the synchronization of Profiles and Accounts from Commerce to Unity will not work until you configure a new PAR.

Configure the Unity Ingestion Pipeline

When you configure a Unity Ingestion Pipeline, you are configuring how to import and manage the data in Unity, as well as selecting the source of the data that Unity uses. You are also configuring the way that data is provided to Unity and master rules that create clustering, augmentation, and promotions rules. Optionally, you can configure a Data Warehouse mapping.

For detailed information on configuring the Unity Ingestion Pipeline, refer to the Unity documentation.

Create Master Entities

Master Entities are data objects within the data model that store unified records. Commerce uses the default Master settings, which include the `MasterCustomer` and `MasterAccount` entities. However, if you would like to create a customized Master Entity, refer to Creating Master Entities in the Unity documentation. Configuring a Master Entity matches data across multiple sources and ensures that there is no duplication. Using the `email` field as the cluster key, records match when the first and last name match. Each cluster is linked to a single master record in a Master Entity, for example `MasterCustomer`. Fields included in the master record are merged from the clustered source records. Note that not all source fields are mapped into the master record

After you have created a Master Entity, you must publish your configuration changes using the Unity administration interface. For detailed information on publishing refer to Publishing changes in the Unity documentation.

Once you have published your changes, check your results to see if your account and customer tables require changes to the ingest configuration. After Unity has ingested data from multiple systems, it moves this data into source data tables within the data warehouse using the Unity Data Warehouse Job. The Data Warehouse Job copies ingested data from staging to the data warehouse and then runs data validation. Once the data has been verified, the data is inserted into the data model and, at this point, you can run the Identity Resource Job

The Identity Resolution Job calculates data density values that belong to Master Entities such as `MasterCustomer`. Data density represents the percentage of records that have values for an attribute. The Identity Resolution Job calculates data density values the first time it runs and then processes the data for the Master Entity. Once the density has initially been calculated, the job will calculate and refresh data density values for nine more times. Once it reaches the tenth job, the data values are calculated and refreshed every seven days. For information on data density and the refer to the Data Density section of the Unity documentation.

Configure a Data Source View

Commerce queries account and profile data using the Customer 360 API. The Customer 360 API cannot access the tables in the data warehouse directly, but instead uses a Data Source Views (DSV). The DSV joins together one or more source or master tables, and then selects the subset of fields that can be searched and returned from queries. Queries are run against a single Data Source View, which are generated by the Customer 360 job, and are run one at a time. Rules and subrules define how the DSV relate to each other. For detailed information on working with Customer 360 jobs, refer to the Customer 360 Concepts section of the Unity documentation.

The definitions include the following:

- `IDGraphRules` - Contains a reference to a DSV object, and any defined subrules. Your searches specify which ID Graph Rule to search, as you can create multiple ID Graph Rules to return different data. For example, you might want to search different data records, such as customer or account data.
- `Subrules` - Each subrule attaches more data to the `IDGraph` table for each record, and references its own DSV object, whose `outputAttributes` object represents the additional data available on the selected record. For example, use a subrule if you want to collect additional attributes. You can create multiple subrules and attach them to one ID Graph Rule so that you can initiate searches that return specific attributes. Note that each

subrule must reference its own DSV, which allows you to search for fields related to the subrule and return related fields.

- Data Source View (DSV) - The DSV specifies the attributes that are populated in the `IDGraph` table, as well as the attributes you can use when performing a search query and what is returned in the search results. DSVs must be referenced by the ID Graph Rule and each subrule within the ID Graph Rule.

For additional information on creating ID Graph Rules, subrules and DSVs, refer to this tutorial in the Unity documentation.

When you create the main DSV:

- The main DSV is the DSV that is associated with the primary `IDGraphRule` object.
- There are two `outputAttributes` with the DSV:
  - Inside operands objects, also known as inner `outputAttributes` objects - These are the column names that you specify after the `SELECT` keywords in an SQL query. You can select as many attributes as needed from the table, which is specified in the `objectName` object.
  - Inside `MCPSQuery` objects, also known as outer `outputAttributes` objects - To use attributes with this object, the attribute must be referenced before the inner `outputAttributes` object. The outer `outputAttributes` object in the `IDGraphRule` object specifies the attributes that you can search across and the attributes that are returned in the search results.

