WebLogic Commerce and Personalization Servers

Order Package Extensions

Technical Article

WebLogic Commerce Server and
WebLogic Personalization Server

Document Edition: 3.1

October 2000

Document	**Edition Date**	**Software Version**
Order Package Extension	October 2000	WebLogic Commerce Server 3.1

Order Package Extension

This document explains how to extend the Order Package data model. The following topics are discussed:

- Extending the Data Model
- Persistence Architecture
- Adding Runtime Attributes to Customer Data
- Adding Runtime Attributes to Other Entities
- Extending the Schema
  - Overview of Approach to Extending the WebLogic Commerce Server (WLCS) Schema
  - Adding Attributes Against WLCS_CUSTOMER, WLCS_ORDER, WLCS_TRANSACTION and WLCS_SHIPPING_METHOD Tables
  - Adding Attributes Against the WLCS_ORDER_LINE Table
  - Adding Attributes Against the WLCS_CREDIT_CARD and WLCS_SHIPPING_ADDRESS Tables
- Transaction Management
- Summary

Extending the Data Model

The User Registration and OrderProcessing packages are two of the core components of the WebLogic Commerce Server 3.1. These packages implement use-cases that deal with customer self-registration, customer management, shopping cart experience, and order processing (including shipping, payment and taxation).

These packages implement some of the most commonly required online commerce scenarios. However, this does not preclude any extensions that are specific to your commerce site. You can extend functionality of the WebLogic Commerce Server to provide more sophisticated and specialized commerce scenarios to meet your business needs. The WebLogic Commerce Server infrastructure supports use-case driven extensibility in the form of the webflow and pipelines. This infrastructure provides you with three forms of extensibility:

- You can rapidly modify the existing use-case flows, by changing the webflow and pipeline configurations.
- You can customize use-cases by adding new input processors and pipelines.

You can implement new use-cases by defining new webflows and pipelines that include custom input processors and pipelines.

For more information on the webflow and pipeline infrastructure, please refer to the topic “Webflow and PipeLine Management” in the BEA product documentation Web site at http://edocs.bea.com/wlcs310/wflopipe/index.htm.

One of the common requirements for implementing such extensions is the ability to access and extend the WebLogic Commerce Server data model and schema. For example, you may want to customize the checkout process of your commerce site to collect a promotion code or gift coupon data, and process the order and payment data accordingly. Similarly, you may like to capture additional shipping instructions from your customers. In this case, apart from extending the checkout webflow/pipeline, you’ll be required to capture, store, and process additional data.

This document presents some possible approaches and guidelines for extending the data model for the User Registration and Order Processing packages of the WebLogic Commerce Server. While this document does not guarantee automatic compatibility of such extensions with future releases of the WebLogic Commerce Server, the approaches discussed in this document try to minimize potential problems, by leveraging the webflow/pipeline infrastructure.

This document addresses the following:

