Siebel Oracle Commerce Implementation Guide

Checkout Integration

Implementation Guide for the Integration of Siebel with Oracle Commerce Checkout

Table of Contents

Introduction

This Implementation Guide (IG) is provided as documentation for the Checkout Integration Reference Implementation (RI). The RI enables Oracle Commerce to capture web orders in a format that provides all of the information required to convert from the Oracle Commerce Order format to the Siebel Quote format.

This version of the document provides the necessary Oracle Commerce functionality required to take the fully configured order and place it on the Siebel system.

This version focuses on shipping and billing details, along with submission of the order to Siebel.

The functionality in this version depends on the previous integrations for Account and Product Catalog and Product Configurator.

Siebel has the concept of both Quotes and Orders. In this version of the integration Oracle Commerce submits the shopping carts in Siebel Quote format. Those Quotes are converted into Siebel Orders before they are sent to the downstream systems.

There are no changes required to Siebel for this integration. The integration uses existing Siebel web services.

Note: for the remainder of this document the terms Siebel Quote and Siebel Order may be used interchangeably but from a technical perspective we are referring to Siebel Quotes.

References

[1] Siebel Oracle Commerce Implementation Guide – Product Configurator Integration v2.0.

[2] Siebel Oracle Commerce Implementation Guide – Order Integration v1.0

[3] Oracle Commerce Programming Guide Version 11.2

[4] Oracle Commerce Platform Programming Guide Version 11.2

[5] Oracle Commerce Repository Guide Version 11.2

Software Dependencies

This guide is based on the following software and respective versions:

- Oracle Siebel Version 15.6.0.0
- Oracle Commerce Version 11.2

Technical Requirements Summary

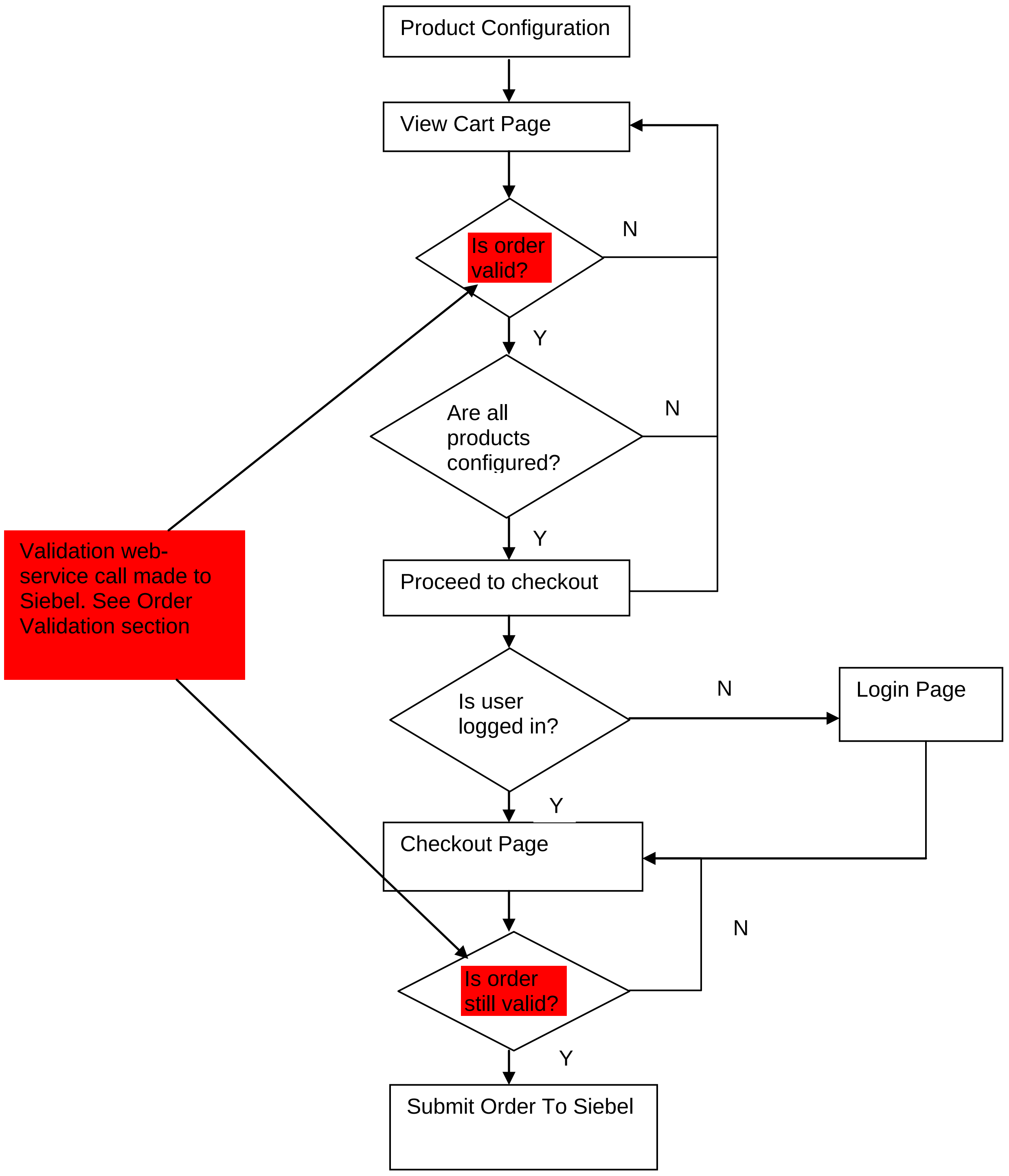
This chapter gives a brief overview of the high level requirements for submission of an order to Siebel with Oracle Commerce.

