BEA WebLogic Portal™

Guide to Managing Purchases and Processing Orders

Version 4.0
Document Date: April 2002

Guide to Managing Purchases and Processing Orders

Document Edition	Part Number	Date	Software Version
4.0.3	N/A	April 2002	WebLogic Portal 4.0, Service Pack 1

Contents

About This Document

This document explains how to use the functionality within the BEA WebLogic Portal™ Order services.

This document includes the following topics:

- Chapter 1, "Overview of Managing Purchases and Processing Orders," which describes the high-level architecture for managing purchases and processing orders. It also provides introductory information about its services.
- Chapter 2, "Discounts," which provides background on how discounts work and examples of how discounts are applied.
- Chapter 3, "Shopping Cart Management Services," which describes the JSP templates, input processors, and Pipelines associated with the shopping cart Web pages.
- Chapter 4, "Shipping Services," which describes the JSP templates, input processors, and Pipelines associated with the shipping Web pages.
- Chapter 5, "Taxation Services," which describes the Tax Web service, the JSP templates, input processors, and Pipelines associated with the Tax Web service, and provides instructions for connecting your enterprise applications to third-party tax calculation products.
- Chapter 6, "Payment Services," which describes the Payment Web service, JSP templates, input processors, and Pipelines associated with the Payment Web service, and provides instructions for connecting your enterprise applications to third-party payment processing products.
- Chapter 7, "Order Summary and Confirmation Services," which describes the JSP templates, input processors, and Pipelines associated with the order summary and confirmation Web pages.

- Chapter 8, “Extending the Data Model,” which explains how to extend Order services.
- Chapter 9, “Using the Order and Payment Management Pages,” which describes how to find and manage customer orders and modify payment transactions.
- Chapter 10, “The Order Processing Database Schema,” which describes the database tables used for order processing activities.

What You Need to Know

This document is intended for the following audiences:

- The business engineer (BE) or JSP content developer, who uses JSP templates and tag libraries to implement interactive Web pages to meet business requirements. This user also maintains simple configuration files.
- The business analyst (BA), who defines the company’s business protocols (processes and rules) for a Web site. This user may set pricing policies and discounts, and may plan promotional advertising.
- The site administrator, who uses the WebLogic Portal administration screens to configure the site’s rules, portals, property sets, user profiles, content delivery, and product catalog.
- The Java or EJB programmer, who creates custom code to insert in the JSP files. This user may also handle complex configuration files.

e-docs Web Site

BEA product documentation is available on the BEA corporate Web site. From the BEA Home page, click on Product Documentation or go directly to the “e-docs” Product Documentation page at http://e-docs.bea.com.

How to Print the Document

You can print a copy of this document from a Web browser, one file at a time, by using the File—>Print option on your Web browser.

A PDF version of this document is available on the WebLogic Portal and WebLogic Personalization Server documentation Home page on the e-docs Web site (and also on the documentation CD). You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access the PDFs, open the WebLogic Portal and WebLogic Personalization Server documentation Home page, click the PDF files button and select the document you want to print.

If you do not have the Adobe Acrobat Reader, you can get it for free from the Adobe Web site at http://www.adobe.com/.

Related Information

The following WebLogic Portal documents contain information that is relevant to using the Order services and understanding how to customize or extend the provided functionality.

- The *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline*.
- The *Guide to Registering Customers and Managing Customer Services*.
- The *Guide to Building a Product Catalog*

Contact Us!

Your feedback on the WebLogic Portal and WebLogic Personalization Server documentation is important to us. Send us e-mail at **docsupport@bea.com** if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the WebLogic Portal and WebLogic Personalization Server documentation.

In your e-mail message, please indicate that you are using the documentation for the WebLogic Portal and WebLogic Personalization Server 4.0 release.

If you have any questions about this version of WebLogic Portal or WebLogic Personalization Server, or if you have problems installing and running WebLogic Portal or WebLogic Personalization Server, contact BEA Customer Support through BEA WebSUPPORT at **www.bea.com**. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

When contacting Customer Support, be prepared to provide the following information:

- Your name, e-mail address, phone number, and fax number
- Your company name and company address
- Your machine type and authorization codes
- The name and version of the product you are using
- A description of the problem and the content of pertinent error messages

Documentation Conventions

The following documentation conventions are used throughout this document.

Convention	Item
boldface text	Indicates terms defined in the glossary.
Ctrl+Tab	Indicates that you must press two or more keys simultaneously.
italics	Indicates emphasis or book titles.

`monospace text`	Indicates code samples, commands and their options, data structures and their members, data types, directories, and filenames and their extensions. Monospace text also indicates text that you must enter from the keyboard.
	Examples:
	`#include <iostream.h> void main ( ) the pointer psz`
	`chmod u+w *`
	`\tux\data\ap`
	`.doc`
	`tux.doc`
	`BITMAP`
	`float`

`monospace boldface text`	Identifies significant words in code. *Example*: `void` **`commit`** `( )`
`monospace italic text`	Identifies variables in code. *Example*: `String` *`expr`*
UPPERCASE TEXT	Indicates device names, environment variables, and logical operators. *Example*s: LPT1 SIGNON OR

Convention	Item
`{ }`	Indicates a set of choices in a syntax line. The braces themselves should never be typed.

`[ ]`	Indicates optional items in a syntax line. The brackets themselves should never be typed.
	Example:
	`buildobjclient [-v] [-o name ] [-f file-list]...`
	`[-l file-list]...`

`|`	Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.

`...`	Indicates one of the following in a command line:
	■ That an argument can be repeated several times in a command line
	■ That the statement omits additional optional arguments
	■ That you can enter additional parameters, values, or other information
	The ellipsis itself should never be typed.
	Example:
	`buildobjclient [-v] [-o name ] [-f file-list]...`
	`[-l file-list]...`

. . .	Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed.

1 Overview of Managing Purchases and Processing Orders

The process customers go through when making a purchase from your Web site is one of the most common but complex aspects of an e-business. To help you get to market faster than your competitors, the BEA WebLogic Portal provides out-of-the-box Order services. These services contains default implementations for the most common e-business order-related functions, such as shopping cart management, taxation, payment, and so on. Moreover, these services allows your site designers to customize the order process without the need for advanced programming skills. Additionally, it is easily extensible for those with advanced technical knowledge. This topic provides you with some background information about purchase management and order processing. It also introduces you to the types of services that are available.

This topic includes the following sections:

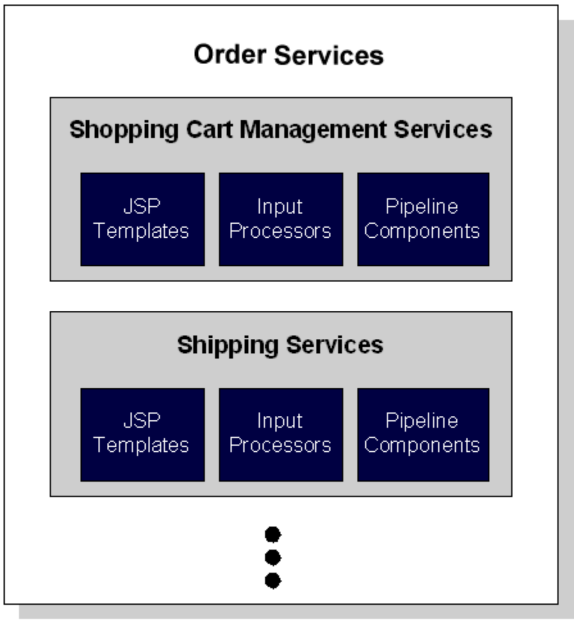
- What Are Order Services?
- High-level Architecture
- Development Roles
- Next Steps

What Are Order Services?

Order services is a collection of services used to facilitate the online ordering process. There are services for shipping, payment, and so on. Together, these services handle all of the tasks necessary to process your customers' orders, from the acceptance of items in their shopping cart to final order confirmation.

As shown in Figure 1-1, each service consists of one or more JavaServer Pages (JSPs) templates and the business logic associated with them. Some of these templates may collect information from your customers, while others will simply display dynamic data your customer previously supplied. Some JSPs may do both. The logic is implemented as a combination of input processors and Pipeline components, each of which can be customized to suit your needs. You can also incorporate the input processors and Pipeline components you create into the Order services.

Figure 1-1 Structure of Order Services



Because all the business logic is managed by a Pipeline and accessed within a Pipeline session, the state of your customer's ordering experience can be maintained. For detailed information about Pipelines (including Pipeline components and Pipeline sessions), see the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline*.

In addition to the services available for order processing, the WebLogic Portal also contains services for browsing the product catalog and registration/user processing. For information on services related to the product catalog, see the *Guide to Building a Product Catalog*. For information on services related to registration and user processing, see the *Guide to Registering Customers and Managing Customer Services*.

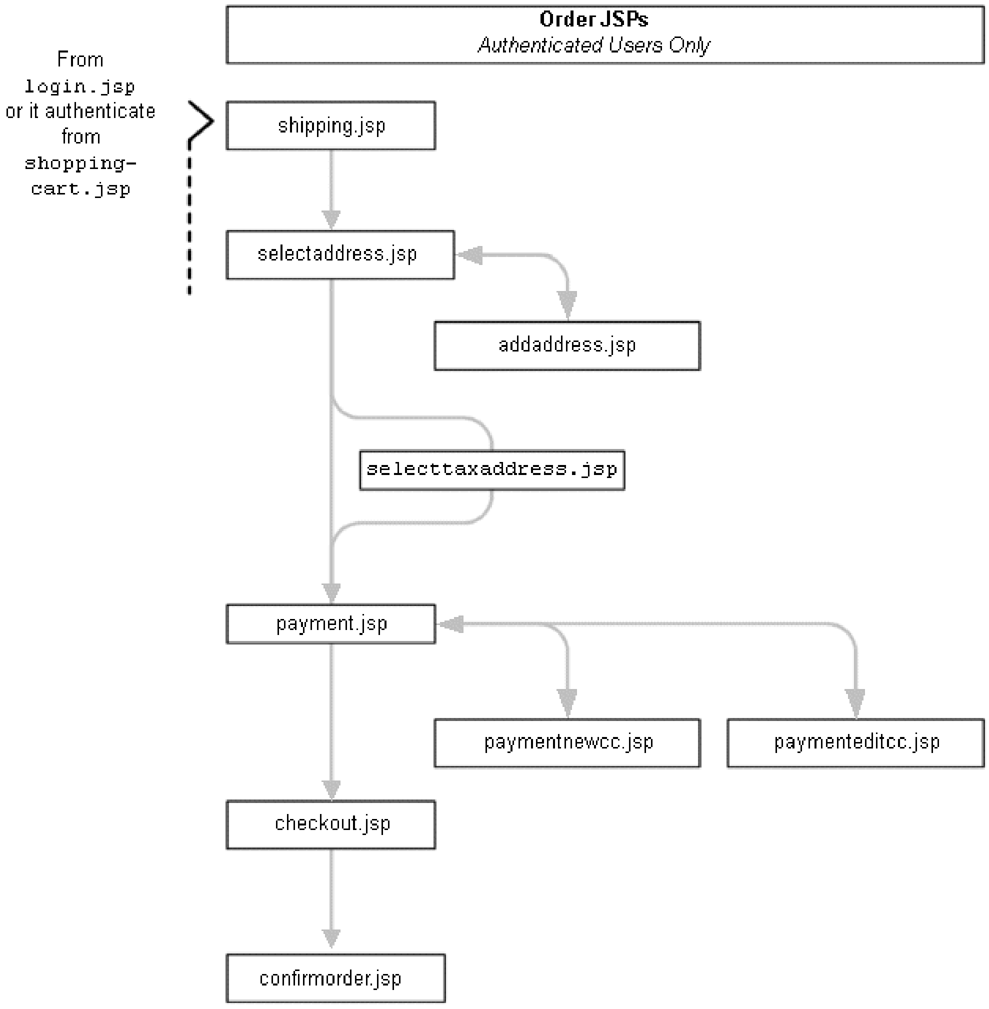
High-level Architecture

Order services is essentially an application that utilizes the Webflow/Pipeline infrastructure. Before you begin to customize or extend this application, however, it is important that you have a high-level understanding of how all the JSP templates in this service work together in the default Webflow. It is also important that you understand how this functionality works in conjunction with the JSP templates in the Registering Customers and Managing Customer services.

- For more information about the default Webflow, see the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline*.
- For more information about the Registering Customers and Managing Customer services, see the *Guide to Registering Customers and Managing Customer Services*.

Figure 1-2 shows the ways in which your customer might move through the JSP templates in the Order services. It also shows where Registering Customers and Managing Customer services comes into play. Only customers who have registered and have a valid username/password combination can browse the order-related pages (any page in the `/order` subdirectory). Additionally, customers who have registered can modify their user profile, check the status of their current order, or even check their order and payment history in the customer self-service pages (using pages in the `/user` subdirectory).

Figure 1-2 Default Webflow for Order Processing



Note: All JSP templates include other templates, making it easy for you to create new pages with the same look and feel.

Whether you are customizing or extending this architecture, everything you need to know about functionality in Order services (including the JSP templates, input processors, and Pipeline components associated with them) is provided in this document. This includes detailed information about the database schema, for those advanced programmers who want to take their e-business site to the next level.

Development Roles

This document is intended for the following audiences:

- The business engineer/JSP content developer, who uses JSP templates and tag libraries to implement interactive Web pages to meet business requirements. This user also maintains simple configuration files.
- The business analyst, who defines the company's business protocols (processes and rules) for a business-to-consumer Web site. This user may set pricing policies and discounts, and may plan promotional advertising.
- The site administrator, who uses WebLogic Portal administration screens to configure the site's rules, portals, property sets, user profiles, content delivery, and product catalog.
- The Java/EJB programmer, who creates custom code to insert in the JSP files. This user may also handle complex configuration files.

Next Steps

Subsequent chapters of this document describe Order services in detail, and provide you with information you need to customize or extend the default implementations to meet your requirements. These chapters are as follows:

- "The Order Processing Database Schema"
- "Shopping Cart Management Services"

- “Shipping Services”
- “Taxation Services”
- “Payment Services”
- “Extending the Data Model”
- “Order Summary and Confirmation Services”

2 Discounts

This topic provides background information about discounts. It does not provide instructions on creating, maintaining, and removing discounts. For instructions on how to perform these tasks, see *Guide to Using the E-Business Control Center*.

This topic includes the following sections:

- Campaign and Stand-Alone Discounts
- Introduction to How Discounts Work
- Discount Management Service
- Association Service
- Price Service
- Triggers and Targets Specifications
- Examples

Campaign and Stand-Alone Discounts

There are two ways to use discounts. You can use discounts targeted to specific customers or have them available to all customers. Discounts targeted to specific customers are called campaing discounts. Discounts available to all customers are called stand-alone discounts.

Notes: In Javadoc and API documentation, campaign discounts are referred to as user discounts and stand-alone discounts are referred to as global discounts.

Discounts are not available if your product license is only for BEA WebLogic Personalization Server.

Introduction to How Discounts Work

Discounts are based on either items or orders. Item discounts modify the price charged for one or more items placed in a shopping cart. Order discounts apply to the order subtotal.

Item discounts are based on the number of items and the properties (SKU and product category) of each item. A discount is applied when particular quantity and property conditions are met. The conditions are defined by the discount definition. For example, when a customer purchases two items where SKU=T123, apply a 15% discount.

Order discounts can be applied to any order or based on the subtotal of the order. For example, you could apply a 10% discount to every order or only to orders with subtotals greater than $50. Additionally, you can specify whether to apply order discounts to the order subtotal or to the shipping cost. For example, you could specify that an order with a subtotal greater than $100 is discounted by $10 or that the order will be shipped for free.

Note: When you specify currency amounts for a discount, the type of currency you use must match the type of currency used for the items in your catalog.

Items that cause a discount to be offered are called *trigger items* and the items that are discounted as a result are called *target items*. Both per item and set-based discounts are triggered based on the item (SKU), product category, or combination of items and product category. The discount can be targeted to the same items that triggered the discount or targeted to other items in the product catalog.

The Discount system is comprised of the Price service, Discount Management service, and Discount Association service. The Price service applies discounts to the items or orders in a shopping cart. The Discount Management service defines and maintains a set of discounts used by the Price service. The Association service is used by campaigns to determine if a particular customer is eligible for specific discounts.

These services work together to provide discounts to your customers. Each service is described in detail in the sections that follow.

Discount Management Service

The Discount Management service defines discounts. Business Analysts or Marketing Professionals can define discounts in the E-Business Control Center. Discount definitions include the duration of the discount, the amount of the discount, the type of discount, the discount limits, and the priority of each discounted item or order.

Definition Parameters

As previously mentioned, discounts are defined in the Discount Management service. Discounts are defined by the following parameters:

- **Discount Name**—the name of the discount.
- **Duration**—the date and time a discount starts and ends.

Notes: Campaign and discount dates are independent from each other. Campaign dates associate discounts to users. Irrespective of anything a campaigns may or may not do, the Price service attempts to apply a discount when the current date and time of the order is within the range of the start and end dates of the discount.

If you deploy a discount in a different time zone from where the discount was defined, it will deploy at the concurrent time in the local time zone. For example, if you set the discount to deploy at 12:00 A.M. Pacific Standard Time, it will deploy at 3:00 A.M. Eastern Standard Time.

- **Discount Types**—two types of discounts exist:
  - Item—this type applies either to individual items in a customer's shopping cart (per item discount) or to a of set items in the customer's shopping cart (set-based discount).
  - Order—this type applies to a customer's order subtotal.

- **Discount Limits**—three types of limits exist:
  - Overall Limit—applies to both per item and set-based limits. This limit is the number of orders to which a discount can be applied for a given customer.

    For example, say your store offers a 10% discount on books with an overall limit of 2. This means that customers can receive the 10% discount for up to two separate orders containing books. Without an overall limit, customers would receive the 10% discount on every book order they placed.
  - Per Item Trigger Limits—the minimum and maximum cardinality for selecting trigger items.

    Minimum Purchase Requirement—the minimum limit that must be reached to trigger the discount.

    Maximum Limit—the maximum number of items of a particular kind to which a discount can be applied.
  - Per Item Target Limits—specifies the number of items to select for the target. Target item limitations are up to or exactly N, where N is a value equal to or greater than 1.
  - Set-Based Triggers—specifies the size of the trigger set. Set-based triggers are specified exactly; the value must be equal to or greater than 1.
  - Set-Based Target Limits—specifies the number of items to select for the target. Target items limitations are up to or exactly N, where N is a value equal to or greater than 1.
- **Discount Priorities**—a discount priority is a setting within the E-Business Control Center that allows you to specify the relative importance of a discount. The discount priority is a value in the range of 1–20, with 1 being the highest priority.
- **Stand-alone Display Description**—applies only to stand-alone discounts. This feature is available so that JSP developers can show a description of the discount to customers.

Note: For campaign discounts, the displayed description is maintained in the association for the user and discount.

- **Active/Deactive Flag**—this feature allows you to deactivate a discount if a mistake is found in a discount. This should be used for emergencies only.

Association Service

The Campaign service uses the Association service to links discounts with particular customers. Campaigns provides the means to target behavior and associate a behavior with a discount. For example, in a campaign, when a customer clicks an ad or fills out a survey, that customer becomes eligible to receive a discount. The customer's behavior results in making an association between a discount and the customer. The Price service uses associations to discount items or orders for particular customers.

The association consists of a Customer ID (`CustomerPk`), a discount identifier (set and discount name), and a discount display description. The Association service maintains a count of uses for each association. The count of uses is the current value of how many times the customer has used the discount. Stand-alone discounts are also tracked in a similar manner. When an order is confirmed, the count of uses is updated.

Price Service

The Price service applies the discounts that are defined in the Discount Management service. The Price service checks with the Association service to determine if a particular customer is eligible for specific discounts. The Discount Management service defines which items and what quantities are required for a discount and which items receive the discounts. The items that qualify for a discount may or may not be the same as the items that receive the discounts. The application of the discount process is defined in terms or triggers and targets. The Shopping service uses the Price service to apply discounts.

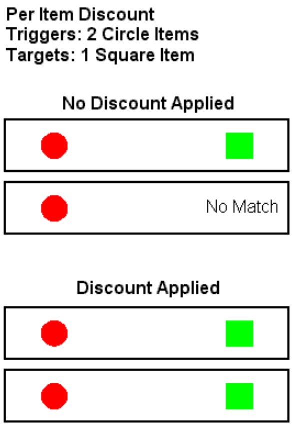
Triggers and Targets Specifications

Triggers and targets specify which items are required to activate a discount and which items are discounted. Recall, that items that cause a discount to be offered are called *trigger items* and the items that are discounted as a result are called *target items*. A discount can be targeted to the same items that triggered the discount or targeted to other items in the product catalog.

Both triggers and target specifications must be satisfied in order for a discount to be applied. The rules for triggers and targets are quite complex. Before introducing these rules, you should understand triggers and targets in relation to per item discounts and set-based discounts. Both per item and set-based discounts are triggered based on the item (SKU), product category, or combination of items and product category.

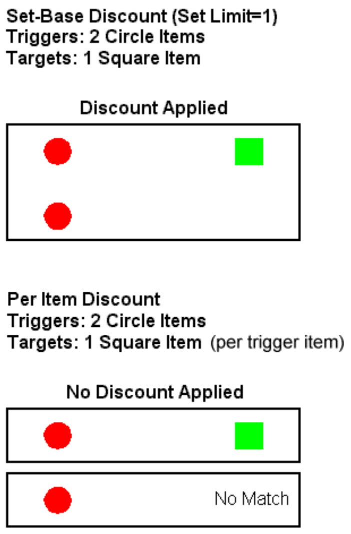
In a per item discount, each individual trigger item must be *paired* with items designated by the target specification. Figure 2-1shows this relationship. Notice that in both cases the triggers and the targets is the same.

Figure 2-1 Per Item Discount Comparison



In set-based discounts, the set of trigger items as a whole are collectively matched with items designated by the target specification. Figure 2-2 shows a comparison of per item and set-based discounts. Both types of discounts have the same number of triggers and targets. However, the results are quite different: for the set-based discount a discount is applied but not for the per item discount.

Figure 2-2 Set-based Discount Versus Per Item Discount



Two Examples of Using Triggers and Targets

Example 1: Per Item Discounts

Trigger Specification: Up to 5 bats
Target Specification: 1 baseball

For per item discounts, select the trigger items and for each trigger item, and then pair each individual trigger item with the items designated by the target specification. If the target specification is 1 item and 5 trigger items are available, then 5 target items (if available) will be discounted. To illustrate this, suppose that your target item is a baseball and your trigger item is your line of baseball bats, all belonging to the same category. If a customer buys one bat from the bat category, the customer will get a free baseball, and if a customer buys two bats, two baseballs will be free, and so on up to five baseballs.

Example 2: Set-based Discounts

Trigger Specification: 5 CDs
Target Specification: 1 CD Wallet

For set-based discounts, select the trigger items that match the pattern described by the trigger specification, and then select a collection of targets that match the target specifications. For example, if the target is a single item and the trigger items are any five items, only 1 target item will be discounted, such as a CD Wallet. For example, if a customer buys any 5 CDs, the customer will get 1 free CD Wallet. If the customer buys only 4 CDs, the customer will not get a CD Wallet. If the customer buys 10 CDs, the customer will get only 1 CD Wallet.

An extensive list of trigger and target examples is in “Examples” on page 2-10. It shows a number of examples to illustrate the different discount combinations.

Consumption Model

To explain how the discounting operation works, a consumption model is used. Before describing how the model works, some terminology needs to be clarified. An item is one particular product represented by its SKU, such as a DVD player where SKU=T123. A line item is a particular product and its quantity, such as DVD player where quantity=3 and SKU=T123. The consumption model is based on items. Each item can be discounted only once; a line item where quantity=N may have up to N discounts. For example, you could offer a discount where your customers would receive a 15% price reduction if they buy two or more cases of dog food on each case up to 10 cases.

The Price Service applies discounts to a pool of items according to the discount definition. When a discount is applied to a pool of items, the set of items (triggers and targets) that match the discount definition are removed from the pool. The Price service continues to apply discounts to the items that match the discount definition and then remove those items from the pool until it runs out of discounts, or until no more items lie within the pool, or until no discounts match the remaining items. Recall that each item can be discounted only once.

The consumption model ensures that the items are consumed as the discount is applied. No item may be used to trigger two items and no item may be discounted more than once.

How Discounts Are Applied

The Price service gets stand-alone discounts for every pricing operation. A pricing operation is the process of examining the contents of a shopping cart or order and applying the appropriate discounts. Recall that the order discount can be applied to either the order subtotal or shipping charges. If a customer is specified in the request to the Price service, that customer's campaign discounts are applied; the Price service calls the Association service to get a list of associations for that customer, and then gets the discounts for those associations.

Discounts are applied in the following manner: The Price service first separates item discounts from order discounts. It then sorts item discounts by the priority, with 1 being the highest priority. Next, the Price service applies discounts to the set of items and computes the subtotal (that is, the sum of the line item prices). At this point, the Price service starts applying the order discounts. It first sorts the order discounts by priority and then applies them. After all the order discounts are applied, the discount process is complete.

Priority

Item or order discounts are sorted by priority from 1 to 20, with 1 being the highest priority. Priority is especially important when two or more discounts refer to a similar collection of items. More specifically, if trigger and target specifications of two or more discounts potentially select the same items, the discounts conflict.

If two or more discounts have the same priority, each discount is still eligible for application. The order in which discounts with the same priority are applied is random. Recall that each item may be discounted only once. A line item with quantity=3 may have three discounts applied. The Price service applies all possible discounts.

Note: For best results, you should avoid conflicting discounts by adjusting the priorities.

How Discounts Are Calculated

There are three methods for adjusting prices on a product: a percentage off discount, a fixed off discount, and a fixed price discount. For each discount method, a calculator (class) exists in the Price service that calculates the new price for an item based on a value, such as 5% or $5. You can use the E-Business Control Center to set these values. Each method is defined in the following list:

- **Percentage Off Discount**—A discount where the price is reduced by a certain percentage, such as 10% off. The calculator applies the following formula:

  newPrice = oldPrice(1 – value), where 0.0 =< value <=1.0 and value is a property of the discount definition. For example, $90 = $100(1 – .1).

- **Fixed Off Discount**—A discount where the price of an item is reduced by a set monetary value such as $5 off. The calculator can never reduce the item price below zero. The calculator applies the following formula:

  newPrice = oldPrice – value, where value is any non-negative monetary value. For example, $45 = $50 – $5.

- **Fixed Price Discount**—A discount where the price is reduced to a particular price. The calculator applies the following formula:

  newPrice = value, where value is any non-negative monetary value. For example, $12 = $12, where the original price was $15.

  You can use a fixed price discount to raise the price of an item.

Note: When you specify currency amounts for a discount, the type of currency you use must match the type of currency used for the items in your catalog.

Examples

This section provides a number of examples for using triggers and targets for item discounts and order discounts.

Item Discounts

This section provides information about the form of item discount rules and examples of the rules. Before the form of the rules can be explained, some terminology needs to be explained. The following list describes this terminology.

- [Square Brackets] denotes optional elements.
- An asterisk (*) denotes zero or more of the preceding element.
- A <discount modifier> refers to the type of calculation: percentage off, fixed off, or fixed price.
- When the term "each qualifying item" is used it refers to each qualifier. In the case of "each set of <x> items" or "the set of all items," the set becomes the one qualifying item.
- Attributes are either SKU or category.

Other elements are defined in the context of the discount rule or explanation.

Form of Discount Rules

Discount rules have a particular structure. The form of each part of a discount rule is presented, along with examples.

General Form

The general form of a discount has the following structure:

<qualifier clause> apply a <discount modifier> discount to <target clause>

Examples

Rule: For all items where SKU=123, apply a 10% discount to each qualifying item.

What It Means: Apply a 10% discount to all items.

Rule: For all items, apply a $5 discount to each qualifying item.

What It Means: Reduce the price of each item by $5.

Qualifier Clause with Property Clauses

The qualifier clause consists of the following forms:

<qualifier phrase> [AND <qualifier phrase>]*

The AND condition allows you to link phrases together.

The complete qualifier phrases consist of the following:

For <qualifier quantity clause> [<property clause> [OR <property clause>]*]

The OR condition allows you to specify conditions based on one set of properties or a different set of properties.

Per Item Discount with an OR Clause Example

Rule: For all items where Category=ABC or SKU=123, apply a 10% discount to each qualifying item.

What It Means: Apply a 10% discount to all items that have a SKU of 123 or belong to category ABC.

Per Item Discount with an AND Clause Example

Rule: For 3 items where Category=ABC and 2 items where SKU=123 items, apply a $5 fixed price discount to each qualifying item.

What It Means: For 5 items in a shopping cart where 3 items belong to Category=ABC and 2 items having SKU=123, reduce the price of each qualifying item by $5.

Other Per Item Discount Examples

Rule: For at least 3 items where SKU=123, apply a $10 fixed off discount to 2 items where Category=books for each qualifying item.

What It Means: If 5 items with SKU=123 exist and 9 items from the Category=books exist, 8 of the Category=books items are discounted. If 2 items with SKU=123 exist and any number of items from the Category=books exists, none of the Category=books items are discounted.

Rule: For between 3 and 5 items where SKU=123, apply a $10 fixed off discount for up to 2 items where Category=books for each qualifying item.

What It Means: If 6 items with SKU=123 exist and 14 items from the Category=books exist, 10 of the Category=books items are discounted. If 4 items with SKU=123 exist and 12 items from the Category=books exist, 8 of the Category=books items are discounted. If 2 items with SKU=123 exist and any number of items from the Category=books exists, none of the Category=books items are discounted.

Set Discounts Examples

Rule: For each set of 2 items where SKU=123, apply a 10% discount to each qualifying item.

What It Means: Apply a 10% discount to every group of 2 items with SKU=123 selected from the shopping cart. If 5 items with SKU=123 exist, 4 are discounted; if 1 item with SKU=123 exists, none are discounted.

Rule: For each set of 2 items where SKU=123, apply a $10 fixed off discount to 2 items.

What It Means: For every group of 2 items of SKU=123, 2 items (of any kind) are discounted. If 5 items with SKU=123 exist and 6 other items exist, 4 of the other items are discounted; if 5 items with SKU=123 exist and 3 other items exist, 2 of the other items are discounted.

Order Rules

The form of order rules is much more simple than the rules for item discounts. The following list describes the basic rules.

Rule: For order subtotal >= $50, apply a 10% discount to qualifying order.

What It Means: For any order subtotal greater then $50 apply a 10% to the order subtotal.

Rule: For order subtotal >= $50 AND order subtotal <= $100 apply a 10% discount to qualifying item.

What It Means: For any order subtotal between $50 and $100, apply a 10% to the order subtotal.

Rule: For order subtotal >= $100, apply a 10% discount to shipping.

What It Means: For any order subtotal greater then $100, apply a 10% to the cost of shipping.

Rule: For order subtotal >= $100 OR order subtotal <= $25 apply a 10% discount to shipping.

What It Means: For any order subtotal less than $25 or greater than $100 apply a 10% discount to the shipping costs.

CHAPTER

3 Shopping Cart Management Services

As in a physical store, a shopping cart is the mechanism used to store items that a customer decides to purchase from your e-business. Implicitly, the cart also stores various types of information related to these items: a unique identifier, a quantity, a price, discounts, taxes, and so on. Customers need to be able to manage their shopping cart by adding and removing items. This topic provides you with information about the Shopping Cart Management Services, which allow your customers to perform these activities.

This topic includes the following sections:

- JavaServer Pages (JSPs)
  - shoppingcart.jsp Template
- Input Processors
  - DeleteProductItemFromShoppingCartIP
  - EmptyShoppingCartIP
  - InitShoppingCartIP
  - UpdateShoppingCartQuantitiesIP
  - UpdateSkuIP
- Pipeline Components
  - DeleteProductItemFromSavedListPC
  - MoveProductItemToSavedListPC
  - MoveProductItemToShoppingCartPC
  - RefreshSavedListPC

- PriceShoppingCartPC
- AddToCartTrackerPC
- RemoveFromCartTrackerPC
- UpdateShoppingCartQuantitiesTrackerPC

JavaServer Pages (JSPs)

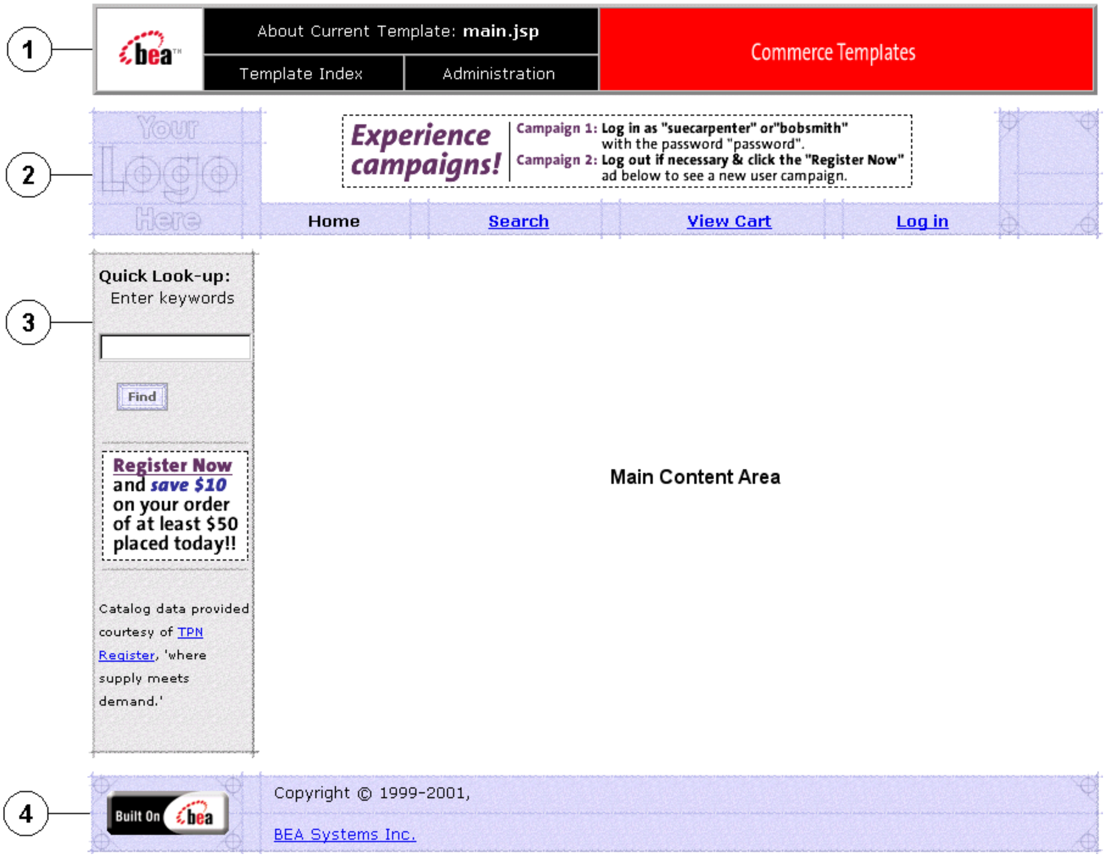
Order services contains one JavaServer Page (JSP) that allows your customers to manage their shopping cart. You can choose to utilize this page in its current form, or adapt it to meet your specific needs. This section describes this page in detail.