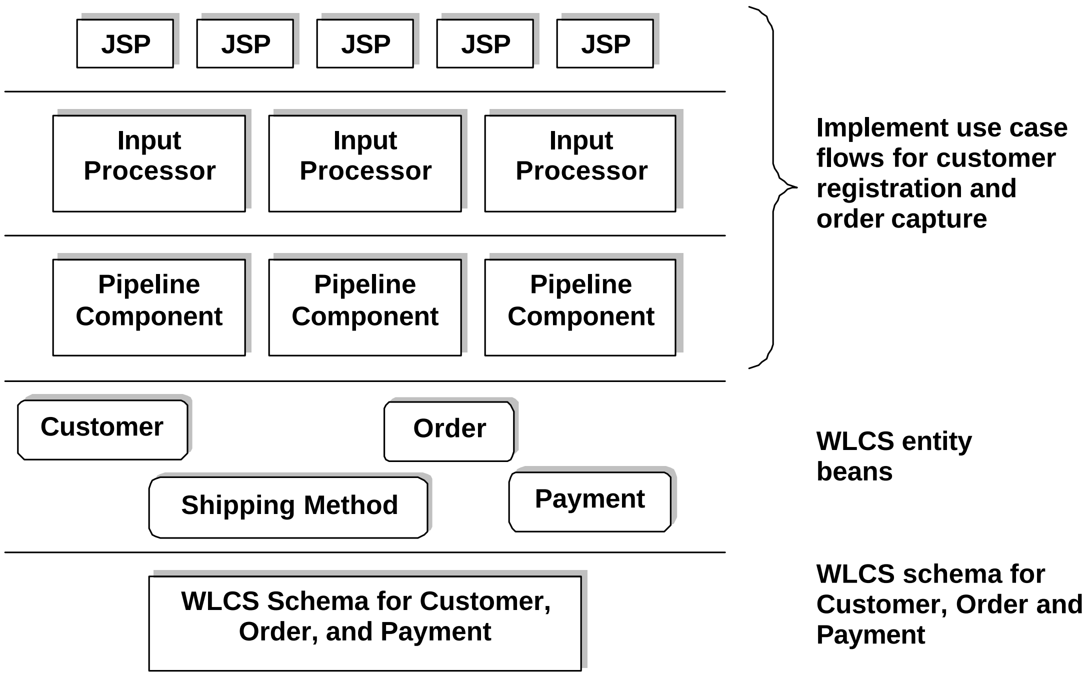
- The WebLogic Commerce Server persistence architecture.
- Adding runtime attributes to customer and order related entities.
- General approach for extending the data model and the schema.
- Extending WLCS_CUSTOMER, WLCS_ORDER, WLCS_TRANSACTION and WLCS_SHIPPING_METHOD tables.
- Extending WLCS_ORDER_LINE table (the case of one-to-many associations).
- How to persist and query additional attributes on entities such as customer, order, and payment transaction?
- How to demarcate transactions with such extensions.

Note: This document does not cover extensions to other WebLogic Commerce Server packages (such as extensions for catalog). You can periodically check the BEA product documentation Web site at http://edocs.bea.com for future updates on how to extend other packages.

Persistence Architecture

Before we go into the approaches for extending the WebLogic Commerce Server schema, let's consider the persistence architecture of WLCS, as shown in Figure 1. This figure shows the persistence architecture for the Customer Registration and Order Processing packages. In this structure, the JSP, Input Processor, and Pipeline Component layers are responsible for implementing the use-case flow. For more information on the JSPs, input processors, and pipeline components in these layers, refer to the topic "Order Processing Package" in the BEA product documentation Web site at http://edocs.bea.com/wlcs310/order/index.htm.

Figure 1: Persistence Architecture for Customer Registration and Order Processing Packages



The pipeline components rely on the following WebLogic Commerce Server entity beans for persisting customer, order, payment, and shipping method data respectively:

```
com.beasys.commerce.ebusiness.customer.Customer
com.beasys.commerce.ebusiness.order.Order
com.beasys.commerce.ebusiness.payment.PaymentTransaction
com.beasys.commerce.ebusiness.shipping.ShippingMethod
```

These entities use the WebLogic Commerce Server tables discussed in the Order Processing Package documentation for persistence.

The following table describes the mapping between these entities and the corresponding WebLogic Commerce Server tables.

Entity: `com.beasys.commerce.ebusiness.customer.Customer`	
WLCS_CUSTOMER	Customer description
WLCS_CREDIT_CARD	Credit cards
WLCS_SHIPPING_ADDRESS	Shipping address
Entity: `com.beasys.commerce.ebusiness.order.Order`	
WLCS_ORDER	Order description
WLCS_ORDER_LINES	Order lines
Entity: `com.beasys.commerce.ebusiness.payment.PaymentTransaction`	

WLCS_TRANSACTION	CyberCash transaction description
WLCS_TRANSACTION_ENTRY	CyberCash transaction entries
Entity: `com.beasys.commerce.ebusiness.shipping.ShippingMethod`	
WLCS_SHIPPING_METHOD	Shipping method description

The pipeline components in the Customer Registration and Order Processing packages manipulate the above tables via the respective entities. The default deployment configuration of these beans is such that all business methods are always executed within a transaction. This is established by setting the `<trans-attribute>` to `Required` in the deployment descriptor. In the default configuration, the pipelines that access these beans are transactional (with the `isTransactional` property set to true in pipeline.properties). Therefore all database access occurs under transactions initiated by the pipeline infrastructure, and the methods on these entities merely participate in those transactions.

