Oracle® Cloud

Integrating Oracle CX Commerce and Oracle CPQ

Release 1

F37077-01

Oracle Cloud Integrating Oracle CX Commerce and Oracle CPQ, Release 1

F37077-01

Contents

Preface

This preface contains the following sections:

Documentation accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

1 Introduction

The Oracle CX Commerce/Oracle CPQ integration lets you configure complex products for purchase in Commerce by using the Oracle CPQ configurator.

Self-service users in Oracle CX Commerce (formerly Oracle Commerce Cloud) can configure complex products for purchase in Commerce using the Oracle CPQ configurator.

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

Note: The integration of Commerce with Oracle CPQ 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 CPQ, OIC, and Oracle CX Commerce to prepare for a Commerce and Oracle CPQ integration.

Audience

This implementation guide is for administrators who are setting up and configuring the integration. The guide assumes administrators have prior Commerce, Oracle CPQ, 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 CPQ 19C or later Base Ref App site set up as described in this implementation guide. The integration between Commerce and Oracle CPQ 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 CPQ catalog.
- Oracle Integration Cloud Service (OIC) 18.3.5 or later.
- The latest Oracle CX Commerce and Oracle CPQ Reference Migration Package and integration files can be downloaded from Integrating Oracle CX Commerce and Oracle CPQ (Doc ID 2214316) on My Oracle Support.

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

2

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 CPQ through a common configuration.

You must import an OIC Integration Package to an OIC environment that connects Commerce and Oracle CPQ through a common configuration.

The OIC Integration Package creates web service connections that allow users to adjust order and quote details in Oracle CPQ, 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.

Topics:

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 CPQ article on My Oracle Support.
2. If you want to implement the integration between Commerce and the Oracle CPQ 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 CPQ 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 CPQ, 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 CPQ.

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 CPQ 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 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 CPQ.
- The *OCCS-CPQ Sync Quote* integration allows Oracle CPQ 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 CPQ.
- The *OCCS-CPQ Update Quote* integration sends information to Oracle CPQ 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 CPQ.

Note: Importing and setting up the OIC Integration Package is a prerequisite to completing the Sync Quote action in Oracle CPQ. 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 CPQ connections

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

Administrators must configure connections from the integrations referenced in the previous section to Oracle CPQ.

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

To configure the Oracle CPQ 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 CPQ getConfigurations API. **Note:** The Oracle CPQ Asset Actions, Get Assets, and GetConfigBom connections are REST based and use the REST Catalog URL. The Oracle CPQ getConfigurations and Oracle CPQ 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 CPQ 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 CPQ connection uses the Basic security policy, so you must enter the login details for your Oracle CPQ account.
8. Click **OK**.
9. Click **Test** to test the connection.

10. Click **Save**.
    The Oracle CPQ connection is now configured for the integration. Repeat steps 1-10 for each of the remaining Oracle CPQ 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 CPQ, Commerce, Oracle CPQ Quote, Oracle CPQ Configure, and Oracle CPQ 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 CPQ

The Sync Quote Action needs to be created for the Oracle CPQ/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 CPQ site

You must complete some preliminary OIC integration setup steps on the Oracle CPQ site for the integration to run successfully.

You must set up the OIC integration on the Oracle CPQ 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
`_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: 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 CPQ. 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 CPQ 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 CX 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 CPQ Features with Oracle CX Commerce* book.

Note: You can also customize configurations of complex assets in Commerce without being redirected to an Oracle CPQ 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 CPQ Configuration API section of the Using Oracle CPQ Features with Oracle CX Commerce book.

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 CX 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 CX Commerce*
- Work with address types in *Extending Oracle CX Commerce*
- Account Details in *Using Oracle CX Commerce*
- Work with account addresses in *Using Oracle CX Commerce*
- Work with account registration requests in *Using Oracle CX 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 the Understand the general procedure for installing and configuring the integration SSEs section of the *Using Oracle CPQ Features with Oracle CX Commerce* guide in the Commerce Help Library.

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 the Understand the general procedure for installing and configuring the integration SSEs section of the *Using Oracle CPQ Features with Oracle CX Commerce* guide in the Commerce Help Library.

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 the Use Asset Based Ordering section of the *Using Oracle CPQ Features with Oracle CX Commerce* book.

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 the integration SSEs section of the *Using Oracle CPQ Features with Oracle CX Commerce* guide in the Commerce Help Library.