Note: For a description of the complete set of JSPs used in the WebLogic Portal Web application and a listing of their locations in the directory structure, see the *E-Commerce JSP Template Summary*.

Common JSP Template Elements

Several elements are common to all JSP commerce templates. The callouts in Figure 3-1 point out each common element; a description of each element follows the figure.

Figure 3-1 Common Commerce JSP Template Elements


1
2
3
4
bea
About Current Template: main.jsp
Template Index
Administration
Commerce Templates
Your Logo Here
Experience campaigns!
Campaign 1: Log in as "suecarpenter" or"bobsmith" with the password "password".
Campaign 2: Log out if necessary & click the "Register Now" ad below to see a new user campaign.
Home
Search
View Cart
Log in
Quick Look-up:
Enter keywords
Find
Register Now and save $10 on your order of at least $50 placed today!!
Catalog data provided courtesy of TPN Register, 'where supply meets demand.'
Main Content Area
Built On bea
Copyright © 1999-2001,
BEA Systems Inc.

1. The Commerce Templates header (`admin.inc`) contains useful information for the benefit of your development team. The import call is:

   ```
   <%@ include file="/commerce/includes/admin.inc" %>
   ```

2. The page header is created by importing the `header.inc` template. It is standard across many of the JSP templates provided by WebLogic Portal. The import call is:

   ```
   <%@ include file="/commerce/includes/header.inc" %>
   ```

3. The left column is created by importing the `leftside.inc` template. It is also a secondary placeholder for advertising. It is standard across many of the JSP templates provided by WebLogic Portal. The import call is:

```
<%@ include file="/commerce/includes/leftside.inc" %>
```

4. The page footer is created by importing the `footer.inc` template. It is standard across many of the JSP templates provided by WebLogic Portal. The import call is:

```
<%@ include file="/commerce/includes/footer.inc" %>
```

shoppingcart.jsp Template

The `shoppingcart.jsp` template (shown in Figure 3-2 and Figure 3-3) displays the items currently in a customer's shopping cart. For each item the customer added to their cart (that is still actively part of the current purchase), the `shoppingcart.jsp` template displays the quantity, the item name, the list price, the actual price, a savings amount, and a subtotal. Following this information, a total price for the order is displayed.

The item quantity is shown in an editable field, allowing customers to change the quantity of the item simply by typing a new quantity and clicking the Update button. For your customers' convenience, the item name is hyperlinked back to its description in the product catalog. For each item in the shopping cart, there is also a Delete button and a Buy Later button. Clicking the Delete button removes the item from the shopping cart, while clicking the Buy Later button causes the item to be moved from the Shopping Cart to the Saved Items list. For each item shown in the Saved Items list, the hyperlinked item name and a brief description are displayed. Additionally, the Delete and Add to Cart buttons in this section allow your customers to remove the item altogether or to move it back to their active Shopping Cart.

Notes: To be able to use the features of the Saved Items list, a customer must have first logged in.

If there are no items in a customer's shopping cart, the Empty Cart, Update, and Check Out buttons will not be available.

If the customer is satisfied with the contents of their shopping cart, the customer can click the Check Out button to begin the checkout process.

Note: If the customer is not logged into your e-commerce site, they will be prompted to do so before continuing to the next part of the checkout process.

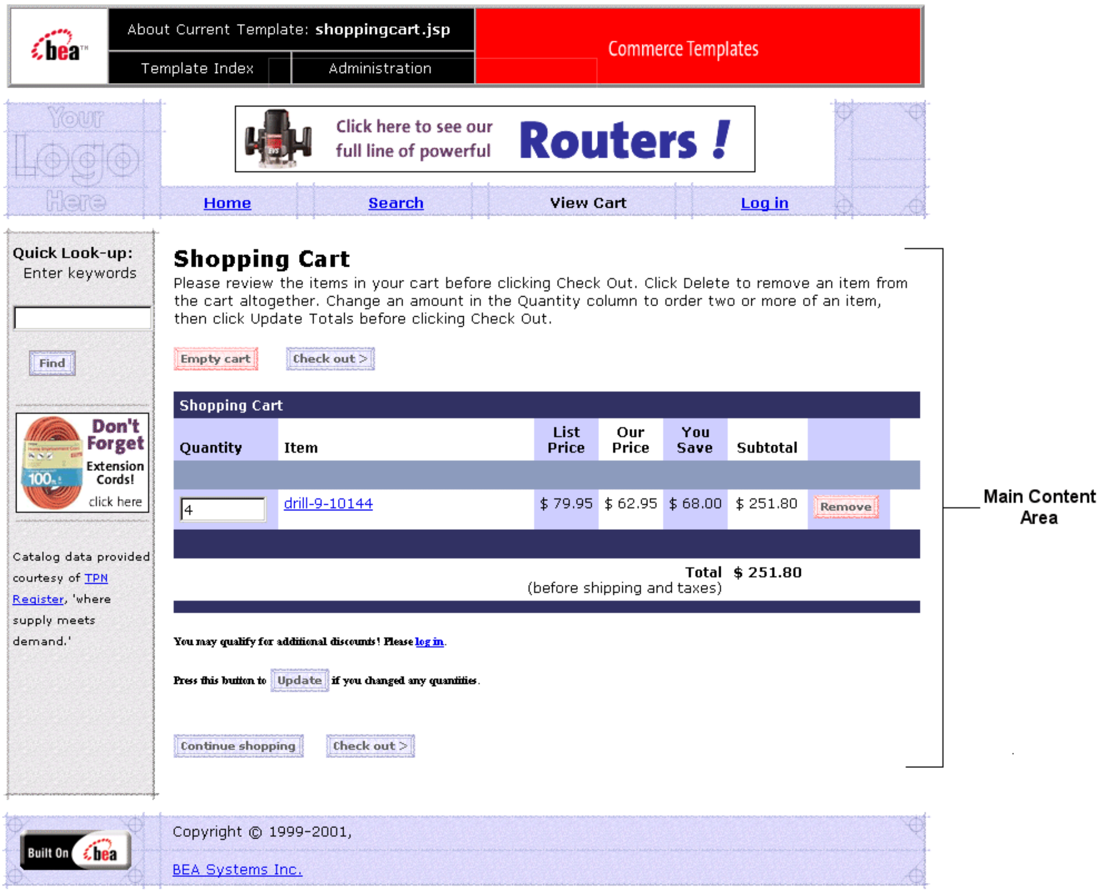
If your customer wants to start over, the customer can click the Empty Cart button to empty the entire contents of the shopping cart (both active and saved). If your customer wants to continue shopping, the customer can click the Continue Shopping button to return to the product catalog.

Sample Browser View

Figure 3-2 and Figure 3-3 show annotated versions of the `shoppingcart.jsp` template; the first figure shows the page for a customer who has not logged in, the second shows the page for a customer who has logged in. The main content area of the template contains both dynamically generated data and static content. The dynamic content on `shoppingcart.jsp` is generated using WebLogic Server and Pipeline JSP tags, which obtain and display the contents for both the active shopping cart and Saved Item list. For the `shoppingcart.jsp` template, the form posts include Empty Cart, Check Out, Remove, Update, and Continue.

Note: For information on other elements in the `shoppingcart.jsp` template, see "Common JSP Template Elements" on page 3-2.

Figure 3-2 Annotated shoppingcart.jsp Template - Customer Not Logged In



Catalog data provided courtesy of TPN Register, 'where supply meets demand.'

Shopping Cart

Please review the items in your cart before clicking Check Out. Click Delete to remove an item from the cart altogether. Change an amount in the Quantity column to order two or more of an item, then click Update Totals before clicking Check Out.

Shopping Cart

Quantity	Item	List Price	Our Price	You Save	Subtotal	
4	drill-9-10144	$ 79.95	$ 62.95	$ 68.00	$ 251.80	Remove

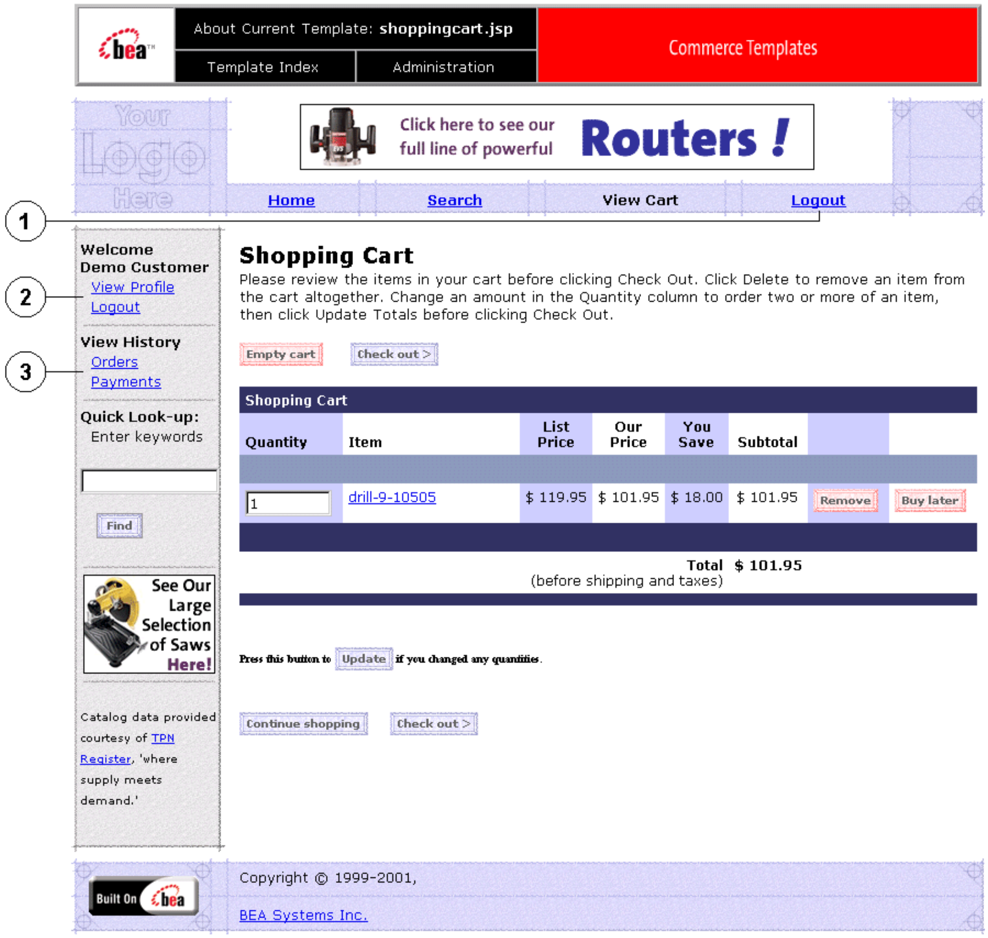
Total $ 251.80
(before shipping and taxes)

You may qualify for additional discounts! Please log in.

Press this button to Update if you changed any quantities.

Copyright © 1999-2001, BEA Systems Inc.

Figure 3-3 Annotated shoppingcart.jsp Template - Customer Logged In



Shopping Cart

Please review the items in your cart before clicking Check Out. Click Delete to remove an item from the cart altogether. Change an amount in the Quantity column to order two or more of an item, then click Update Totals before clicking Check Out.

Quantity	Item	List Price	Our Price	You Save	Subtotal		
1	drill-9-10505	$ 119.95	$ 101.95	$ 18.00	$ 101.95	Remove	Buy later

Total $ 101.95
(before shipping and taxes)

Press this button to Update if you changed any quantities.

Catalog data provided courtesy of TPN Register, 'where supply meets demand.'

Copyright © 1999-2001, BEA Systems Inc.

In Figure 3-3, the following changes occur after the user has logged in:

1. The Login link changes to Logout.
2. A welcome section appears that shows the customer’s name, a link to view that customer’s profile, and a link to logout.
3. A view history section appears that shows the customer’s order and payment history.

Location in the WebLogic Portal Directory Structure

You can find the `shoppingcart.jsp` template file at the following location, where `PORTAL_HOME` is the directory in which you installed WebLogic Portal.

`%PORTAL_HOME%\applications\wlcsApp\wlcs\commerce\shoppingcart.jsp` (Windows)
`$PORTAL_HOME/applications/wlcsApp/wlcs/commerce/shoppingcart.jsp` (UNIX)

Tag Library Imports

The `shoppingcart.jsp` template uses WebLogic Server and Pipeline JSP tags. Therefore, the template includes the following JSP tag libraries:

```
<%@ taglib uri="weblogic.tld" prefix="wl" %>
<%@ taglib uri="webflow.tld" prefix="webflow" %>
<%@ taglib uri="i18n.tld" prefix="i18n" %>
```

Note: For more information on the WebLogic Server JSP tags, see “JSP Tag Reference” in the *Guide to Building Personalized Applications*. For more information about the Pipeline JSP tags, see the *Guide to Registering Customers and Managing Customer Services*.

These files reside in the `lib` directory within `PORTAL_HOME`.

Java Package Imports

The `shoppingcart.jsp` template uses Java classes in the following packages and therefore includes these import statements:

```
<%@ page import="java.util.*" %>
<%@ page import="java.text.*" %>
<%@ page import="com.beasys.commerce.axiom.units.*" %>
<%@ page import="examples.wlcs.sampleapp.shoppingcart.*" %>
<%@ page import="examples.wlcs.sampleapp.price.service.DiscountPresentation" <%@
page import="examples.wlcs.sampleapp.price.quote.OrderAdjustment" %>
<%@ page import="examples.wlcs.sampleapp.price.quote.AdjustmentDetail" %>
<%@ page import="com.beasys.commerce.webflow.HttpRequestConstants" %>
<%@ page import="com.beasys.commerce.webflow.PipelineSessionConstants" %>
<%@ page import="com.bea.p13n.appflow.webflow.WebflowJSPHelper" %>
```

Location in Default Webflow

Customers can arrive at `shoppingcart.jsp` template from any product catalog page by clicking the View Cart button. If the customer is satisfied with the contents of their shopping cart as shown on this page, the customer can initiate the checkout process by clicking the Check Out button. If this is the case, the next page is the shipping information page (`shipping.jsp`).

Note: If the customer has not yet logged into the site and clicks the Check Out button, the customer will be prompted to log in at the `login.jsp` template (prior to loading the `shipping.jsp` template). For more information about the `login.jsp` template, see the *Guide to Registering Customers and Managing Customer Services*.

If customers click a link to an individual product item to review detailed information about that product item, the next page is the appropriate product catalog page. If they click on the Update, Empty Cart, Delete, or Save for Later buttons, they are returned to the shopping cart page (`shoppingcart.jsp`) after the appropriate input processor or Pipeline has been executed to record the modification.

This JSP is in the `sampleapp_order` namespace.

Note: For more information about the default Webflow, see "Overview of Managing Purchases and Processing Orders" on page 1-1.

Events

Every time a customer clicks a button to manage the contents of their shopping cart, it is considered an event. Each event triggers a particular response in the default Webflow that allows the customer to continue. While this response can be to load another JSP, it is usually the case that an input processor and/or Pipeline is invoked first. Table 3-1 provides information about these events and the business logic they invoke.

Table 3-1 shoppingcart.jsp Events

Event	Webflow Response(s)
`--`	`InitShoppingCartIP`
`--`	`RefreshSavedList`

Table 3-1 shoppingcart.jsp Events

Event	Webflow Response(s)
`button.checkout`	`InitShippingMethodListIP`
`button.deleteItemFromShoppingCart`	`DeleteProductItemFromShoppingCartIP`
`button.deleteItemFromSavedList`	`UpdateSkuIP` `DeleteProductItemFromSavedList`
`button.emptyShoppingCart`	`EmptyShoppingCartIP`
`button.moveItemToSavedList`	`UpdateSkuIP` `MoveProductItemToSavedList`
`button.moveItemToShoppingCart`	`UpdateSkuIP` `MoveProductItemToShoppingCart`
`button.updateShoppingCartQuantities`	`UpdateShoppingCartQuantitiesIP`

Table 3-2 briefly describes each of the Pipelines from Table 3-1. For more information about individual Pipeline components, see "Pipeline Components" on page 3-19.

Table 3-2 Shopping Cart Pipelines

Pipeline	Description
`RefreshSavedList`	Contains `RefreshSavedListPC` and is not transactional.
`DeleteProductItemFromSavedList`	Contains `DeleteProductItemFromSavedListPC` and `PriceShoppingCartPC`, and is transactional.
`MoveProductItemToSavedList`	Contains `MoveProductItemToSavedListPC` and `PriceShoppingCartPC`, and is transactional.
`MoveProductItemToShoppingCart`	Contains `MoveProductItemToShoppingCartPC` and `PriceShoppingCartPC`, and is transactional.

Notes: Although the `InitShoppingCartIP` and `RefreshSavedList` Pipeline are associated with the `shoppingcart.jsp` template, they are not triggered by events on the page. Rather, both are executed before the `shoppingcart.jsp` is viewed. The `InitShoppingCartIP` input processor creates an empty shopping cart in preparation for the customer's shopping experience, while the `RefreshSavedList` Pipeline retrieves a customer's list of previously saved shopping cart items.

For information about the `AddProductItemToShoppingCartPC`, a Pipeline component invoked in a Pipeline prior to display of the `shoppingcart.jsp` template, see the "Product Catalog JSP Templates and Tag Library" in the *Guide to Building a Product Catalog*.

Dynamic Data Display

One purpose of the `shoppingcart.jsp` template is to display the data specific to a customer's shopping experience for their review. This is accomplished on `shoppingcart.jsp` using a combination of WebLogic Server and Pipeline JSP tags and accessor methods/attributes.

First, the `getProperty` JSP tag retrieves the `SHOPPING_CART` and `SAVED_SHOPPING_CART` attributes from the Pipeline session. Table 3-3 provides more detailed information on these attributes.

Table 3-3 shoppingcart.jsp Pipeline Session Attributes

Attribute	Type	Description
`PipelineSessionConstants.SAVED_SHOPPING_CART`	`examples.wlcs.sampleapp.shoppingcart.ShoppingCart`	The saved shopping cart (source of the saved items).
`PipelineSessionConstants.SHOPPING_CART`	`examples.wlcs.sampleapp.shoppingcart.ShoppingCart`	The currently active shopping cart.

Listing 3-1 illustrates how these attributes are retrieved from the Pipeline session using the `getProperty` JSP tag.

Listing 3-1 Retrieving Shopping Cart Attributes

```
<webflow:getProperty id="shoppingCart"
property="<%=PipelineSessionConstants.SHOPPING_CART%>"
type="examples.wlcs.sampleapp.shoppingcart.ShoppingCart" scope="session"
namespace="sampleapp_main" />
```

```
<webflow:getProperty id="savedShoppingCart"
property="<%=PipelineSessionConstants.SAVED_SHOPPING_CART%>"
type="examples.wlcs.sampleapp.shoppingcart.ShoppingCart" scope="session"
namespace="sampleapp_main" />
```

Note: For more information on the `getProperty` JSP tag, see the *Guide to Registering Customers and Managing Customer Services*.

The data stored within the Pipeline session attributes is accessed by using accessor methods/attributes within Java scriptlets. Table 3-4 provides more detailed information about these methods for `ShoppingCart` (also `savedShoppingCart`), while Table 3-5 provides this information for `ShoppingCartLine`.

Table 3-4 ShoppingCart Accessor Methods/Attributes

Method/Attribute	Description
`getShoppingCartLineCollection()`	A collection of the individual lines in the shopping cart (that is, `ShoppingCartLine`).
`getTotal`	In this instance, the total tax specified by the `OrderConstants.LINE_TAX` parameter. **Note:** The `getTotal()` method also allows you to combine different total types. For more information, see the *Javadoc*.

Because the `getShoppingCartLineCollection()` method allows you to retrieve a collection of the individual lines within a shopping cart, there are also accessor methods/attributes you can use to break apart the information contained within each line. Table 3-5 provides information about these methods/attributes.

Table 3-5 ShoppingCartLine Accessor Methods/Attributes

Method/Attribute	Description
`getQuantity()`	The quantity of the item.
`getProductItem()`	The product item in the shopping cart line.

Table 3-5 ShoppingCartLine Accessor Methods/Attributes (Continued)

Method/Attribute	Description
`getUnitPrice()`	The current price for the item at the time it was added to the shopping cart. May be different from MSRP.
`getBaseTotal(int totalType)`	The total before discounts.

Listing 3-2 provides an example of how these accessor methods/attributes are used within Java scriptlets.

Note: The `ProductItem` object is described in the *Guide to Building a Product Catalog*.

Listing 3-2 Using Accessor Methods Within shoppingcart.jsp Java Scriptlets

```
<<td align="right" valign="top" bgcolor="#CCCCFF"><div class="tabletext" nowrap>
<%-- The i18n tag allows the "currency.properties" file to substitute a display    --%>
<%-- currency value (e.g "$") for the returned 3 letter ISO4217 code (e.g. "USD"). --%>
        <i18n:getMessage bundleName="/commerce/currency" messageName="<%=
shoppingCartLine.getProductItem().getMsrp().getCurrency() %>"/>
<%= WebflowJSPHelper.priceFormat(
shoppingCartLine.getProductItem().getMsrp().getValue() ) %></div>
      </td>

      <td align="right" valign="top"><div class="tabletext" nowrap>
        <i18n:getMessage bundleName="/commerce/currency" messageName="<%=
shoppingCartLine.getUnitPrice().getCurrency() %>"/>
<%= WebflowJSPHelper.priceFormat( shoppingCartLine.getUnitPrice().getValue() ) %></div>
      </td>

      <td align="right" valign="top" bgcolor="#CCCCFF"><div class="tabletext" nowrap>
        <i18n:getMessage bundleName="/commerce/currency" messageName="<%=
shoppingCartLine.getBaseSavings().getCurrency() %>"/>
<%= WebflowJSPHelper.priceFormat( shoppingCartLine.getBaseSavings().getValue() )
%></div>
      </td>

      <td align="right" valign="top"><div class="tabletext" nowrap>
```

```
        <i18n:getMessage bundleName="/commerce/currency" messageName="<%=
shoppingCartLine.getBaseTotal().getCurrency() %>"/>
<%= WebflowJSPHelper.priceFormat( shoppingCartLine.getBaseTotal().getValue() ) %>
</div>
      </td>
```

Note: For more information on the WebLogic Server JSP tags, see "JSP Tag Reference" in the *Guide to Building Personalized Applications*.

Form Field Specification

Another purpose of the `shoppingcart.jsp` template is to allow customers to make changes to their shopping cart using various HTML form fields. These form fields are also used to pass needed information to the Webflow.

The form fields used in the `shoppingcart.jsp` template, and a description for each of them, are listed in Table 3-6.

Table 3-6 shoppingcart.jsp Form Fields

Parameter Name	Type	Description
`"event"`	Hidden	Indicates which event has been triggered. It is used by the Webflow to determine what happens next.
`"origin"`	Hidden	The name of the current page (`shoppingcart.jsp`), used by the Webflow.
`HttpRequestConstants. CATALOG_ITEM_SKU`	Hidden	SKU of the item that the event is to operate on.
`NewQuantity_<`*`SKU`*`>` Where <*SKU*> is replaced with the SKU of the item on the shopping cart line.	Textbox	The new quantity for the item in the shopping cart. It is the only form field on this page that requires input from the customer.

Note: Parameters that are literals in the JSP code are shown in quotes, while non-literals will require scriptlet syntax (such as `<%= HttpRequestConstants.CATALOG_ITEM_SKU %>`) for use in the JSP.

Input Processors

This section provides a brief description of each input processor associated with the Shopping Cart Management Services JSP template(s).

Note: For information about the `InitShippingMethodListIP` input processor, see the input processors listed in "Shipping Services" on page 4-1.

DeleteProductItemFromShoppingCartIP

Class Name	`examples.wlcs.sampleapp.shoppingcart.webflow.` `DeleteProductItemFromShoppingCartIP`
Description	Removes the item from the shopping cart.
Required `HTTPServletRequest` Parameters	`HttpRequestConstants.CATALOG_ITEM_SKU`
Required Pipeline Session Attributes	`PipelineSessionConstants.SHOPPING_CART` `PipelineSessionConstants.CATALOG_ITEM`
Updated Pipeline Session Attributes	`PipelineSessionConstants.SHOPPING_CART`
Removed Pipeline Session Attributes	None
Validation	None
Exceptions	`ProcessingException`, thrown if the required request parameters or required Pipeline session attributes are not available.

EmptyShoppingCartIP

Class Name	`examples.wlcs.sampleapp.shoppingcart.webflow.` `EmptyShoppingCartIP`
Description	Creates a new shopping cart and stores it in the Pipeline session. The old shopping cart is discarded.
Required `HTTPServletRequest` Parameters	None
Required Pipeline Session Attributes	None

Updated Pipeline Session Attributes	`PipelineSessionConstants.SHOPPING_CART`
	`PipelineSessionConstants.UPDATED_QUANTITY_DELTAS`
	`PipelineSessionConstants.UPDATED_PRODUCT_ITEMS`

Removed Pipeline Session Attributes	None
Validation	None
Exceptions	None

InitShoppingCartIP

Class Name	`examples.wlcs.sampleapp.shoppingcart.webflow.` `InitShoppingCartIP`
Description	Initializes the active shopping cart prior to loading the `shoppingcart.jsp` template. If the shopping cart already exists, this input processor does nothing.
Required `HTTPServletRequest` Parameters	None

Required Pipeline Session Attributes	None
Updated Pipeline Session Attributes	`PipelineSessionConstants.SHOPPING_CART` `PipelineSessionConstants.UPDATED_QUANTITY_DELTAS`
Removed Pipeline Session Attributes	None
Validation	None
Exceptions	None

UpdateShoppingCartQuantitiesIP

Class Name	`examples.wlcs.sampleapp.shoppingcart.webflow.` `UpdateShoppingCartQuantitiesIP`
Description	Validates the quantity fields for each line and sets those quantities in the shopping cart. If the quantity is zero, it will delete the item from the shopping cart.
Required `HTTPServletRequest` Parameters	`NewQuantity_<SKU>` Where <*SKU*> is replaced with the SKU of the item on the shopping cart line.
Required Pipeline Session Attributes	`PipelineSessionConstants.SHOPPING_CART`
Updated Pipeline Session Attributes	`PipelineSessionConstants.SHOPPING_CART` `PipelineSessionConstants.UPDATED_QUANTITY_DELTAS` `PipelineSessionConstants.UPDATED_PRODUCT_ITEMS`
Removed Pipeline Session Attributes	None
Validation	Verifies that the quantity fields only contain positive integers.
Exceptions	`ProcessingException`, thrown if the required request parameters or required Pipeline session attributes are not available.

UpdateSkuIP

Class Name	`examples.wlcs.sampleapp.shoppingcart.webflow.UpdateSkuIP`
Description	Reads the SKU from the HTTP request and places it into the Pipeline session.
Required `HTTPServletRequest` Parameters	`HttpRequestConstants.CATALOG_ITEM_SKU`
Required Pipeline Session Attributes	None
Updated Pipeline Session Attributes	`PipelineSessionConstants.CATALOG_ITEM_SKU`
Removed Pipeline Session Attributes	None
Validation	None
Exceptions	`ProcessingException`, thrown if the required request parameters are not available.

Pipeline Components

This section provides a brief description of each Pipeline component associated with the Shopping Cart Management Services JSP template(s).

Notes: For information about the `AddProductItemToShoppingCartPC`, invoked prior to display of the `shoppingcart`.jsp template, see "the Product Catalog JSP Templates and Tag Library" in the *Guide to Building a Product Catalog*.

Some Pipeline components extend other, base Pipeline components. For more information on the base classes, see the *Javadoc*.

DeleteProductItemFromSavedListPC

Class Name	`examples.wlcs.sampleapp.shoppingcart.pipeline.DeleteProductItemFromSavedListPC`
Description	Removes the item from the saved list and updates the `WLCS_SAVED_ITEM_LIST` table in the database.

Required Pipeline Session Attributes	`PipelineSessionConstants.CATALOG_ITEM_SKU`
	`PipelineSessionConstants.SAVED_SHOPPING_CART`
	`PipelineSessionConstants.USER_NAME`

Updated Pipeline Session Attributes	`PipelineSessionConstants.SAVED_SHOPPING_CART`
Removed Pipeline Session Attributes	None
Type	Session bean
JNDI Name	`examples.wlcs.sampleapp.shoppingcart.pipeline.DeleteProductItemFromSavedListPC`
Exceptions	`PipelineException`, thrown if the required Pipeline session attributes are not available.

MoveProductItemToSavedListPC

Class Name	`examples.wlcs.sampleapp.shoppingcart.pipeline.` `MoveProductItemToSavedListPC`
Description	Removes the item from the shopping cart, adds it to the saved list, and then updates the `WLCS_SAVED_ITEM_LIST` table in the database.

Required Pipeline Session Attributes	`PipelineSessionConstants.CATALOG_ITEM_SKU`
	`PipelineSessionConstants.SAVED_SHOPPING_CART`
	`PipelineSessionConstants.SHOPPING_CART`
	`PipelineSessionConstants.USER_NAME`

Updated Pipeline Session Attributes	`PipelineSessionConstants.SAVED_SHOPPING_CART`
	`PipelineSessionConstants.SHOPPING_CART`
	`PipelineSessionConstants.CATALOG_ITEM`
	`PipelineSessionConstants.QUANTITY`

Removed Pipeline Session Attributes	None
Type	Session bean
JNDI Name	`examples.wlcs.sampleapp.shoppingcart.pipeline.` `MoveProductItemToSavedListPC`
Exceptions	`PipelineException`, thrown if the required Pipeline session attributes are not available.

MoveProductItemToShoppingCartPC

Class Name	`examples.wlcs.sampleapp.shoppingcart.pipeline.` `MoveProductItemToShoppingCartPC`
Description	Removes the item from the saved list, adds it to the shopping cart with a quantity of 1, and then updates the `WLCS_SAVED_ITEM_LIST` table in the database.

Required Pipeline Session Attributes	`PipelineSessionConstants.CATALOG_ITEM_SKU`
	`PipelineSessionConstants.SAVED_SHOPPING_CART`
	`PipelineSessionConstants.SHOPPING_CART`
	`PipelineSessionConstants.USER_NAME`

Updated Pipeline Session Attributes	`PipelineSessionConstants.SAVED_SHOPPING_CART`
	`PipelineSessionConstants.SHOPPING_CART`
	`PipelineSessionConstants.CATALOG_ITEM`

Removed Pipeline Session Attributes	None
Type	Session bean
JNDI Name	`examples.wlcs.sampleapp.shoppingcart.` `pipeline.MoveProductItemToShoppingCartPC`
Exceptions	`PipelineException`, thrown if the required Pipeline session attributes are not available.

RefreshSavedListPC

Class Name	`examples.wlcs.sampleapp.shoppingcart.pipeline.RefreshSavedListPC`
Description	Queries the `WLCS_SAVED_ITEM_LIST` table and refreshes the saved shopping cart in the Pipeline session. The saved list is only refreshed if the saved shopping cart does not exist in the Pipeline session.
Required Pipeline Session Attributes	`PipelineSessionConstants.USER_NAME`
Updated Pipeline Session Attributes	`PipelineSessionConstants.SAVED_SHOPPING_CART`
Removed Pipeline Session Attributes	None
Type	Session bean
JNDI Name	`examples.wlcs.sampleapp.shoppingcart.pipeline.RefreshSavedListPC`
Exceptions	`PipelineException`, thrown if the required Pipeline session attributes are not available.

PriceShoppingCartPC

Class Name	`examples.wlcs.sampleapp.shoppingcart.pipeline.PriceShoppingCartPC`
Description	Invokes the Pricing Service to compute the line totals, discounts, shopping cart total and shopping cart discounts
Required Pipeline Session Attributes	`PipelineSessionConstants.SHOPPING_CART` `PipelineSessionConstants.USER_NAME`

Updated Pipeline Session Attributes	`PipelineSessionConstants.SHOPPING_CART`
Removed Pipeline Session Attributes	None
Type	Java object
JNDI Name	None
Exceptions	`PipelineException`, thrown if the Pricing Service fails in any way

AddToCartTrackerPC

Class Name	`examples.wlcs.sampleapp.tracking.pipeline.AddToCartTrackerPC`
Description	Fires an `AddToCartEvent` describing which item was just added to the cart. For more information about this event, see Event Details in the *Guide to Events and Behavior Tracking*.

Required Pipeline Session Attributes	`PipelineSessionConstants.CATALOG_ITEM`
	`PipelineSessionConstants.HTTP_SESSION_ID`
	`PipelineSessionConstants.USER_NAME`
	`PipelineSessionConstants.STOREFRONT`
	`PipelineSessionConstants.CUSTOM_REQUEST`

Updated Pipeline Session Attributes	None
Removed Pipeline Session Attributes	None
Type	Java object
JNDI Name	None
Exceptions	None

RemoveFromCartTrackerPC

Class Name	`examples.wlcs.sampleapp.tracking.pipeline.RemoveFromCartTrackerPC`
Description	Fires a `RemoveFromCartEvent` describing which item was just added to the cart. For more information about this event, see Event Details in the *Guide to Events and Behavior Tracking*.

Required Pipeline Session Attributes	`PipelineSessionConstants.CATALOG_ITEM`
	`PipelineSessionConstants.HTTP_SESSION_ID`
	`PipelineSessionConstants.USER_NAME`
	`PipelineSessionConstants.STOREFRONT`
	`PipelineSessionConstants.CUSTOM_REQUEST`

Updated Pipeline Session Attributes	None
Removed Pipeline Session Attributes	None
Type	Java object
JNDI Name	None
Exceptions	None

UpdateShoppingCartQuantitiesTrackerPC

Class Name	`examples.wlcs.sampleapp.tracking.pipeline.UpdateShoppingCartQuantitiesTrackerPC`
Description	For each shopping cart line, if more items in the line were selected, fires an `AddToCartEvent`; if fewer items in that line were selected, fires a `RemoveFromCartEvent`; if the number of items in that line is the same as before, no event is fired.