Adding Runtime Attributes to Customer Data

The simplest possible extension is to add run-time attributes to the entities in the Customer Registration and the Order Processing packages. In the WebLogic Commerce Server, run-time attributes can be added on these entities without having to change the underlying database schema.

Although all the above entities in the WebLogic Commerce Server share the same basic structure, there are some differences in the way you can add run-time attributes to the customer entity, and the other entities.

The Customer entity of the WebLogic Commerce Server is a component that relies on the Unified User Profile (UUP) technology of the WebLogic Commerce Server. A UUP for customer data allows the abstraction of a customer to be seamlessly integrated into the WebLogic Personalization Server. Apart from personalization, this approach allows you to use the user management tools of the WebLogic Personalization Server to administer customer data, and maps the customer identity into a WebLogic Personalization Server-administered groups and the RDMBS security realm. For more information on unified user profiles, see the document “Creating and Managing Users” available online at http://edocs.bea.com/wlcs310/p13n/users.htm.

In addition to the above information, the notion of unified user profile can be used to add run-time attributes to customer data without having to modify the underlying schema.

You can find examples of adding attributes for customer data in the pipeline components under the `com.beasys.commerce.ebusiness.customer.pipeline` package. To add attributes to the customer data, the WebLogic Commerce Server customer registration package provides an abstract pipeline component `com.beasys.commerce.ebusiness.customer.pipeline.UpdateUserPC` with the following method:

```
public void setCustomerProperty(String key, Object value,
                                Customer customer)
                            throws java.rmi.RemoteException
```

This method takes a property name (“key”), the value of the property (“value”), and a reference to the customer entity (“customer”). For instance, you may use the following pipeline component to add a new attribute called “preference” for a given customer:

```
public class MyPC extends UpdateUserPC {
   public void updateCustomer(PipelineSession pSession,
                              Customer customer,
                                    CustomerValue customerValue)
                                    throws PipelineFatalException
    {
       try {
         setCustomerProperty("preference", “Loves music”,
                                 customer);
       }
    }
}
```

Given a customer, you can use the following snippet in your JSPs to read such runtime attributes:

```
<um:getProfile profileKey="<%=request.getRemoteUser()%>"
profileType="WLCS_Customer" />
<!-- Get the “preference” -->
<um:getPropertyAsString propertyName="preference" />
```

In the above example, the `request.getRemoteUser()` method returns the login name of the customer accessing the page. The `profileType` is a UUP name, and WebLogic Commerce Server specifies the customer entity as a UUP of type “`WLCS_Customer`.” The `<um:getPropertyAsString>` tag is one of the user management tags to extract user attributes in JSP pages. For more documentation on user management tags, please refer to http://edocs.bea.com/wlcs310/p13ndev/jsptags.htm.

Before you attempt to consider adding run-time attributes to the customer data, please bear in mind that this approach is meant only for quickly adding attributes without changing the schema. The WebLogic Commerce Server persists run-time attributes in tables that are internal to WebLogic Commerce Server. Consequently, you cannot execute SQL level operations on such data.

Adding Runtime Attributes to Other Entities

For the entities the order-processing package (viz., `com.beasys.commerce.ebusiness.shipping.Order`, `com.beasys.commerce.ebusiness.shipping.PaymentTransaction`, and `com.beasys.commerce.ebusiness.shipping.ShippingMethod`), there exists a similar mechanism for adding runtime attributes. All the entities in the order-processing package extend the `com.beasys.commerce.foundation.ConfigurableEntity` interface, which provides the following methods for adding and manipulating run-time attributes.

```
  public void setProperty(String key, Object value)
                        throws java.rmi.RemoteException
```

Using this method you can set a new property on an entity. You can use the following method to access attribute later:

```
  public Object getProperty(String key)
                        throws java.rmi.RemoteException
```

This method returns a previously added property.