Checkout Requirements

- The user must be logged in. Anonymous users will not be supported.
- The logged in account must have billing and delivery information
- Oracle Commerce must submit orders completed by the configuration process to Siebel in Order Quote format
- All products must have been configured using the Product Configurator

Pre Checkout

The diagram below describes the process flow of pre-checkout and post-configuration validation, to order submission:



Order Validation

Order Validation is provided by the **validateOrderWithSiebel** method of **SiebelOrderTools**. This method accepts an Oracle Commerce order, and converts it to a Siebel Quote structure using the **createExecuteQuotingInput** method. It then sets the Pricing Flag, the Check Eligibility Flag and the Verify Promotion Flag.

```
ExecuteQuotingInput createExecuteQuotingInput(Order pOrder)
```

This method uses the convertOrderToSiebelQuote (See the SiebelOrderTools class description in the Order Integration Guide) to create Quote from the Oracle Commerce Order. It then un-sets all the flags on the input:

- pricingFlag
- checkEligibilityFlag
- verifyPromotionFlag
- syncQuoteFlag
- queryQuoteFlag
- deltaSpcActionSpcCodeSpcFlag
- calculateShippingCostFlag
- repricingFlag
- calculateTaxFlag

Once the input is created, this is passed to the callQuotingWebSerice method:

```
List<String> callQuotingWebService(ExecuteQuotingInput pInput)
```

This method calls the web-service, and returns any errors to the UI.

Logged In Profile Verification

Before proceeding to the Checkout Screen, the user must be logged in. The logic to check this is provided by the **getOrderProfileIsLoggedInUser** method of **SiebelProfileTools**. This method gets the Profile associated with the Order, and passes it to the **isLoginUser** method, which checks the security status for the profile.

Product Configuration Validation

Before proceeding to the Checkout Screen, all products must be configured. As we have now left the configuration process, there is no longer guaranteed access to the Product Configurator model, so we need to know if all the products have been configured by reference to the Order.

To this end, the SiebelCommerceItem (a wrapper for CommerceItem. See Order Integration Integration Guide) has a property **configured** which is *false* as default. When *endConfig* is called on a Root Product, this property is updated to *true*.

This property can then be checked by page logic to decide whether to allow the user to progress to the Checkout process.

For a detailed description of the ProductConfiguratorInstance model, please refer to the Product Configurator Integration Guide.

View Cart JSP

This page represents the end of the Product Configuration process, and the user will be directed here after configuration has finished.

From this page, the user can see their Shopping Cart, the hierarchy of Commerce Items and associated attributes, and has the options to remove products, or reconfigure them (see Product Configurator Integration Guide).

This page also provides the validation which will allow/disallow the user to progress to the Checkout by using the logic described in the above sections.

To call into the validation logic, the page makes use of the CheckoutValidationDroplet.