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 the Understand the general procedure for installing and configuring the integration SSEs section of the *Using Oracle CPQ Features with Oracle CX Commerce* guide in the Commerce Help Library.

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 of the *Using Oracle CPQ Features with Oracle CX Commerce* guide in the Commerce Help Library.

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 CPQ 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 the Understand the general procedure for installing and configuring the integration SSEs section of the *Using Oracle CPQ Features with Oracle CX Commerce* guide in the Commerce Help Library.

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 CPQ 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 CPQ 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 CPQ 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 CPQ /rest/v9/config{prodFamVarName}.{prodLineVarName}.{modelVarName}/actions/_configure. This endpoint supports a direct Oracle CPQ 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 CPQ 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 CPQ Configuration API section of the *Using Oracle CPQ Features with Oracle CX Commerce* book.

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 CX Commerce* book.

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

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 the Understand the general procedure for installing and configuring the integration SSEs section of the *Using Oracle CPQ Features with Oracle CX Commerce* guide in the Commerce Help Library.

3 Set Up Oracle CPQ

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

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

Topics:

Understand general set up for Oracle CPQ

Some general set up procedures for Oracle CPQ need to be completed for the integration to run successfully.

You must complete the following Oracle CPQ general set up procedures:

- Enable Guest Access to Oracle CPQ
- Add Template Dependencies to File Manger
- Make Oracle CPQ Stylesheet Edits
- Synchronize Oracle CPQ Parts with Commerce SKUs

Enable Guest Access to Oracle CPQ

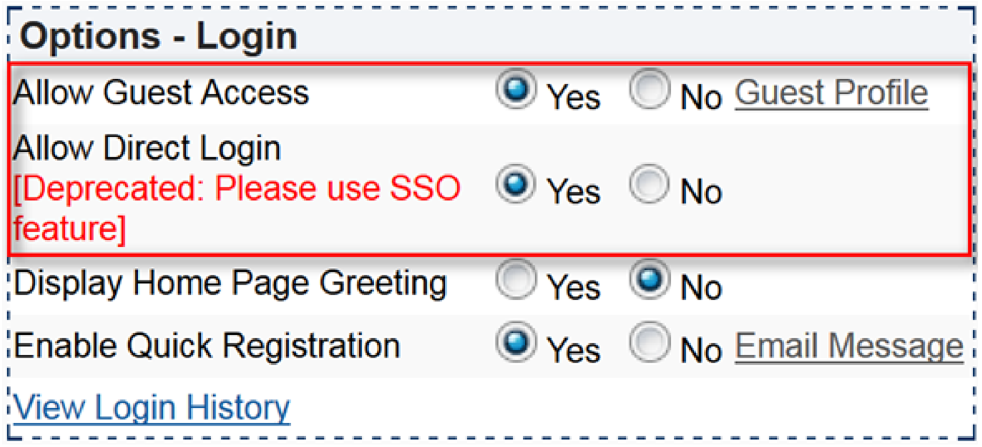
Administrators can allow multiple self-service users in Commerce to access an Oracle CPQ site as a guest user from an iFrame displaying within Commerce. When Commerce punches in to Oracle CPQ for configuring items, the system uses sessions for unregistered users (i.e. guest users). When self-service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

Note: You can now customize the configurations of complex products in Commerce without being redirected to an Oracle CPQ 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 CPQ hosted iFrame. Refer to the Using Oracle CPQ Features with Oracle CX Commerce guide in the Oracle CX 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 CPQ interface displays in German when the user accesses Oracle CPQ. The user's currency and locale preferences are also passed from Commerce and display in Oracle CPQ.

To enable guest access to Oracle CPQ:

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 CPQ.
4. If multi-currency support from Commerce is required, set Allow Direct Login [Deprecated: Please use SSO feature] to Yes.