Required Pipeline Session Attributes	`PipelineSessionConstants.UPDATED_PRODUCT_ITEMS`
	`PipelineSessionConstants.UPDATED_QUANTITY_DELTAS`
	`PipelineSessionConstants.HTTP_SESSION_ID`
	`PipelineSessionConstants.USER_NAME`
	`PipelineSessionConstants.STOREFRONT`
	`PipelineSessionConstants.CUSTOM_REQUEST`

Updated Pipeline Session Attributes	None
Removed Pipeline Session Attributes	None
Type	Java object
JNDI Name	None
Exceptions	None

CHAPTER

4 Shipping Services

In Order services, Shipping Services record the shipping information related to a customer's order and calculate shipping costs. This topic describes the Shipping Services in detail, and provides information about how you can customize them to meet your specific needs.

This topic includes the following sections:

- JavaServer Pages
  - shipping.jsp Template
  - selectaddress.jsp Template
  - addaddress.jsp Template
- Input Processors
  - InitShippingMethodListIP
  - UpdateShippingAddressIP
  - ValidateAddressIP
  - ValidateShippingInfoIP
- Pipeline Components
  - AddShippingAddressPC
  - CalculateShippingPC
  - DeleteShippingAddressPC

JavaServer Pages

Shipping Services in Order services consist of three JavaServer Pages (JSPs) that you can use as is, or customize to your own liking. This section describes each of these pages in detail.

Note: For a description of the complete set of JSPs used in the WebLogic PortalWeb application and a listing of their locations in the directory structure, see the E-Commerce JSP Template Summary.

shipping.jsp Template

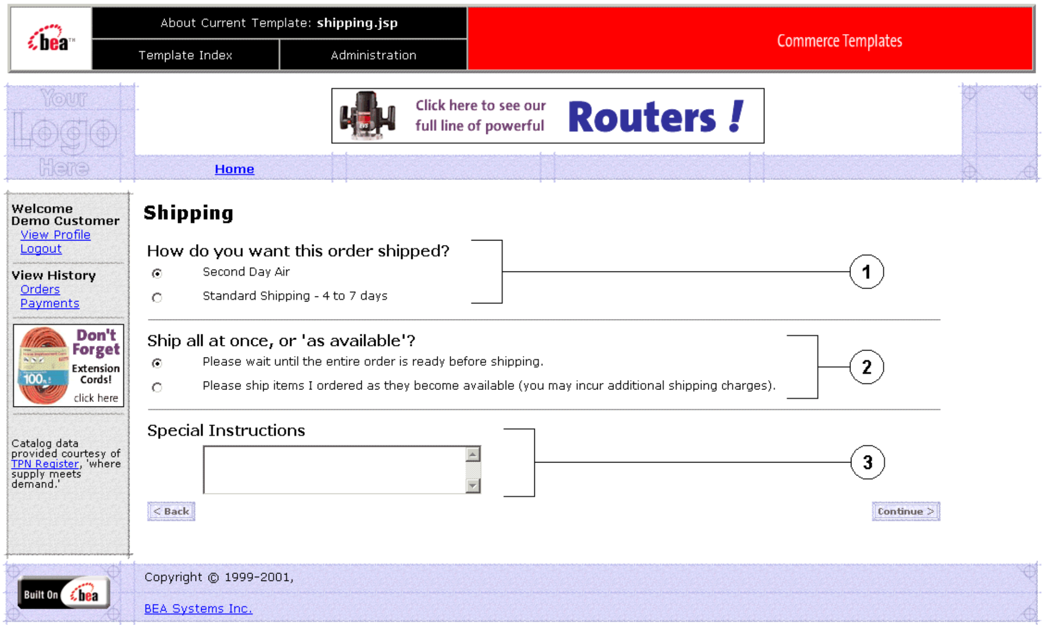
The `shipping.jsp` template (shown in Figure 4-1) allows the customer to select and input shipping details for the order. Shipping details include the shipping method (such as standard, second day air, and so on), shipping preference (all at once or as items become available) and any special shipping instructions the customer may want to specify.

If the customer is satisfied with the shipping details for the order, the customer can click the Continue button to continue to the next part of the checkout process. If the customer had forgotten something or wanted to do something else to their order, the customer can click the Back button instead.

Sample Browser View

Figure 4-1 shows an annotated version of the `shipping.jsp` template. A discription of the annotated regions follow the figure.

Figure 4-1 Annotated shipping.jsp Template



Shipping

How do you want this order shipped?
(●) Second Day Air
( ) Standard Shipping - 4 to 7 days

Ship all at once, or 'as available'?
(●) Please wait until the entire order is ready before shipping.
( ) Please ship items I ordered as they become available (you may incur additional shipping charges).

Special Instructions

Catalog data provided courtesy of TPN Register, 'where supply meets demand.'

Copyright © 1999-2001,
BEA Systems Inc.

The numbers in the following list refer to the numbered regions in the figure:

1. This region displays dynamic data related to the possible shipping methods. This is accomplished using a combination of WebLogic Server and Pipeline JSP tags that obtain and display each shipping method. Along with the other shipping details described in regions 2 and 3, the form posts the customer's selected shipping method.

2. This region, called the splitting preference, does not contain dynamic data. There are only two preferences: wait until the entire order is ready before shipping or ship the items as they become available. Along with the other shipping details described in regions 1 and 3, the form posts the customer’s selected splitting preference.

3. This region of the `shipping.jsp` template contains a simple input box, allowing the customer to enter any special instructions with regard to shipping. Again, no dynamic data is displayed in this region. Along with the other shipping details described in regions 1 and 2, the form posts any special instructions the customer specifies.

Note: For information on other elements in the `shipping.jsp` template, see "Common JSP Template Elements" on page 3-2.

Location in the WebLogic Portal Directory Structure

You can find the `shipping.jsp` template file at the following location, where `PORTAL_HOME` is the directory in which you installed WebLogic Portal:

`%PORTAL_HOME%applications\wlcsApp\wlcs\commerce\order\shipping.jsp` (Windows)
`$PORTAL_HOME/applications/wlcsApp/wlcs/commerce/order/shipping.jsp` (UNIX)

Tag Library Imports

The `shipping.jsp` template uses WebLogic Server and Pipeline JSP tags. Therefore, the template includes the following JSP tag libraries:

```
<%@ taglib uri="weblogic.tld" prefix="wl" %>
<%@ taglib uri="webflow.tld" prefix="webflow" %>
```

Note: For more information on the WebLogic Server JSP tags, see "JSP Tag Reference" in the *Guide to Building Personalized Applications*. For more information about the Pipeline JSP tags, see the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline*.

These files reside in the `lib` directory within `PORTAL_HOME`.

Java Package Imports

The `shipping.jsp` template uses Java classes in the following packages and therefore includes these import statements:

```
<%@ page import="java.util.*" %>
<%@ page import="java.text.*" %>
<%@ page import="examples.wlcs.sampleapp.shipping.*" %>


<%@ page import="com.beasys.commerce.webflow.HttpRequestConstants" %>
<%@ page import="com.beasys.commerce.webflow.PipelineSessionConstants" %>
```

Location in Default Webflow

The `shipping.jsp` template follows the page where the customer manages their shopping cart (`shoppingcart.jsp`), or any product catalog page where the customer clicks the View Cart button. The next page allows the customer to select a shipping address (`selectaddress.jsp`).

This template is in the sampleapp_order namespace.

Notes: If the customer has not yet logged into the site and clicks the Check Out button on the shopping cart page, the customer will be prompted to log in at the `login.jsp` template prior to loading the `shipping.jsp`. For more information about the `login.jsp` template, see the *Guide to Registering Customers and Managing Customer Services*.

For more information about the default Webflow, see "Overview of Managing Purchases and Processing Orders" on page 1-1.

Events

The `shipping.jsp` template presents a customer with two buttons, each of which is considered an event. Each event triggers a particular response in the default Webflow that allows customers to continue. While this response can be to load another JSP, it is usually the case that an input processor or Pipeline is invoked first. Table 4-1 provides information about these events and the business logic they invoke.

Table 4-1 shipping.jsp Events

Event	Webflow Response(s)
`button.back`	No business logic required. Loads `shoppingcart.jsp`.
`button.continue`	`ValidateShippingInfoIP.`

Dynamic Data Display

One purpose of the `shipping.jsp` template is to display information about the possible shipping methods for the order. This is accomplished on `shipping.jsp` using a combination of WebLogic Server JSP tags, Pipeline JSP tags and accessor methods/attributes.

First, the `getProperty` JSP tag retrieves the `SHIPPING_METHOD_LIST` attribute from the Pipeline session. Table 4-2 provides more detailed information about this attribute.

Table 4-2 shipping.jsp Dynamic Data Specification

Attribute	Type	Description
`PipelineSessionConstants.SHIPPING_METHOD_LIST`	List of `examples.wlcs.sampleapp.shipping.ShippingMethodValue`	The list of available shipping methods.

Listing 4-1 illustrates how this attribute is retrieved from the Pipeline session.

Listing 4-1 Retrieving the Shipping Method Attribute

```
<webflow:getProperty id="shippingMethodListObject"
property="<%=PipelineSessionConstants.SHIPPING_METHOD_LIST%>"
type="java.util.List" scope="session" namespace="sampleapp_main" />
```

Note: For more information on the `getProperty` JSP tag, see the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline*.

The data stored within this Pipeline session attribute is then accessed by using accessor methods/attributes within Java scriptlets. Table 4-3 provides more detailed information about these methods for `ShippingMethodValue`.

Table 4-3 ShippingMethodValue Accessor Methods/Attributes

Method/Attribute	Description
`description`	A description of the shipping method.
`identifier`	Key in the database for the shipping method.

Listing 4-2 illustrates how these accessor methods/attributes are used within Java scriptlets.

Listing 4-2 Using Accessor Methods Within shipping.jsp Java Scriptlets

```
<wl:repeat set="<%=shippingMethodList%>" id="shippingMethodValue"
type="ShippingMethodValue" count="100">
    <tr>
      <td width="1%" valign="top">
        <!-- put up a button for each of item -->
        <%
          if((previousShippingMethodValue != null &&
shippingMethodValue.identifier.equals(previousShippingMethodValue.identifier)) ||
             (previousShippingMethodValue == null && defaultShippingMethod == true) )
          {
            shippingMethodCheckedStatus = "CHECKED";
            defaultShippingMethod = false;
          }
          else
          {
            shippingMethodCheckedStatus = "";
          }

        %>
        <input <%=shippingMethodCheckedStatus%> type="radio"
name="<%=HttpRequestConstants.SHIPPING_METHOD%>"
value="<%=shippingMethodValue.identifier%>">
      </td>
      <td valign="top">
        <div class="tabletext"><%=shippingMethodValue.description%></div>
      </td>
    </tr>
</wl:repeat>
```

Note: For more information on the WebLogic Server JSP tags, see “JSP Tag Reference” in the *Guide to Building Personalized Applications*.

Form Field Specification

Other purposes of the `shipping.jsp` template are to collect information from the customer and to pass hidden information to the Webflow. The form fields used in the `shipping.jsp` template, and a description for each of these form fields, are listed in Table 4-4.

Table 4-4 shipping.jsp Form Fields

Parameter Name	Type	Description
`“event”`	Hidden	Indicates whether an event has been triggered. It is used by the Webflow to determine what happens next.
`“origin”`	Hidden	The name of the current page (`shipping.jsp`), used by the Webflow.
`HttpRequestConstants.SHIPPING_METHOD`	Radio button	Identifies the shipping method the customer selects.
`HttpRequestConstants.SPLITTING_PREFERENCE_CODE`	Radio button	String representing the splitting preference the customer selects.
`HttpRequestConstants.SPLITTING_PREFERENCE_SPLIT_LOCAL`	Hidden	Choice for letting customers choose to ship items separately as they become available.
`HttpRequestConstants.SPLITTING_PREFERENCE_NO_SPLIT_LOCAL`	Hidden	Choice for letting customers choose to ship items all at once when they are all available.
`HttpRequestConstants.SPECIAL_INSTRUCTIONS`	Textbox	Any special instructions the customer specifies.

Note: Parameters that are literals in the JSP code are shown in quotes, while non-literals will require JSP scriptlet syntax (such as `<%= HttpRequestConstants.SPLITTING_PREFERENCE_CODE %>`) for use in the JSP.

selectaddress.jsp Template

The `selectaddress.jsp` template (shown in Figure 4-2) displays a list of shipping addresses that have previously been associated with the customer. If the customer clicks the Use button associated with a particular address, that address will be used as the shipping address and the customer will continue to the next part of the checkout process.

If the customer wants to delete an address that is shown, the customer can click the Delete button associated with that address. To add a new shipping address, the customer can click the Add Address button. To go back to the previous page, the customer can click the Back button instead.

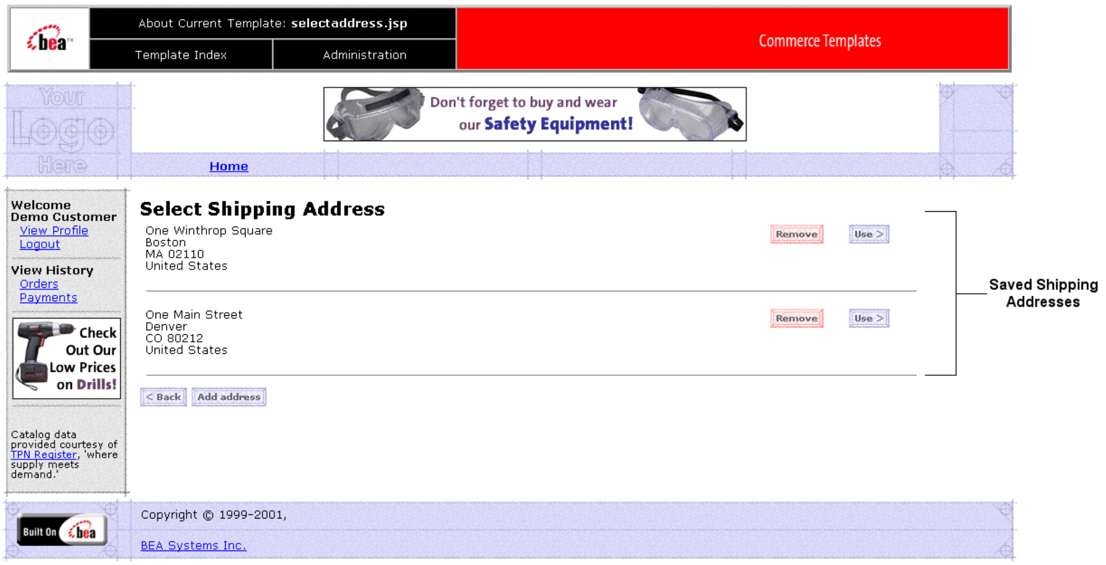
Sample Browser View

Figure 4-2 shows an annotated version of the `selectaddress.jsp` template. The Select Shipping Address region contains dynamically displayed data of the customer's saved shipping addresses. This is accomplished using a combination of WebLogic Server and WebLogic Portal JSP tags that obtain and display the addresses. Posts to the form can indicate use of a listed address or deletion of a listed address.

Notes: The customer can also initiate entry of a new shipping address from the `selectaddress.jsp` template. For more information about the `addaddress.jsp` template, see "addaddress.jsp Template" on page 4-17.

For information on other elements in the `selectaddress.jsp` template, see "Common JSP Template Elements" on page 3-2.

Figure 4-2 Annotated selectaddress.jsp Template



Select Shipping Address

One Winthrop Square
Boston
MA 02110
United States

One Main Street
Denver
CO 80212
United States

Location in the WebLogic Portal Directory Structure

You can find the `selectaddress.jsp` template file at the following location, where `PORTAL_HOME` is the directory in which you installed WebLogic Personalization Server:

`%PORTAL_HOME%\applications\wlcsApp\wlcs\commerce\order\`
`selectaddress.jsp` (Windows)
`$PORTAL_HOME/applications/wlcsApp/wlcs/commerce/order/`
`selectaddress.jsp` (UNIX)

Tag Library Imports

The `selectaddress.jsp` template uses existing WebLogic Server and the WebLogic Portal's User Management and Personalization JSP tags. It also uses Pipeline JSP tags. Therefore, the template includes the following JSP tag libraries:

```
<%@ taglib uri="webflow.tld" prefix="webflow" %>
<%@ taglib uri="um.tld" prefix="um" %>
<%@ taglib uri="es.tld" prefix="es" %>
```

Note: For more information on the WebLogic Server JSP tags or the WebLogic Portal JSP tags, see JSP Tag Reference” in the *Guide to Building Personalized Applications*. For more information about the Pipeline JSP tags, see the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline*.

These files reside in the `lib` directory within `PORTAL_HOME`.

Java Package Imports

The `selectaddress.jsp` template uses Java classes in the following packages and therefore includes these import statements:

```
<%@ page import="java.util.*" %>
<%@ page import="java.text.*" %>
<%@ page import="com.beasys.commerce.axiom.contact.*" %>
<%@ page import="examples.wlcs.sampleapp.shipping.*" %>
<%@ page import="examples.wlcs.sampleapp.customer.*" %>
<%@ page import="com.beasys.commerce.webflow.HttpRequestConstants" %>
```

Location in Default Webflow

The page prior to the `selectaddress.jsp` template in the default Webflow is either the shipping details page (`shipping.jsp`) or the page where the customer enters a new shipping address (`addaddress.jsp`).

If the customer deletes an existing shipping address, the `selectaddress.jsp` is reloaded after the appropriate input processor and/or Pipeline has executed. If the customer is satisfied with selecting an address from the list of choices, they proceed to the payment information page (`payment.jsp`).

This template is in the sampleapp_order namespace.

Note: For more information about the default Webflow, see “Overview of Managing Purchases and Processing Orders” on page 1-1.

Events

The `selectaddress.jsp` template presents a customer with several buttons, each of which is considered an event. These events trigger a particular response in the default Webflow that allows customers to continue. While this response can be to load another JSP, it is usually the case that an input processor or Pipeline is invoked first. Table 4-5 provides information about these events and the business logic they invoke.

Table 4-5 selectaddress.jsp Events

Event	Web Flow Response(s)
`button.back`	No business logic required. Loads `shipping.jsp`.
`button.addNewShippingAddress`	No business logic required. Loads `addaddress.jsp`.
`button.deleteShippingAddress`	`UpdateAddressKeyIP` `DeleteShippingAddress`
`button.useShippingAddress`	`UpdateShippingAddressIP` `TaxVerifyShippingAddress` `CalculateShippingCost` `TaxCalculateLineLevel`

Table 4-6 briefly describes each of the Pipelines from Table 4-5. For more information about individual Pipeline components, see "Pipeline Components" on page 4-26.

Table 4-6 Select Shipping Address Pipelines

Pipeline	Description
`TaxVerifyShippingAddress`	Contains `TaxVerifyShippingAddressPC` and is not transactional.
`CalculateShippingCost`	Contains `CalculateShippingCostPC` and is not transactional.
`TaxCalculateLineLevel`	Contains `TaxCalculateLineLevelPC` and is not transactional.
`DeleteShippingAddress`	Contains `DeleteShippingAddressPC` and is not transactional.

Dynamic Data Display

One purpose of the `selectaddress.jsp` template is to display the shipping addresses a customer previously entered. This is accomplished on `selectaddress.jsp` using two of the WebLogic Portal's User Management JSP tags.

First, the `getProfile` JSP tag is used to set the customer profile (context) for which the shipping addresses should be retrieved, as shown in Listing 4-3.

Listing 4-3 Setting the Customer Context

```
<um:getProfile
    profileKey="<%=request.getRemoteUser()%>"
    profileType="WLCS_Customer" />
```

Next, the `getProperty` JSP tag is used to retrieve a cached copy of the possible shipping addresses for the customer from the database, as shown in Listing 4-4.

Listing 4-4 Retrieving the ShippingAddressMap for the Customer

```
<um:getProperty propertySet="CustomerProperties"
propertyName="shippingAddressMap" id="shippingAddressMap" />
```

You can now iterate through the shipping addresses contained within the `shippingAddressMap`, as shown in Listing 4-5.

Listing 4-5 Iterating Through the Shipping Addresses

```
<%
Iterator iterator =((Map)shippingAddressMap).keySet().iterator();
  while(iterator.hasNext())
  {
    String addressKey = (String)iterator.next();
   Address shippingAddress = (Address)((Map)shippingAddressMap).get(addressKey);
%>
```

Note: For more information on the WebLogic Portal's JSP tags, see "JSP Tag Reference" in the *Guide to Building Personalized Applications*.

Lastly, the data contained within `shippingAddress` is accessed by using accessor methods/attributes within Java scriptlets. Table 4-7 provides more detailed information about these methods for `Address`.

Table 4-7 Address Accessor Methods/Attributes

Method/Attribute	Description
`getStreet1()`	The first line of the customer's street address.
`getStreet2()`	The second line of the customer's street address.
`getCity()`	The city in the customer's address.
`getState()`	The state in the customer's address.
`getPostalCode()`	The zip/postal code in the customer's address.
`getCountry()`	The country in the customer's address.

Listing 4-6 illustrates how these accessor methods/attributes are used within Java scriptlets.

Listing 4-6 Using Accessor Methods Within selectaddress.jsp Java Scriptlets

```
  <%
  Iterator iterator =((Map)shippingAddressMap).keySet().iterator();
  while(iterator.hasNext())
  {
    String addressKey = (String)iterator.next();
    Address shippingAddress = (Address)((Map)shippingAddressMap).get(addressKey);
  %>

  <table width="90%" border="0" cellpadding="6" cellspacing="0">

    <tr>
      <td align="left" valign="top" width="40%" nowrap>

      <p><%= shippingAddress.getStreet1() %><br>

           <% if( shippingAddress.getStreet2().length() != 0) {%>
           <%= shippingAddress.getStreet2() %><br>
           <% } %>
           <%= shippingAddress.getCity() %><br>
           <%= shippingAddress.getState() %> <%= shippingAddress.getPostalCode() %><br>
           <%= shippingAddress.getCountry() %>
  </td>

  <td align="left" valign="top" width="5%" >
<%
    String extraParams = HttpRequestConstants.ADDRESS_KEY + "=" + addressKey;
%>
    <div class="commentary">
   <a href="<webflow:createWebflowURL event="button.deleteShippingAddress"
httpsInd="calculate" namespace="sampleapp_order" extraParams="<%= extraParams %>" />"><img
src="<webflow:createResourceURL resource="/commerce/images/btn_delete.gif" />"
border="0"></a>

    </div>

  </td>
  <td align="left" valign="top" width="5%" >

    <div class="commentary">
```

```
      <a href="<webflow:createWebflowURL event="button.useShippingAddress"
httpsInd="calculate" namespace="sampleapp_order" extraParams="<%= extraParams %>" />"><img
src="<webflow:createResourceURL resource="/commerce/images/btn_use.gif" />" border="0"></a>
    </div></td>
    </tr>
    <tr>
  <td colspan="3">
  <hr size="1">
  </td>
  </tr>
</table>

 <%
 }
%>
```

Form Field Specification

The `selectaddress.jsp` template does not make use of any form fields.

addaddress.jsp Template

The `addaddress.jsp` template (shown in Figure 4-3) collects information about a new shipping address from the customer. This information includes two lines of a street address (one required), a city, a state, a zip code, and a country (all required).

When the customer clicks the Save button, the shipping address entered on this page is added to the list of addresses from which customers can select for this and future orders (`selectaddress.jsp`). Otherwise, the customer can click the Back button to return to the previous page.

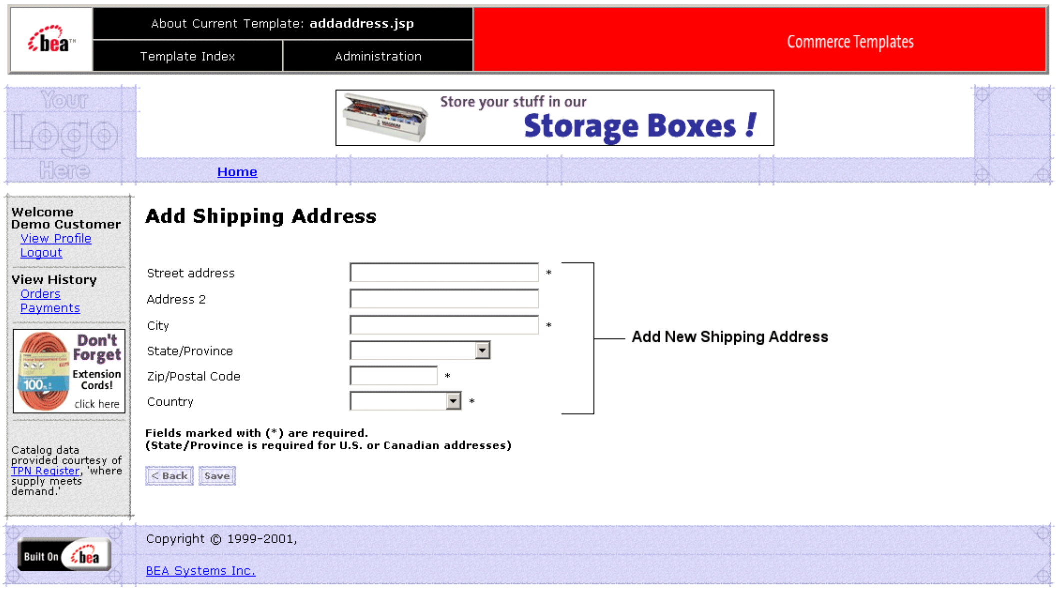
Sample Browser View

Figure 4-3 shows an annotated version of the `addaddress.jsp` template. The Add Shipping Address region provides the customer with a series of form fields for entering a new shipping address. Required fields are indicated by an asterisk (*). This region utilizes the `states.jsp` and `countries.jsp` template files. The “include” calls in `addaddress.jsp` are:

```
<%@ include file="/commerce/includes/countries.inc" %>
<%@ include file="/commerce/includes/footer.inc" %>
<%@ include file="/commerce/includes/stylesheet.inc" %>
<%@ include file="/commerce/includes/admin.inc" %>
<%@ include file="/commerce/includes/header.inc" %>
<%@ include file="/commerce/includes/leftside.inc" %>
<%@ include file="/commerce/includes/states.inc" %>
```

Note: For information on other elements in the `addaddress.jsp` template, see “Common JSP Template Elements” on page 3-2.

Figure 4-3 Annotated addaddress.jsp Template



Add Shipping Address

Street address	*
Address 2	
City	*
State/Province	
Zip/Postal Code	*
Country	*

Fields marked with (*) are required.
(State/Province is required for U.S. or Canadian addresses)

Catalog data provided courtesy of TPN Register, 'where supply meets demand.'

Copyright © 1999-2001, BEA Systems Inc.

Location in the WebLogic Portal Directory Structure

You can find the `addaddress.jsp` template file at the following location, where `PORTAL_HOME` is the directory in which you installed WebLogic Portal:

`%PORTAL_HOME%\applications\wlcsApp\wlcs\commerce\order\`
`addaddress.jsp` (Windows)
`$PORTAL_HOME/applications/wlcsApp/wlcs/commerce/order/`
`addaddress.jsp` (UNIX)

Tag Library Imports

The `addaddress.jsp` template uses Webflow and Pipeline JSP tags. Therefore, the template includes the following JSP tag libraries:

```
<%@ taglib uri="webflow.tld" prefix="webflow" %>
```

Note: For more information on the Webflow and Pipeline JSP tags, see the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline*.

These files reside in the `lib` directory within `PORTAL_HOME`.

Java Package Imports

The `addaddress.jsp` template uses Java classes in the following packages and therefore includes these import statements:

```
<%@ page import="javax.servlet.*" %>
<%@ page import="javax.servlet.http.*" %>
<%@ page import="com.beasys.commerce.webflow.tags.*" %>
<%@ page import="com.beasys.commerce.axiom.contact.*" %>
<%@ page import="examples.wlcs.sampleapp.customer.*" %>
<%@ page import="com.beasys.commerce.webflow.HttpRequestConstants" %>
```

Location in Default Webflow

The `addaddress.jsp` template follows the page where the customer selects from a list of possible shipping addresses (`selectaddress.jsp`). Once the customer saves the new address, the customer is returned to the `selectaddress.jsp` template.

Note: For more information about the default Webflow, see "Overview of Managing Purchases and Processing Orders" on page 1-1.

Events

The `addaddress.jsp` template presents a customer with two buttons, each of which is considered an event. These events trigger a particular response in the default Webflow that allows customers to continue. While this response can be to load another JSP, it is usually the case that an input processor or Pipeline is invoked first. Table 4-8 provides information about these events and the business logic they invoke.

Table 4-8 addaddress.jsp Events

Event	Webflow Response(s)
`button.back`	No business logic required. Loads `selectaddress.jsp`.
`button.addNewShippingAddress`	`ValidateAddressIP` `AddShippingAddress`

Table 4-9 briefly describes each of the Pipelines from Table 4-8. For more information about individual Pipeline components, see "Pipeline Components" on page 4-26.

Table 4-9 Add Shipping Address Pipelines

Pipeline	Description
`AddShippingAddress`	Contains `AddShippingAddressPC` and is not transactional.

Dynamic Data Display

No dynamic data is presented on the `addaddress.jsp` template. However, the `addaddress.jsp` template does make use of code similar to that found in the `newaddresstemplate.jsp` template. Namely, it uses the same code to indicate when customers enter incorrect input or fail to provide information for a required field. For more information about the `newaddresstemplate.jsp` template, see "About the Included newaddresstemplate.jsp Template" in the *Guide to Registering Customers and Managing Customer Services*.

Form Field Specification

The purpose of the `addaddress.jsp` template is to allow customers to enter a new shipping address using various HTML form fields. It is also used to pass needed information to the Webflow.

The form fields used in the `addaddress.jsp` template, and a description for each of these form fields are listed in Table 4-10.

Table 4-10 addaddress.jsp Form Fields

Parameter Name	Type	Description
`“event”`	Hidden	Indicates which event has been triggered. It is used by the Webflow to determine what happens next.
`“origin”`	Hidden	The name of the current page (`addaddress.jsp`), used by the Webflow.
`HttpRequestConstants.CUSTOMER_SHIPPING_ADDRESS1`	Textbox	The first line of the shipping street address.
`HttpRequestConstants.CUSTOMER_SHIPPING_ADDRESS2`	Textbox	The second line of the shipping street address.
`HttpRequestConstants.CUSTOMER_SHIPPING_CITY`	Textbox	The city in the shipping address.
`HttpRequestConstants.CUSTOMER_SHIPPING_STATE`	Textbox	The state in the shipping address.
`HttpRequestConstants.CUSTOMER_SHIPPING_ZIPCODE`	Textbox	The zip/postal code in the shipping address.
`HttpRequestConstants.CUSTOMER_SHIPPING_COUNTRY`	Textbox	The country in the shipping address.

Note: Parameters that are literals in the JSP code are shown in quotes, while non-literals will require JSP scriptlet syntax (such as `<%= HttpRequestConstants.CUSTOMER_SHIPPING_CITY %>`) for use in the JSP.

Input Processors

This section provides a brief description of each input processor associated with the Shipping Services JSP template(s).

InitShippingMethodListIP

Class Name	`examples.wlcs.sampleapp.shipping.webflow.InitShippingMethodListIP`
Description	Obtains a list of all shipping methods from the database and populates the Pipeline session with a list of `ShippingMethodValue` objects. This list is cached, so this input processor does not continuously access the database. Accessing the list multiple times within one session has no additional effect.
Required `HTTPServletRequest` Parameters	None
Required Pipeline Session Attributes	`PipelineSessionConstants.SHOPPING_CART`
Updated Pipeline Session Attributes	`PipelineSessionConstants.SHIPPING_METHOD_LIST`
Removed Pipeline Session Attributes	None
Validation	None
Exceptions	`ProcessingException`, thrown if no shopping cart exits.

UpdateShippingAddressIP

Class Name	`examples.wlcs.sampleapp.shipping.webflow.UpdateShippingAddressIP`
Description	Updates the shipping address attribute in the Pipeline session based on the address the customer selects.
Required `HTTPServletRequest` Parameters	`HTTPRequestConstants.ADDRESS_KEY`
Required Pipeline Session Attributes	None
Updated Pipeline Session Attributes	`PipelineSessionConstants.SHIPPING_ADDRESS`
Removed Pipeline Session Attributes	None
Validation	None
Exceptions	None

ValidateAddressIP

Class Name	`examples.wlcs.sampleapp.shipping.webflow.` `ValidateAddressIP`
Description	Validates the address and places it in the Pipeline session.

Required `HTTPServletRequest` Parameters	`HttpRequestConstants.CUSTOMER_SHIPPING_ADDRESS1`
	`HttpRequestConstants.CUSTOMER_SHIPPING_ADDRESS2`
	`HttpRequestConstants.CUSTOMER_SHIPPING_CITY`
	`HttpRequestConstants.CUSTOMER_SHIPPING_STATE`
	`HttpRequestConstants.CUSTOMER_SHIPPING_ZIPCODE`
	`HttpRequestConstants.CUSTOMER_SHIPPING_COUNTRY`

Required Pipeline Session Attributes	None
Updated Pipeline Session Attributes	`PipelineSessionConstants.ADDRESS`
Removed Pipeline Session Attributes	None
Validation	Verifies that the required fields contain values.
Exceptions	`ProcessingException`, thrown if the required request parameters or required Pipeline session attributes are not available.

ValidateShippingInfoIP

Class Name	`examples.wlcs.sampleapp.shipping.webflow.` `ValidateShippingInfoIP`
Description	Places the shipping method, splitting preference, and special instructions into the Pipeline session.

Required `HTTPServletRequest` Parameters	`HttpRequestConstants.SHIPPING_METHOD`
	`HttpRequestConstants.SPLITTING_PREFERENCE`
	`HttpRequestConstants.SPECIAL_INSTRUCTIONS`
	`HttpRequestConstants.SPLITTING_PREFERENCE_CODE`

Required Pipeline Session Attributes	None

Updated Pipeline Session Attributes	`PipelineSessionConstants.SHIPPING_METHOD`
	`PipelineSessionConstants.SPLITTING_PREFERENCE`
	`PipelineSessionConstants.SPECIAL_INSTRUCTIONS`
	`PipelineSessionConstants.SPLITTING_PREFERENCE_CODE`

Removed Pipeline Session Attributes	None
Validation	Verifies that the required fields contain values.
Exceptions	`ProcessingException`, thrown if the required request parameters or required Pipeline session attributes are not available.