The following example shows how you would extract five attributes from the Customer table. This includes `ID`, `SourceCustomerID`, `SourceID`, `Email` and `AccountID`. These attributes are pulled from the inner `outputAttributes` object and then exposes those in the outer `outputAttributes` object. This allows you to search them and return search results.

Issue a `POST` command to `{{Tenant URL}}/{{api-metadata_v1}}/{{MCPS Tenant Key}}/metadata/datasourceviews`.

This is the example for the Profile DSV:

```
{
    "tenantId": 100052,
    "name": "OCC_Profile_DSV",
    "versionTS": 1611872531438,
    "active": true,
    "dataSourceViewID": "OCC_Profile_DSV",
    "sourceTableQuery": {
        "MCPSQuery": {
            "tenantId": 0,
            "operation": {
                "ctype": ".SetOperation",
                "tenantId": 0,
                "operands": [{
                        "ctype": ".ObjectSet",
                        "tenantId": 0,
                        "name": "c1",
                        "objectName": "Customer",
                        "outputAttributes": [{
                                "atype": ".ReferenceAttribute",
                                "tableName": "c1",
                                "attributeName": "ID"
                            },
                            {
                                "atype": ".ReferenceAttribute",
                                "tableName": "c1",
                                "attributeName": "SourceCustomerID"
                            },
                            {
                                "atype": ".ReferenceAttribute",
                                "tableName": "c1",
                                "attributeName": "SourceID"
                            },
                            {
                                "atype": ".ReferenceAttribute",
                                "tableName": "c1",
                                "attributeName": "Email"
                            },
                            {
                                "atype": ".ReferenceAttribute",
                                "tableName": "c1",
                                "attributeName": "AccountID"
                            }
                        ],
                        "distinct": false
                    },
                    null
                ],
                "operator": "INTERSECTION",
                "distinct": false,
                "outputAttributes": [{
                        "atype": ".ReferenceAttribute",
                        "tableName": "c1",
                        "attributeName": "ID"
                    },
                    {
                        "atype": ".ReferenceAttribute",
                        "tableName": "c1",
                        "attributeName": "SourceCustomerID"
                    },
                    {
                        "atype": ".ReferenceAttribute",
                        "tableName": "c1",
                        "attributeName": "SourceID"
                    },
                    {
                        "atype": ".ReferenceAttribute",
                        "tableName": "c1",
                        "attributeName": "Email"
                    },
                    {
                        "atype": ".ReferenceAttribute",
                        "tableName": "c1",
                        "attributeName": "AccountID"
                    }
```

```
                ],
                "joinConditions": []
            },
            "type": "DW"
        }
    }
}
```

The following example is for the Account DSV:

```
{
        "tenantId": 100052,
        "name": "OCC_Account_DSV",
        "versionTS": 1628229524814,
        "active": true,
        "lastModifiedBy": "fc524ec322dd4bdca13c06e6261cc725",
        "createdBy": "fc524ec322dd4bdca13c06e6261cc725",
        "createdTS": 1628229524814,
        "dataSourceViewID": "OCC_Account_DSV",
        "sourceTableQuery": {
            "MCPSQuery": {
                "tenantId": 0,
                "operation": {
                    "ctype": ".SetOperation",
                    "tenantId": 0,
                    "operands": [
                        {
                            "ctype": ".ObjectSet",
                            "tenantId": 0,
                            "name": "a1",
                            "objectName": "Account",
                            "outputAttributes": [
                                {
                                    "atype": ".ReferenceAttribute",
                                    "tableName": "a1",
                                    "attributeName": "ID"
                                },
                                {
                                    "atype": ".ReferenceAttribute",
                                    "tableName": "a1",
                                    "attributeName": "SourceAccountID"
                                },
                                {
                                    "atype": ".ReferenceAttribute",
                                    "tableName": "a1",
                                    "attributeName": "SourceID"
                                }
                            ],
                            "distinct": false
                        },
                        null
                    ],
                    "operator": "INTERSECTION",
                    "distinct": false,
                    "outputAttributes": [
                        {
                            "atype": ".ReferenceAttribute",
                            "tableName": "a1",
                            "attributeName": "ID"
                        },
                        {
                            "atype": ".ReferenceAttribute",
                            "tableName": "a1",
                            "attributeName": "SourceAccountID"
                        },
                        {
                            "atype": ".ReferenceAttribute",
                            "tableName": "a1",
                            "attributeName": "SourceID"
                        }
                    ],
                    "joinConditions": []
                },
                "type": "DW"
            }
        }
    }
    }
}
```