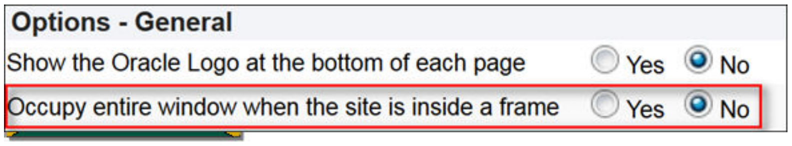


Options - Login

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 CPQ 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 CPQ 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 CPQ 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 CPQ 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}}]
}
```

Make Oracle CPQ Stylesheet Edits

Oracle recommends administrators hide Oracle CPQ navigation options outside the scope of the integration from Commerce self-service users.

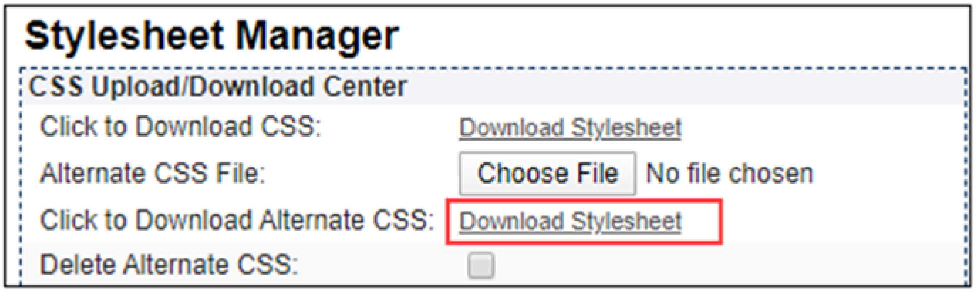
Hide the Oracle CPQ Home Button

By hiding the Oracle CPQ Home button, the Oracle CPQ configurator opens whenever users access Oracle CPQ. 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 CPQ 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.



Stylesheet Manager

CSS Upload/Download Center

Click to Download CSS:	Download Stylesheet
Alternate CSS File:	Choose File No file chosen
Click to Download Alternate CSS:	Download Stylesheet
Delete Alternate CSS:	☐

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;
}
```

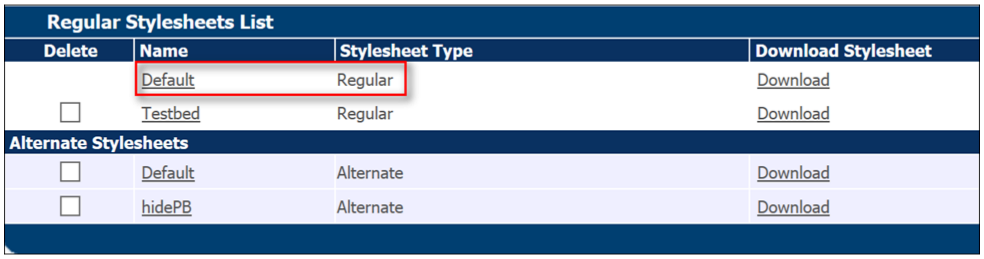
Note: If the **Home** button shows both a label and an icon, administrators cannot hide the label using only CSS. From the Admin Home page, navigate to **Style and Templates** > **Navigation Menus** > **Subheader** > **Home** > **Edit**. Choose **Icon** for **Display**. The **Home** button is then hidden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

Hide Price Books

Oracle CPQ 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 CPQ Parts with Commerce SKUs

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

To synchronize Oracle CPQ 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 CPQ 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 CX Commerce and Oracle CPQ integration, repeat the above procedure to add the corresponding part in Oracle CPQ.
- In addition toOracle CPQ 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 CX Commerce set up

You must complete preliminary Commerce set up steps in Oracle CPQ for the integration to run successfully.

This topic contains the Commerce set up steps that you must complete in Oracle CPQ.

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_ExternalOrderPriceQuantity_t	Integer	none
CC Expiration Date	cC_ExpirationDate_t	Date	none
CC Agent Id	cC_AgentId_t	Text Field	none
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.

Attribute Label	Variable Name	Attribute Type	Additional Settings
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

Attribute Label	Variable Name	Attribute Type	Additional Settings
Price (List)*	listPrice_l	Currency	**Default Value:** Use the formula provided in the *Apply Formulas* section.
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 3-1 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 3-2 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: 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
_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 CPQ 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 CPQ 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 CPQ configuration set up

Specific set up procedures must be completed for the Commerce/Oracle CPQ Configuration integration to run successfully.

This topic contains the configuration set up procedures that you must complete in Oracle CPQ.

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 CPQ and Commerce.