Pipeline Components

This section provides a brief description of each Pipeline component associated with the Shipping Services JSP template(s).

Notes: For information about the `TaxVerifyShippingAddressPC` and `TaxCalculateLineLevelPC` Pipeline components, see Chapter 5, "Taxation Services."

Some Pipeline components extend other, base Pipeline components. For more information on the base classes, see the *Javadoc*.

AddShippingAddressPC

Class Name	`examples.wlcs.sampleapp.shipping.pipeline.AddShippingAddressPC`
Description	Adds the address to the list of customer shipping addresses stored for the customer.
Required Pipeline Session Attributes	`PipelineSessionConstants.ADDRESS` `PipelineSessionConstants.ADDRESS_KEY`
Updated Pipeline Session Attributes	None
Removed Pipeline Session Attributes	None
Type	Java object
JNDI Name	None
Exceptions	`PipelineException`, thrown when the Pipeline component cannot update the address information in the database.

CalculateShippingPC

Class Name	`examples.wlcs.sampleapp.shipping.pipeline.CalculateShippingPC`
Description	Calculates the per-line cost of shipping for each line in the shopping cart. The implementation only uses a simple per-shipping method cost calculation. When integrating with a shipping provider, this Pipeline component should be rewritten to perform more specific cost calculations.
Required Pipeline Session Attributes	`PipelineSessionConstants.SHOPPING_CART` `PipelineSessionConstants.SHIPPING_METHOD`
Updated Pipeline Session Attributes	`PipelineSessionConstants.SHOPPING_CART`
Removed Pipeline Session Attributes	None
Type	Java object
JNDI Name	None
Exceptions	`PipelineException`, thrown if the required request parameters or required Pipeline session attributes are not available.

DeleteShippingAddressPC

Class Name	`examples.wlcs.sampleapp.shipping.pipeline.DeleteShippingAddressPC`
Description	Uses the address key in the Pipeline session to locate the correct customer shipping address, then removes it from the list.
Required Pipeline Session Attributes	`PipelineSessionConstants.ADDRESS_KEY`
Updated Pipeline Session Attributes	None
Removed Pipeline Session Attributes	None
Type	Java object
JNDI Name	None
Exceptions	`PipelineException`, thrown when the Pipeline component cannot update the shipping address information in the database.

CHAPTER

5 Taxation Services

WebLogic Portal includes a flexible taxation service that lets you connect to a third-party tax calculation product for determine the accurate tax rates imposed on the sale or use of each item at the state, country, city, and district levels. This topic describes the Taxation Service in detail.

This topic includes the following sections:

Introduction to Web Services

Web services are stand-alone software components, available over the Internet, that you can bind into your enterprise applications for immediate functionality. Web services are self-describing, self-contained, modular applications that can be mixed and matched with other Web services. You do not need to understand the internal workings of the Web services. You only need to know how to connect your enterprise applications to these services.

Web services can be visible or invisible to your site visitors. For example, a visible Web service can be a stock ticker that appears in a portlet on your site; or, to use two examples of invisible Web services, used in e-commerce (and shipped with WebLogic Portal), Web services that facilitate the handling of online payment and taxation for purchases. This chapter and the next cover these two payment and taxation Web services.

The main reason that Web services are plug-and-play is because they use standard, proven Internet technologies such as HTTP, HTML, and XML. The characteristics of Web services include the following:

- Accessibility via the Web
- Exposure of an XML interface
- Ability to be located via a registry
- Use of XML messages over standard Web protocols
- Support of loosely coupled connections between systems

Ultimately, Web services allow companies and individuals to rapidly and economically make their services available worldwide.

The following sections describe the core standards of Web services. These standards are used in the WebLogic Portal Payment and Taxation services.

Simple Object Access Protocol (SOAP)

Simple Object Access Protocol (SOAP) is an XML-based standard for sending and receiving messages over the Internet, using transports like HTTP. A service request is embodied in a SOAP message and HTTP posted to a Service Provider. The response is then synchronously returned via the same HTTP session, embodied in a SOAP response message.

For more information, go to www.w3.org/TR/SOAP/.

Web Services Description Language (WSDL)

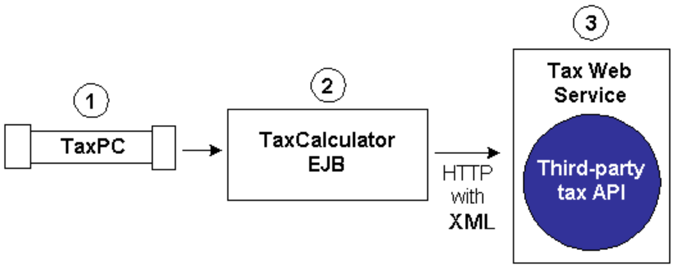
Web Services Description Language (WSDL) is an XML-based standard that describes the services a business offers and provides a way for individuals and other businesses to access those services electronically. In more technical terms, WSDL describes the programmatic interface of a Web service, allowing companies who use a Web service to craft the program statements that invoke the Web service.

WSDL is the cornerstone of the Universal Description, Discovery, and Integration (UDDI) initiative. UDDI is an XML-based registry for businesses worldwide that enables businesses to list themselves and their services on the Internet. WSDL is the language used to do this.

How the Taxation Service Works

Figure 5-1 shows the basic architecture of WebLogic Portal's taxation service. The key component of this architecture is the Tax Web Service, which is the connection point between WebLogic Portal and any third-party tax calculation product.

Figure 5-1 taxation Service Architecture



Following is a description of each piece of the architecture.

1	The TaxPC Pipeline component controls the taxation sequence in the Webflow and instantiates the TaxCalculator EJB. For information on Webflow, see the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline*.
2	The TaxCalculator EJB handles the business logic for the tax calculation. The EJB converts its taxation service calls to XML for transporting SOAP messages to the Tax Web service.
3	The TaxCalculator EJB in WebLogic Portal sends SOAP messages to the Web service, where those SOAP messages are converted to the language of the third-party product's API. The third-party taxation service applies taxes to orders and sends calculated tax amounts or exceptions back to the enterprise application. **Caution:** The default Tax Web service that ships with WebLogic Portal automatically applies a 5% tax to an order. This default application of taxes is not designed for production use. You must integrate with your third-party vendor's tax service to calculate taxes properly.

There are two keys to connecting an enterprise application with a Tax Web service:

- Sending the proper SOAP messages from the TaxCalculator EJB to the Tax Web Service

- In the Tax Web Service, translating the SOAP messages into the language of the third-party product's API

For information on connecting an enterprise application to a Web service, see "Integrating with a Taxation Service" on page 5-15.

JavaServer Pages (JSPs)

The Taxation Services consist of one JavaServer Page (JSP) that you can use as is, or customize to meet your business requirements. This section describes this page in detail.

Note: For a description of the complete set of JSPs used in the WebLogic Portal Web application and a listing of their locations in the directory structure, see the *E-Commerce JSP Template Summary*.

selecttaxaddress.jsp Template

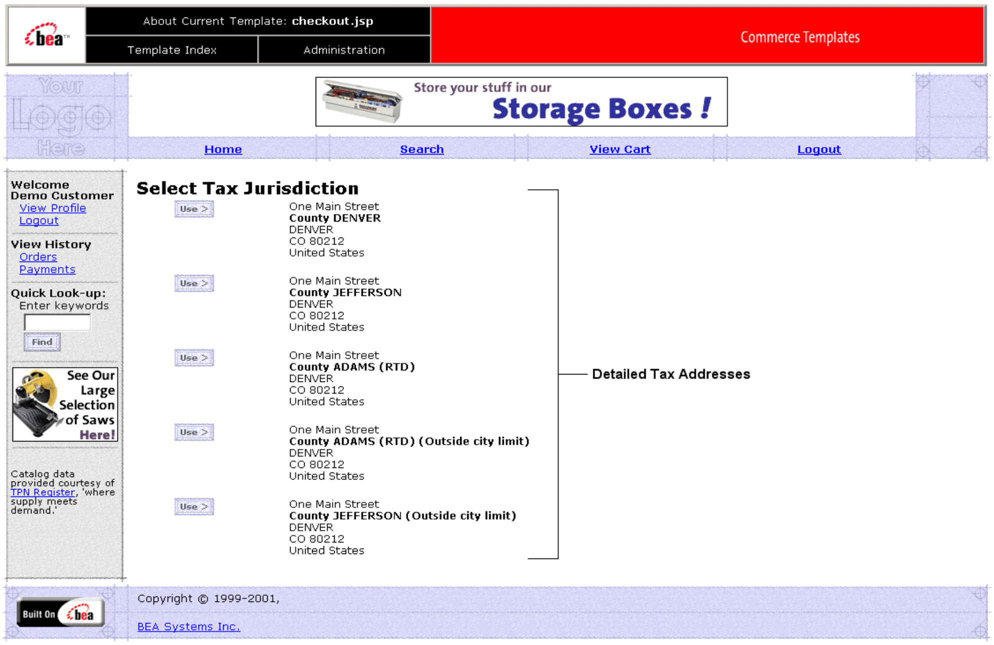
In cases where a customer provides a shipping address that does not resolve to a unique GeoCode (a Tax Web service code that is used to determine taxes based on jurisdiction), the `selecttaxaddress.jsp` template (shown in Figure 5-2) allows the customer to select from a list of more specific shipping addresses.

Sample Browser View

Figure 5-2 shows the `selecttaxaddress.jsp` template. The Select Tax Jurisdiction region uses a combination of WebLogic Server and Pipeline JSP tags to obtain and display a list of more detailed addresses, from which the customer can select.

Note: For information on other elements in the `selecttaxaddress.jsp` template, see "Common JSP Template Elements" on page 3-2.

Figure 5-2 Annotated selecttaxaddress.jsp Template



Catalog data provided courtesy of TPN Register, 'where supply meets demand.'

Select Tax Jurisdiction

Use >	One Main Street **County DENVER** DENVER CO 80212 United States
Use >	One Main Street **County JEFFERSON** DENVER CO 80212 United States
Use >	One Main Street **County ADAMS (RTD)** DENVER CO 80212 United States
Use >	One Main Street **County ADAMS (RTD) (Outside city limit)** DENVER CO 80212 United States
Use >	One Main Street **County JEFFERSON (Outside city limit)** DENVER CO 80212 United States

Detailed Tax Addresses

Copyright © 1999-2001,

BEA Systems Inc.

Location in the WebLogic Portal Directory Structure

You can find the `selecttaxaddress.jsp` template file at the following location, where `PORTAL_HOME` is the directory in which you installed WebLogic Portal:

`%PORTAL_HOME%\applications\wlcsApp\wlcs\commerce\order\`
`selecttaxaddress.jsp` (Windows)
`$PORTAL_HOME/applications/wlcsApp/wlcs/commerce/order/`
`selecttaxaddress.jsp` (UNIX)

Tag Library Imports

The `selecttaxaddress.jsp` template uses existing WebLogic Server and Pipeline JSP tags. Therefore, the template includes the following JSP tag libraries:

```
<%@ taglib uri="es.tld" prefix="es" %>
<%@ taglib uri="weblogic.tld" prefix="wl" %>
<%@ taglib uri="webflow.tld" prefix="webflow" %>
```

Note: For more information on the WebLogic Server JSP tags, see “JSP Tag Reference” in the *Guide to Building Personalized Applications*. For more information about the Pipeline JSP tags, see the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline*.

These files reside in the `lib` directory within `PORTAL_HOME`.

Java Package Imports

The `selecttaxaddress.jsp` template uses Java classes in the following packages and therefore includes these import statements:

```
<%@ page import="java.util.*" %>
<%@ page import="java.text.*" %>
<%@ page import="com.beasys.commerce.axiom.contact.*" %>
<%@ page import="com.beasys.commerce.ebusiness.shipping.*" %>

<%@ page import="com.beasys.commerce.webflow.HttpRequestConstants" %>
<%@ page import="com.beasys.commerce.webflow.PipelineSessionConstants" %>
<%@ page import="com.bea.p13n.appflow.webflow.WebflowJSPHelper" %>
<%@ page import="com.bea.p13n.appflow.webflow.SessionManagerFactory" %>
<%@ page import="com.bea.p13n.appflow.common.PipelineSession" %>
<%@ page import="com.bea.p13n.appflow.common.internal.AppflowConstants" %>
```

Location in Default Webflow

Note: The `selecttaxaddress.jsp` template is only displayed if the customer provides a shipping address that is not specific enough. Otherwise, it is bypassed.

The page prior to the `selecttaxaddress.jsp` template in the default Webflow is the page where the customer selects a shipping address (`selectaddress.jsp`). After the customer has selected an address from the list of choices presented on `selecttaxaddress.jsp`, they proceed to the payment information page (`payment.jsp`).

The template is in the sampleapp_order namespace.

Note: For more information about the default Webflow, see “Overview of Managing Purchases and Processing Orders” on page 1-1.

Included JSP Templates

The following JSP templates are included in the `selecttaxaddress.jsp` template:

- `header.inc`, which creates the top banner.
- `admin.inc`, which is used on all pages and presents the top red-and-black banner with links to the main WLCS Administration screen, to this template index, and to a *.jsp.html file for the current template. You should remove this from the JSP when you deploy it.
- `leftside.inc`, which presents quick look-up and a promotional ad; for authenticated users, it also presents a personalized message to the user, customer profile link, order history link, and payment history link.
- `footer.inc`, which creates a horizontal footer at the bottom of the page.

Events

The `selecttaxaddress.jsp` template presents a customer with two buttons, each of which is considered an event. These events trigger a particular response in the default Webflow that allows customers to continue. While this response can be to load another JSP, it is usually the case that an input processor or Pipeline is invoked first. Table 5-1 provides information about these events and the business logic they invoke.

Table 5-1 selecttaxaddress.jsp Events

Event	Webflow Response(s)
`button.checkout`	`UpdateTaxShippingAddressIP`

Dynamic Data Display

The only purpose of the `selecttaxaddress.jsp` template is to display variations on a shipping address that the customer has already entered. This is accomplished on `selecttaxaddress.jsp` using a combination of WebLogic Server and Pipeline JSP tags, and accessor methods/attributes.

First, the `getProperty` JSP tag retrieves the `AVS_SHIPPING_ADDRESSES` attribute from the Pipeline session. Table 5-2 shows more detailed information about this attribute.

Table 5-2 selecttaxaddress.jsp Pipeline Session Attributes

Attribute	Type	Description
`PipelineSessionConstants.` `AVS_SHIPPING_ADDRESSES`	List of `com.beasys.commerce.axiom.contact.Address`	List of the possibilities for the more detailed shipping address.

Listing 5-1 illustrates how this attribute is retrieved from the Pipeline session.

Listing 5-1 Retrieving the Address Selection Attribute

```
<webflow:getProperty id="addressesObject"
property="<%=PipelineSessionConstants.AVS_SHIPPING_ADDRESSES%>"
type="java.lang.Object" scope="session" namespace="sampleapp_main" />
```

Note: For more information on the `getProperty` JSP tag, see the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline*.

The data stored within this attribute is then accessed by using accessor methods/attributes within Java scriptlets. Table 5-3 provides more detailed information on these methods/attributes for `Address`.

Table 5-3 Address Accessor Methods/Attributes

Method/Attribute	Description
`getStreet1()`	The first line of the street in the shipping address.
`getStreet2()`	The second line of the street in the shipping address.
`getCity()`	The city in the shipping address.
`getCounty()`	The county in the shipping address.
`getState()`	The state in the shipping address.
`getPostalCode()`	The zip/postal code in the shipping address.
`getCountry()`	The country in the shipping address.

Since there are multiple addresses, you must also use the WebLogic Server JSP tag to iterate through each of the addresses, as shown in Listing 5-2.

Listing 5-2 Using <wl> and Accessor Methods in selecttaxaddress.jsp

```
<wl:repeat set="<%=addressesObject%>" id="address" type="Address" count="100">

<%
  String extraParams = HttpRequestConstants.TAX_SHIPPING_ADDRESS + "=" + address.getGeoCode();
%>

  <table width="90%" border="0" cellpadding="3" cellspacing="0">
    <tr>
      <td align="left" valign="top" width="15%">
        <a href="<webflow:createWebflowURL event="button.checkout" httpsInd="calculate"
         namespace="sampleapp_order" extraParams="<%= extraParams %>" />">
       <img border=0 src="<webflow:createResourceURL resource="/commerce/images/btn_use.gif"
/>">
       </a>

      <td align="left" valign="top">
        <div class="tabletext"><%= address.getStreet1() %><br>
      <% if(address.getStreet2().length() != 0) { %>
      <%=address.getStreet2()%><br>
      <% } %>
      <b>County  <%= address.getCounty() %></b><br>
      <%= address.getCity() %><br>
      <%= address.getState() %> <%= address.getPostalCode() %><br>
      <%= address.getCountry() %><br>
      &nbsp;</div>

      </td>
    </tr>
  </table>
</wl:repeat>
```

Note: For more information on the WebLogic Server JSP tags, see “JSP Tag Reference” in the *Guide to Building Personalized Applications*.

Form Field Specification

Besides allowing a customer to select a more detailed shipping address, the `selecttaxaddress.jsp` template also passes hidden information to the Webflow. The form fields used in the `selecttaxaddress.jsp` template, and a description for each of these form fields are listed in Table 5-4.

Table 5-4 selectataxddress.jsp Form Fields

Parameter Name	Type	Description
`“event”`	Hidden	Indicates which event has been triggered. It is used by the Webflow to determine what happens next.
`“origin”`	Hidden	The name of the current page (`selecttaxaddress.jsp`), used by the Webflow.
`PipelineSessionConstants.` `TAX_SHIPPING_ADDRESS`	Hidden	Identifies the more specific address selected by the customer.

Note: Parameters that are literals in the JSP code are shown in quotes, while non-literals will require JSP scriptlet syntax (such as `<%= PipelineSessionConstants.TAX_SHIPPING_ADDRESS %>`) for use in the JSP.

Input Processors

This section provides a brief description of each input processor associated with the Taxation Services JSP template(s).

DecideShippingAddressPageIP

Class Name	`examples.wlcs.sampleapp.tax.webflow.DecideShippingAddressPageIP`
Description	Makes the decision about whether to display `selecttaxaddress.jsp` based on the number of address variations returned from the Tax Web service. If a single address is found, this input processor updates the shipping address, returns successfully, and allows the Webflow to proceed to `payment.jsp`. Otherwise, this input processor redirects the Webflow to `selecttaxaddress.jsp`.
Required `HTTPServletRequest` Parameters	None
Required Pipeline Session Attributes	`PipelineSessionConstants.SHIPPING_ADDRESS` `PipelineSessionConstants.AVS_SHIPPING_ADDRESSES`
Updated Pipeline Session Attributes	`PipelineSessionConstants.SHIPPING_ADDRESS` (in the case of a single address)
Removed Pipeline Session Attributes	None
Validation	None
Exceptions	`None`

UpdateShippingAddressIP

Class Name	`examples.wlcs.sampleapp.shipping.webflow.UpdateShippingAddressIP`
Description	Updates the shipping address attribute in the Pipeline session based on the tax address the customer selects.

Required `HTTPServletRequest` Parameters	`HTTPRequestConstants.TAX_SHIPPING_ADDRESS`
Required Pipeline Session Attributes	`PipelineSessionConstants.SHIPPING_ADDRESS` `PipelineSessionConstants.AVS_SHIPPING_ADDRESSES`
Updated Pipeline Session Attributes	`PipelineSessionConstants.SHIPPING_ADDRESS`
Removed Pipeline Session Attributes	None
Validation	None
Exceptions	None

Pipeline Components

This section provides a brief description of each Pipeline component associated with the Taxation Services JSP template(s).

Note: Some Pipeline components extend other, base Pipeline components. For more information on the base classes, see the *Javadoc*.

TaxCalculateLineLevelPC

Class Name	`examples.wlcs.sampleapp.tax.pipeline.TaxCalculateLineLevelPC`
Description	Calculates the tax and provides line-level information about the taxability of an item. This Pipeline component is used to display the tax information to the customer.

Required Pipeline Session Attributes	`PipelineSessionConstants.SHOPPING_CART` `PipelineSessionConstants.SHIPPING_ADDRESS`
Updated Pipeline Session Attributes	`PipelineSessionConstants.SHOPPING_CART`
Removed Pipeline Session Attributes	None
Type	Java class
JNDI Name	None
Exceptions	None

TaxCalculateAndCommitLineLevelPC

Class Name	`examples.wlcs.sampleapp.tax.pipeline.TaxCalculateAndCommitLineLevelPC`
Description	Calculates the tax and provides line-level information about the taxability of an item. The results are logged to the Tax Web service audit file so that correct payment can be made to taxing jurisdictions, or to generate tax reports.
Required Pipeline Session Attributes	`PipelineSessionConstants.SHOPPING_CART` `PipelineSessionConstants.SHIPPING_ADDRESS`
Updated Pipeline Session Attributes	`PipelineSessionConstants.SHOPPING_CART`
Removed Pipeline Session Attributes	None
Type	Java class
JNDI Name	None
Exceptions	None

TaxVerifyShippingAddressPC

Class Name	`examples.wlcs.sampleapp.tax.pipeline.` `TaxVerifyShippingAddressPC`
Description	Ensures that the shipping address is descriptive enough to properly calculate taxation for an order based on jurisdiction.
Required Pipeline Session Attributes	`PipelineSessionConstants.SHIPPING_ADDRESS`
Updated Pipeline Session Attributes	`PipelineSessionConstants.AVS_SHIPPING_ADDRESSES`
Removed Pipeline Session Attributes	None
Type	Java class
JNDI Name	None
Exceptions	`TaxSystemException`, thrown if processing could not occur due to system level problems (for example, some data files are missing). `TaxUserException`, thrown if processing could not occur due to invalid user input.

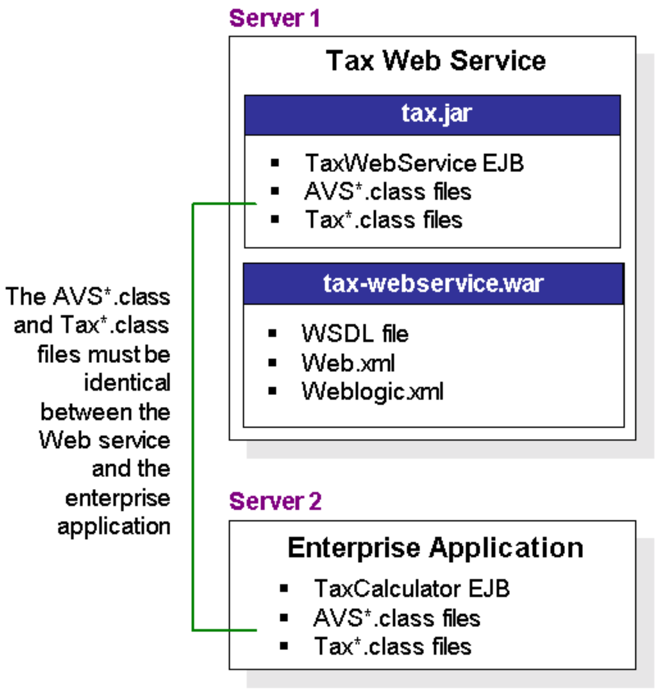
Integrating with a Taxation Service

The Tax Web Service that is installed with WebLogic Portal provides a default framework for handling tax calculations on transactions received from the default TaxCalculator EJB. The business methods within the TaxCalculator EJB implement a standard workflow that is associated with the completion of order taxation. (The Tax Web service, by comparison, is a stateless session EJB wrapped in code that makes it a Web service.)

Integrating your enterprise applications with the Tax Web Service involves modifying either the TaxCalculator EJB or the Tax Web Service, depending on who will host the Web service: your organization or the third-party tax calculation vendor.

In either case, it helps to understand the connection relationship between the pieces in the WebLogic Portal taxation services and the pieces in the Tax Web Service. Figure 5-3 illustrates the connection between the two.

Figure 5-3 The Relationship Between the Tax Web Service and the TaxCalculator EJB



Caution: The default Tax Web service that is shipped with WebLogic Portal automatically applies a 5% tax to an order. This default application of taxes is not designed for production use. You must integrate with your third-party vendor's tax service to calculate taxes properly.

If the Third-Party Vendor Hosts the Web Service

If the third-party vendor hosts the Tax Web Service, the vendor will integrate the Web service with their product's API.

Here is what your organization must do to connect to the vendor-hosted Web service:

1. If the vendor has modified any of `AVS*.class` or `Tax*.class` files in the Web service's `tax.jar` file, copy those modifications into your enterprise application. You can find the source code in:

   ```
   PORTAL_HOME\applications\wlcsApp\src\examples\wlcs\
   sampleapp\tax\
   ```

   Compile the source files.

2. Make any vendor-required modifications to the TaxCalculator EJB in your enterprise application so that it makes appropriate SOAP calls to the vendor's tax Web service. You can find the source code for the TaxCalculator EJB in the following files:

   ```
   PORTAL_HOME\applications\wlcsApp\src\examples\wlcs\
   sampleapp\tax\TaxCalculator*.java
   ```

   Compile the source files.

3. After you compile your source code, add the class files to the `wlcsSample.jar` in the `wlcsApp` application directory. When you add the class files to the JAR, make sure you maintain their relative directory structure.

4. Run the EJB compiler (`ejbc`) on the `wlcsSample.jar` file.

5. In the WebLogic Server Console for portalServer, select Deployments > Applications > wlcsApp > Service Configuration > Tax Service Client, and in the Tax Calculator WSDL field, modify the URL to the tax vendor's WSDL file. Click Apply in the Console to apply the new URL. The new URL is written to the following file: `PORTAL_HOME\applications\wlcsApp\META-INF\ application-config.xml`.

   Note: At startup, WebLogic Server reads the `application-config.xml` file, so it knows where to find the Web service.

If Your Organization Hosts the Web Service

If your organization hosts the Tax Web Service, we strongly recommend that you deploy the Web service on a separate instance of WebLogic Server (that is, use a separate Java Virtual Machine [JVM]) than what your enterprise applications are running on. This way, your enterprise applications are insulated from failures or incidents in the Web service.

Here is what you must do if your organization hosts the Tax Web Service:

Caution: These are general, simplified guidelines for integrating with a vendor's API. In actual practice, such integration requires close collaboration with your vendor. We strongly recommend you contact your vendor for assistance.

1. Obtain your third-party vendor's tax calculation product API.

2. Modify the TaxWebService EJB (the Web service EJB) so that it translates SOAP calls into the language of the third-party product's API. You can find the source code for the TaxWebService EJB in the following files:

   ```
   PORTAL_HOME\applications\taxWSApp\src\examples\wlcs\
   sampleapp\tax\TaxWebService*.java
   ```

   Compile the source files.

3. After you have compiled the source code, replace the class files in `tax.jar`, located in the `taxWSApp` directory. When you add the class files to the JAR, make sure you maintain their relative directory structure.

4. Use the Web service generator (`wsgen`) on the `tax.jar` file to build a file called `tax-webservice.war`, as shown in Figure 5-3.

   For information on using `wsgen`, see *Programming WebLogic Server Web Services* at http://e-docs.bea.com/wls/docs61/webServices/index.html.

5. Make any necessary modifications to the TaxCalculator EJB in the `wlcsApp` application so that it makes appropriate SOAP calls to the TaxWebService EJB. You can find the source code for the TaxCalculator EJB in the following files:

   ```
   PORTAL_HOME\applications\wlcsApp\src\examples\wlcs\
   sampleapp\tax\TaxCalculator*.java
   ```

   Compile the source files.

6. After you compile your source code, add the class files to `wlcsSample.jar` in the `wlcsApp` application directory. When you add the class files to the JAR, make sure you maintain their relative directory structure.

7. Run the EJB compiler (`ejbc`) on the `wlcsSample.jar` file.

8. In the WebLogic Server Console for portalServer, select Deployments > Applications > wlcsApp > Service Configuration > Tax Service Client, and in the Tax Calculator WSDL field, modify the URL to the WSDL file. Click Apply in the Console to apply the new URL. The new URL is written to the following file: `PORTAL_HOME\applications\wlcsApp\META-INF\application-config.xml`.

   Note: At startup, WebLogic Server reads the `application-config.xml` file, so it knows where to find the Web service.

CHAPTER

6 Payment Services

WebLogic Portal includes a flexible payment service that lets you connect to a third-party payment product for authorizing and settling orders. The payment service itself currently allows credit card payments to be made. This topic describes the Payment Services in detail.

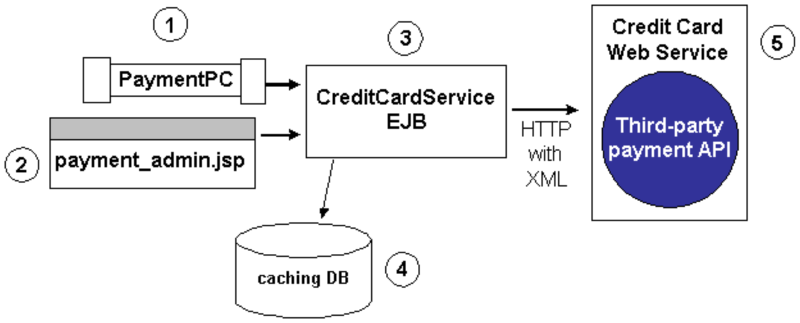
This topic includes the following sections:

How the Payment Service Works

For an introduction to Web services, see "Introduction to Web Services" on page 5-2 in Chapter 5, "Taxation Services."

Figure 6-1 shows the basic architecture of WebLogic Portal's payment service. The key component of this architecture is the Credit Card Web Service, which is the connection point between WebLogic Portal and any third-party payment product.

Figure 6-1 Payment Service Architecture



Following is a description of each piece of the architecture.

1	The PaymentPC Pipeline component instantiates and makes calls to the CreditCardService EJB. For information on Pipeline components, see the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline*.
2	The payment JSPs provide various functions, including form fields and information that the customer uses to make payment. The `payment_admin.jsp`, which calls the CreditCardService EJB, also lets you search transactions and resubmit them if necessary. On submit, the JSP passes the customer payment data to the CreditCardService EJB.
3	The CreditCardService EJB handles the business logic for the payment. The EJB converts payment service calls to XML for transporting SOAP messages to the Credit Card Web Service.

4	The CreditCardService EJB persists the payment transactions.
5	The CreditCardService EJB in WebLogic Portal sends SOAP messages to the Credit Card Web Service (a stateless session EJB), where those SOAP messages are converted to the language of the third-party product's API. The third-party payment service processes payments and sends confirmation or exceptions back to the application. **Caution:** The default Payment Web service that ships with WebLogic Portal always sends payment information through without any errors, as if it were connected to and approved by a third-party payment service. This default processing of payment is not designed for production use. You must integrate with your third-party vendor's payment service to process payment correctly.

Because no industry standards exist yet for handling payment tasks (such as authorization, retry, and settle), third-party payment products have their own APIs handling these tasks. There are two keys to connecting an enterprise application with a third-party payment Web service:

- Sending the proper SOAP messages from the CreditCardService EJB in your enterprise application to the payment Web service
- In the payment Web service, translating the SOAP messages into the language of the third-party product's API

For information on connecting an enterprise application to a Web service, see "Integrating with a Payment Service" on page 6-26.

JavaServer Pages (JSPs)

A primary goal of Order services is to allow you to quickly establish a fully-functioning e-commerce site. To this end, the Payment Service provides you with a Java Server Page (JSP) template that you can use as is, or customize to better meet your needs. This section describes this page in detail.

Note: For a description of the complete set of JSPs used in the WebLogic Portal Web application and a listing of their locations in the directory structure, see the *E-Commerce Summary of JSP Templates* documentation.

payment.jsp Template

If a customer has already specified payment information in their user profile, the `payment.jsp` template (shown in Figure 6-2) provides the customer with a list of credit cards (by type and last 4 digits) for selection. Customers wanting to use an existing credit card can simply click its associated Use button to proceed to the next part of the checkout process.

Note: For more information about user profiles, see "Customer Profile Services" in the *Guide to Registering Customers and Managing Customer Services*.

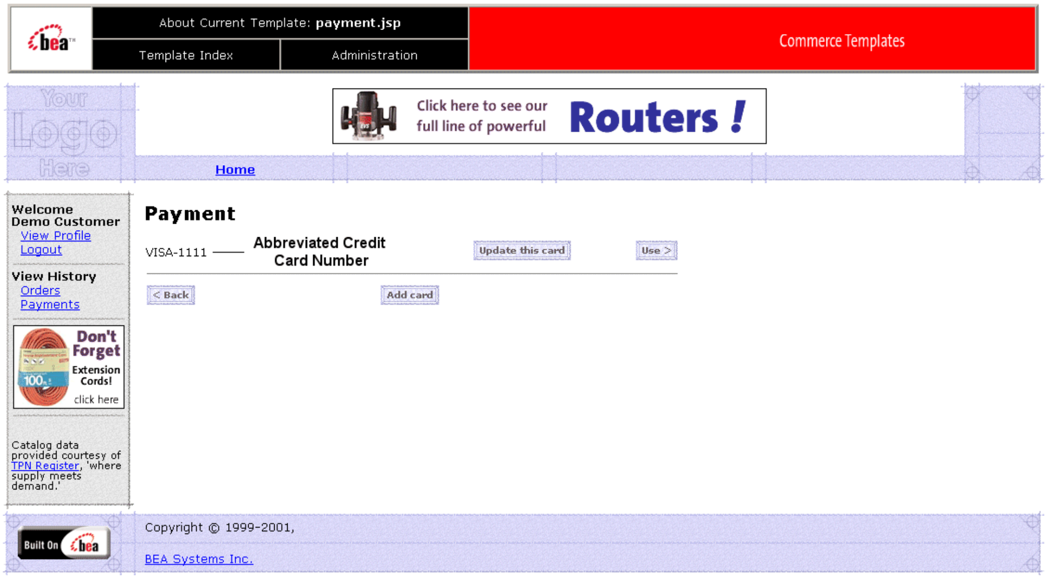
Customers can also choose to update the information associated with this credit card by clicking the Update This Card button. If your customer wants to use a credit card they have never used on your e-commerce site before, the customer can click the Add Card button to add it to the list (using the `paymentnewcc.jsp` template). If a customer wants to go back to the previous page, the customer can click the Back button.