CheckoutValidationDroplet

Component Configuration:

```
$class=atg.siebel.order.droplet.CheckoutValidationDroplet
$scope=request
orderTools=/atg/commerce/order/OrderTools
shoppingCart=/atg/commerce/ShoppingCart
```

Class Description:

This droplet's **service** method calls into the Order Validation and Logged In Validation logic described above, then renders output params accordingly.

Example Usage:

```
<c:set var="validOrder">true</c:set>
  <dsp:droplet name="/atg/siebel/order/CheckoutValidationDroplet">
    <dsp:oparam name="orderNotValid">
      <c:set var="validOrder">false</c:set>
      <dsp:getvalueof var="errors" param="errors" />
    </dsp:oparam>
  </dsp:droplet>
```

Checkout

When the user has come this far, all Item Pricing has already been calculated, and Order has a total for the items (based on Recurring and Non Recurring costs. See Order Integration Guide), and tax. We now need to reconcile the Shipping Methods with Siebel, and provide calculators for each type of shipping method.

Shipping Methods

In this Reference Integration, we have configured 3 Shipping pre-calculators which correspond to 3 shipping methods which Siebel allows. These configurations are just for demonstration purposes, and are based on existing Oracle Commerce shipping calculators. It is expected that customers will implement their own Shipping calculators.

Siebel Shipping Pricing Engine

The SiebelShippingPricingEngine extends the ShipingPricingEngineImpl in order to add functionality to map the Shipping Method to a String which Shipping Carrier that Siebel will recognise, as Siebel will expect the name of a Shipping Carrier in its Quote.

This done via configuring a Map of Shipping Methods to Shipping Carriers:

shippingMethodShippingCarrierMap=Ground=UPS,Next Day=DHL,2 Day Service=US Mail

When SiebelOrderTools is converting an Order to a Quote for submission to Siebel, it calls into SiebelShippingPricingEngine to get the name of the carrier for the quote based on the order's hardgood shipping group's shipping method.

Checkout JSP

This user is redirected to the Checkout after order and logged in verification has been performed on the View Cart page. Here, they may change shipping methods, choose Billing and Shipping profiles to use for the order, navigate to pages to manage these (see Account Integration Guide), and place the order, representing the submission to Siebel.

This page must also perform order and logged in validation, to ensure a user cannot directly access the page without having gone through the configuration and logging in process. The page uses the CheckoutValidationDroplet to ensure this:

Sample usage:

```
<dsp:droplet name="/atg/siebel/order/CheckoutValidationDroplet">
    <dsp:oparam name="notLoggedIn">
      <dsp:droplet name="/atg/dynamo/droplet/Redirect">
        <dsp:param name="url"
value="../index.jsp?successUrl=/siebel/checkout/checkout.jsp"/>
      </dsp:droplet>
    </dsp:oparam>
    <dsp:oparam name="orderNotValid">
      <dsp:droplet name="/atg/dynamo/droplet/Redirect">
        <dsp:param name="url" value="../configurator/view_cart.jsp"/>
      </dsp:droplet>
    </dsp:oparam>
  </dsp:droplet>
```

This has the effect of performing order validation every time the page renders. Therefore, when the user chooses a new Shipping Method, he will be redirected back to the Checkout page, it will render and trigger an order validation call.

Once the user has applied a shipping method, the ‘Place Order’ button will become available.

SiebelExpressCheckoutFormHandler

This form handler kicks off the Oracle Commerce order submission pipeline and performs any functionality on the order needed to facilitate that, including copying the Shipping and Billing information to the order’s hardgood shipping and payment groups.

It also calls into SiebelProfileTools before each user gesture, which facilitates making sure the user is logged in before they are able to perform each of these actions before providing redirections to pages which manage the functionality (eg manage billing and shipping addresses).

The SiebelExpressCheckoutFormHandler then invokes the order submission process, and redirects the user to a confirmation page.

Order Placement

The order has now been placed on Oracle Commerce, and an order exists in the OrderRepository awaiting submission to Siebel.

Order Submission

To facilitate Order Submission to Siebel, the Reference Integration makes use of Patchbay, and Oracle Commerce implementation of a SQLJMS provider (see http://docs.oracle.com/cd/E26180_01/Platform.94/ATGProgGuide/html/s1109declaringjmsproviders01.html).