Note: Ensure the appropriate Commerce Product Families and Product Lines are created in Oracle CPQ prior to starting the Client-Side Integration. Refer to the Configuration > Product Families articles within the Oracle CPQ 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 CPQ. 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 CPQ. These include domains that are allowed to load the Oracle CPQ in an iFrame and domains that Oracle CPQ 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 CPQ Models Corresponding to Products in Commerce

You must create Oracle CPQ 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 CPQ 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 CPQ 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 CPQ Administration Help.
4. Create a hiding rule configured as follows:

Condition	Action Attribute

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

4

Set Up Subscription Ordering in Oracle CPQ

The subscription ordering feature requires some set up when integrating Oracle CPQ and Commerce.

The following features require specific attention when integrating Oracle CPQ and Commerce and running the subscription ordering feature.

Topics:

For information about setting up Subscription or asset based orders within Oracle CPQ, refer to the ABO implementation guide and the Oracle CPQ 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 CPQ integration.

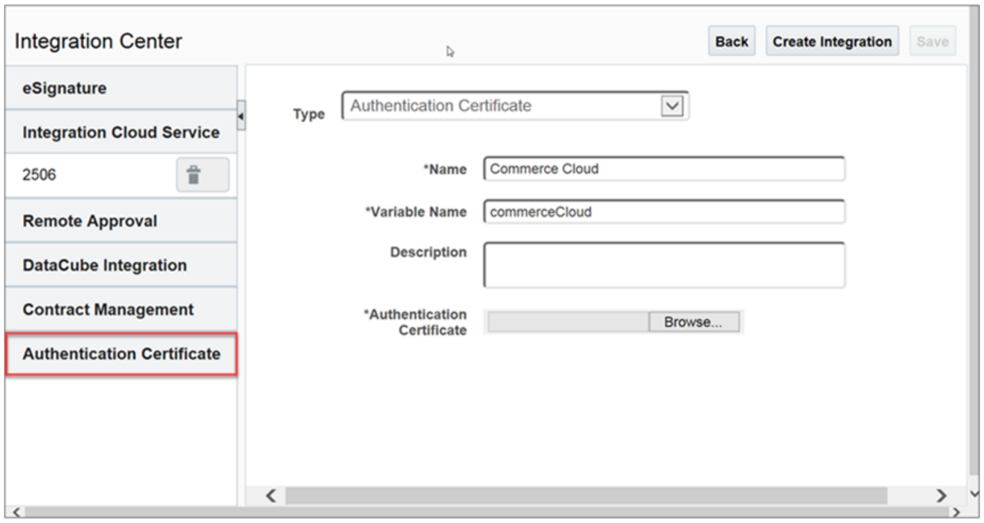
Oracle CPQ provides an Authentication Certificate integration type in the Integration Center to support access token-based authentication. This integration type allows Oracle CX Commerce self-service users to securely access Oracle CPQ to modify or reconfigure a Subscription Ordering asset-based Configuration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

Note: Refer to the *Custom Asset Attributes* article within the Oracle CPQ 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 CPQ 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 CPQ configuration steps to support asset based ordering.

Oracle CPQ 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 CPQ which the upgrade offer applies to.
- **currentOffer** – Maps to a configurable attribute on the root config model in Oracle CPQ. 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 CX Commerce to Oracle CPQ, 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 CX 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 CPQ 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 the Use Asset Based Ordering section of the *Using Oracle CPQ with Oracle CX Commerce* book (20A or greater version).

You can also customize configurations of complex assets in Commerce without being redirected to an Oracle CPQ 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 CPQ Configuration API section of the *Using Oracle CPQ Features with Oracle CX Commerce* book.

5

Enable Integrations in Commerce

Some configuration procedures need to be completed in order to enable the features of this integration.

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

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

Topics:

Enable Oracle CPQ configuration integration

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

To enable the Oracle CPQ 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 CPQ 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 CPQ 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 CPQ 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 CPQ catalog.
8. Enter the **Product Line** variable name. This should match the Product Line variable name of a configurable product in the Oracle CPQ catalog.
9. Enter the **Product Family** variable name. This should match the Product Family variable name of a configurable product in the Oracle CPQ 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 CPQ quoting integration

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

To enable the Oracle CPQ 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 CPQ 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 CPQ 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 CX 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.