For more information, including the API, refer to http://edocs.bea.com/wlcs310/javadoc/wlcs/index.html.

Extending the Schema

The following are some of the common drivers for extending the WebLogic Commerce Server schema:

- Extending the schema of the WebLogic Commerce Server to meet your existing schema.
- Enhancing the WebLogic Commerce Server to modify or add new functionality.
- Both these drivers manifest in the following:
- Modifying (or sometimes adding) the templates to render and/or collection additional data from the user interface.
- Modifying the webflow to change the flow of user interaction.
- Extending the WebLogic Commerce Server schema.

Note: Almost all the data in the Order-Processing package is meaningful across your business, for you may want to apply SQL level semantics for creating, updating and querying. Depending on the nature and scale of your commerce site, the WebLogic Commerce Server and your backend applications may depend on this data. Any extension to the schema of the order-processing package cannot be represented with runtime attributes, as runtime attributes cannot be accessed directly via standard SQL.

Here is an example scenario. Consider a new attribute called "tracking number" on your Order. Typically this is an attribute generated after order fulfillment by your backend order fulfillment application. You may want to display this tracking number on WebLogic Commerce Server order history pages for customers to view the tracking information. This is a domain specific attribute that best be persisted in the WLCS_ORDER table (or another table that you created for this purpose).

In this section, let's the consider the following cases, and discuss approaches that meet the above needs:

1. Adding attributes against WLCS_CUSTOMER, WLCS_ORDER, WLCS_TRANSACTION and WLCS_SHIPPING_METHOD
2. Adding attributes against WLCS_ORDER_LINE WLCS_SHIPPING_ADDRESS, and WLCS_CREDIT_CARD tables

Note: These two cases are discussed separately because the tables in case 2 participate in a one-to-many association with WLCS_ORDER and WLCS_CUSTOMER tables in case 1.

Overview of Approach to Extending the WebLogic Commerce Server Schema

The following figure presents an overview of the approach for "extending the WebLogic Commerce Server schema" and NOT for "integrating the WebLogic Commerce Server schema with your existing schema" or for "mapping the WebLogic Commerce Server schema onto your existing schema."