Sample Browser View

Figure 6-2 shows an annotated version of the `payment.jsp` template. The Payment region uses a combination of the WebLogic Server and WebLogic Portal JSP tags to obtain and display the customer's saved credit card(s).

Note: For information on other elements in the `payment.jsp` template, see "Common JSP Template Elements" on page 3-2.

Figure 6-2 Annotated payment.jsp Template


About Current Template: payment.jsp
Template Index
Administration
Commerce Templates
Click here to see our full line of powerful Routers !
Home
Welcome Demo Customer
View Profile
Logout
View History
Orders
Payments
Don't Forget Extension Cords! click here
Catalog data provided courtesy of TPN Register, 'where supply meets demand.'
Payment
VISA-1111
Abbreviated Credit Card Number
Update this card
Use >
< Back
Add card
Copyright © 1999-2001,
BEA Systems Inc.
Built On bea

Location in the WebLogic Portal Directory Structure

You can find the `payment.jsp` template file at the following location, where `PORTAL_HOME` is the directory in which you installed WebLogic Portal:

`%PORTAL_HOME%\applications\wlcsApp\wlcs\commerce\order\payment.jsp` (Windows)
`$PORTAL_HOME/applications/wlcsApp/wlcs/commerce/order/payment.jsp` (UNIX)

Tag Library Imports

The `payment.jsp` template uses existing WebLogic Server and WebLogic Portal's User Management JSP tags. Therefore, the template includes the following JSP tag libraries:

```
<%@ taglib uri="webflow.tld" prefix="webflow" %>
<%@ taglib uri="um.tld" prefix="um" %>
<%@ taglib uri="weblogic.tld" prefix="wl" %>
```

Note: For more information on the WebLogic Server JSP tags or the WebLogic Portal JSP tags, see "JSP Tag Reference" in the *Guide to Building Personalized Applications*.

These files reside in the `lib` directory within `PORTAL_HOME`.

Java Package Imports

The `payment.jsp` template uses Java classes in the following packages and therefore includes these import statements:

```
<%@ page import="javax.servlet.*" %>
<%@ page import="javax.servlet.http.*" %>
<%@ page import="com.beasys.commerce.webflow.tags.*" %>
<%@ page import="com.beasys.commerce.axiom.contact.*" %>
<%@ page import="com.beasys.commerce.ebusiness.customer.*" %>

<%@ page import="com.beasys.commerce.webflow.HttpRequestConstants" %>
```

Location in Default Webflow

Customers arrive at `payment.jsp` from the page where they select their shipping address (`selectaddress.jsp`). If they choose to add a new credit card, they will be directed to the `paymentnewcc.jsp` template. If the customer chooses to edit one of the cards that appears in the list, the customer will be directed to the `paymenteditcc.jsp` template. After selecting a credit card for payment, customers move on to the final page in the checkout process, where they can review their order prior to committing it (`checkout.jsp`).

Note: For more information about the default Webflow, see "Overview of Managing Purchases and Processing Orders" on page 1-1.

Included JSP Templates

The following JSP templates are included in the `payment.jsp` template:

- `header.inc`, which creates the top banner.
- `admin.inc`, which is used on all pages and presents the top red-and-black banner with links to the main WLCS Administration screen, to this template

index, and to a *.jsp.html file for the current template. You should remove this from the JSP when you deploy it.

- `leftside.inc`, which presents quick look-up and a promotional ad; for authenticated users, it also presents a personalized message to the user, customer profile link, order history link, and payment history link.
- `footer.inc`, which creates a horizontal footer at the bottom of the page.

Events

The `payment.jsp` template presents a customer with several buttons, each of which is considered an event. These events trigger a particular response in the default Webflow that allows customers to continue. While this response can be to load another JSP, it is usually the case that an input processor or Pipeline is invoked first. Table 6-1 provides information about these events and the business logic they invoke.

Table 6-1 payment.jsp Events

Event	Webflow Response(s)
`button.addNewCreditCard`	No business logic required. Loads `paymentnewcc.jsp`.
`button.continue`	`AuthorizePaymentIP`
`button.updatePaymentInfo`	No business logic required. Loads `paymenteditcc.jsp`.

Dynamic Data Display

The purpose of the `payment.jsp` template is to display a list of the customer's previously saved credit cards. This is accomplished on the `payment.jsp` template using a combination of WebLogic Server and WebLogic Portal JSP tags and accessor methods/attributes.

First, the `getProfile` JSP tag is used to set the customer profile (context) for which the credit cards should be retrieved, as shown in Listing 6-1.

Listing 6-1 Setting the Customer Context

```
<um:getProfile
       profileKey="<%=request.getRemoteUser()%>"
       profileType="WLCS_Customer" />
```

Next, the `getProperty` JSP tag is used to retrieve a cached copy of the possible credit cards for the customer from the database, as shown in Listing 6-2.

Listing 6-2 Retrieving the CreditCardsMap for the Customer

```
<um:getProperty propertySet="CustomerProperties"
propertyName="creditCardsMap" id="creditCardsMapObject" />
```

You can now iterate through the credit cards contained within the `creditCardsMap` (using the WebLogic Server JSP tag) and display each credit card in the collection (using a Java scriptlet) as shown in Listing 6-3.

Listing 6-3 Iterating Through and Displaying the Credit Cards

```
<table>
  <wl:repeat
   set="<%=((Map)creditCardsMapObject).keySet().iterator()%>"
   id="creditCard" type="String" count="100000">
    <tr>
    <!-- Output the credit card name -->
      <td width="50%"><div class="tabletext"><%=creditCard%></div></td>
    <!-- The update button -->
      <td width="30%" align="right">
      <%
     String extraParams = HttpRequestConstants.CREDITCARD_KEY + "=" + creditCard;
      %>

      <a
      href="<webflow:createWebflowURL event="button.updatePaymentInfo"
httpsInd="calculate" namespace="sampleapp_order" extraParams="<%= extraParams
```

```
%>" />"><img src="<webflow:createResourceURL
resource="/commerce/images/btn_updatecard.gif" />" border="0">
      </a>
      </td>
      <!-- The use button -->
      <td width="20%" align="right">
      <a
      href="<webflow:createWebflowURL event="button.continue"
httpsInd="calculate" namespace="sampleapp_order" extraParams="<%= extraParams
%>" />"><img src="<webflow:createResourceURL
resource="/commerce/images/btn_use.gif" />" border="0">
      </a>
      </td>
    </tr>
    <tr>
      <td colspan="3"><hr size="1"></td>
    </tr>
  </wl:repeat>
</table>
```

Note: For more information on the WebLogic Server JSP tags or the WebLogic Portal JSP tags, see "JSP Tag Reference" in the *Guide to Building Personalized Applications*.

Form Field Specification

The `payment.jsp` template does not make use of any form fields.

paymentnewcc.jsp Template

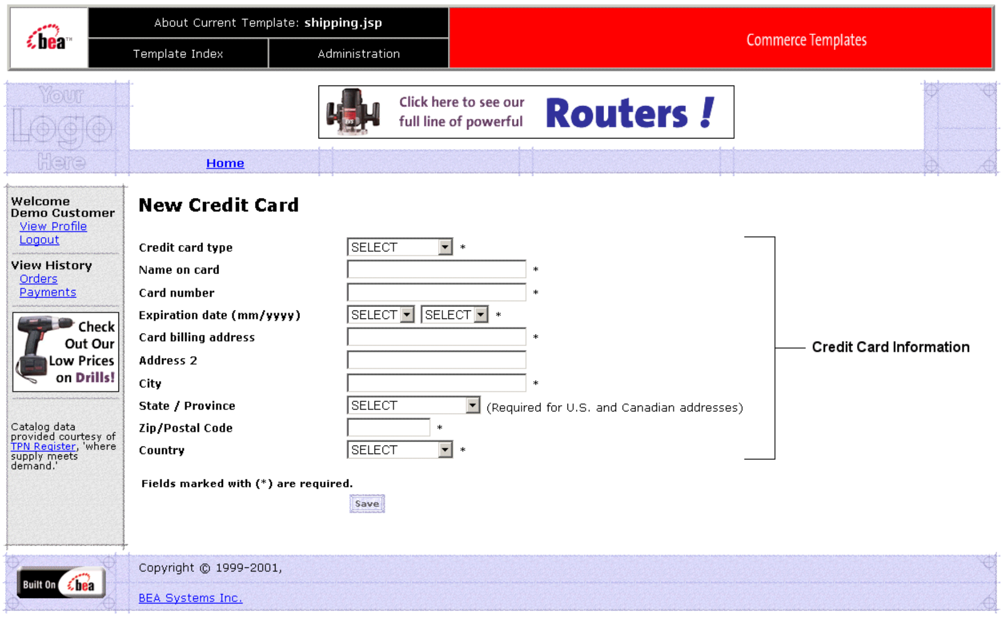
The `paymentnewcc.jsp` template (shown in Figure 6-3) allows customers to enter information about a new credit card, which will be added to their profile. This information includes the credit card type (VISA, MasterCard, and so on), the name on the card, the card number, the card expiration date (month and 4-digit year), and the billing address (including a street address, city, state, zip/postal code, and country). The customer must click the Save button for the new credit card to be added to the customer's list of credit cards.

Sample Browser View

Figure 6-3 shows an annotated version of the `paymentnewcc.jsp` template. The New Credit Card region provides customers with a series of form fields that allow customers to add a credit card. This region utilizes the form fields defined in the included `newcctemplate.jsp` template file, which itself includes the `states.jsp` and `countries.jsp` template files. The import call in `paymentnewcc.jsp` is:

```
<%@ include file="/commerce/includes/newcctemplate.jsp" %>
```

Figure 6-3 Annotated paymentnewcc.jsp Template


About Current Template: shipping.jsp
Template Index
Administration
Commerce Templates
Your Logo Here
Click here to see our full line of powerful Routers !
Home
Welcome Demo Customer
View Profile
Logout
View History
Orders
Payments
Check Out Our Low Prices on Drills!
Catalog data provided courtesy of TPN Register, 'where supply meets demand.'
Credit Card Information
Built On bea

New Credit Card

Credit card type	SELECT *
Name on card	*
Card number	*
Expiration date (mm/yyyy)	SELECT SELECT *
Card billing address	*
Address 2	
City	*
State / Province	SELECT (Required for U.S. and Canadian addresses)
Zip/Postal Code	*
Country	SELECT *

Fields marked with (*) are required.

Save

Copyright © 1999-2001,
BEA Systems Inc.

Location in the WebLogic Portal Directory Structure

You can find the `paymentnewcc.jsp` template file at the following location, where `PORTAL_HOME` is the directory in which you installed Commerce services:

`%PORTAL_HOME%\applications\wlcsApp\wlcs\commerce\order\paymentnewcc.jsp` (Windows)
`$PORTAL_HOME/applications/wlcsApp/wlcs/commerce/order/paymentnewcc.jsp` (UNIX)

Tag Library Imports

The `paymentnewcc.jsp` template uses Pipeline and Webflow JSP tags. Therefore, the template includes the following JSP tag libraries:

```
<%@ taglib uri="webflow.tld" prefix="webflow" %>
```

Note: For more information on the Webflow and Pipeline JSP tags, see the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline*.

These files reside in the `lib` directory within `PORTAL_HOME`.

Java Package Imports

The `paymentnewcc.jsp` template uses Java classes in the following packages and therefore includes these import statements:

```
<%@ page import="javax.servlet.*" %>
<%@ page import="javax.servlet.http.*" %>
<%@ page import="com.beasys.commerce.webflow.tags.*" %>
<%@ page import="com.beasys.commerce.axiom.contact.*" %>
<%@ page import="com.beasys.commerce.ebusiness.customer.*" %>
<%@ page import="com.beasys.commerce.webflow.HttpRequestConstants" %>
```

Location in Default Webflow

Customers arrive at the `paymentnewcc.jsp` template from the page where they are given the option of selecting a credit card from their profile (`payment.jsp`). When customers are finished with this page, customers are returned to the `payment.jsp` template so customers can make their selection.

This template is in the sampleapp_order namespace.

Note: For more information about the default Webflow, see “Overview of Managing Purchases and Processing Orders” on page 1-1.

Included JSP Templates

The following JSP templates are included in the `paymentnewcc.jsp` template:

- `header.inc`, which creates the top banner.

- `admin.inc`, which is used on all pages and presents the top red-and-black banner with links to the main WLCS Administration screen, to this template index, and to a *.jsp.html file for the current template. You should remove this from the JSP when you deploy it.
- `leftside.inc`, which presents quick look-up and a promotional ad; for authenticated users, it also presents a personalized message to the user, customer profile link, order history link, and payment history link.
- `footer.inc`, which creates a horizontal footer at the bottom of the page.
- `newcctemplate.inc`, described in "Customer Registration and Login Services" in the *Guide to Registering Customers and Managing Customer Services*.

Events

The `paymentnewcc.jsp` template presents a customer with a single button, which is considered an event. This event triggers a particular response in the default Webflow that allows customers to continue. While this response can be to load another JSP, it is usually the case that an input processor or Pipeline is invoked first. Table 6-2 provides information about these events and the business logic they invoke.

Table 6-2 paymentnewcc.jsp Events

Event	Webflow Response(s)
`button.save`	`UpdatePaymentInfoIP`

Dynamic Data Display

No dynamic data is displayed on the `paymentnewcc.jsp` template.

Form Field Specification

The purpose of the `paymentnewcc.jsp` template is to provide form fields that allow the customer to enter new credit card information. It also passes hidden information to the Webflow. The form fields used in the `paymentnewcc.jsp` template, and a description for each of these form fields, are listed in Table 6-3.

You could add additional fields if your payment service required them.

Table 6-3 paymentnewcc.jsp Form Fields

Parameter Name	Type	Description
`“event”`	Hidden	Indicates which event has been triggered. It is used by the Webflow to determine what happens next.
`“origin”`	Hidden	The name of the current page (`paymentnewcc.jsp`), used by the Webflow.
`HttpRequestConstants.CUSTOMER_CREDITCARD_TYPE`	Listbox	The type of the customer’s credit card.
`HttpRequestConstants.CUSTOMER_CREDITCARD_HOLDER`	Textbox	The name on the credit card.
`HttpRequestConstants.CUSTOMER_CREDITCARD_NUMBER`	Textbox	The number of the customer’s credit card.
`HttpRequestConstants.CUSTOMER_CREDITCARD_MONTH`	Listbox	The month of the customer’s credit card expiration date.
`HttpRequestConstants.CUSTOMER_CREDITCARD_YEAR`	Listbox	The year of the customer’s credit card expiration date.
`HttpRequestConstants.CUSTOMER_CREDITCARD_ADDRESS1`	Textbox	The first line in the customer’s billing address.
`HttpRequestConstants.CUSTOMER_CREDITCARD_ADDRESS2`	Textbox	The second line in the customer’s billing address.
`HttpRequestConstants.CUSTOMER_CREDITCARD_CITY`	Textbox	The city in the customer’s billing address.
`HttpRequestConstants.CUSTOMER_CREDITCARD_STATE`	Listbox	The state in the customer’s billing address.
`HttpRequestConstants.CUSTOMER_CREDITCARD_ZIPCODE`	Textbox	The zip/postal code in the customer’s billing address.
`HttpRequestConstants.CUSTOMER_CREDITCARD_COUNTRY`	Listbox	The country in the customer’s billing address.

Note: Parameters that are literals in the JSP code are shown in quotes, while non-literals will require scriptlet syntax (such as `<%= HttpRequestConstants.CUSTOMER_CREDIT_CARD_COUNTRY %>`) for use in the JSP.

paymenteditcc.jsp Template

The `paymenteditcc.jsp` template (shown in Figure 6-4) allows your customers to modify information about one of the credit cards shown in the credit card list. Editable information includes the name on the credit card, the expiration date (month and 4-digit year), and the billing address (including street address, city, state, zip/postal code, and country). The customer must click the Save button to save the modifications to their credit card.

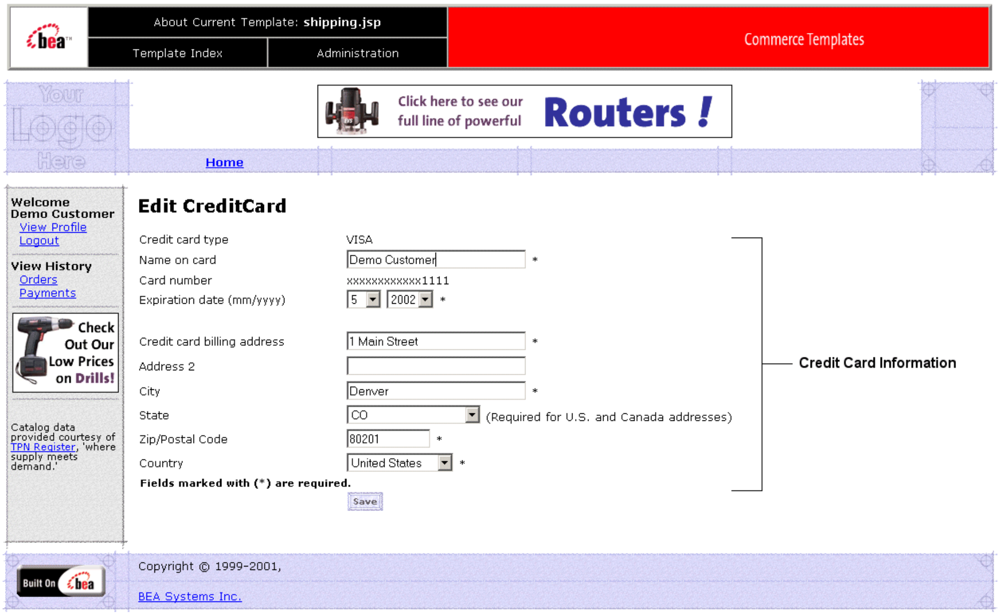
Sample Browser View

Figure 6-4 shows an annotated version of the `paymenteditcc.jsp` template. The Edit Credit Card region provides customers with a series of form fields that allow customers to edit a credit card. This region utilizes the form fields defined in the included `editcctemplate.jsp` template file, which itself includes the `states.jsp` and `countries.jsp` template files. The import call in `paymenteditcc.jsp` is:

```
<%@ include file="/commerce/includes/editcctemplate.inc” %>
```

Note: For information on other elements in the `paymenteditcc.jsp` template, see “Common JSP Template Elements” on page 3-2.

Figure 6-4 Annotated paymenteditcc.jsp Template


About Current Template: **shipping.jsp**
Template Index
Administration
Commerce Templates
Click here to see our full line of powerful Routers !
Your Logo Here
Home
Welcome Demo Customer
View Profile
Logout
View History
Orders
Payments
Check Out Our Low Prices on Drills!
Catalog data provided courtesy of TPN Register, 'where supply meets demand.'
Built On bea
Copyright © 1999-2001,
BEA Systems Inc.
Credit Card Information

Edit CreditCard

Credit card type	VISA
Name on card	Demo Customer *
Card number	xxxxxxxxxxxx1111
Expiration date (mm/yyyy)	5 2002 *
Credit card billing address	1 Main Street *
Address 2	
City	Denver *
State	CO (Required for U.S. and Canada addresses)
Zip/Postal Code	80201 *
Country	United States *

Fields marked with (*) are required.

Save

Location in the WebLogic Portal Directory Structure

You can find the `paymenteditcc.jsp` template file at the following location, where `PORTAL_HOME` is the directory in which you installed WebLogic Portal:

`%PORTAL_HOME%\applications/wlcsApp\wlcs\commerce\order\`
`paymenteditcc.jsp` (Windows)
`$PORTAL_HOME/applications/wlcsApp/wlcs/commerce/order/`
`paymenteditcc.jsp` (UNIX)

Tag Library Imports

The `paymenteditcc.jsp` template uses the existing WebLogic Portal's User Management JSP tags, and the Pipeline and Webflow JSP tags. Therefore, the template includes the following JSP tag libraries:

```
<%@ taglib uri="webflow.tld" prefix="webflow" %>
<%@ taglib uri="um.tld" prefix="um" %>
```

Note: For more information on the Webflow and Pipeline JSP tags, see the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline*. For more information on the WebLogic Portal JSP tags, see "JSP Tag Reference" in the *Guide to Building Personalized Applications*.

These files reside in the `lib` directory within `PORTAL_HOME`.

Java Package Imports

The `paymenteditcc.jsp` template uses Java classes in the following packages and therefore includes these import statements:

```
<%@ page import="javax.servlet.*" %>
<%@ page import="javax.servlet.http.*" %>
<%@ page import="com.beasys.commerce.webflow.tags.*" %>
<%@ page import="com.beasys.commerce.axiom.contact.*" %>
<%@ page import="com.beasys.commerce.ebusiness.customer.*" %>
<%@ page import="com.beasys.commerce.webflow.HttpRequestConstants" %>
```

Location in Default Webflow

Customers arrive at `paymenteditcc.jsp` template from the page where they are given the option of selecting a credit card from their profile (`payment.jsp`). When customers are finished with this page, they are returned to the `payment.jsp` template so they can make their selection.

This template is in the sampleapp_order namespace.

Note: For more information about the default Webflow, see "Overview of Managing Purchases and Processing Orders" on page 1-1.

Included JSP Templates

The following JSP templates are included in the `paymenteditcc.jsp` template:

- `header.inc`, which creates the top banner.
- `admin.inc`, which is used on all pages and presents the top red-and-black banner with links to the main WLCS Administration screen, to this template index, and to a *.jsp.html file for the current template. You should remove this from the JSP when you deploy it.
- `leftside.inc`, which presents quick look-up and a promotional ad; for authenticated users, it also presents a personalized message to the user, customer profile link, order history link, and payment history link.
- `footer.inc`, which creates a horizontal footer at the bottom of the page.
- `editcctemplate.inc`, described in "Customer Profile Services" in the *Guide to Registering Customers and Managing Customer Services*.

Events

The `paymenteditcc.jsp` template presents a customer with a single button, which is considered an event. This event triggers a particular response in the default Webflow that allows customers to continue. While this response can be to load another JSP, it is usually the case that an input processor or Pipeline is invoked first. Table 6-4 provides information about these events and the business logic they invoke.

Table 6-4 paymenteditcc.jsp Events

Event	Webflow Response(s)
`button.save`	`UpdatePaymentInfoIP`

Dynamic Data Display

One purpose of the `paymenteditcc.jsp` template is to prepare the credit card information a customer had previously entered, so the `editcctemplate.jsp` template can display this information in the payment information form fields. This is accomplished on the `paymenteditcc.jsp` template using a combination WebLogic Portal's User Management JSP tags and accessor methods/attributes.

First, the `getProfile` JSP tag is used to set the customer profile (context) for which the customer information should be retrieved, as shown in Listing 6-4.

Listing 6-4 Setting the Customer Context

```
<um:getProfile
  profileKey="<%=request.getRemoteUser()%>"
  profileType="WLCS_Customer" />
```

Note: For more information on the WebLogic Portal User Management JSP tags, see "JSP Tag Reference" in the *Guide to Building Personalized Applications*.

Next, the `getProperty` JSP tag is used to obtain the customer's list of credit cards (and related billing information), which is then initialized with data from the customer object, as shown in Listing 6-5.

Listing 6-5 Obtaining the Customer's Credit Cards and Billing Information

```
<um:getProperty propertySet="CustomerProperties" propertyName="creditCardsMap"
id="creditCards" />
<%
    String creditCardKey =
request.getParameter(HttpRequestConstants.CREDITCARD_KEY);
    CreditCard defaultCreditCard = null;
    if(creditCardKey != null)
    {
        defaultCreditCard = (CreditCard)((Map)creditCards).get(creditCardKey);
    }
    else
    {
        defaultCreditCard = CreditCardHome.create();
    }
    Address billingAddress = (Address) defaultCreditCard.getBillingAddress();
%>
```

The data stored within the `defaultCreditCard` and `billingAddress` objects can now be accessed by calling accessor methods/attributes within Java scriptlets. Table 6-5 provides more detailed information about the methods/attributes for the default credit card, while Table 6-6 provides more information about the accessor methods/attributes on `billingAddress`.

Table 6-5 defaultCreditCard Accessor Methods/Attributes

Method/Attribute	Description
`getType()`	The credit card type (VISA, MasterCard, AMEX, and so on).
`getName()`	The credit card holder's name.
`getDisplayNumber()`	The credit card number for display (12 Xs and last 4 digits).
`getNumber()`	The credit card number.
`getExpirationDate()`	The credit card's expiration date.

Table 6-6 billingAddress Accessor Methods/Attributes

Method/Attribute	Description
`getStreet1()`	The first line in the customer's billing street address.
`getStreet2()`	The second line in the customer's billing street address.
`getCity()`	The city in the customer's billing address.
`getCounty()`	The county in the customer's billing address.
`getState()`	The state in the customer's billing address.
`getPostalCode()`	The zip/postal code in the customer's billing address.
`getCountry()`	The country in the customer's billing address.

Form Field Specification

Another purpose of the `paymenteditcc.jsp` template is to provide the form fields for the customer's modifications and to pass hidden information to the Webflow. The form fields used in the `paymenteditcc.jsp`, and a description for each of these form fields, are listed in Table 6-7.

You could add additional fields if your payment service required them.

Table 6-7 paymenteditcc.jsp Form Fields

Parameter Name	Type	Description
`“event”`	Hidden	Indicates which event has been triggered. It is used by the Webflow to determine what happens next.
`“origin”`	Hidden	The name of the current page (`paymenteditcc.jsp`), used by the Webflow.
`HttpRequestConstants.CUSTOMER_CREDITCARD_TYPE`	Listbox	The type of the customer's credit card.
`HttpRequestConstants.CUSTOMER_CREDITCARD_HOLDER`	Textbox	The name on the credit card.
`HttpRequestConstants.CUSTOMER_CREDITCARD_NUMBER`	Textbox	The number of the customer's credit card.
`HttpRequestConstants.CUSTOMER_CREDITCARD_MONTH`	Listbox	The month of the customer's credit card expiration date.
`HttpRequestConstants.CUSTOMER_CREDITCARD_YEAR`	Listbox	The year of the customer's credit card expiration date.
`HttpRequestConstants.CUSTOMER_CREDITCARD_ADDRESS1`	Textbox	The first line in the customer's billing address.
`HttpRequestConstants.CUSTOMER_CREDITCARD_ADDRESS2`	Textbox	The second line in the customer's billing address.
`HttpRequestConstants.CUSTOMER_CREDITCARD_CITY`	Textbox	The city in the customer's billing address.

Table 6-7 paymenteditcc.jsp Form Fields (Continued)

Parameter Name	Type	Description
`HttpRequestConstants.` `CUSTOMER_CREDITCARD_STATE`	Listbox	The state in the customer's billing address.
`HttpRequestConstants.` `CUSTOMER_CREDITCARD_ZIPCODE`	Textbox	The zip/postal code in the customer's billing address.
`HttpRequestConstants.` `CUSTOMER_CREDITCARD_COUNTRY`	Listbox	The country in the customer's billing address.

Note: Parameters that are literals in the JSP code are shown in quotes, while non-literals will require scriptlet syntax (such as `<%= HttpRequestConstants.CUSTOMER_CREDIT_CARD_COUNTRY %>`) for use in the JSP.

payment_admin.jsp, paymenthistory.jsp, and payment_info.jsp Templates

Following are descriptions of the payment_admin.jsp, the payment_info.jsp, and the paymenthistory.jsp.

payment_admin.jsp

The `payment_admin.jsp` is the main page used for payment administration. All payment administration functions go through this page. It interfaces with the CreditCardService EJB for many operations.

payment_info.jsp

The `payment_info.jsp` provides developer-level information about the Payment JSP files. This file is, out of the box, only available as a popup from the main admin page when you click the More Explanation link under the Payment Administration section.

paymenthistory.jsp

The `paymenthistory.jsp` lets customers view their payment history by clicking the Payments link under View History in the left side of the window.

Input Processors

This section provides a brief description of each input processor associated with the Payment Services JSP template(s).

PaymentAuthorizationIP

Class Name	`examples.wlcs.sampleapp.payment.webflow.` `PaymentAuthorizationIP`
Description	Retrieves the shopping cart from the Pipeline session, the `CreditCardMapKey` from the request, and determines the total price of the order associated with the shopping cart. Adds the amount and credit card associated with the key to the Pipeline session.
Required `HTTPServletRequest` Parameters	`HttpRequestConstants.CREDITCARD_KEY`
Required Pipeline Session Attributes	`PipelineSessionConstants.SHOPPING_CART`
Updated Pipeline Session Attributes	`PipelineSessionConstants.PAYMENT_CREDIT_CARD` `PipelineSessionConstants.PAYMENT_AUTHORIZATION_AMOUNT`
Removed Pipeline Session Attributes	None
Validation	Verifies that the credit card key is valid and that it references an existing credit card.

Exceptions	`ProcessingException`, thrown for invalid types of `CREDITCARD_KEY`, `PAYMENT_CREDIT_CARD`, or `SHOPPING_CART`. Also thrown if these attributes are not available.

UpdatePaymentInfoIP

Class Name	`examples.wlcs.sampleapp.customer.webflow.` `UpdatePaymentInfoIP`
Description	Processes the customer's input from `paymentnewcc.jsp` and `paymenteditcc.jsp`. Retrieves the customer name from the Pipeline session, creates a new `CustomerValue` object, and sets it in the Pipeline session.

Required `HTTPServletRequest` Parameters	`HttpRequestConstants.CUSTOMER_CREDITCARD_TYPE`
	`HttpRequestConstants.CUSTOMER_CREDITCARD_HOLDER`
	`HttpRequestConstants.CUSTOMER_CREDITCARD_NUMBER`
	`HttpRequestConstants.CUSTOMER_CREDITCARD_MONTH`
	`HttpRequestConstants.CUSTOMER_CREDITCARD_YEAR`
	`HttpRequestConstants.CUSTOMER_CREDITCARD_ADDRESS1`
	`HttpRequestConstants.CUSTOMER_CREDITCARD_ADDRESS2`
	`HttpRequestConstants.CUSTOMER_CREDITCARD_CITY`
	`HttpRequestConstants.CUSTOMER_CREDITCARD_STATE`
	`HttpRequestConstants.CUSTOMER_CREDITCARD_ZIPCODE`
	`HttpRequestConstants.CUSTOMER_CREDITCARD_COUNTRY`

Required Pipeline Session Attributes	`PipelineSessionConstants.USER_NAME`
Updated Pipeline Session Attributes	`PipelineSessionConstants.CUSTOMER`
Removed Pipeline Session Attributes	None
Validation	Verifies that the required fields contain values.

Exceptions	`InvalidInputException`, thrown if invalid credit card information is obtained from the `HttpServletRequest`.

Pipeline Components

This section provides a brief description of each Pipeline component associated with the Payment Services JSP templates.

Note: Some Pipeline components extend other, base Pipeline components. For more information on the base classes, see the *Javadoc*.

PaymentAuthorizationHostPC

Class Name	`examples.wlcs.sampleapp.payment.pipeline.` `PaymentAuthorizationHostPC`

Description	Authorizes a given credit card for a specified amount. Used for host-based payment models, shown in the `weblogiccommerce.properties` file as:
	`HOST_AUTH_CAPTURE`
	`HOST_AUTH_CAPTURE_AVS`
	`HOST_POST_AUTH_CAPTURE`
	`HOST_POST_AUTH_CAPTURE_AVS`

Required Pipeline Session Attributes	`PipelineSessionConstants.PAYMENT_CREDIT_CARD`
	`PipelineSessionConstants.PAYMENT_AUTHORIZATION_AMOUNT`
	`PipelineSessionConstants.ORDER_HANDLE` (Request scope)

Updated Pipeline Session Attributes	None
Removed Pipeline Session Attributes	None
Type	Java object

JNDI Name	None
Exceptions	`AuthorizationFailureException`, thrown when the credit card being used for authorization is invalid (that is, the number or other associated information is incorrect). `AuthorizationRejectedException`, thrown when the credit card used for authorization is valid but cannot be authorized (overdrawn, expired, and so on). `PipelineNonFatalException`, thrown when the external payment service is unavailable. The transaction is recorded for retry. `PipelineException`, thrown when there is a configuration error, a general service error, or a system-level exception from a back-end component.

PaymentAuthorizationTerminalPC

Class Name	`examples.wlcs.sampleapp.payment.pipeline.` `PaymentAuthorizationTerminalPC`

Description	Authorizes a given credit card for a specified amount. Used for terminal-based payment models, shown in your application's `Meta-inf\application-config.xml` file as the following MBean properties:
	`AUTO_MARK_AUTO_SETTLE`
	`AUTO_MARK_AUTO_SETTLE_AVS`
	`AUTO_MARK_MANUAL_SETTLE`
	`AUTO_MARK_MANUAL_SETTLE_AVS`
	`MANUAL_MARK_AUTO_SETTLE`
	`MANUAL_MARK_AUTO_SETTLE_AVS`
	`MANUAL_MARK_MANUAL_SETTLE`
	`MANUAL_MARK_MANUAL_SETTLE_AVS`

Required Pipeline Session Attributes	`PipelineSessionConstants.PAYMENT_CREDIT_CARD`
	`PipelineSessionConstants.PAYMENT_AUTHORIZATION_AMOUNT`
	`PipelineSessionConstants.ORDER_HANDLE` (Request scope)

Updated Pipeline Session Attributes	None
Removed Pipeline Session Attributes	None
Type	Java object
JNDI Name	None
Exceptions	`AuthorizationFailureException`, thrown when the credit card being used for authorization is invalid (that is, the number or other associated information is incorrect). `AuthorizationRejectedException`, thrown when the credit card used for authorization is valid but cannot be authorized (overdrawn, expired, and so on). `PipelineNonFatalException`, thrown when the external payment service is unavailable. The transaction is recorded for retry. `PipelineFatalException`, thrown when there is a configuration error, a general service error, or a system-level exception from a back-end component.