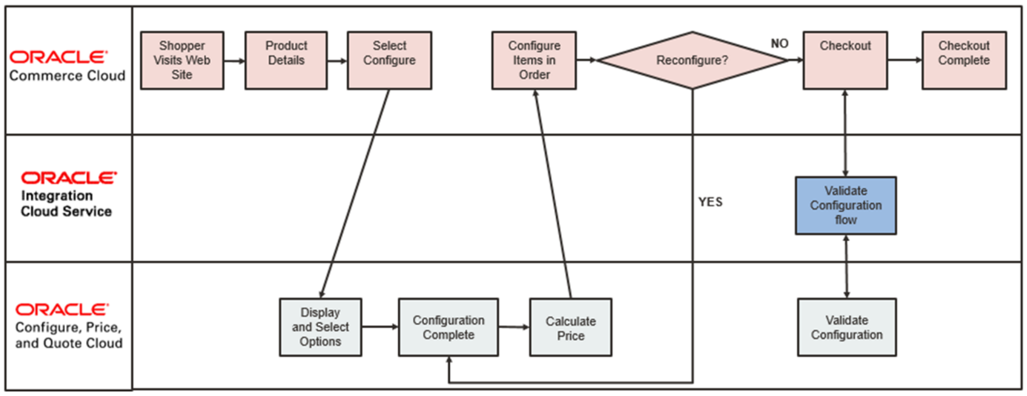
Only the configured product, which has external recurring charge details is considered as a subscription line items in OIC layer and the rest of the items in the order are filtered out.

A

Appendix A: Understand the Configurator Flow

A Configurator process flow occurs between Oracle CPQ and Commerce during the integration.

The following presents a diagram of the integration Configurator Flow:

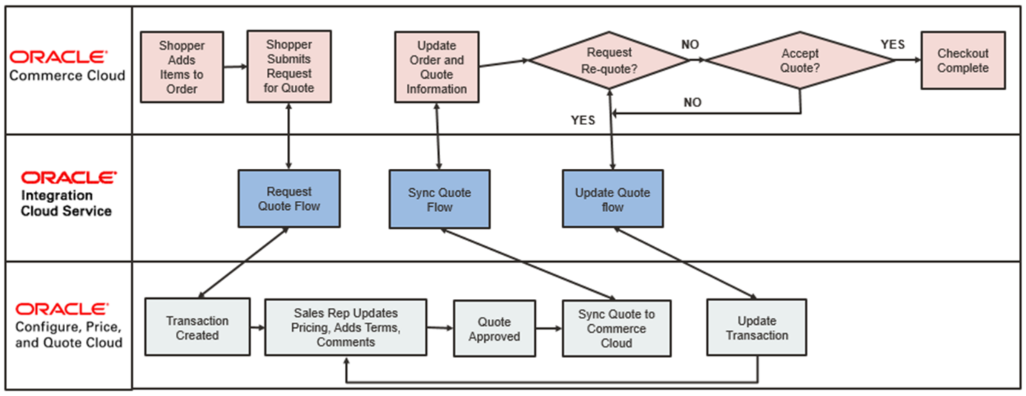


B

Appendix B: Understand the Request for Quote Flow

A Request for Quote process flow occurs between Oracle CPQ and Commerce during the integration.

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


ORACLE Commerce 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
ORACLE Integration Cloud Service
Request Quote Flow
Sync Quote Flow
Update Quote flow
ORACLE Configure, Price, and Quote Cloud
Transaction Created
Sales Rep Updates Pricing, Adds Terms, Comments
Quote Approved
Sync Quote to Commerce Cloud
Update Transaction

C

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 CPQ.

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

After all Oracle CPQ 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
	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

Integration Flow	Target Variable Name	Mapping	Comments
	_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
	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

Integration Flow	Target Variable Name	Mapping	Comments
	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>`
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

Integration Flow	Target Variable Name	Mapping	Comments
	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
	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

Integration Flow	Target Variable Name	Mapping	Comments
	externalData	attributes	None
	billingAccountId	billingAccount	None
	externalRecurringChargeFrequency	field4	Corresponds to part custom field 4 in Oracle CPQ
	childItems	for-each(children), for-each(partNumber)	None
	catalogRefId	partNumber	None
	configuratorId	lineId	None
	externalRecurringDuration	field6	Corresponds to part custom field 6 in Oracle CPQ
	externalPrice	_price_unit_price_each	None
	assetId	id	None
	actionCode	oRCL_ABO_ActionCode_l	None
	serviceId	id	None
	activationDate	startDate	None

D

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.oracle.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.oracle.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;
```

E

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;
```

F

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;
```