Figure 2: Extending the Data Model

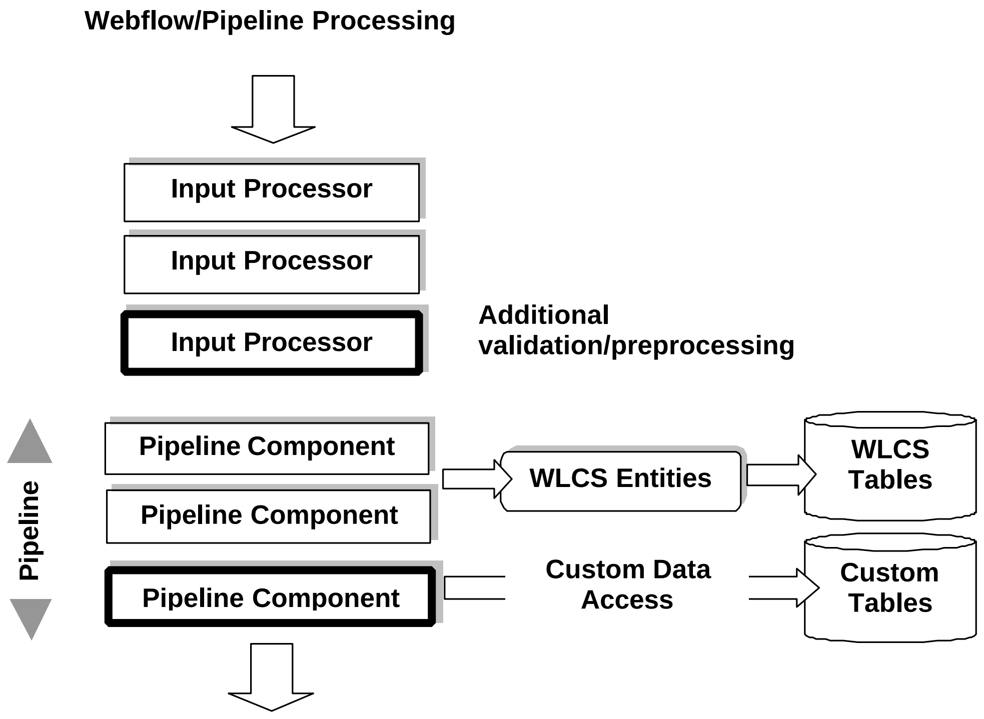


Figure 2 demonstrates how a given webflow/pipeline processing can be modified to process additional data, without modifying existing input processors and pipeline components. In Figure 2, the blocks with thick borders are new input processors and pipeline components inserted to process the additional data. While the WebLogic Commerce Server pipeline components manage the WebLogic Commerce Server data via the WebLogic Commerce Server entities, the new pipeline component in the pipeline may directly access the data via plain JDBC, or indirectly via another layer of custom entity beans. Alternatively, the new pipeline component may also delegate this data access to legacy data access mechanisms.

As we shall discuss in a later section, depending on whether the additional data should be processed with in the same transaction, or in a new transaction, or no transaction at all, you can split the above pipeline into more than one pipeline each of which will have its own transaction setting.

Adding Attributes Against WLCS_CUSTOMER, WLCS_ORDER, WLCS_TRANSACTION and WLCS_SHIPPING_METHOD Tables

Let's now consider the case of the customer, order, payment, and shipping method tables. The general approach is as follows:

Step 1: Design new tables

For each of the above tables, design new table(s) for the additional attributes with the same primary key. For instance, for extending order data, consider a new table with ORDER_ID as the primary key. Although it is tempting to extend the WebLogic Commerce Server tables for such attributes, we recommend against doing so, as it could lead to compatibility issues and potential name collision issues with future releases of WebLogic Commerce Server.

Step 2: Modify corresponding JSP templates

If the new data is user-entered, modify the corresponding JSP templates to add new fields in the forms.

Step 3: Implement new input processor

Implement a new input processor to read validate/preprocess the new data. Since input processors can be chained against a webflow event, adding a new input processor gives you more flexibility when compared to modifying an existing input processor for the same input processor chain. After validating the data, add the collected data to the pipeline session for further processing in the pipeline. Depending on whether such data is required beyond the scope of the current HTTP request or not, use the appropriate scope (session scope or request scope) while adding data to the pipeline session.

Step 4: Include the new input processor.

Modify the webflow.properties to include the new input processor.

Step 5: Implement a new pipeline component.

Implement a new pipeline component to extract the additional data from the pipeline session, and write to the new tables. Obtain the primary key from the respective entity. For example, for storing additional attributes for the order entity, call the `getIdentifier()` method on the order entity. This method returns the primary key for the WLCS_ORDER table for the current order.

Step 6: Obtain a database connection.

To obtain a database connection, use the `getConnection()` method in the abstract base class `com.beasys.commerce.foundation.pipeline.CommercePipelineComponent`. You may recall that all pipeline components extend this abstract class. This method returns a connection from the `commercePool` setup in the `weblogic.properties` file. However, if you want to use a different connection pool, modify the property `commerce.jdbc.pool.url` property in the `weblogiccommerce.properties` file to point to a different data source wrapping the new connection pool.

Step 7: Include the new pipeline component.

Modify the pipeline.properties to include the new pipeline component.

To query for such additional data, you may follow a similar procedure.

Adding Attributes Against the WLCS_ORDER_LINE Table

In the WebLogic Commerce Server, an Order entity aggregates a collection of `OrderLine` objects, with each `OrderLine` object representing an order line in the database in the WLCS_ORDER_LINE table, with ORDER_LINE_ID as the primary key.

These collections are internally based on the Java collections API, with primary keys generated while storing the order entity.

The following procedure applies in case you want to extend the WLCS_ORDER_LINE table.

Step 1: Design a new table.

Design a new table for the additional attributes with the same primary key. For extending the ORDER_LINE table, consider a new table with ORDER_LINE_ID as the primary key.

Step 2: Modify the corresponding JSP template.

If the new data is user-entered, modify the corresponding JSP templates to add new fields in the forms.