Create the subrule DSV

The subrule DSV contains properties similar to the main DSV.

- The subrule DSV attaches additional data to the `IDGraphRule` object.
- There are two `outputAttributes` with the subrule DSV
  - Inside operands objects, also known as inner `outputAttributes` objects - These are the column names that you specify after the SELECT keywords in an SQL query. You can select as many attributes as needed from the table, which is specified in the `objectName` object.
  - Inside `MCPSQuery` objects, also known as outer `outputAttributes` objects - To use attributes with this object, the attribute must be referenced before the inner `outputAttributes` object. The outer `outputAttributes` object in the `IDGraphRule` object specifies the attributes that you can search across and the attributes that are returned in the search results.

The following example of a subrule DSV shows how collect the data from two tables, the `Customer` (c1) and the `MasterCustomer` (m1) and then join them on the attribute ID.

Issue a `POST` command to `{{Tenant URL}}/{{api-metadata_v1}}/{{MCPS Tenant Key}}/metadata/datasourceviews`. The `ID` of the DSV is `OCC_Profile_Subrule_DSV`.

The following is an example of the Profile subrule DSV:

```
{
   "tenantId": 100052,
   "name": "OCC_Profile_Subrule_DSV",
   "versionTS": 1620369103427,
   "active": true,
   "lastModifiedBy": "fc524ec322dd4bdca13c06e6261cc725",
   "createdBy": "fc524ec322dd4bdca13c06e6261cc725",
   "createdTS": 1620369103427,
   "dataSourceViewID": "OCC_Profile_Subrule_DSV",
   "sourceTableQuery": {
       "MCPSQuery": {
           "tenantId": 0,
           "operation": {
               "ctype": ".SetOperation",
               "tenantId": 0,
               "operands": [{
                       "ctype": ".ObjectSet",
                       "tenantId": 0,
                       "name": "c1",
                       "objectName": "Customer",
                       "outputAttributes": [{
                               "atype": ".ReferenceAttribute",
                               "tableName": "c1",
                               "attributeName": "ID"
                           },
                           {
                               "atype": ".ReferenceAttribute",
                               "tableName": "c1",
                               "attributeName": "SourceCustomerID"
                           },
                           {
                               "atype": ".ReferenceAttribute",
                               "tableName": "c1",
                               "attributeName": "SourceID"
                           },
                           {
                               "atype": ".ReferenceAttribute",
                               "tableName": "c1",
                               "attributeName": "AccountID"
                           }
                       ],
                       "distinct": false
                   },
                   {
                       "ctype": ".ObjectSet",
                       "tenantId": 0,
                       "name": "m1",
                       "objectName": "MasterCustomer",
                       "outputAttributes": [{
                               "atype": ".ReferenceAttribute",
                               "tableName": "m1",
                               "attributeName": "ID"
                           },
                           {
                               "atype": ".ReferenceAttribute",
                               "tableName": "m1",
                               "attributeName": "Email"
                           },
                           {
```

```
                "atype": ".ReferenceAttribute",
                "tableName": "m1",
                "attributeName": "FirstName"
            },
            {
                "atype": ".ReferenceAttribute",
                "tableName": "m1",
                "attributeName": "LastName"
            },
            {
                "atype": ".ReferenceAttribute",
                "tableName": "m1",
                "attributeName": "Age"
            },
            {
                "atype": ".ReferenceAttribute",
                "tableName": "m1",
                "attributeName": "Gender"
            },
            {
                "atype": ".ReferenceAttribute",
                "tableName": "m1",
                "attributeName": "AOV"
            },
            {
                "atype": ".ReferenceAttribute",
                "tableName": "m1",
                "attributeName": "City"
            },
            {
                "atype": ".ReferenceAttribute",
                "tableName": "m1",
                "attributeName": "Country"
            },
            {
                "atype": ".ReferenceAttribute",
                "tableName": "m1",
                "attributeName": "JobTitle"
            },
            {
                "atype": ".ReferenceAttribute",
                "tableName": "m1",
                "attributeName": "months_since_last_purchase"
            },
            {
                "atype": ".ReferenceAttribute",
                "tableName": "m1",
                "attributeName": "days_since_last_purchase"
            },
            {
                "atype": ".ReferenceAttribute",
                "tableName": "m1",
                "attributeName": "first_purchase_date"
            },
            {
```

```
            "atype": ".ReferenceAttribute",
            "tableName": "m1",
            "attributeName": "okToEmail"
        },
        {
            "atype": ".ReferenceAttribute",
            "tableName": "m1",
            "attributeName": "okToText"
        },
        {
            "atype": ".ReferenceAttribute",