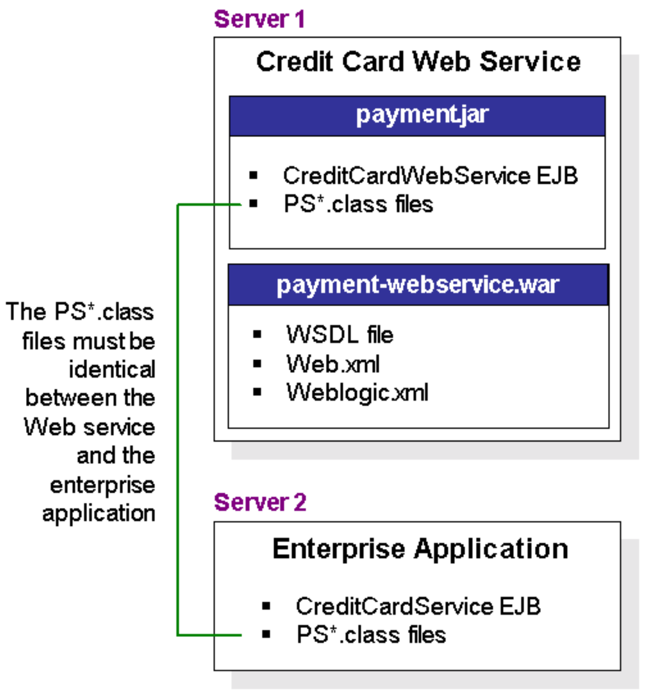
Integrating with a Payment Service

The Credit Card Web Service that is installed with WebLogic Portal provides a default framework for handling authorization, capture, and settlement of credit card transactions received from the default CreditCardService EJB in the enterprise application. The business methods within the CreditCardService EJB implement a standard workflow that is associated with the completion of credit card transactions, and the current state of the transaction is maintained and each action is journaled. (The Credit Card Web Service, by comparison, is a stateless session EJB wrapped in code that makes it a Web service.)

Integrating your enterprise applications with the payment Web service involves modifying either the CreditCardService EJB and/or the Credit Card Web Service, depending on who will host the Web service: your organization or the third-party payment vendor.

In either case, it helps to understand the connection relationship between the pieces in the WebLogic Portal payment services and the pieces in the Credit Card Web Service. Figure 6-5 illustrates the connection between the two.

Figure 6-5 The Relationship Between the Credit Card Web Service and the CreditCardService EJB



Caution: The default payment Web service that is shipped with WebLogic Portal always sends payment information through without any errors, as if it were connected to and approved by a third-party payment service. This default processing of payment is not designed for production use. You must integrate with your third-party vendor's payment service to process payment correctly.

If the Third-Party Vendor Hosts the Web Service

If the third-party vendor hosts the Credit Card Web Service, the vendor will integrate the Web service with their product's API.

Here is what your organization must do to connect to the vendor-hosted Web service:

1. If the vendor has modified any of `PS*.class` files in the Web service's `payment.jar` file, copy those modifications in your enterprise application. You can find the source code for these classes in:

   ```
   PORTAL_HOME\applications\wlcsApp\src\examples\wlcs\
   sampleapp\payment\PS*.java
   ```

   Compile the source files.

2. Make any vendor-required modifications to the CreditCardService EJB in your enterprise application so that it makes appropriate SOAP calls to the vendor's payment Web service. You can find the source code for the CreditCardService EJB in the following files:

   ```
   PORTAL_HOME\applications\wlcsApp\src\examples\wlcs\
   sampleapp\payment\CreditCardService*.java
   ```

   Compile the source files.

3. After you compile your source code, add the class files to the `wlcsSample.jar` in `wlcsApp` application directory. When you add the class files to the JAR, make sure you maintain their relative directory structure.

4. Run the EJB compiler (`ejbc`) on the `wlcsSample.jar` file.

5. In the WebLogic Server Console for portalServer, select Deployments > Applications > wlcsApp > Service Configuration > Payment Service Client, and in the Payment Web Service WSDL field, modify the URL to the payment vendor's WSDL file. Click Apply in the Console to apply the new URL. The new URL is written to the following file: `PORTAL_HOME\applications\wlcsApp\META-INF\application-config.xml`.

   Note: At startup, WebLogic Server reads the `application-config.xml` file, so it knows where to find the Web service.

If Your Organization Hosts the Web Service

If your organization hosts the Credit Card Web Service, we strongly recommend that you deploy the Web service on a separate instance of WebLogic Server (that is, use a separate Java Virtual Machine [JVM]) than what your enterprise applications are running on. This way, your enterprise applications are insulated from failures or incidents in the Web service.

Here is what you must do if your organization hosts the Credit Card Web Service:

Caution: These are general, simplified guidelines for integrating with a vendor's API. In actual practice, such integration requires close collaboration with your vendor. We strongly recommend you contact your vendor for assistance.

1. Obtain your third-party vendor's payment product API.
2. Modify the CreditCardWebService EJB (the Web service EJB) so that it translates SOAP calls into the language of the third-party product's API. (See "Default Payment Services Shipped With WebLogic Portal" on page 6-30.) You can find the source code for the CreditCardWebService EJB in the following files:

   ```
   PORTAL_HOME\applications\paymentWSApp\src\examples\wlcs\
   sampleapp\payment\CreditCardWebService*.java
   ```

   Compile the source files.
3. After you have compiled the source code, replace the class files in `payment.jar`, located in the `paymentWSApp` directory. When you add the class files to the JAR, make sure you maintain their relative directory structure.
4. Use the Web service generator (`wsgen`) on the `payment.jar` file to build a file called `payment-webservice.war`, as shown in Figure 6-5.

   For information on using `wsgen`, see *Programming WebLogic Server Web Services* at http://e-docs.bea.com/wls/docs61/webServices/index.html.
5. Make any necessary modifications to the CreditCardService EJB in the `wlcsApp` application so that it makes appropriate SOAP calls to the CreditCardWebService EJB. You can find the source code for the CreditCardService EJB in the following files:

`PORTAL_HOME\applications\wlcsApp\src\examples\wlcs\sampleapp\payment\CreditCardService*.java`

Compile the source files.

6. After you compile the source files, add the class files to `wlcsSample.jar` in the `wlcsApp` application directory. When you add the file to the JAR, maintain its relative directory structure.
7. Run the EJB compiler (`ejbc`) on the `wlcsSample.jar` file.
8. In the WebLogic Server Console for portalServer, select Deployments > Applications > wlcsApp > Service Configuration > Payment Service Client, and in the Payment Web Service WSDL field, modify the URL to the payment WSDL file. Click Apply in the Console to apply the new URL. The new URL is written to the following file: `PORTAL_HOME\applications\wlcsApp\META-INF\application-config.xml`.

   Note: At startup, WebLogic Server reads the `application-config.xml` file, so it knows where to find the Web service.

Default Payment Services Shipped With WebLogic Portal

The CreditCardWebService EJB is a stateless session bean that provides services related to the authorization, capture, and settlement of credit card transactions. The CreditCardWebService EJB serves as an interface behind which integrations with various payment solutions can be implemented. The current state of each transaction is maintained, and each action is journaled, by the CreditCardService EJB in the `wlcsApp` application.

Caution: The CreditCardWebService EJB that is shipped with WebLogic Portal is designed to give you an example of the different payment services you can use. The default Web service always sends payment information through without any errors, as if it were connected to and approved by a third-party payment service. This default processing of payment is not designed for production use. You must integrate with your third-party vendor's payment service to process payment correctly.

General characteristics of transactions are described in the following list:

- Each transaction is initiated with a request to authorize. This authorization generally results in the creation of a persistent `PaymentTransaction`. The state of the payment and the key for that `PaymentTransaction` is returned in a `TransactionResponse` as well as service specific information. A handle for that `PaymentTransaction` can be obtained from the `TransactionResponse`.

- In the event that the initial authorization fails due to a failure to connect to the payment authorization service, it is possible to retry the authorization using the reauthorize method.
- An authorized transaction can be captured or settled depending on how the service is configured.
- An entire transaction can be completed in a single `AuthorizeAndCapture`.

You can configure the Payment service to work with your business model. The methods/entry points are described in detail in the sections that follow.

Authorize

Use this method in the CreditCardService EJB only for terminal-based payment models. This entry point validates the credit card number and reserves credit on the supplied card for the amount specified. When validated, it creates a new entry in the `WLCS_TRANSACTION` table that records the incident and sets the state based on the payment model. The amount of the transaction is deducted from the *open to buy* in the customer's credit balance. However, the funds are not transferred to the merchant until settling.

Note: Merchants who are using a terminal-based processor must perform a capture and settlement procedure before the funds from the sale are transferred to their account. This is accomplished by a subsequent call to `Capture` and/or `Settle`, depending on the Auto Mark/Auto Settle processor configuration.

AuthorizeAndCapture

Use this method only for host-based payment models. This entry point validates the credit card number and reserves credit on the supplied card for the amount specified. When validated, it creates a new entry in the `WLCS_TRANSACTION` table that records the incident and sets the state based on the payment model. The amount of the transaction is deducted from the *open to buy* in the customer's credit balance. However, the funds are not transferred to the merchant until settling.

Note: Merchants who are using a host-based post-authorization capture processor must perform a capture and settlement procedure before the funds from the sale are transferred to their account.

BatchQuery

Use this method to update and reconcile the status of a transaction committed in a given batch. This entry point determines if a particular transaction has failed, and is essential for payment processors where the status of an item cannot be determined correctly from the output fields of a batch-commit message. `BatchQuery` always returns success on the query and creates a `TransactionEntry` to reflect this success.

Note: This method is implemented as a pass through to the underlying service provider. Subsequently, all return information is service specific. For details on return codes and results, see your service providers documentation.

QueryTransactions

Use this method to query the Payment server for transactions that match the designated parameters. You need only to supply non-null values for those parameters that you want to query against. However, you must supply at least one non-null parameter. Always returns OK.

Note: This method is implemented as a pass through to the underlying service provider. Subsequently, all return information is service specific. For details on return codes and results, see your service provider documentation.

Reauthorize

Use this method only for terminal-based payment models. This method attempts to authorize a payment transaction that is in the retry state. After authorization attempt, the payment transaction is updated with the current date and a transaction entry is added to the payment transaction. The modified payment transaction and any service-specific results are then returned.

ReauthorizeAndCapture

Use this method only for host-based payment models. This method attempts to authorize a payment transaction that is in the retry state. After authorization attempt, the payment transaction is updated with the current date and a transaction entry is added to the payment transaction. The modified payment transaction and any service-specific results are then returned.

Settle

Use this method only for terminal-based payment models with a manual-settle processor configuration. This method finalizes a transaction by transferring a portion of the funds previously captured from the customer's account to the merchant's account. The amount can be less than or equal to the captured amount. Always returns settle success.

VoidTransaction

This method aborts previously submitted transactions. Returns OK. The following transactions can be voided:

- `HOST_AUTH_CAPTURE` transactions in the pending settlement state.
- `HOST_POST_AUTH_CAPTURE` transactions in the pending settlement state.
- `AUTO_MARK_AUTO_SETTLE` transactions in the pending settlement state.
- `MANUAL_MARK_AUTO_SETTLE` transactions in the pending settlement state.
- `AUTO_MARK_MANUAL_SETTLE` transactions in the captured state.
- `MANUAL_MARK_MANUAL_SETTLE` transactions in the captured state.

Credit Card Encryption

For information on credit card encryption, see "Credit Card Security Service" in the *Security Guide*.

CHAPTER

7 Order Summary and Confirmation Services

Prior to submitting their order, your customers will want to review an order summary that includes information about the items they have decided to purchase, as well as other information (shipping, payment, and tax) related to their order. Following order submission, it is customary to provide your customers with a confirmation page, which customers can save and later use to check on the status of their order. The Order Summary and Confirmation Services allow you to do just that, and this topic describes how.

This topic includes the following sections:

- JavaServer Pages (JSPs)
  - checkout.jsp Template
  - confirmorder.jsp Template
- Input Processors
- Pipeline Components
  - CommitOrderPC
  - ResetCheckoutPC
  - PurchaseTrackerPC

JavaServer Pages (JSPs)

This section describes the JavaServer Pages (JSPs) used to implement the Order Summary and Confirmation Services. You can use them on your own e-commerce site, or customize them to meet your requirements.

Note: For a description of the complete set of JSPs used in the WebLogic Portal Web application and a listing of their locations in the directory structure, see the *E-Commerce JSP Template Summary*.

checkout.jsp Template

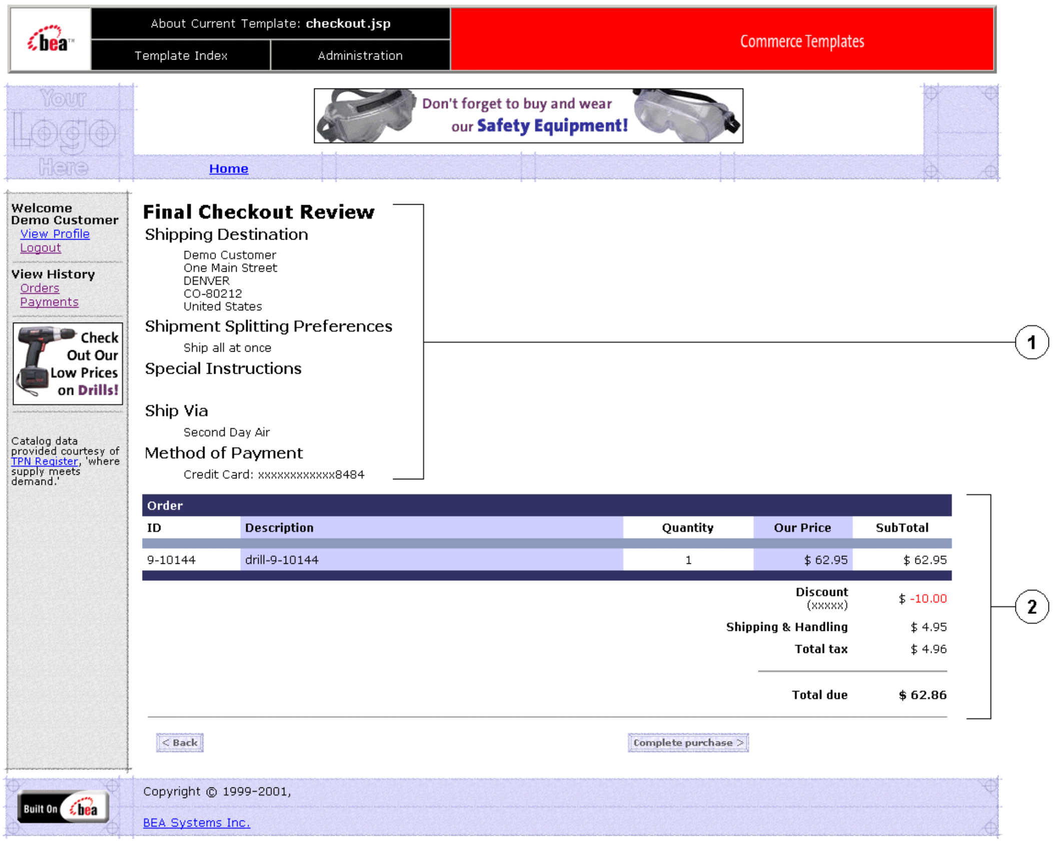
The `checkout.jsp` template (shown in Figure 7-1) provides a customer with a final look at all the details of their order, before the customer commits or cancels the order. Information displayed includes the shipping address, shipping details, a list of the items ordered (including the item name, short description, quantity, price, and subtotal), shipping and handling costs, tax costs, and total cost.

Customers must click the Complete Purchase button to commit their order. Customers wishing to return to the previous page can click the Back button instead.

Sample Browser View

Figure 7-1 shows an annotated version of the `checkout.jsp` template. A description of the annotated regions follow the figure.

Figure 7-1 Annotated checkout.jsp Template



Final Checkout Review

Shipping Destination

Demo Customer
One Main Street
DENVER
CO-80212
United States

Shipment Splitting Preferences

Ship all at once

Special Instructions

Ship Via

Second Day Air

Method of Payment

Credit Card: xxxxxxxxxxxx8484

Order

ID	Description	Quantity	Our Price	SubTotal
9-10144	drill-9-10144	1	$ 62.95	$ 62.95

Discount (xxxxx)	$ -10.00
Shipping & Handling	$ 4.95
Total tax	$ 4.96
Total due	$ 62.86

The numbers in the following list refer to the numbered regions in the figure:

1. The Final Checkout Review region uses a combination of WebLogic Portal and Pipeline JSP tags to obtain and display the shipping address, splitting preferences, and shipping method. This provides the customer with a final look at this shipping information as it was entered on previous JSP templates.

2. The Order region uses a combination of WebLogic Portal and Pipeline JSP tags to obtain and display the customer's current shopping cart. This provides the customer with a final look at the contents of their shopping cart (including item name, description, quantity, price, and subtotal), and the discount, shipping, tax, and total amounts for the entire order.

Location in the WebLogic Portal Directory Structure

You can find the `checkout.jsp` template file at the following location, where `PORTAL_HOME` is the directory in which you installed WebLogic Portal:

`%PORTAL_HOME%\applications\wlcsApp\wlcs\commerce\order\checkout.jsp` (Windows)
`$PORTAL_HOME/applications/wlcsApp/wlcs/commerce/order/checkout.jsp` (UNIX)

Tag Library Imports

The `checkout.jsp` template uses existing WebLogic Server JSP tags, and WebLogic Portal's User Management and Personalization JSP tags. It also uses Pipeline JSP tags. Therefore, the template includes the following JSP tag libraries:

```
<%@ taglib uri="webflow.tld" prefix="webflow" %>
<%@ taglib uri="weblogic.tld" prefix="wl" %>
<%@ taglib uri="um.tld" prefix="um" %>
<%@ taglib uri="es.tld" prefix="es" %>
<%@ taglib uri="i18n.tld" prefix="i18n" %>
```

For more information on the WebLogic Server JSP tags or the WebLogic Portal JSP tags, see "JSP Tag Reference" in the *Guide to Building Personalized Applications*. For more information about the Pipeline JSP tags, see the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline*.

These files reside in the `lib` directory within `PORTAL_HOME`.

Java Package Imports

The checkout.jsp template uses Java classes in the following packages and therefore includes these import statements:

```
<%@ page import="java.util.*" %>
<%@ page import="java.text.*" %>
<%@ page import="com.beasys.commerce.axiom.units.*" %>
<%@ page import="com.beasys.commerce.axiom.contact.*" %>
<%@ page import="examples.wlcs.sampleapp.shoppingcart.*" %>
<%@ page import="examples.wlcs.sampleapp.price.service.DiscountPresentation" %>
<%@ page import="examples.wlcs.sampleapp.price.quote.OrderAdjustment" %>
<%@ page import="examples.wlcs.sampleapp.price.quote.AdjustmentDetail" %>
<%@ page import="examples.wlcs.sampleapp.price.quote.AdjustmentType" %>

<%@ page import="com.beasys.commerce.webflow.HttpRequestConstants" %>
<%@ page import="com.beasys.commerce.webflow.PipelineSessionConstants" %>
<%@ page import="com.bea.p13n.appflow.webflow.WebflowJSPHelper" %>
```

Location in Default Webflow

Customers arrive at the checkout.jsp template from the payment information page (payment.jsp). If customers choose to commit their order, they will continue to the order confirmation page (confirmorder.jsp). If customers choose to cancel, they will be sent back to the payment page (payment.jsp).

Note: For more information about the default Webflow, see “Overview of Managing Purchases and Processing Orders” on page 1-1.

Events

The checkout.jsp template presents a customer with two buttons, each of which is considered an event. These events trigger a particular response in the default Webflow that allows customers to continue. While this response can be to load another JSP, it is usually the case that an input processor or Pipeline is invoked first. Table 7-1 provides information about these events and the business logic they invoke.

Table 7-1 checkout.jsp Events

Event	Webflow Response(s)
button.back	No business logic required. Loads payment.jsp.

Table 7-1 checkout.jsp Events

Event	Webflow Response(s)
`button.purchase`	`CommitOrder`

Table 7-2 briefly describes each of the Pipelines from Table 7-1. For more information about individual Pipeline components, see "Pipeline Components" on page 7-20.

Table 7-2 Checkout Review Pipelines

Pipeline	Description
`CommitOrder`	Contains `CommitOrderPC`, `AuthorizePaymentPC`, `CalculateTaxLineLevelCommitPC`, and is transactional.
`PurchaseTracker`	Contains `PurchaseTrackerPC`, `ResetCheckoutPC`, and is not transactional.

Dynamic Data Display

The purpose of the `checkout.jsp` template is to display the data specific to a customer's shopping experience for their final review. This is accomplished on the `checkout.jsp` template using a combination of Pipeline and WebLogic Portal JSP tags and accessor methods/attributes.

First, the `getProfile` JSP tag is used to set the customer profile (context) for which the customer information should be retrieved, as shown in Listing 7-1.

Listing 7-1 Setting the Customer Context

```
<um:getProfileprofileKey="<%=request.getRemoteUser()%>"
profileType="WLCS_Customer" />
```

Note: For more information on the WebLogic Portal JSP tags, see "JSP Tag Reference" in the *Guide to Building Personalized Applications*.

Next, the `getProperty` JSP tag retrieves the `SHIPPING_ADDRESS` and `SHOPPING_CART` attributes from the Pipeline session. Table 7-3 provides more detailed information on these attributes.

Table 7-3 checkout.jsp Pipeline Session Attributes

Attributes	Type	Description
`PipelineSessionConstants.SHIPPING_ADDRESS`	`com.beasys.commerce.axiom.contact.Address`	The address the order is being shipped to.
`PipelineSessionConstants.SHIPPING_METHOD`	`examples.wlcs.sampleapp.shipping.shippingMethodValue`	Identifies the shipping method the customer selected.
`PipelineSessionConstants.SHOPPING_CART`	`examples.wlcs.sampleapp.shoppingcart.ShoppingCart`	The shopping cart that was ordered.
`PipelineSessionConstants.SPLITTING_PREFERENCE`	`java.lang.String`	The splitting preference the customer selected.
`PipelineSessionConstants.SPECIAL_INSTRUCTIONS`	`java.lang.String`	Any special instructions the customer specifies.
`PipelineSessionConstants.ORDER_ADJUSTMENTS`	`examples.wlcs.sampleapp.price.quote.Quote`	Adjustments to the order and order lines.
`PipelineSessionConstants.PAYMENT_CREDIT_CARD`	`com.beasys.commerce.axiom.contact.CreditCard`	The user's credit card.

Listing 7-2 illustrates how some of these attributes are retrieved from the Pipeline session.

Listing 7-2 Retrieving Check Out Attributes

```
<webflow:getProperty id="shippingMethodValue"
property="<%=PipelineSessionConstants.SHIPPING_METHOD%>"
type="examples.wlcs.sampleapp.shipping.ShippingMethodValue" scope="session"
namespace="sampleapp_main" />

<webflow:getProperty id="shippingAddress"
property="<%=PipelineSessionConstants.SHIPPING_ADDRESS%>"
type="com.beasys.commerce.axiom.contact.Address" scope="session"
namespace="sampleapp_main" />
```

```
<webflow:getProperty id="shoppingCart"
property="<%=PipelineSessionConstants.SHOPPING_CART%>"
type="examples.wlcs.sampleapp.shoppingcart.ShoppingCart" scope="session"
namespace="sampleapp_main" />
```

Note: For more information on the `getProperty` JSP tag, see the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline*.

For the data stored in the customer profile and retrieved using the `getProfile` JSP tag, use the `getPropertyAsString` JSP tag to display the customer information, as shown in Listing 7-3.

Listing 7-3 Displaying Data Stored in the Customer's Profile

```
<div class="tabletext">
  <um:getPropertyAsString propertySet="CustomerProperties"
propertyName="firstName"/> <um:getPropertyAsString
propertySet="CustomerProperties" propertyName="lastName"/><br>
    <%=shippingAddress.getStreet1()%><br>
    <!-- implent street2 using es -->
    <% if( shippingAddress.getStreet2().length() != 0 ) { %>
    <%=shippingAddress.getStreet2()%><br>
    <% } %>
    <%=shippingAddress.getCity()%><br>
    <%String stateZip  = shippingAddress.getState()+ "-" +
shippingAddress.getPostalCode();%>
    <%=stateZip%><br>
    <%= shippingAddress.getCountry() %><br>
</div>
```

Note: For more information on the WebLogic Portal JSP tags, see "JSP Tag Reference" in the *Guide to Building Personalized Applications*.

The data stored within the Pipeline session attributes (retrieved using the `getProperty` JSP tag) is displayed by using accessor methods/attributes within Java scriptlets. Table 7-4 provides more detailed information on these methods/attributes for `Address`, `ShoppingCart`, and `ShoppingCartLine`.

Table 7-4 Address Accessor Methods/Attributes

Method/Attribute	Description
`getStreet1()`	The first line in the customer's street address.
`getStreet2()`	The second line in the customer's street address.
`getCity()`	The city in the customer's address.
`getState()`	The state in the customer's address.
`getPostalCode()`	The zip/postal code in the customer's address.
`getCountry()`	The country in the customer's address.

Table 7-5 ShoppingCart Accessor Methods/Attributes

Method/Attribute	Description
`getShoppingCartLineCollection()`	The individual lines in the shopping cart (that is, `ShoppingCartLine`).
`getTotal(int totalType)`	The total amount specified by the `totalType` parameter. The relevant parameter is: `ShoppingCartConstants.LINE_TAX` **Note:** The `getTotal()` method also allows you to combine different total types. For more information, see the Javadoc.

Because the `getShoppingCartLineCollection()` method allows you to retrieve a collection of the individual lines within a shopping cart, there are also accessor methods/attributes you can use to break apart the information contained within each line. Table 7-6 provides information about these methods/attributes.

Table 7-6 ShoppingCartLine Accessor Methods/Attributes

Method/Attribute	Description
`getQuantity()`	The quantity of the item.
`getProductItem()`	The product item in the shopping cart line.
`getUnitPrice()`	The current price for the item at the time it was added to the shopping cart. May be different from MSRP.
`getBaseTotal()`	The total before discounts.
`getDiscountPresentations()`	Returns an array list of `DiscountPresentation` objects.

Listing 7-4 illustrates how these accessor methods/attributes are used within Java scriptlets.

Listing 7-4 Using Accessor Methods/Attributes Within checkout.jsp Java Scriptlets

```
<wl:repeat set="<%=shoppingCart.getShoppingCartLineCollection().iterator()%>"
id="shoppingCartLine" type="ShoppingCartLine" count="100000">
  <tr>
    <td colspan="8" bgcolor="#899ABC"><img src="<webflow:createResourceURL
resource="/commerce/images/shim.gif" />" width="62" height="1"></td>
  </tr>

  <tr>
    <td nowrap valign="top">
      <div class="tabletext"><%=
shoppingCartLine.getProductItem().getKey().getIdentifier() %>
      </div>
    </td>

    <td valign="top" bgcolor="#CCCCFF">
      <div class="tabletext"><%= shoppingCartLine.getProductItem().getName() %>
      </div>
    </td>

    <td align="center" valign="top">
```

```
      <div class="tabletext"><%= WebflowJSPHelper.quantityFormat(
shoppingCartLine.getQuantity()) %>
      </div>
    </td>

    <td align="right" valign="top" bgcolor="#CCCCFF" nowrap>
      <div class="tabletext">
     <%-- The i18n tag allows the "currency.properties" file to substitute a display --%>
     <%-- currency value (e.g "$") for the returned 3 letter ISO4217 code (e.g. "USD").
--%>
      <i18n:getMessage bundleName="/commerce/currency" messageName="<%=
shoppingCartLine.getUnitPrice().getCurrency() %>"/>&nbsp;<%=
WebflowJSPHelper.priceFormat( shoppingCartLine.getUnitPrice().getValue() ) %>
      </div>
    </td>

    <td align="right" valign="top" nowrap><div class="tabletext" nowrap>
      <% // Calculate the Subtotal
      //double lineTotal = (shoppingCartLine.getQuantity() *
shoppingCartLine.getUnitPrice().getValue());
      %>
      <i18n:getMessage bundleName="/commerce/currency" messageName="<%=
shoppingCartLine.getBaseTotal().getCurrency() %>"/>&nbsp;<%=
WebflowJSPHelper.priceFormat( shoppingCartLine.getBaseTotal().getValue() ) %>
      </div>
    </td>
  </tr>
...
</wl>
```

Form Field Specification

The `checkout.jsp` template does not make use of any form fields.

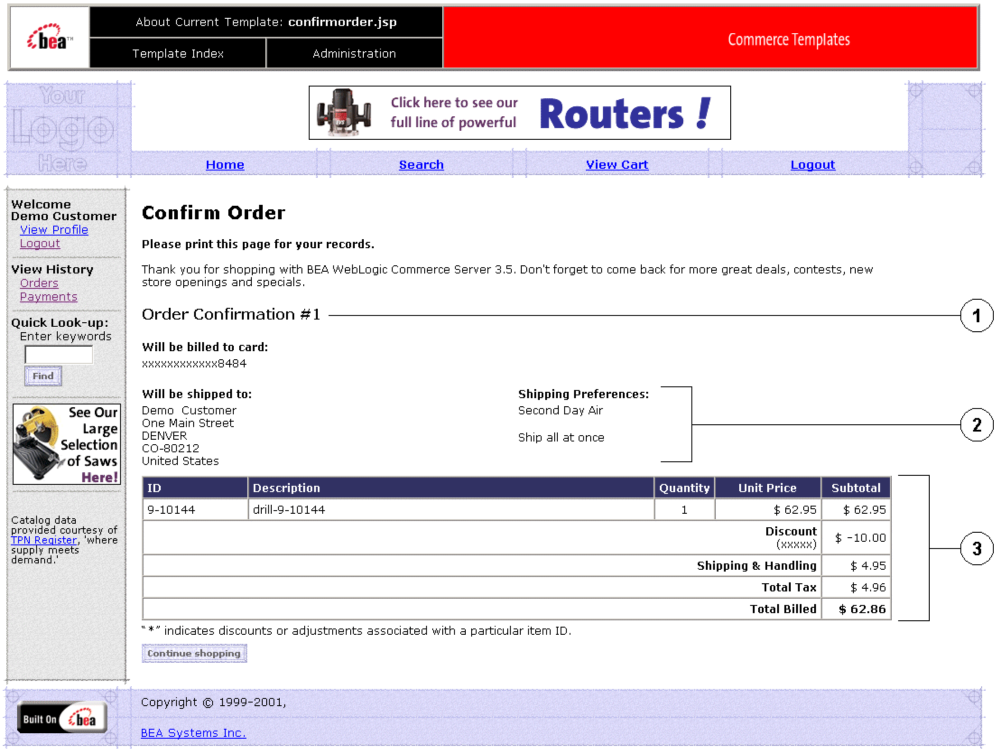
confirmorder.jsp Template

The `confirmorder.jsp` template (shown in Figure 7-2) displays the information about the customer's order after they have committed it. This information is the same as that shown in the `checkout.jsp` template, but also includes an order confirmation number customers can use to access information about the order in the future. The `confirmorder.jsp` template also provides the customer with a Continue Shopping button that will bring the customer back to the product catalog.

Sample Browser View

Figure 7-2 shows an annotated version of the `confirmorder.jsp` template. A description of the annotated regions follow the figure.

Figure 7-2 Annotated confirmorder.jsp Template



Catalog data provided courtesy of TPN Register, 'where supply meets demand.'

Confirm Order

Please print this page for your records.

Thank you for shopping with BEA WebLogic Commerce Server 3.5. Don't forget to come back for more great deals, contests, new store openings and specials.

Order Confirmation #1

Will be billed to card:
xxxxxxxxxxxx8484

Will be shipped to:
Demo Customer
One Main Street
DENVER
CO-80212
United States

Shipping Preferences:
Second Day Air

Ship all at once

ID	Description	Quantity	Unit Price	Subtotal
9-10144	drill-9-10144	1	$ 62.95	$ 62.95
			Discount (xxxxx)	$ -10.00
			Shipping & Handling	$ 4.95
			Total Tax	$ 4.96
			Total Billed	$ 62.86

"*" indicates discounts or adjustments associated with a particular item ID.

Copyright © 1999-2001,
BEA Systems Inc.

The numbers in the following list refer to the numbered regions in the figure:

1. This region contains the dynamically generated order confirmation number, which customers can use on subsequent visits to check the status of their order. It is displayed using Pipeline JSP tags and accessor methods/attributes.
2. This region uses a combination of WebLogic Portal and Pipeline JSP tags to obtain and display the shipping address, splitting preferences, and shipping method. Together with the information in region 4 and region 6, this provides the customer with a record of the shipping information as it was entered on previous JSP templates.
3. This region uses a combination of WebLogic Portal and Pipeline JSP tags to obtain and display the customer's shopping cart. Together with the information in region 4 and region 5, this provides the customer with a record of their shopping cart (including item name, description, quantity, price, and subtotal), and the shipping, tax, and total amounts for the order.

Location in the WebLogic Portal Directory Structure

You can find the `confirmorder.jsp` template file at the following location, where `PORTAL_HOME` is the directory in which you installed WebLogic Portal:

`%PORTAL_HOME%\applications\wlcsApp\wlcs\commerce\order\confirmorder.jsp` (Windows)
`$PORTAL_HOME/applications/wlcsApp/wlcs/commerce/order/confirmorder.jsp` (UNIX)

Tag Library Imports

The `confirmorder.jsp` template uses existing WebLogic Server and WebLogic Portal's User Management and Personalization JSP tags. It also uses Pipeline JSP tags. Therefore, the template includes the following JSP tag libraries:

```
<%@ taglib uri="webflow.tld" prefix="webflow" %>
<%@ taglib uri="weblogic.tld" prefix="wl" %>
<%@ taglib uri="um.tld" prefix="um" %>
<%@ taglib uri="es.tld" prefix="es" %>
<%@ taglib uri="i18n.tld" prefix="i18n" %>
```

Note: For more information on the WebLogic Server JSP tags or the WebLogic Portal JSP tags, see “JSP Tag Reference” in the *Guide to Building Personalized Applications*. For more information about the Pipeline JSP tags, see the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline*.

These files reside in the `lib` directory within `PORTAL_HOME`.

Java Package Imports

The `confirmorder.jsp` template uses Java classes in the following packages and therefore includes these import statements:

```
<%@ page import="java.util.*" %>
<%@ page import="java.text.*" %>
<%@ page import="com.beasys.commerce.axiom.units.*" %>
<%@ page import="com.beasys.commerce.axiom.contact.*" %>
<%@ page import="com.beasys.commerce.axiom.util.helper.*;" %>
<%@ page import="examples.wlcs.sampleapp.order.*" %>
<%@ page import="examples.wlcs.sampleapp.catalog.*" %>
<%@ page import="examples.wlcs.sampleapp.shipping.*" %>
<%@ page import="com.beasys.commerce.util.*; " %>

<%@ page import="com.beasys.commerce.webflow.HttpRequestConstants" %>
<%@ page import="com.beasys.commerce.webflow.PipelineSessionConstants" %>
<%@ page import="com.bea.p13n.appflow.webflow.WebflowJSPHelper" %>
```

Location in Default Webflow

Customers arrive at `confirmorder.jsp` template from the final checkout page (`checkout.jsp`). The default Webflow does not define a subsequent JSP template.