Step 3: Implement a new input processor.

Implement a new input processor to read validate/preprocess the new data. The procedure is similar to that of step 3 of the previous section.

Step 4: Include the new input processor.

Modify the webflow.properties to include the new input processor.

Step 5: Implement a new pipeline component.

Implement a new pipeline component to extract the additional data from the pipeline session, and write to the new tables. However, since the primary key for the WLCS_ORDER_LINE table is internal to the WebLogic Commerce Server, consider the following code snippet in your new pipeline component for obtaining the ORDER_LINE_ID for a given order line:

```
String orderId = null;
order.getIdentifier();
String sku =  ...; // Get the sku from the corresponding line
                   // in the shopping cart.
try {
   Connection c = getConnection();
   String statement = "SELECT ORDER_LINE_ID FROM \
       WLCS_ORDER_LINE WHERE ORDER_ID = ? AND PRODUCT_ID = ?";
   PreparedStatement preparedStatement = null;
   preparedStatement = c.prepareStatement(statement);
   preparedStatement.setObject(1, ordereId);
   preparedStatement.setObject(2, sku);
   ResultSet rs = preparedStatement.executeQuery();
   // The result set should now have a row containing
   // the ORDER_LINE_ID. Add your custom JDBC here to
   // persist the additional data for the order line.
```

Step 6: Update the deployment descriptor.

Before you deploy the new pipeline component, there is one additional step to be perfomed – that is to update the deployment descriptor of the order entity as follows:

- Unjar the `lib\ebusiness.jar` into a temporary directory
- Open the `weblogic-ejb-jar.xml` file. You can find it under the META-INF subdirectory from where you unjared.
- In this file, search for the following entry, and add the text marked in bold.

```
<weblogic-enterprise-bean>
  <ejb-name>
    com.beasys.commerce.ebusiness.order.Order
  </ejb-name>
  <persistence-descriptor>
    <is-modified-method-name>
      isModified
     </is-modified-method-name>
     <delay-updates-until-end-of-tx>
```

```
            false
          </delay-updates-until-end-of-tx>
        </persistence-descriptor>
              <reference-descriptor>
                 ...
        </reference-descriptor>
              <enable-call-by-reference>true</enable-call-by-
reference>
        <jndi-name>
           com.beasys.commerce.ebusiness.order.Order
        </jndi-name>
      </weblogic-enterprise-bean>
```

- Jar the contents of the temporary directory, and run the ejbc to create a new ebusiness.jar
- Replace the `lib\ebusiness.jar` with the newly created `ebusiness.jar`.

Step 6 ensures that the order and order-line data is available for executing queries in the new pipeline component.

Step 7: Include the new pipeline component.

Modify the pipeline.properties to include the new pipeline component.

Adding Attributes Against the WLCS_CREDIT_CARD and WLCS_SHIPPING_ADDRESS Tables

The following procedure applies in case you want to extend the WLCS_ORDER_LINE table.

Step 1: Design new tables.

For each of the above tables, design new table(s) for the additional attributes with the same primary key. For extending the WLCS_CREDIT_CARD table, consider a new table with CREDIT_CARD_ID as the primary key. Similarly for the WLCS_SHIPPING_ADDRESS table, consider a new table with SHIPPING_ADDRESS_ID as the primary key.

Step 2: Modify corresponding JSP templates.

If the new data is user-entered, modify the corresponding JSP templates to add new fields in the forms.

Step 3: Add "mapKey" attribute to the Pipeline Session.

Modify the com.beasys.commerce.ebusiness.customer.webflow.UpdatePaymentInfoIP to add the "`mapKey`" attribute to the `PipelineSession`. Similarly, in the case of shipping address, add the "`ShippingAddressMapKey`" attribute to the `PipelineSession` in the `com.beasys.commerce.ebusiness.customer.webflow.UpdateShippingInfoIP`

Step 4: Implement new input processor.

Implement a new input processor to read validate/preprocess the new data. Reconfigure webflow.properties to include the new input processor.

Step 5: Implement new pipeline component.