            "tableName": "m1",
            "attributeName": "Phone"
        },
        {
            "atype": ".ReferenceAttribute",
            "tableName": "m1",
            "attributeName": "State"
        },
        {
            "atype": ".ReferenceAttribute",
            "tableName": "m1",
            "attributeName": "top_brand"
        },
        {
            "atype": ".ReferenceAttribute",
            "tableName": "m1",
            "attributeName": "top_product"
        },
        {
            "atype": ".ReferenceAttribute",
            "tableName": "m1",
            "attributeName": "top_product_category"
        },
        {
            "atype": ".ReferenceAttribute",
            "tableName": "m1",
            "attributeName": "total_order_count"
        },
        {
            "atype": ".ReferenceAttribute",
            "tableName": "m1",
            "attributeName": "total_purchase_count"
        },
        {
            "atype": ".ReferenceAttribute",
            "tableName": "m1",
            "attributeName": "total_return_count"
        },
        {
            "atype": ".ReferenceAttribute",
            "tableName": "m1",
            "attributeName": "total_spent_amt"
        },
        {
                "atype": ".ReferenceAttribute",
                "tableName": "m1",
                "attributeName": "ZipCode"
            }
        ],
        "distinct": false
    }
],
"operator": "INTERSECTION",
"distinct": false,
"outputAttributes": [{
        "atype": ".ReferenceAttribute",
        "tableName": "c1",
        "attributeName": "ID"
    },
    {
        "atype": ".ReferenceAttribute",
        "tableName": "c1",
        "attributeName": "SourceCustomerID"
    },
    {
        "atype": ".ReferenceAttribute",
        "tableName": "c1",
        "attributeName": "SourceID"
    },
    {
        "atype": ".ReferenceAttribute",
        "tableName": "c1",
        "attributeName": "AccountID"
    },
    {
        "atype": ".ReferenceAttribute",
        "tableName": "m1",
        "attributeName": "FirstName"
    },
    {
        "atype": ".ReferenceAttribute",
        "tableName": "m1",
        "attributeName": "LastName"
    },
    {
        "atype": ".ReferenceAttribute",
        "tableName": "m1",
        "attributeName": "Email"
    },
    {
        "atype": ".ReferenceAttribute",
        "tableName": "m1",
        "attributeName": "Age"
    },
    {
        "atype": ".ReferenceAttribute",
        "tableName": "m1",
        "attributeName": "Gender"
    },
    {
        "atype": ".ReferenceAttribute",
        "tableName": "m1",
        "attributeName": "AOV"
    },
    {
        "atype": ".ReferenceAttribute",
        "tableName": "m1",
        "attributeName": "City"
    },
    {
        "atype": ".ReferenceAttribute",
        "tableName": "m1",
        "attributeName": "Country"
    },
    {
        "atype": ".ReferenceAttribute",
        "tableName": "m1",
        "attributeName": "JobTitle"
    },
    {
        "atype": ".ReferenceAttribute",
        "tableName": "m1",
        "attributeName": "months_since_last_purchase"
    },
    {
        "atype": ".ReferenceAttribute",
        "tableName": "m1",
        "attributeName": "days_since_last_purchase"
    },
    {
        "atype": ".ReferenceAttribute",
        "tableName": "m1",
        "attributeName": "first_purchase_date"
    },
    {
        "atype": ".ReferenceAttribute",
        "tableName": "m1",
        "attributeName": "okToEmail"
    },
    {
        "atype": ".ReferenceAttribute",
        "tableName": "m1",
        "attributeName": "okToText"
    },
    {
        "atype": ".ReferenceAttribute",
        "tableName": "m1",
        "attributeName": "Phone"
    },
    {
        "atype": ".ReferenceAttribute",
        "tableName": "m1",
        "attributeName": "State"
    },
                {
                    "atype": ".ReferenceAttribute",
                    "tableName": "m1",
                    "attributeName": "top_brand"
                },
                {
                    "atype": ".ReferenceAttribute",
                    "tableName": "m1",
                    "attributeName": "top_product"
                },
                {
                    "atype": ".ReferenceAttribute",
                    "tableName": "m1",
                    "attributeName": "top_product_category"
                },
                {
                    "atype": ".ReferenceAttribute",
                    "tableName": "m1",
                    "attributeName": "total_order_count"
                },
                {
                    "atype": ".ReferenceAttribute",
                    "tableName": "m1",
                    "attributeName": "total_purchase_count"
                },
                {
                    "atype": ".ReferenceAttribute",
                    "tableName": "m1",
                    "attributeName": "total_return_count"
                },
                {
                    "atype": ".ReferenceAttribute",
                    "tableName": "m1",
                    "attributeName": "total_spent_amt"
                },
                {
                    "atype": ".ReferenceAttribute",
                    "tableName": "m1",
                    "attributeName": "ZipCode"
                }
            ],
            "joinConditions": [{
                "left": {
                    "atype": ".ReferenceAttribute",
                    "tableName": "c1",
                    "attributeName": "ID"
                },
                "right": {
                    "atype": ".ReferenceAttribute",
                    "tableName": "m1",
                    "attributeName": "ID"
                }
            }]
        },
        "type": "DW"
        }
    }
}
```