The template is in the sampleapp_order namespace.

Note: For more information about the default Webflow, see “Overview of Managing Purchases and Processing Orders” on page 1-1.

Events

There are no events associated with the `confirmorder.jsp` template.

Dynamic Data Display

The purpose of the `confirmorder.jsp` template is to display the data specific to a customer's shopping experience along with a unique order confirmation number. This is accomplished on the `confirmorder.jsp` template using a combination of Pipeline and WebLogic Portal JSP tags and accessor methods/attributes.

First, the `getProfile` JSP tag is used to set the customer profile (context) for which the customer information should be retrieved, as shown in Listing 7-5.

Listing 7-5 Setting the Customer Context

```
<um:getProfile
  profileKey="<%=request.getRemoteUser()%>"
  profileType="WLCS_Customer" />
```

Note: For more information on the WebLogic Portal JSP tags, see "JSP Tag Reference" in the *Guide to Building Personalized Applications*.

Next, the `getProperty` JSP tag retrieves the `ORDER_VALUE` and `SHIPPING_METHOD` attributes from the Pipeline session. Table 7-7 provides more detailed information about these attributes.

Table 7-7 confirmorder.jsp Pipeline Session Attributes

Attribute	Type	Description
`PipelineSessionConstants.ORDER_VALUE`	List of `com.beasys.commerce.ebusiness.order.OrderValue`	List of the orders available for the customer.
`PipelineSessionConstants.SHIPPING_METHOD`	`examples.wlcs.sampleapp.shipping.ShippingMethodValue`	The method being used to ship the order.
`PipelineSessionConstants.CREDIT_CARD_KEY`	`java.lang.String`	The key of the credit card.

Listing 7-6 illustrates how these attributes are retrieved from the Pipeline session.

Listing 7-6 Retrieving Order Confirmation Attributes

```
<webflow:getProperty id="orderValue"
property="<%=PipelineSessionConstants.ORDER_VALUE%>" type="OrderValue"
scope="request" namespace="sampleapp_main" />

<webflow:getProperty id="creditCard"
property="<%=PipelineSessionConstants.CREDITCARD_KEY%>" type="java.lang.String"
scope="session" namespace="sampleapp_main" />

<webflow:getProperty id="shippingMethodValue"
property="<%=PipelineSessionConstants.ORDER_SHIPPING_METHOD%>"
type="examples.wlcs.sampleapp.shipping.ShippingMethodValue" scope="request"
namespace="sampleapp_main" />
```

Note: For more information on the `getProperty` JSP tag, see the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline*.

For the data stored in the customer profile and retrieved using the `getProfile` JSP tag, use the `getPropertyAsString` JSP tag to display the customer information, as shown in Listing 7-7.

Listing 7-7 Displaying Data Stored in the Customer's Profile

```
<um:getPropertyAsString propertySet="CustomerProperties"
propertyName="firstName" />&nbsp;

<um:getPropertyAsString propertySet="CustomerProperties"
propertyName="lastName" /><br>
```

Note: For more information on the WebLogic Portal JSP tags, see "JSP Tag Reference" in the *Guide to Building Personalized Applications*.

The data stored within the Pipeline session attributes (retrieved using the `getProperty` JSP tag) is displayed by using accessor methods/attributes within Java scriptlets. Table 7-8 through Table 7-11 provide more detailed information on these methods/attributes for `Address`, `ShippingMethodValue`, `OrderValue`, and `Orderline`.

Table 7-8 Address Accessor Methods/Attributes

Method/Attribute	Description
`getStreet1()`	The first line in the customer's street address.
`getStreet2()`	The second line in the customer's street address.
`getCity()`	The city in the customer's address.
`getState()`	The state in the customer's address.
`getPostalCode()`	The zip/postal code in the customer's address.
`getCountry()`	The country in the customer's address.

Table 7-9 ShippingMethodValue Accessor Methods/Attributes

Method/Attribute	Description
`description`	A description of the shipping method.
`identifier`	Key in the database for the shipping method.

Table 7-10 OrderValue Accessor Methods/Attributes

Method/Attribute	Description
`createdDate`	The date the customer's order was created.
`identifier`	Key in the database for the order.
`getTotal(int totalType)`	The total amount specified by the `totalType` parameter. The relevant parameter is `OrderConstants.LINE_TAX` **Note:** The `getTotal()` method also allows you to combine different total types. For more information, see the Javadoc.
`orderLines`	A collection of the lines in the shopping cart that make up the customer's order.

Table 7-10 OrderValue Accessor Methods/Attributes (Continued)

Method/Attribute	Description
`price`	The total price as a money object.

Because the `orderLines` attribute allows you to retrieve the individual lines within an order, it also has accessor methods/attributes you can use to display the information contained within each line. These methods/attributes are listed in Table 7-11.

Table 7-11 OrderLine Accessor Methods/Attributes

Method/Attribute	Description
`getProductIdentifier()`	The name (identifier) for the shopping cart item.
`getDescription()`	A description of the shopping cart item.
`getQuantity()`	The quantity of the shopping cart item.
`getUnitPrice()`	The unit price for the shopping cart item.

Listing 7-8 illustrates how these accessor methods/attributes are used within Java scriptlets.

Listing 7-8 Using Accessor Methods Within confirmorder.jsp Java Scriptlets

```
<%--Iterate through order to get all order lines --%>
<wl:repeat set="<%=orderValue.orderLines.iterator()%>" id="orderLine"
type="OrderLine" count="100000">
  <tr>
    <td valign="top" align="left" nowrap>
      <div class="tabletext"><%= orderLine.getProductIdentifier() %></div>
    </td>

    <td valign="top" align="left">
      <div class="tabletext"><%= orderLine.getDescription() %></div>
    </td>

    <td align="center" valign="top">
```

```
      <div class="tabletext"><%= WebflowJSPHelper.quantityFormat(
orderLine.getQuantity()) %></div>
    </td>

    <td align="right" valign="top" nowrap>
      <div class="tabletext">
      <i18n:getMessage bundleName="/commerce/currency" messageName="<%=
orderLine.getUnitPrice().getCurrency() %>"/>&nbsp;<%=
WebflowJSPHelper.priceFormat( orderLine.getUnitPrice().getValue() ) %>
      </div>
      </td>

      <td align="right" valign="top" nowrap>
  <%
// Calculate the line subtotal without adjustments/discounts
double orderLineTotal = (orderLine.getQuantity() *
orderLine.getUnitPrice().getValue());
%>
        <div class="tabletext">
<i18n:getMessage bundleName="/commerce/currency" messageName="<%=
orderLine.getUnitPrice().getCurrency() %>"/>&nbsp;<%=
WebflowJSPHelper.priceFormat( orderLineTotal ) %>
</div>
      </td>
    </tr>
...
</wl>
```

For a code example of the `ShoppingCart` and `ShoppingCartLine` accessor methods/attributes, see “Shopping Cart Management Services” on page 3-1.

Form Field Specification

The `confirmorder.jsp` template does not make use of any form fields.

Input Processors

No input processors are used in the Order Summary and Confirmation Services JSP template(s).

Pipeline Components

This section provides a brief description of each Pipeline component associated with the Order Summary and Confirmation Services JSP template(s).

Note: Some Pipeline components extend other, base Pipeline components. For more information on the base classes, see the Javadoc.

CommitOrderPC

Class Name	`examples.wlcs.sampleapp.order.pipeline.CommitOrderPC`
Description	Reads all the information about a customer's order from the Pipeline session and creates an `Order` entity bean. This is committed to the database in the `WLCS_ORDER` and `WLCS_ORDER_LINE` tables. The `OrderValue` object for the order is then stored in the Pipeline session.

Required Pipeline Session Attributes	`PipelineSessionConstants.USER_NAME`
	`PipelineSessionConstants.SHOPPING_CART`
	`PipelineSessionConstants.SPLITTING_PREFERENCE`
	`PipelineSessionConstants.SPECIAL_INSTRUCTIONS`
	`PipelineSessionConstants.ORDER_CONFIRMATION_NUMBER`
	`PipelineSessionConstants.SHIPPING_ADDRESS`
	`PipelineSessionConstants.ORDER_ADJUSTMENTS`
	`PipelineSessionConstants.SHIPPING_METHOD`
	`PipelineSessionConstants.DISCOUNT_IDS`
	`PipelineSessionConstants.GLOBAL_DISCOUNTS_IDS`

Updated Pipeline Session Attributes	`PipelineSessionConstants.ORDER_HANDLE` (Request scope) `PipelineSessionConstants.ORDER_VALUE` (Request scope) `PipelineSessionConstants.ORDER_SHIPPING_METHOD` (Request scope) `PipelineSessionConstants.PAYMENT_AUTHORIZATION_ACCOUNT`
Removed Pipeline Session Attributes	`PipelineSessionConstants.SHIPPING_METHOD`
Type	Java object
JNDI Name	None
Exceptions	`PipelineException`, thrown when the required Pipeline session attributes are not available or if the shopping cart is empty.

ResetCheckoutPC

Class Name	`examples.wlcs.sampleapp.order.pipeline.` `ResetCheckoutPC`
Description	Removes all Pipeline session attributes relating to the customer's checkout process.
Required Pipeline Session Attributes	None
Updated Pipeline Session Attributes	None

Removed Pipeline Session Attributes	`PipelineSessionConstants.SHOPPING_CART`
	`PipelineSessionConstants.SHIPPING_ADDRESS`
	`PipelineSessionConstants.SPLITTING_PREFERENCE`
	`PipelineSessionConstants.SHIPPING_METHOD`
	`PipelineSessionConstants.SPECIAL_INSTRUCTIONS`
	`PipelineSessionConstants.PAYMENT_AUTHORIZATION_AMOUNT`
	`PipelineSessionConstants.VERAZIP_SHIPPING_ADDRESS`
	`PipelineSessionConstants.PAYMENT_CREDIT_CARD`

Type	Java object
JNDI Name	None
Exceptions	None

PurchaseTrackerPC

Class Name	`examples.wlcs.sampleapp.tracking.pipeline.PurchaseTracker PC`
Description	Fires events: first, a `PurchaseCartEvent` for the entire order that is being placed; second, one `BuyEvent` per Order Line (SKU) that is being purchased. For more information about this event, see Event Details in the *Guide to Events and Behavior Tracking*.

Required Pipeline Session Attributes	`PipelineSessionConstants.ORDER_VALUE`
	`PipelineSessionConstants.HTTP_SESSION_ID`
	`PipelineSessionConstants.USER_NAME`
	`PipelineSessionConstants.CATALOG_CATEGORY`
	`PipelineSessionConstants.STOREFRONT`
	`PipelineSessionConstants.CUSTOM_REQUEST`

Updated Pipeline Session Attributes	None
Removed Pipeline Session Attributes	None

Type	Java Object
JNDI Name	None
Exceptions	None

CHAPTER

8 Extending the Data Model

This chapter explains how to extend Order services. The following topics are discussed:

- Data Model Extensions
- Persistence Architecture
- Adding Run-Time Attributes to Customer Data
- Adding Run-Time Attributes to Other Entities
- Extending the Schema
  - Overview of Approach to Extending the WebLogic Portal Schema
  - Adding Attributes Against the 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

Data Model Extensions

Registering Customers and Managing Customer services and Order services are two core components of the WebLogic Portal 4.0. These services implement use-cases that deal with customer self-registration, customer management, shopping cart experience, and order processing (including shipping, payment and taxation).

These services implement 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 WebLogic Portal to provide more sophisticated and specialized commerce scenarios to meet your business needs. The Commerce services infrastructure of WebLogic Portal 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 to include custom input processors and Pipelines.

For more information on the WebFlow and Pipeline infrastructure, see the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline*.

One of the common requirements for implementing such extensions is the ability to access and extend the Commerce services 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 then process the order and payment data accordingly. Similarly, you may want 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 chapter presents some possible approaches and guidelines for extending the data model of WebLogic Portal. While this chapter does not guarantee automatic compatibility of such extensions with future releases of WebLogic Portal, the approaches discussed in this chapter try to minimize potential problems, by leveraging the WebFlow/Pipeline infrastructure.

This chapter addresses the following:

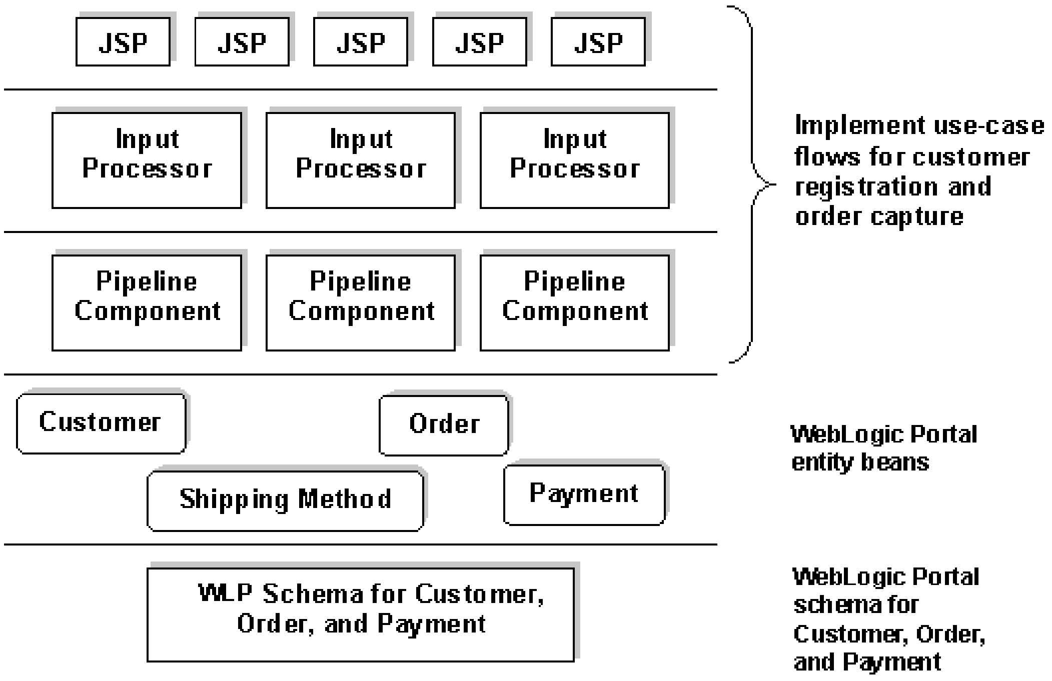
- The Commerce services persistence architecture.
- Adding run-time attributes to customer and order related entities.
- General approach for extending the data model and the schema.
- Extending the WLCS_CUSTOMER, WLCS_ORDER, WLCS_TRANSACTION and WLCS_SHIPPING_METHOD tables.
- Extending the WLCS_ORDER_LINE 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 chapter does not cover extensions to other Commerce services (such as extensions for building a product catalog). You can periodically check the WebLogic Portal documentation for future updates on how to extend other services.

Persistence Architecture

Before we go into the approaches for extending the WebLogic Portal Commerce services schema, consider the persistence architecture of Commerce services shown in Figure 8-1. This figure shows the persistence architecture for the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline*. In this structure, the JSP, Input Processor, and Pipeline component layers are responsible for implementing the use-case flow. Specific information on the JSPs, input processors, and Pipeline components in these layers are discussed throughout this chapter.

Figure 8-1 Persistence Architecture for Registering Customers and Managing Customer Services and Order Services


JSP
JSP
JSP
JSP
JSP
Input Processor
Input Processor
Input Processor
Pipeline Component
Pipeline Component
Pipeline Component
Implement use-case flows for customer registration and order capture
Customer
Order
Shipping Method
Payment
WebLogic Portal entity beans
WLP Schema for Customer, Order, and Payment
WebLogic Portal schema for Customer, Order, and Payment

Pipeline components rely on the following WebLogic Portal entity beans for persisting customer, order, payment, and shipping method data respectively:

- `examples.wlcs.sampleapp.customer.Customer`
- `examples.wlcs.sampleapp.order.Order`
- `examples.wlcs.sampleapp.payment.PaymentTransaction`
- `examples.wlcs.sampleapp.shipping.ShippingMethod`

For persistence, these entities use the WebLogic Portal tables discussed in the Chapter 10, "The Order Processing Database Schema."

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

Table	Description
Entity: `examples.wlcs.sampleapp.customer.Customer`	
`WLCS_CUSTOMER`	Customer description
`WLCS_CREDIT_CARD`	Credit cards
`WLCS_SHIPPING_ADDRESS`	Shipping address
Entity: `examples.wlcs.sampleapp.order.Order`	
`WLCS_ORDER`	Order description
`WLCS_ORDER_LINES`	Order lines
Entity: `examples.wlcs.sampleapp.payment.PaymentTransaction`	
`WLCS_TRANSACTION`	Transaction description
`WLCS_TRANSACTION_ENTRY`	Transaction entries
Entity: `examples.wlcs.sampleapp.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 Run-Time 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 Portal, 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 Portal Commerce services 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 Portal is a component that relies on the Unified User Profile (UUP) technology of WebLogic Portal. A UUP for customer data allows the abstraction of a customer to be seamlessly integrated into WebLogic Portal. Apart from personalization, this approach allows you to use the user management tools of WebLogic Portal to administer customer data, and maps the customer identity into a WebLogic Portal-administered groups and the RDMBS security realm. For more information on unified user profiles, see “Creating and Managing Users” in the *Guide to Building Personalized Applications*.

In addition to the information in the previous paragraph, the notion of the 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 `examples.wlcs.sampleapp.customer.pipeline` package. To add attributes to the customer data, the WebLogic Portal Registration Package provides an abstract Pipeline component `examples.wlcs.sampleapp.customer.pipeline.UpdateUserPC`, as shown in Listing 8-1.

Listing 8-1 Adding Attributes to Customer Data

```
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 run-time 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 Portal 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, see the “JSP Tag Reference Library” in the *Guide to Building Personalized Applications*.

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. WebLogic Portal persists run-time attributes in tables that are internal to WebLogic Portal. Consequently, you cannot execute SQL level operations on such data.

Adding Run-Time Attributes to Other Entities

For the entities in Order services such as `examples.wlcs.sampleapp.shipping.Order`, `examples.wlcs.sampleapp.shipping.PaymentTransaction`, and `examples.wlcs.sampleapp.shipping.ShippingMethod`, there exists a similar mechanism for adding run-time attributes. All the entities in the Order services 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 the attribute later:

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

This method returns a previously added property.

For more information, including the API, see the JavaDoc.

Extending the Schema

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

- Extending the schema of the Commerce services to meet your existing schema.
- Enhancing the Commerce services to modify or add new functionality.

Both these drivers manifest in the following:

- Modifying (or sometimes adding) the templates to render and/or collect additional data from the user interface.

- Modifying the WebFlow to change the flow of user interaction.
- Extending the Commerce services schema.

Note: Almost all the data in the Order services is meaningful across your business, so you may want to apply SQL level semantics for creating, updating, and querying. Depending on the nature and scale of your commerce site, the Commerce services and your back-end applications may depend on this data. Any extension to the schema of the Order services cannot be represented with run-time attributes, as run-time 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 back-end order fulfillment application. You may want to display this tracking number on WebLogic Portal order history pages for customers to view the tracking information. This is a domain-specific attribute that can best be persisted in the WLCS_ORDER table (or another table that you created for this purpose).

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

1. Adding attributes against the WLCS_CUSTOMER, WLCS_ORDER, WLCS_TRANSACTION, and WLCS_SHIPPING_METHOD tables.
2. Adding attributes against the 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 Portal Schema

The following figure presents an overview of the approach for extending the Commerce services schema and *not* for integrating the Commerce services schema with your existing schema or for mapping the Commerce services schema onto your existing schema.

Figure 8-2 Extending the Data Model

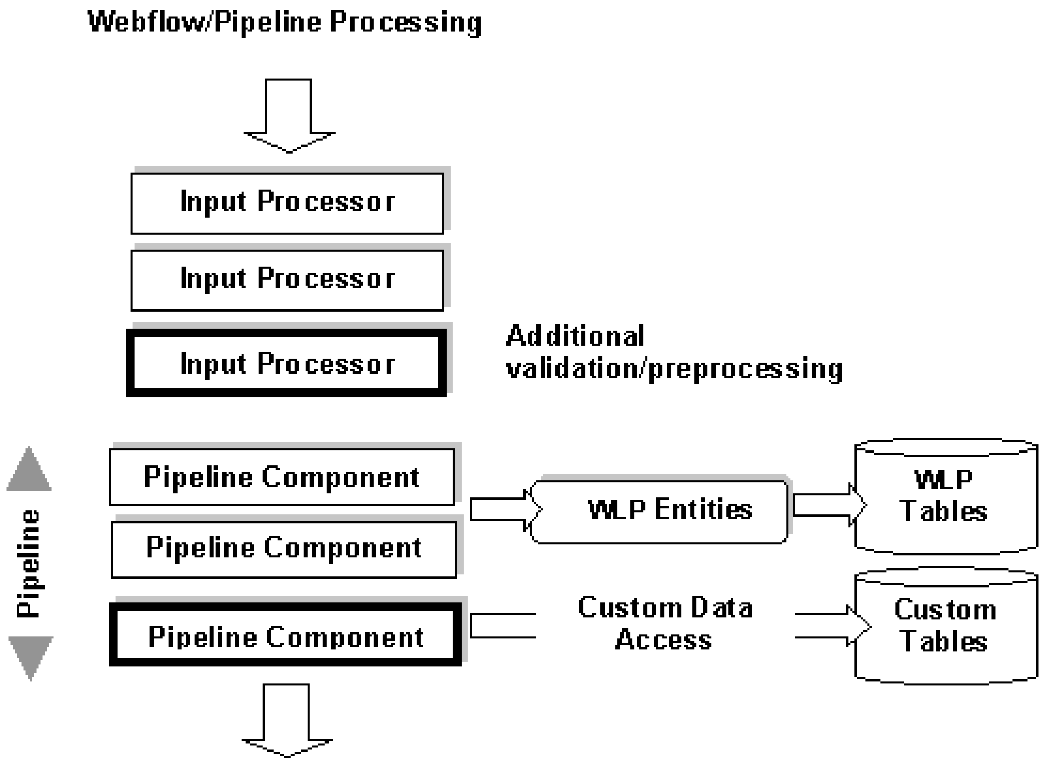


Figure 8-2 demonstrates how a given WebFlow/Pipeline processing can be modified to process additional data, without modifying existing input processors and Pipeline components. In Figure 8-2, the blocks with heavy borders are new input processors and Pipeline components inserted to process the additional data. While the Commerce services Pipeline components manage the Commerce services data via the Commerce services 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 within the same transaction, within a new transaction, or no transaction at all, you can split the above Pipeline into more than one Pipeline where each will have its own transaction setting.

Adding Attributes Against the 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 Commerce services 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 Commerce services.

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 `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 Portal Commerce services, 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 Commerce services, examine the code snippet shown in Listing 8-2 in your new Pipeline component for obtaining the ORDER_LINE_ID for a given order line.

Listing 8-2 Implementing a New Pipeline Component

```
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, another step has to be perfomed, which 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 entry shown in Listing 8-3, and add the text marked in bold.

Listing 8-3 Updating the Deployment Descriptor

```
<weblogic-enterprise-bean>
    <ejb-name>
        examples.wlcs.sampleapp.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>
            examples.wlcs.sampleapp.order.Order
        </jndi-name>
    </weblogic-enterprise-bean>
```

- Jar the contents of the temporary directory, and run the EJB compiler 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 the new fields in the forms.

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

Modify the `examples.wlcs.sampleapp.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 `examples.wlcs.sampleapp.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.

To extract the additional data from the `PipelineSession` and write to the new tables, you need to implement a new Pipeline component. However, since the primary keys for the WLCS_CREDIT_CARD and WLCS_SHIPPING_ADDRESS tables are internal to the Commerce services, consider using the code snippet shown in Listing 8-4 in your new Pipeline component for obtaining the primary keys. Although this snippet describes the steps for credit card data, the same procedure applies to shipping address data.

Listing 8-4 Implementing a New Pipeline Component

```
// 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 the 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`.
- Go to c:\temp\ebusiness\META-INF, and open `weblogic-ejb-jar.xml` file.
- In this file, search for the entry shown in Listing 8-5, and then add the text marked in bold.

Listing 8-5 Modifying the Deployment Descriptor