Implement a new pipeline component to extract the additional data from the `PipelineSession`, and write to the new tables. However, since the primary keys for the WLCS_CREDIT_CARD and

WLCS_SHIPPING_ADDRESS tables are internal to the WebLogic Commerce Server, consider the following code snippet in your new pipeline component for obtaining the primary keys. Although the following snippet describes the steps for credit card data, the same procedure applies to shipping address data.

```
   // Get the customer ID
   String customerId = null;
   customer.getIdentifier();

   // Get the map key for the credit card from the
   // pipeline session. Refer to Step 3.
   String mapKey = pipelineSession.getAttribute(“mapKey”);
   try {
      Connection c = getConnection();
      String statement = "SELECT CREDIT_CARD_ID FROM \
         WLCS_CREDIT_CARD WHERE CUSTOMER_ID = ? AND MAP_KEY =
?";
      PreparedStatement preparedStatement = null;
      preparedStatement = c.prepareStatement(statement);
      preparedStatement.setObject(1, customerId);
      preparedStatement.setObject(2, mapKey);

      ResultSet rs = preparedStatement.executeQuery();
      // The result set should now have a row containing
      // the CREDIT_CARD_CARD_ID.
      // Add your custom JDBC for your tables here.
```

Step 6: Modify deployment descriptor.

Similar to the case of order-line attributes, modify the deployment descriptor for the Customer entity.

- Unjar the lib\ebusiness.jar into a temporary directory, say for instance,
- `jar –xvf lib\ebusiness.jar c:\temp\ebusiness`
- Goto c:\temp\ebusiness\META-INF, and open weblogic-ejb-jar.xml file.
- In this file, search for the following entry, and add the text marked in bold.

```
<weblogic-enterprise-bean>
   <ejb-name>
     com.beasys.commerce.ebusiness.customer.Customer
   </ejb-name>
   <persistence-descriptor>
     <is-modified-method-name>
```

```
                isModified
              </is-modified-method-name>
              <delay-updates-until-end-of-tx>
                 false
              </delay-updates-until-end-of-tx>
            </persistence-descriptor>
                 <reference-descriptor>
                    ...
            </reference-descriptor>
                 <enable-call-by-reference>true</enable-call-by-
reference>
            <jndi-name>
               com.beasys.commerce.ebusiness.customer.Customer
            </jndi-name>
          </weblogic-enterprise-bean>
```

- Jar the contents of the temporary directory, and run the ejbc to create a new ebusiness.jar
- Replace the lib\ebusiness.jar with the newly created ebusiness.jar.

Step 7: Include new pipeline component.

Modify the pipeline.properties to include the new pipeline component.

Transaction Management

In the webflow/pipeline infrastructure, you can declaratively demarcate pipelines within transactions. Although the default pipeline configuration has certain default settings on the pipelines, you should reconsider your options while deploying your extensions on the WebLogic Commerce Server.

Depending on how you're customizing a use case flow, consider if the new pipeline component should participate in a preexisting pipeline? The answer depends on whether the database access in the new pipeline component is part of another unit of work or not.

In cases such as capturing additional order/order line information, add the new pipeline component to CommitOrder pipeline. This is a transactional pipeline, and therefore the updates made in the new pipeline component would happen in the same transaction as that of the CommitOrder pipeline.

If the database access new pipeline component is independent of any existing pipelines, define a new pipeline with the new pipeline component. Note that, you can chain multiple pipelines. For instance, consider four pipeline components A, B, C, and D. If A, B, and C are required to execute within a single transaction, while D is not, define two different pipelines (one consisting of A, B, and C), and the other consisting of D. Set the first pipeline to be transactional, and depending on whether D should execute in its own transaction or no transaction at all, specify the second pipeline to be transactional or not.

Summary

This document discusses on the data extensibility mechanisms for the customer registration and order processing packages of the WebLogic Commerce Server 3.1 product. The approaches suggested in this document leverage the webflow/pipeline infrastructure to extend the out-of-the-box data model to meet your business-specific commerce scenarios. The goal of these approaches has been to minimize the code dependency impact on the out-of-the-box functionality of the WebLogic Commerce Server.