The following is an example of an Account subrule DSV. Note that the account subrule DSV `POST` request does not contain a reference to the Account object because you are not referencing any attributes from that object. If you need to reference an attribute from an object in Unity, you would need to specify the object from which you would be referencing. As in the case of the Profile subrule DSV, you referenced the attributes from the `Customer` object as c1 and the `MasterCustomer` objects as m1.

```
{
        "tenantId": 100052,
        "name": "OCC_Account_Subrule_DSV",
        "versionTS": 1628229554895,
        "active": true,
        "lastModifiedBy": "fc524ec322dd4bdca13c06e6261cc725",
        "createdBy": "fc524ec322dd4bdca13c06e6261cc725",
        "createdTS": 1628229554895,
        "dataSourceViewID": "OCC_Account_Subrule_DSV",
        "sourceTableQuery": {
            "MCPSQuery": {
                "tenantId": 0,
                "operation": {
                    "ctype": ".SetOperation",
                    "tenantId": 0,
                    "operands": [
                        {
                            "ctype": ".ObjectSet",
                            "tenantId": 0,
                            "name": "m1",
                            "objectName": "MasterAccount",
                            "outputAttributes": [
                                {
                                    "atype": ".ReferenceAttribute",
                                    "tableName": "m1",
                                    "attributeName": "ID"
                                },
                                {
                                    "atype": ".ReferenceAttribute",
                                    "tableName": "m1",
                                    "attributeName": "AddressLine1"
                                },
                                {
                                    "atype": ".ReferenceAttribute",
                                    "tableName": "m1",
                                    "attributeName": "AddressLine2"
                                },
                                {
                                    "atype": ".ReferenceAttribute",
                                    "tableName": "m1",
                                    "attributeName": "City"
                                },
                                {
```

```
                "atype": ".ReferenceAttribute",
                "tableName": "m1",
                "attributeName": "Country"
            },
            {
                "atype": ".ReferenceAttribute",
                "tableName": "m1",
                "attributeName": "Name"
            },
            {
                "atype": ".ReferenceAttribute",
                "tableName": "m1",
                "attributeName": "State"
            },
            {
                "atype": ".ReferenceAttribute",
                "tableName": "m1",
                "attributeName": "Type"
            },
            {
                "atype": ".ReferenceAttribute",
                "tableName": "m1",
                "attributeName": "ZipCode"
            }
        ],
        "distinct": false
    },
    null
],
"operator": "INTERSECTION",
"distinct": false,
"outputAttributes": [
    {
        "atype": ".ReferenceAttribute",
        "tableName": "m1",
        "attributeName": "ID"
    },
    {
        "atype": ".ReferenceAttribute",
        "tableName": "m1",
        "attributeName": "AddressLine1"
    },
    {
        "atype": ".ReferenceAttribute",
        "tableName": "m1",
        "attributeName": "AddressLine2"
    },
    {
        "atype": ".ReferenceAttribute",
        "tableName": "m1",
        "attributeName": "City"
    },
    {
        "atype": ".ReferenceAttribute",
        "tableName": "m1",
                        "attributeName": "Country"
                    },
                    {
                        "atype": ".ReferenceAttribute",
                        "tableName": "m1",
                        "attributeName": "Name"
                    },
                    {
                        "atype": ".ReferenceAttribute",
                        "tableName": "m1",
                        "attributeName": "State"
                    },
                    {
                        "atype": ".ReferenceAttribute",
                        "tableName": "m1",
                        "attributeName": "Type"
                    },
                    {
                        "atype": ".ReferenceAttribute",
                        "tableName": "m1",
                        "attributeName": "ZipCode"
                    }
                ],
                "joinConditions": []
            },
            "type": "DW"
        }
    }
}
```