This ensures that any order submission messages are saved to a database, so that none are lost in the event of a catastrophic server failure.

The following message-filter is added to the configuration file:

/atg/dynamo/messaging/dynamoMessagingSystem.xml

```
<message-filter>
      <nucleus-name>
        /atg/siebel/order/submit/SiebelOrderSourceSink
      </nucleus-name>

      <input-port>
        <port-name>
          DEFAULT
        </port-name>

        <input-destination>
          <destination-name>
            patchbay:/Fulfillment/SubmitOrder
          </destination-name>
          <destination-type>
            Topic
          </destination-type>
        </input-destination>

      </input-port>

      <output-port>
        <port-name>
          DEFAULT
        </port-name>
      </output-port>

      <output-port>
```

```
        <port-name>
          OrderSubmitPort
        </port-name>
        <output-destination>
          <destination-name>
            patchbay:/Fulfillment/SubmitOrder
          </destination-name>
          <destination-type>
            Topic
          </destination-type>
        </output-destination>
      </output-port>

    </message-filter>
```

This configures Patch Bay to listen for order submission events. When a message is received, control is passed to the SiebelOrderSourceSink **receiveMessage** method.

SiebelOrderSourceSink

This component is added to the list of Initial components, to be started with the server.

Component Configuration:

```
$class=atg.siebel.order.submit.SiebelOrderSourceSink
$scope=global
orderManager=/atg/commerce/order/OrderManager
transactionManager=/atg/dynamo/transaction/TransactionManager
orderStatePropertyName=state
webServiceHelper=/atg/siebel/configurator/WebServiceHelper
```

Class Description:

The main function of this class is performed in the **receiveMessage** method:

1. The method receives a message containing an Order number

2. A transaction is started

3. The Order is loaded using the OrderManager, the OrderState is set to SIEBEL_PROCESSING, and the Order is saved. This is performed as a single atomic action inside a transaction to protect against any system failures. This way, we can assume any orders with the state SIEBEL_PROCESSING have not been submitted to Siebel.

4. The transaction is ended.

5. A new thread is started and control is passed to the SiebelOrderSubmitter object’s **run** method, thereby each Order is submitted to Siebel in it’s own thread.

doStart:

This method is called on startup of the component. It gets a list of all orders with SIEBEL_PROCESSING state, and submits them to Siebel, using the SiebelOrderSubmitter, under the assumption that a catastrophic server failure occurred and any Orders with that state have yet to be submitted.

SiebelOrderSubmitter

This class implements the Runnable interface, and performs the Order submission process via webservice, so that each Order is submitted in it's own thread.

When submitting each order, a transaction is started, the method calls into the SiebelOrderTools **syncOrderToSiebel** method (which does the work of submitting the order), the Order state is set to submitted, the order is updated and the transaction is ended, thereby ensure the action of submitting and updating the order is atomic.

The **syncOrderToSiebel** method uses the generic QuotingWebService (using the utility methods on SiebelOrderTools to create the input's payload and send the request), and sets the following flags:

- PricingFlag
- CheckEligibilityFlag
- SyncQuoteFlag
- VerfityPromotionFlag

This has the effect of persisting the order on Siebel. At this stage, the order has now been submitted to Siebel.

For a detailed description of the SiebelOrderTools class, please refer to the Order Integration Guide.

Fulfillment:

Once an order is placed, the order will go into the Fulfillment/SubmitOrder Topic.

To fulfil this the responsible module is fulfilment.

This module runs on only one server.

This module has the following responsibilities:

- Read the messages from Topic (dynamoMessagingServices.xml).
- Makes WebService call "ExecuteQuoting" that places a record in Siebel.
- Updates the repository.(table name - dcspp_order)

Running Fulfillment modules:

In single server environment: Siebel.Fulfillment module must be started directly which in turn starts Siebel.

In multi-server environment: Siebel.Fulfillment module must be started with one server and the siebel module will be started with other server.

Only one instance of the Siebel.Fulfillment is running at any one time.

This is because the SubmitOrder destination is a Topic, and can therefore be read by more than 1 server. Running more than instance of Siebel.Fulfillment will cause order contention issues.