Create the IDGraphRule object

The `IDGraphRule` object is a lean JSON object that contains references to the DSV objects you created earlier. The subrules object may contain none or many `IDGraphRules` objects that act as subrules for that specific `IDGraphRule`.

Issue a `POST` command with an empty body to `{{Tenant URL}}/{{api-metadata_v1}}/{{MCPS Tenant Key}}/metadata/idgraphrules`. The `ID` of the `IDGraphRuleObject` is `occ7`.

The following is an example of the Profile `IDGraphRule`:

```
{
    "tenantId": 100052,
    "name": "occ7",
    "versionTS": 1611942055656,
    "active": true,
    "ruleId": "occ7",
    "inputDsvId": "OCC_Profile_DSV",
    "mainIdentifier": {
        "atype": ".DSVAttribute",
        "dsvId": "OCC_Profile_DSV",
        "attributeName": "ID"
    },
    "dedupeTargetId": "customer_mastercustomer",
    "subRules": [{
        "tenantId": 100052,
        "name": "OCC Profile Subrule",
        "versionTS": 1611790281085,
        "active": true,
        "ruleId": "occ7",
        "subRuleId": "OCC_Profile_Subrule",
        "masterEntity": true,
        "inputDsvId": "OCC_Profile_Subrule_DSV",
        "identifier": {
            "atype": ".DSVAttribute",
            "dsvId": "OCC_Profile_Subrule_DSV",
            "attributeName": "ID"
        },
        "maxRecords": 100
    }]
}
```

The following is an example of the Account `IDGraphRule`:

```
{
        "tenantId": 100052,
        "name": "occ9",
        "versionTS": 1628299175767,
        "active": true,
        "lastModifiedBy": "fc524ec322dd4bdca13c06e6261cc725",
        "createdBy": "fc524ec322dd4bdca13c06e6261cc725",
        "createdTS": 1628299175767,
        "ruleId": "occ9",
        "inputDsvId": "OCC_Account_DSV",
        "mainIdentifier": {
            "atype": ".DSVAttribute",
            "dsvId": "OCC_Account_DSV",
            "attributeName": "ID"
        },
        "dedupeTargetId": "account_masteraccount",
        "subRules": [
            {
                "tenantId": 100052,
                "name": "OCC Account Subrule",
                "versionTS": 1628229619713,
                "active": true,
                "createdTS": 1628229619713,
                "ruleId": "occ9",
                "subRuleId": "OCC_Account_Subrule",
                "masterEntity": true,
                "inputDsvId": "OCC_Account_Subrule_DSV",
                "identifier": {
                    "atype": ".DSVAttribute",
                    "dsvId": "OCC_Account_Subrule_DSV",
                    "attributeName": "ID"
                },
                "maxRecords": 100
            }
        ]
    }
```

Run a Publish Job

Note that all metadata, including the Ingest Job, the Master Entities, the DSV and the 360 rule definitions, must go through the publishing process and changes must be promoted before taking effect.

Once you have created all of the objects, you publish them by running a Publish Job. To do this issue a `POST` command with an empty body to `{{Tenant URL}}/api-admin/v1/{{MCPS Tenant Key}}/admin/tenant/publish`. This initiates a Publish Job that you can track using the **Jobs** page. This job usually takes up to ten minutes to complete.

For information on Publishing Jobs, refer to Data Publishing section in the Unity documentation.

Run a Customer 360 Job

Once the Publish Job has completed, you must run a Customer 360 Job. This job ensures that you get the search results that you expected. To do this, issue `POST` command with an empty body to `{{Tenant URL}}/api-admin/v1/{{MCPS Tenant Key}}/admin/job/IDGRAPH/IdGraph/start`. Once the Customer 360 Job has completed, you can search customer data.

For information on working with Customer 360 Jobs, refer to the Customer 360 Jobs Unity documentation.

Configure Commerce for Oracle Unity

Once you have set up your Oracle Unity configuration, you need to configure Oracle Commerce to work with Unity.

To complete the configuration, you must configure the integration settings in the Oracle Integrations page. You must also configure the property mapping for profile and account enrichment. To use Commerce with Unity:

- Enable Unity Integration from the Oracle Integrations Page.
- Define profile and account enrichment.
- Optionally define new custom properties on Account and Profile types to be populated with Unity data.
- Define property mappings using the `/enrichmentMaps` endpoint.
- Define audiences using enriched profile properties.

Enable Unity Integration from the Oracle Integrations Page

Perform the following steps to set up a Unity integration in Commerce

1. Use the administration interface to select **Settings** and then **Oracle Integrations**.
2. Use the drop down menu to select **Unity Configuration**.
3. Click the checkbox to enable the Unity integration.

Define profile and account enrichment

Once you have enabled the Unity integration, you are presented with the **Unity Configuration** page, which requires you to:

- Enter the Server URL
- Enter the Channel ID
- Enter the Channel Token
- Add Unity Users.

You must set the **Unity Profile Attributes** by selecting the attributes of the data objects available for the Profile, such as Account, Address and Customer. The following attributes are for Account:

- Account Name - The name of the account.
- DUNS# - The Dunn & Bradstreet 9 digit business identifier.
- Employee Total - The number of employees within the company.
- NAICS Code - The North American Industry Classification System 6 digit code for the business type.
- SIC Code - The Standard Industrial Classification 4 digit code for the company industry.
- Year Established - The year the company was established.
- Account Type - The type of the account.
- Account Status - The status of the account.
- Annual Revenue - The annual revenue of the company.
- Line of Business - The line of business.

The following attributes are for Address:

- Account ID
- Address Line 1
- Address Line 2
- City
- State
- AV is Valid - A flag that indicates if the address is valid or invalid
- Commerce Address Type - The type of address, for example, Shipping or Billing.
- Latitude
- Longitude

Set the Unity Behavioral Attributes, which include:

- Customer ID
- Severity - The severity level of the incident, including Critical, Major and Minor
- Sentiment
- Coupon Code

Set the Unity Transactional Attributes, which include:

- Customer ID
- Order Date

- Quantity
- Discount
- Subtotal
- Tax Total

Set the Unity Product Attributes:

- Product ID
- Product Name

Set any other attributes necessary for your environment.

Once you have selected the attributes, save the page. Commerce creates the profile extension table with the selected attributes.

Obtain the Unity client credentials

Use the **Identity Cloud Service** administration console for the Unity instance. Then navigate to your Oracle Unity Data Platform application instance by accessing **Oracle Cloud Services** and selecting `CXUNITYSI-commerceoudp`). You can obtain a client ID/ Secret using the **Configuration** tab. Test your configuration using the **Provisioning** tab. Refer to How to Access Oracle Identity Cloud Service for detailed information on working with Identity Cloud Services.

Before you can configure an integration, you need to collect the Oracle Unity API configuration settings. You will need the following information:

Property	Description
`Token URL`	This is the URL to the Identity Cloud Service Oauth2 Token API. Use the following format: `https://idcs-`*`xxxxxxx`*`.identity.oraclecloud.com/oauth2/v1/token`
`Username`	The name of the Unity API user with access to the Unity API. You must have the Instance Admin role access.
`Password`	The Unity API user's password.
`Client ID`	The client ID of the Unity application.
`Client Secret`	The client secret for the Unity application.
`Tenant URL`	The URL of your Unity instance.
`IDCS App Scope`	The IDCS scope for the Unity application in the form `https://`*`unityTenantUrl`*`/cxunity`.
`Tenant Key`	The MCPS tenant access key. If you do not know this key, contact your Unity administrator.

You must also configure the Oracle Object Store (OOS), which contains the Object Store Pre-Authenticated Request (PAR) URL, which is the URL used by Commerce to upload incremental export files. For information on creating this URL, refer to Using pre-authenticated requests.

Map your properties

To synchronize Profile and Account properties from Unity into Commerce, you need to define the mappings that specify which Unity Account or Customer attributes map to which Commerce Organization or Profile properties. To do this, you must use the Commerce API.

Unity attributes must be mapped onto existing Profile and Organization properties. You may want to create custom properties to receive these mappings. Before defining the property mappings, the Organization and Profile types should be extended using the `itemTypes` and `shopperTypes` endpoints.

The following is an example of an `itemTypes` endpoint:

```
PUT  /ccadmin/1/itemTypes/organization

{
  "specifications": [
    {
     "label": "Account Notes",
     "id": "internal_notes",
     "default": null,
     "required": false,
     "localizable": false,
     "internalOnly": false,
     "textSearchable": false,
     "searchable": false,
     "multiSelect": false,
     "type": "richText",
     "uiEditorType": "richText"
    }
  ]
}
```

The following is an example of a `shopperTypes` endpoint:

```
PUT   /ccadmin/v1/shopperTypes/users

{
    "properties": {
        "social_media_rating": {
            "label": "Social Media Rating",
            "type": "shortText",
            "uiEditorType": "shortText",
            "internalOnly": false,
            "default": null,
            "required": false,
            "audienceVisibility": "all"
        },
        "top_brand": {
            "label": "Favorite Brand",
            "type": "shortText",
            "uiEditorType": "shortText",
            "internalOnly": false,
            "default": null,
            "required": false,
            "audienceVisibility": "all"
        }
    }
}
```

Define properties using the `/enrichmentMaps` endpoint

Once you have extended the properties, you must define property mappings using the `/enrichmentMaps` endpoint. For example:

```
PUT  /ccadmin/v1/enrichmentMaps

{
    "IDGraphRuleName": "occ1",
    "subRuleName": "OCC_Profile_Subrule",
    "configMap": [
        {
            "sourceName": "Social_Score",
            "sourceType": "string",
            "targetName": "social_media_rating"
        },
        {
            "sourceName": "top_brand",
            "sourceType": "string",
            "targetName": "top_brand"
        }
    ],
    "source": "customer",
    "target": "profile"
}
```

Configure Commerce Audiences with Unity Data

You can include shopper data in Commerce that you obtain from your Oracle Unity instance, allowing you to personalize your shopper's visit using Audiences.

Oracle Unity consolidates data from various systems. Unity's data model contains both extended profile and account attributes, as well as intelligent and behavioral attributes.

You can define Commerce Audiences to provide a personalized shopper experience based on these enriched profile and account attributes. For detailed information on working with Commerce Audiences, refer to the Define Audiences section.

Define your Audience Rules using the **Audience Rule Definition** page in the Commerce administration interface.

The Audience rule builder will show all available profile or account properties, including those that are mapped from Unity. Because you can add new attributes over time, it is important that you refresh these attributes regularly.

Glossary

Index