```
<weblogic-enterprise-bean>
    <ejb-name>
        examples.wlcs.sampleapp.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>
            examples.wlcs.sampleapp.customer.Customer
        </jndi-name>
    </weblogic-enterprise-bean>
```

- Jar the contents of the temporary directory, and run the EJB compiler 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 WebLogic Portal.

Depending on how you're customizing a use-case flow, consider if the new Pipeline component should participate in a pre-existing 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 accessing the 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.

CHAPTER

9 Using the Order and Payment Management Pages

Customers who make purchases from your e-commerce site often want access to information about their current and past orders. If these customers cannot find what they are looking for using the customer self-service pages or simply prefer the human contact received by calling your e-business, an administrator of your site can locate this information for your customers using the Order Management pages. Additionally, the Order and Payment Management pages allow a site administrator to review and modify the status of order and payment transactions that have been initiated on the WebLogic Portal.

The Order and Payment Management pages ship as part of the WebLogic Portal Administration Tools Web Application. As such, they are not a part of the site that requires modification. This topic describes how an administrator can use the Order and Payment Management pages.

This topic includes the following sections:

- Starting the WebLogic Portal Administration Tools
- Using the Order Management Search Page
  - Searching for an Order by Customer ID
  - Searching for an Order by Order Identifier Number
  - Searching for an Order by Date Range
- Updating Order Status

    - Changing Order Status
- Using the Payment Management Search Page
    - Searching for a Payment by Customer ID
    - Searching for a Payment by Status
    - Authorizing, Capturing, and Settling Payments

Starting the WebLogic Portal Administration Tools

Before you can use the Order and Payment Management pages, you need to start the server and load the WebLogic Portal Administration Tools page in your Web browser.

To start the server on a Windows system, you can either:

- Run `StartPortal.bat` from the command line in the `PORTAL_HOME` directory, where `PORTAL_HOME` is the directory where you installed the WebLogic Portal.
- From the Start menu, select Programs → BEA WebLogic E-Business Platform → BEA WebLogic Portal 4.0 → Start BEA WebLogic Portal.

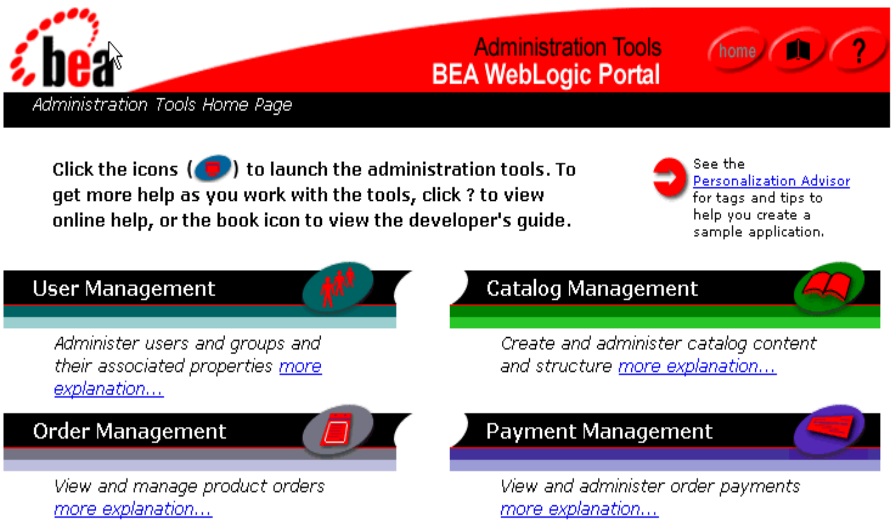
To start the server on a UNIX system, run `StartPortal.sh` from the command line in the `PORTAL_HOME` directory, where `PORTAL_HOME` is the directory where you installed the WebLogic Portal.

The Administration Tools page (shown in Figure 9-1) is an entry page into all of the available WebLogic Portal Administration Tools. To load this page, use one of the following methods:

- Specify the URL for the page (`http://<server>:<port>/<app_name>Tools/index.jsp`) in your Web browser, where <server> is the name of the server running WebLogic Portal (such as `localhost`), <port> is the port number that WebLogic Portal is running on on the server (such as 7501), and <app_name> is the the name of your enterprise application directory beneath `PORTAL_HOME\applications`.

- From the Start menu on a Windows system, select Programs → BEA WebLogic E-Business Platform → BEA WebLogic Portal 4.0 → Administration Tools.

Figure 9-1 WebLogic Portal Administration Tools Page



Click the icons ( ) to launch the administration tools. To get more help as you work with the tools, click ? to view online help, or the book icon to view the developer's guide.

See the Personalization Advisor for tags and tips to help you create a sample application.

User Management

Administer users and groups and their associated properties more explanation...

Catalog Management

Create and administer catalog content and structure more explanation...

Order Management

View and manage product orders more explanation...

Payment Management

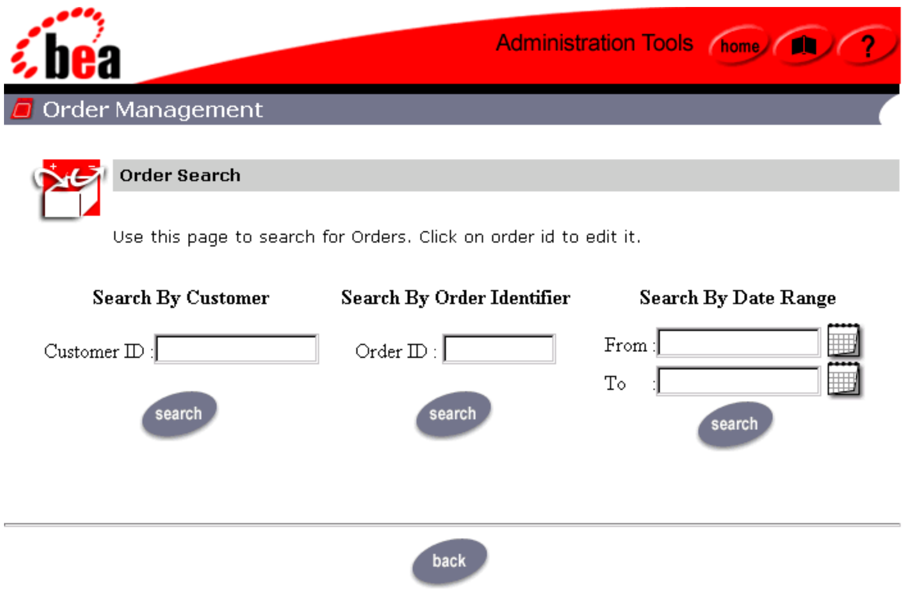
View and administer order payments more explanation...

To look up customers' orders, click the icon shown on the Order Management section titlebar to load the Order Management Search Page; to look up a customer's payment transactions, click the icon shown on the Payment Management section titlebar to load the Payment Management Search Page.

Using the Order Management Search Page

The Order Management search page (shown in Figure 9-2) appears when you click the icon on the Order Management section titlebar. This section explains the three different searches that are available to an administrator for order management.

Figure 9-2 The Order Management Search Page

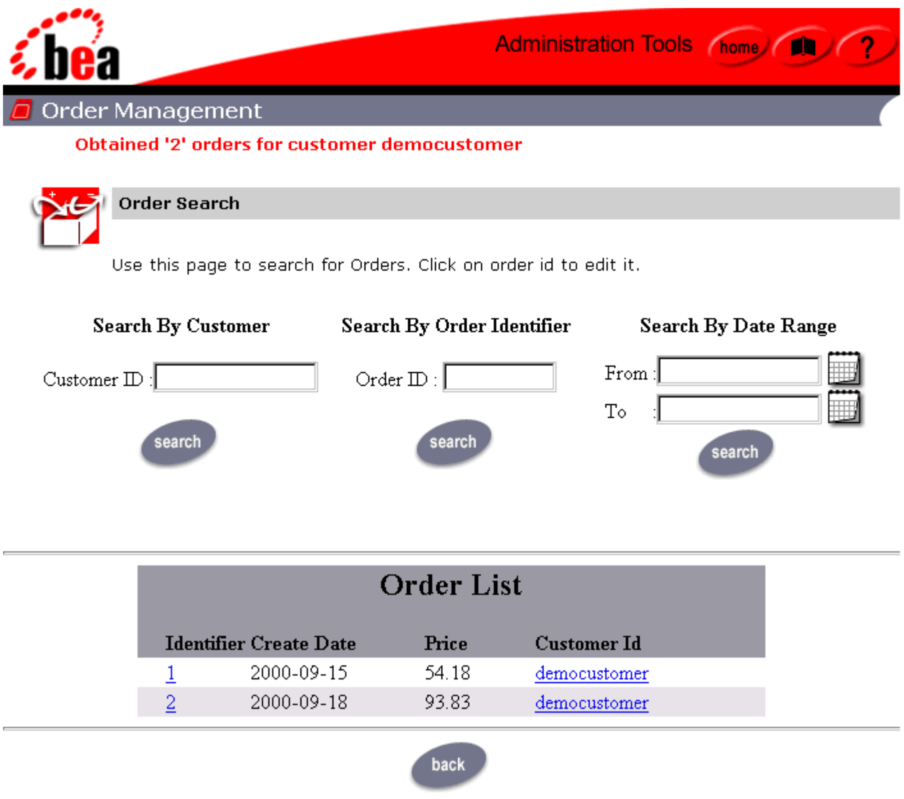


Use this page to search for Orders. Click on order id to edit it.

Searching for an Order by Customer ID

After a customer places an order on your e-commerce site, they may call to learn more about their order. One of the ways in which an administrator of the site can search is by using the customer's login ID. Simply enter the customer's ID into the appropriate form field and click the Search button. A text message appears at the top of the page, indicating how many orders were found for the search. The actual results appear below the search fields in an Order List, as shown in Figure 9-3.

Figure 9-3 Sample Results for Order Search by Customer ID



Obtained '2' orders for customer democustomer

Order Search

Use this page to search for Orders. Click on order id to edit it.

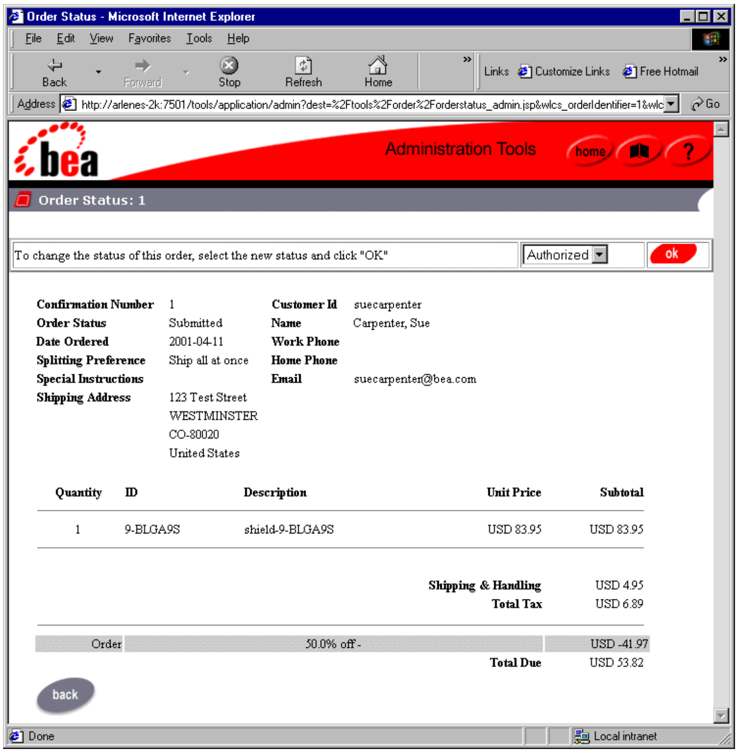
Search By Customer	Search By Order Identifier	Search By Date Range
Customer ID :	Order ID :	From :
		To :

Order List

Identifier	Create Date	Price	Customer Id
1	2000-09-15	54.18	democustomer
2	2000-09-18	93.83	democustomer

The Order List shows the Order Identifier number, the date the customer placed the order, and the price of the order. To see details for a particular order (including the product items ordered, shipping information, tax, and so on), click the hyperlinked Order Identifier number to load the Order Status page (shown in Figure 9-4). To return to the main Administration Tools page instead, click the Back button.

Figure 9-4 Sample Order Status Page



Address http://arlenes-2k:7501/tools/application/admin?dest=%2Ftools%2Forder%2Forderstatus_admin.jsp&wlcs_orderIdentifier=1&wlc

Order Status: 1

To change the status of this order, select the new status and click "OK" Authorized ok

Confirmation Number	1	**Customer Id**	suecarpenter
Order Status	Submitted	**Name**	Carpenter, Sue
Date Ordered	2001-04-11	**Work Phone**	
Splitting Preference	Ship all at once	**Home Phone**	
Special Instructions		**Email**	suecarpenter@bea.com
Shipping Address	123 Test Street WESTMINSTER CO-80020 United States		

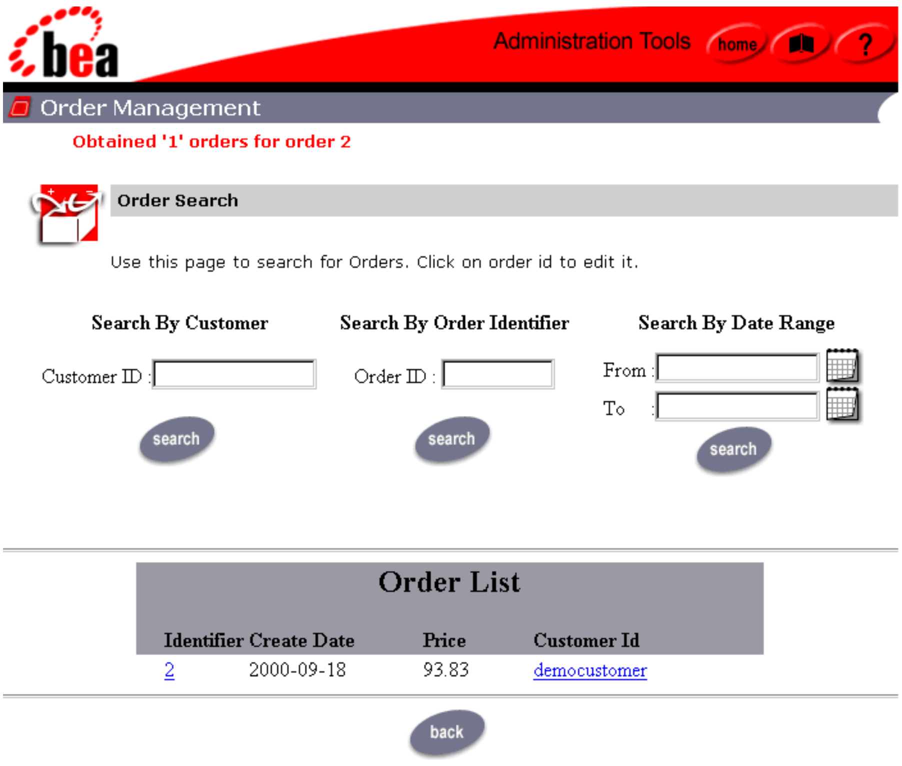
Quantity	ID	Description	Unit Price	Subtotal
1	9-BLGA9S	shield-9-BLGA9S	USD 83.95	USD 83.95
			Shipping & Handling	USD 4.95
			Total Tax	USD 6.89
Order		50.0% off -		USD -41.97
			Total Due	USD 53.82

Click the Back button at the bottom of the Order Status page to return to the Order Management search/results page.

Searching for an Order by Order Identifier Number

Another way in which an administrator of the site can search for a customer's order is by using the customer's Order Identifier number. This number is specified on the customer's order confirmation page after they submit an order to your system. Simply enter the customer's Order Identifier number into the appropriate form field and click the Search button. A text message appears at the top of the page, indicating how many orders were found for the search. The actual results appear below the search fields in an Order List, as shown in Figure 9-5.

Figure 9-5 Sample Results for Order Search by Order Identifier Number



Obtained '1' orders for order 2

Order Search

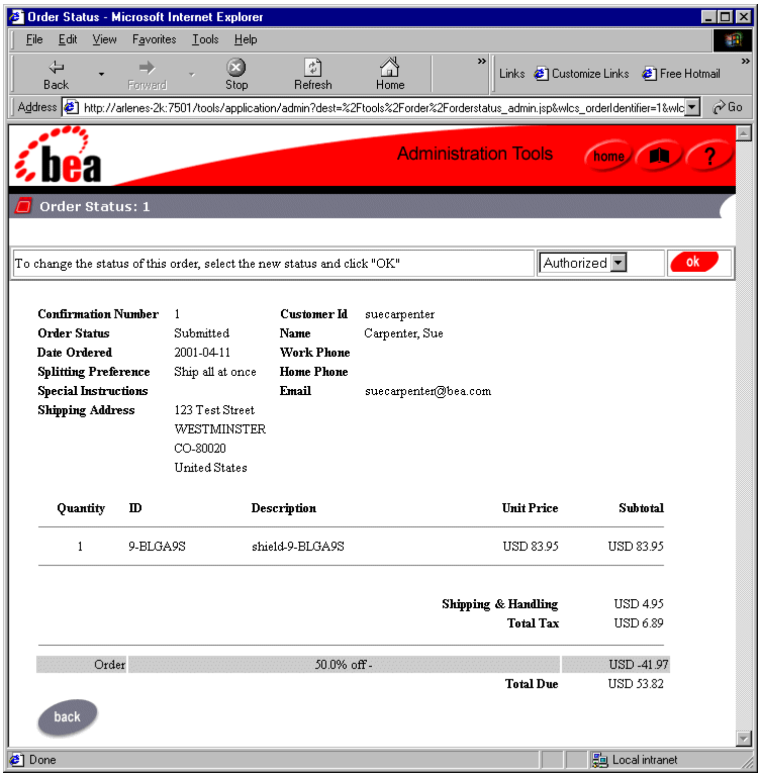
Use this page to search for Orders. Click on order id to edit it.

Order List

Identifier	Create Date	Price	Customer Id
2	2000-09-18	93.83	democustomer

The Order List shows the Order Identifier number, the date the customer placed the order, and the price of the order. To see details for a particular order (including the product items ordered, shipping information, tax, and so on), click the hyperlinked Order Identifier number to load the Order Status page (shown in Figure 9-6). To return to the main Administration Tools page instead, click the Back button.

Figure 9-6 Sample Order Status Page


Order Status - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home
Links Customize Links Free Hotmail
Address http://arlenes-2k:7501/tools/application/admin?dest=%2Ftools%2Forder%2Forderstatus_admin.jsp&wlcs_orderIdentifier=1&wlc
Go
Administration Tools
home
Done
Local intranet

Order Status: 1

To change the status of this order, select the new status and click "OK" — Authorized — ok

Confirmation Number	1	**Customer Id**	suecarpenter
Order Status	Submitted	**Name**	Carpenter, Sue
Date Ordered	2001-04-11	**Work Phone**	
Splitting Preference	Ship all at once	**Home Phone**	
Special Instructions		**Email**	suecarpenter@bea.com
Shipping Address	123 Test Street WESTMINSTER CO-80020 United States		

Quantity	ID	Description	Unit Price	Subtotal
1	9-BLGA9S	shield-9-BLGA9S	USD 83.95	USD 83.95
			Shipping & Handling	USD 4.95
			Total Tax	USD 6.89
Order		50.0% off -		USD -41.97
			Total Due	USD 53.82

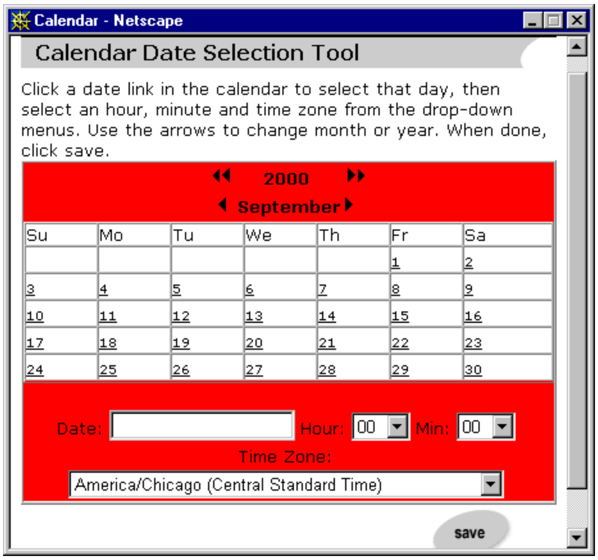
back

Click the Back button at the bottom of the Order Status page to return to the Order Management search/results page.

Searching for an Order by Date Range

Another way in which an administrator of the site can search for a customer's order is by using a date range. Date ranges must be specified using the Calendar Date Selection Tool, shown in Figure 9-7.

Figure 9-7 The Calendar Date Selection Tool



Click a date link in the calendar to select that day, then select an hour, minute and time zone from the drop-down menus. Use the arrows to change month or year. When done, click save.

Su	Mo	Tu	We	Th	Fr	Sa
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

Date: Hour: 00 Min: 00

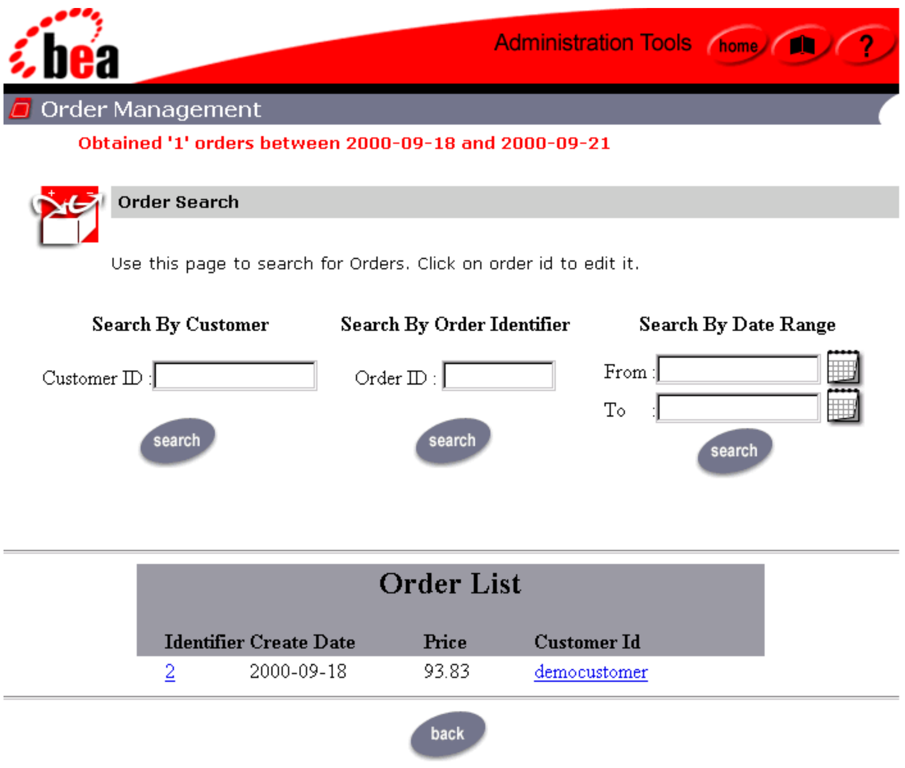
Time Zone: America/Chicago (Central Standard Time)

After clicking the Save button, the date, hour, minute and time zone you select with the Calendar Date Selection Tool appears in the From and To form fields, and you can now just click the Search button.

Note: The results for searches by date range are inclusive. That is, if you search for orders placed between July 22, 2000 and August 24, 2000, results will include orders placed on July 22 and orders placed on August 24.

A text message appears at the top of the page, indicating how many orders were found for the search. The actual results appear below the search fields in an Order List, as shown in Figure 9-8.

Figure 9-8 Sample Results for Order Search by Date Range



Obtained '1' orders between 2000-09-18 and 2000-09-21

Use this page to search for Orders. Click on order id to edit it.

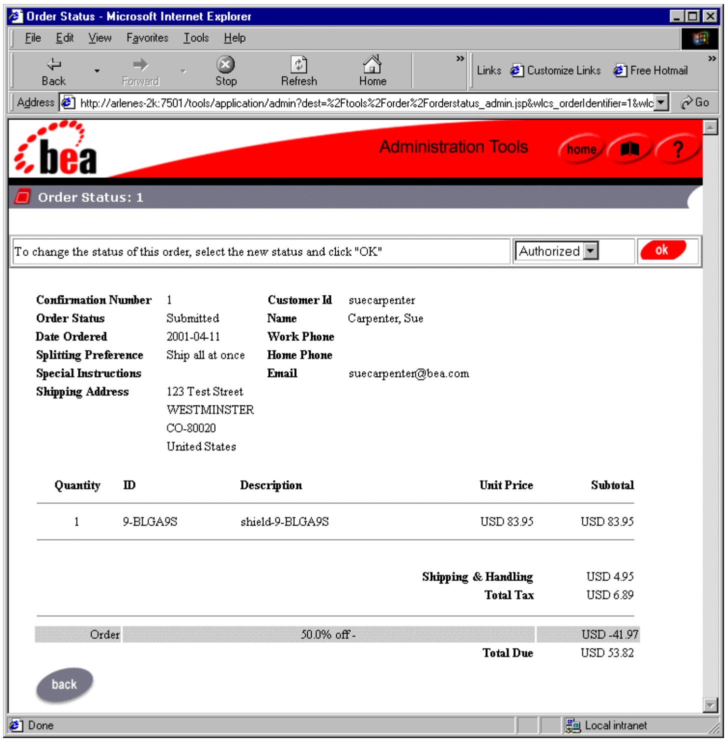
Search By Customer	Search By Order Identifier	Search By Date Range
Customer ID :	Order ID :	From :
		To :

Order List

Identifier	Create Date	Price	Customer Id
2	2000-09-18	93.83	democustomer

The Order List shows the Order Identifier number, the date the customer placed the order, and the price of the order. To see details for a particular order (including the product items ordered, shipping information, tax, and so on), click the hyperlinked Order Identifier number to load the Order Status page (shown in Figure 9-9). To return to the main Administration Tools page instead, click the Back button.

Figure 9-9 Sample Order Status Page



http://arlenes-2k:7501/tools/application/admin?dest=%2Ftools%2Forder%2Forderstatus_admin.jsp&wlcs_orderIdentifier=1&wlc

Order Status: 1

To change the status of this order, select the new status and click "OK" — Authorized

Confirmation Number	1	**Customer Id**	suecarpenter
Order Status	Submitted	**Name**	Carpenter, Sue
Date Ordered	2001-04-11	**Work Phone**	
Splitting Preference	Ship all at once	**Home Phone**	
Special Instructions		**Email**	suecarpenter@bea.com
Shipping Address	123 Test Street WESTMINSTER CO-80020 United States		

Quantity	ID	Description	Unit Price	Subtotal
1	9-BLGA9S	shield-9-BLGA9S	USD 83.95	USD 83.95
			Shipping & Handling	USD 4.95
			Total Tax	USD 6.89
Order		50.0% off -		USD -41.97
			Total Due	USD 53.82

Click the Back button at the bottom of the Order Status page to return to the Order Management search/results page.

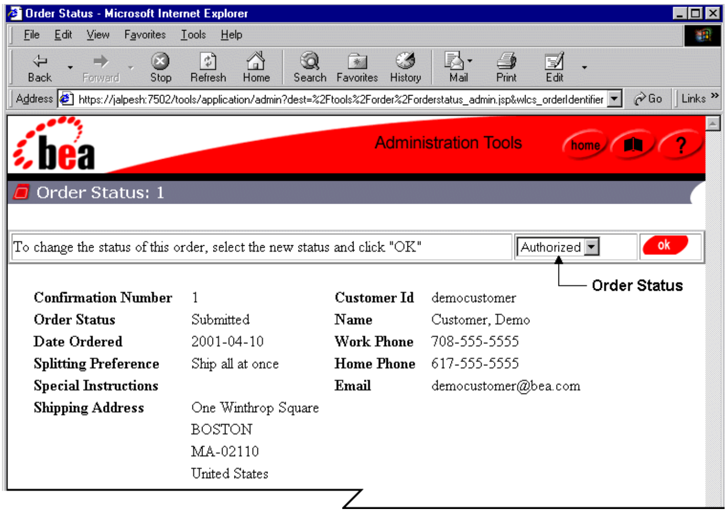
Updating Order Status

This section tells you how to change the status of an order and how to tailor the order status to your business.

Changing Order Status

The Order Status Page (shown in Figure 9-10) appears after you click the hyperlinked Order Identifier number on the Order List page. This section describes how to change the status of an order.

Figure 9-10 Sample Order Status Page


Order Status - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit
Address https://jalpesh:7502/tools/application/admin?dest=%2Ftools%2Forder%2Forderstatus_admin.jsp&wlcs_orderIdentifier
Go Links
bea
Administration Tools
home
?
Order Status: 1
ok
Order Status

To change the status of this order, select the new status and click "OK" — Authorized

Confirmation Number	1	**Customer Id**	democustomer
Order Status	Submitted	**Name**	Customer, Demo
Date Ordered	2001-04-10	**Work Phone**	708-555-5555
Splitting Preference	Ship all at once	**Home Phone**	617-555-5555
Special Instructions		**Email**	democustomer@bea.com
Shipping Address	One Winthrop Square BOSTON MA-02110 United States		

To change the status of an order, click the drop-down arrow on the Order Status list, select the new status, and then click the OK button. After a new status is entered, new entries appear in the Order Status list. These entries reflect the sequence of order status. For example, the initial Order Status list might contain the following:

- Authorized
- Cancelled

- Rejected

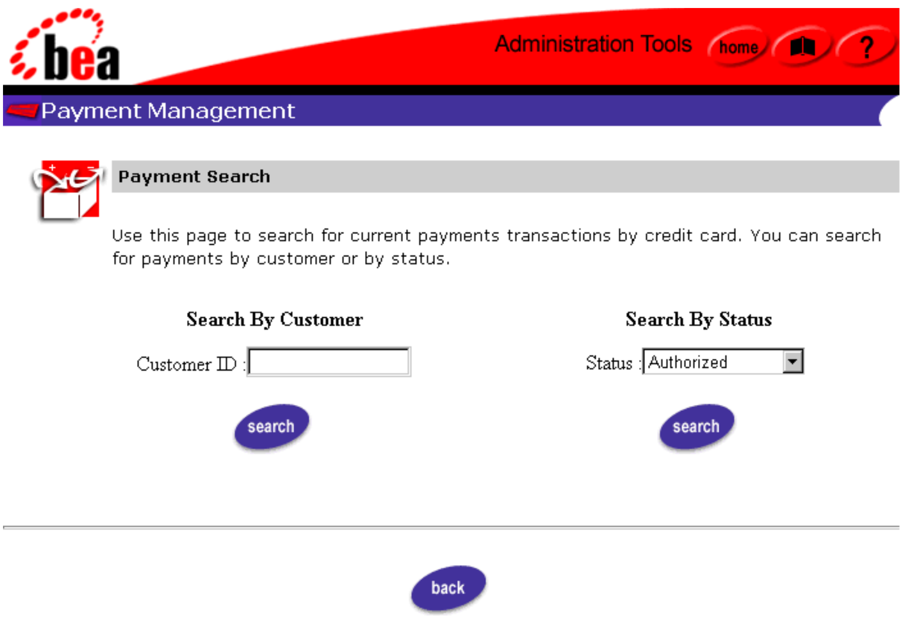
If you change the order status to Authorized, the Order Status list might contain the following options:

- Backordered
- Cancelled
- Shipped

Using the Payment Management Search Page

The Payment Management search page (shown in Figure 9-11) appears when you click the icon on the Payment Management section titlebar. This section explains the three different searches and transaction modification activities that are available to an administrator for payment management.

Figure 9-11 The Payment Management Search Page

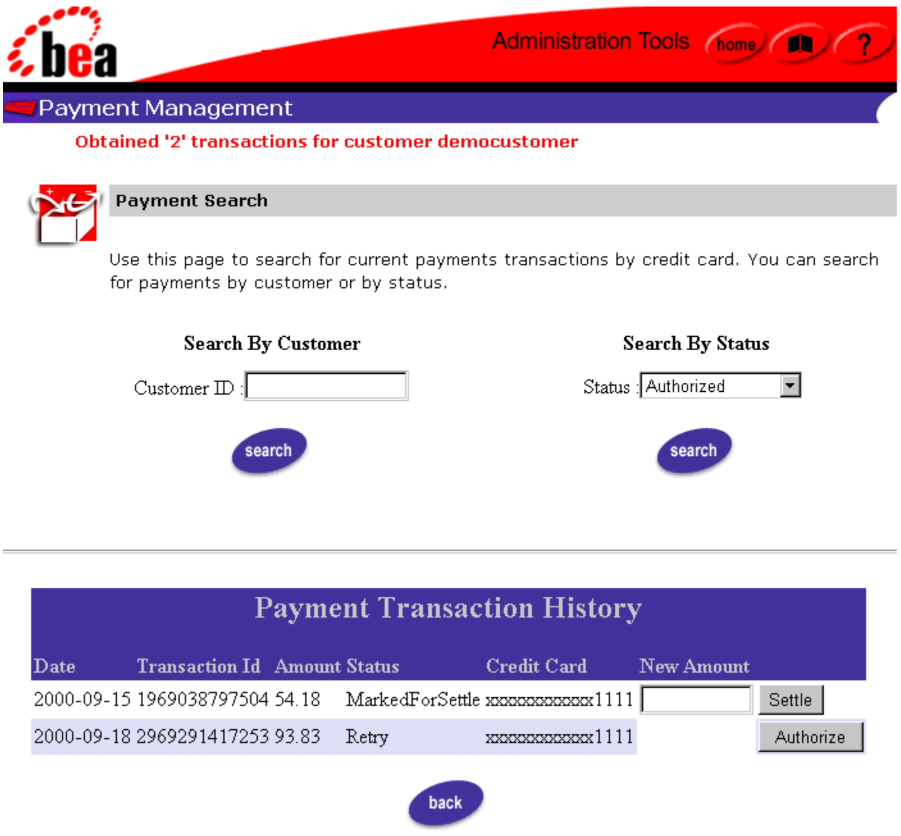


Use this page to search for current payments transactions by credit card. You can search for payments by customer or by status.

Searching for a Payment by Customer ID

After a customer places an order on your e-commerce site, they may call to find out the status of their payment. One of the ways in which an administrator of the site can search is by using the customer's login ID. Simply enter the customer's ID into the appropriate form field and click the Search button. A text message appears at the top of the page, indicating how many payments were found for the search. The actual results will appear below the search fields in the Payment Transaction History, as shown in Figure 9-12.

Figure 9-12 Sample Results for Payment Search by Customer ID



Obtained '2' transactions for customer democustomer

Payment Search

Use this page to search for current payments transactions by credit card. You can search for payments by customer or by status.

Status : Authorized

Payment Transaction History

Date	Transaction Id	Amount	Status	Credit Card	New Amount	
2000-09-15	1969038797504	54.18	MarkedForSettle	xxxxxxxxxxxx1111		Settle
2000-09-18	2969291417253	93.83	Retry	xxxxxxxxxxxx1111		Authorize

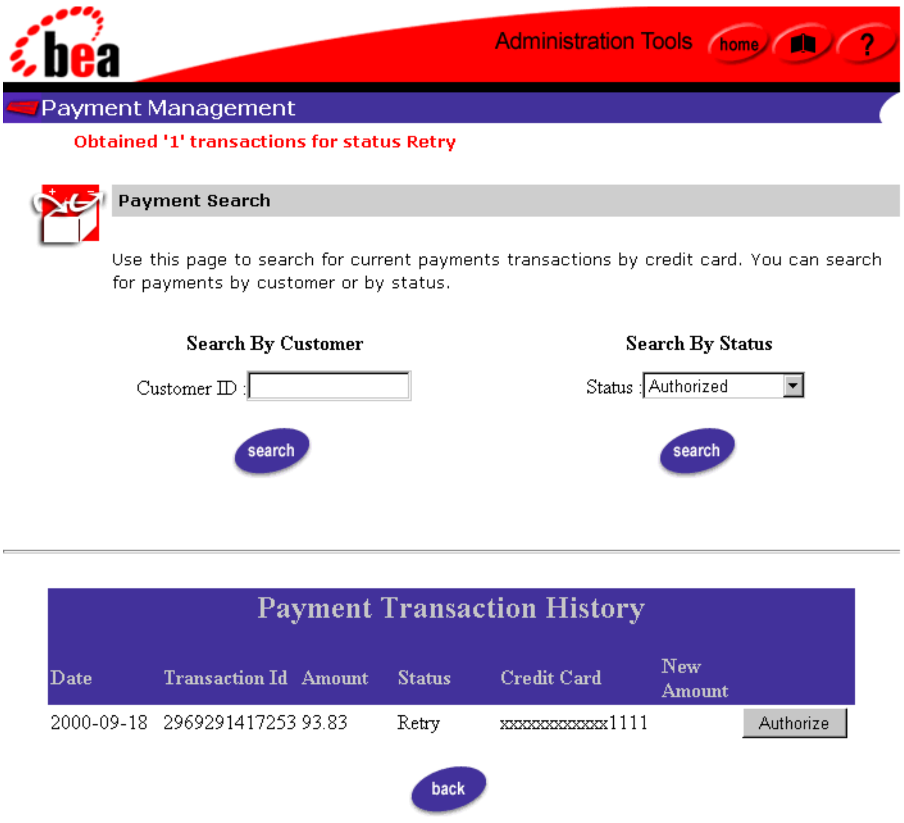
For a detailed explanation of the Payment Transaction History fields and further payment management activities, refer to “Authorizing, Capturing, and Settling Payments” on page 9-17.

To perform another search, type your query in the form field. To return to the main Administration Tools page instead, click the Back button.

Searching for a Payment by Status

Another way that an administrator of the site can search is by using a payment status (Authorized, MarkedForSettle, PendingSettle, Settled, Rejected, and Retry). Simply select the status from the Status pull-down menu and click the Search button. A text message appears at the top of the page, indicating how many payments were found for the status. The actual results will appear below the search fields in the Payment Transaction History, as shown in Figure 9-13.

Figure 9-13 Sample Results for Payment Search by Status



Obtained '1' transactions for status Retry

Payment Search

Use this page to search for current payments transactions by credit card. You can search for payments by customer or by status.

Search By Customer

Customer ID :

Search By Status

Status : Authorized

Payment Transaction History

Date	Transaction Id	Amount	Status	Credit Card	New Amount	
2000-09-18	2969291417253	93.83	Retry	xxxxxxxxxxxx1111		Authorize

For a detailed explanation of the Payment Transaction History fields and further payment management activities, refer to “Authorizing, Capturing, and Settling Payments” on page 9-17.

To perform another search, type your query in the form field. To return to the main Administration Tools page instead, click the Back button.

Authorizing, Capturing, and Settling Payments

The Payment Transaction History section (which appears in the lower portion of the Payment Management search page after a search is performed) shows information about each payment transaction, including the date, the transaction ID, the payment amount, the payment status, and a masked version of the credit card that was used to complete the transaction.

Table 9-1 provides a description for each of the possible payment status values.

Table 9-1 Payment Status Values

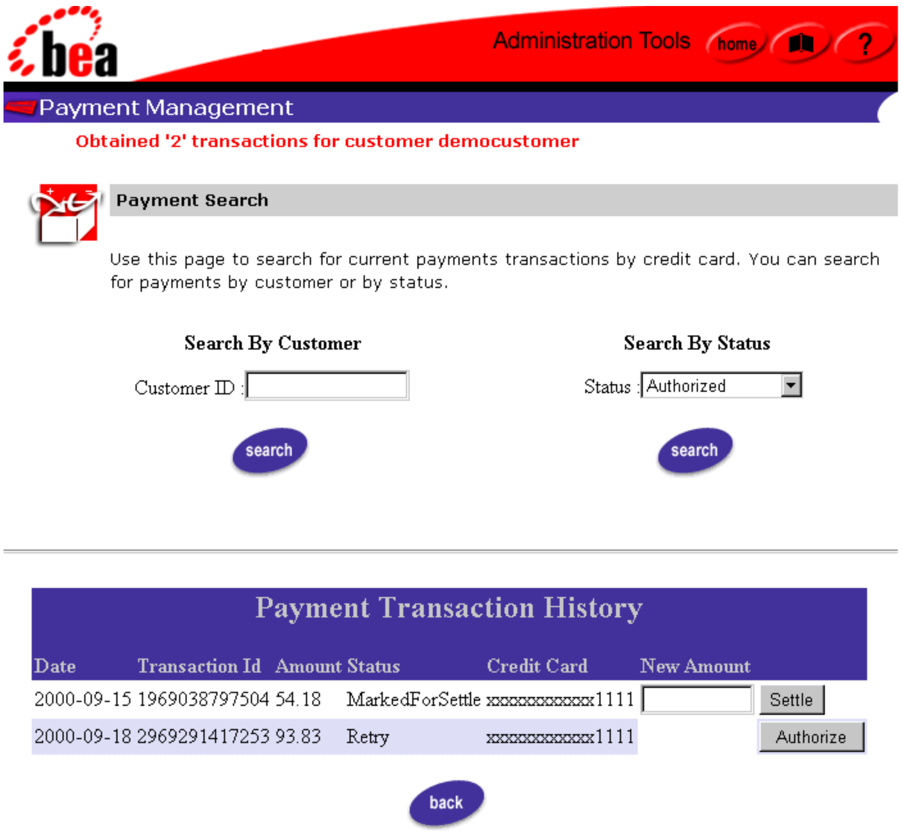
Status	Description
Authorized	The transaction has been successfully authorized, and is awaiting capture and settlement.
MarkedForSettle	The transaction has been batched for settlement (captured).
PendingSettle	The transaction settlement process has been initiated.
Settled	The transaction has been settled.
Rejected	Authorization for the transaction was rejected.
Retry	The transaction has been recorded, but authorization was either unsuccessful or has been deferred.

In order for a merchant to obtain the funds associated with a payment transaction, the transaction must be authorized, captured, and settled. Depending on the status of the transaction, a text field and associated button may appear at the end of the line in the Payment Transaction History section, making it possible to manually change the state of the transaction.

Authorizing the Transaction

If the status of the order is set to Retry, an Authorize button will appear at the end of the line (as shown in Figure 9-14).

Figure 9-14 Payment Transaction History With Authorize Button



Obtained '2' transactions for customer democustomer

Payment Search

Use this page to search for current payments transactions by credit card. You can search for payments by customer or by status.

Payment Transaction History

Date	Transaction Id	Amount	Status	Credit Card	New Amount	
2000-09-15	1969038797504	54.18	MarkedForSettle	xxxxxxxxxxxx1111		Settle
2000-09-18	2969291417253	93.83	Retry	xxxxxxxxxxxx1111		Authorize

Pressing this button will cause the WebLogic Portal product to connect to the Payment Web service, and to reserve credit from the customer's account on behalf of the merchant. A transaction is placed in the Retry state if you have configured the server to defer authorization of payments, or if the Payment Service was unavailable due to a system failure. In such cases, the business will not fulfill the order until the status on the associated payment transaction has been set to Authorized.

Authorization will change the state of the transaction in different ways, depending on the payment model in use. In a soft goods scenario (AUTO_MARK_AUTO_SETTLE or HOST_AUTH_CAPTURE), the transaction will transition directly to the PendingSettle state and remain there until it is settled.

Capturing the Transaction

If the payment model is one of the MANUAL_MARK_* or HOST_AUTH_POST_AUTH models and has been authorized, it is now necessary to capture that transaction. To capture the transaction, specify the amount that is to be captured in the text field, and click the Capture button. Capturing the funds associated with an order generally takes place after the order has been fulfilled. In some cases, the amount of the transaction may be less than the total original amount that was authorized. This is true in cases where the order was partially shipped.

Settling the Transaction

If a transaction has been captured and if the WebLogic Portal product has been configured for a *_MANUAL_SETTLE payment model, the transaction will be assigned the MarkedForSettle state. To settle the transaction, specify the amount that is to be settled in the text field, and click the Settle button. The amount may only be less than or equal to the capture amount.

Note: The WebLogic Portal will not set transactions to a Rejected status. This state is provided so that it may be set by third-party order management systems in the event that a payment transaction is considered unrecoverable. Additionally, the current implementation of the Administration Tools does not allow you to query the state of a Rejected transaction or move it to the Settled state.

CHAPTER

10 The Order Processing Database Schema

This topic describes the database schema for Managing Purchases and Processing Orders services. Understanding this schema will be helpful to those who may be customizing or extending the technologies provided in the product.

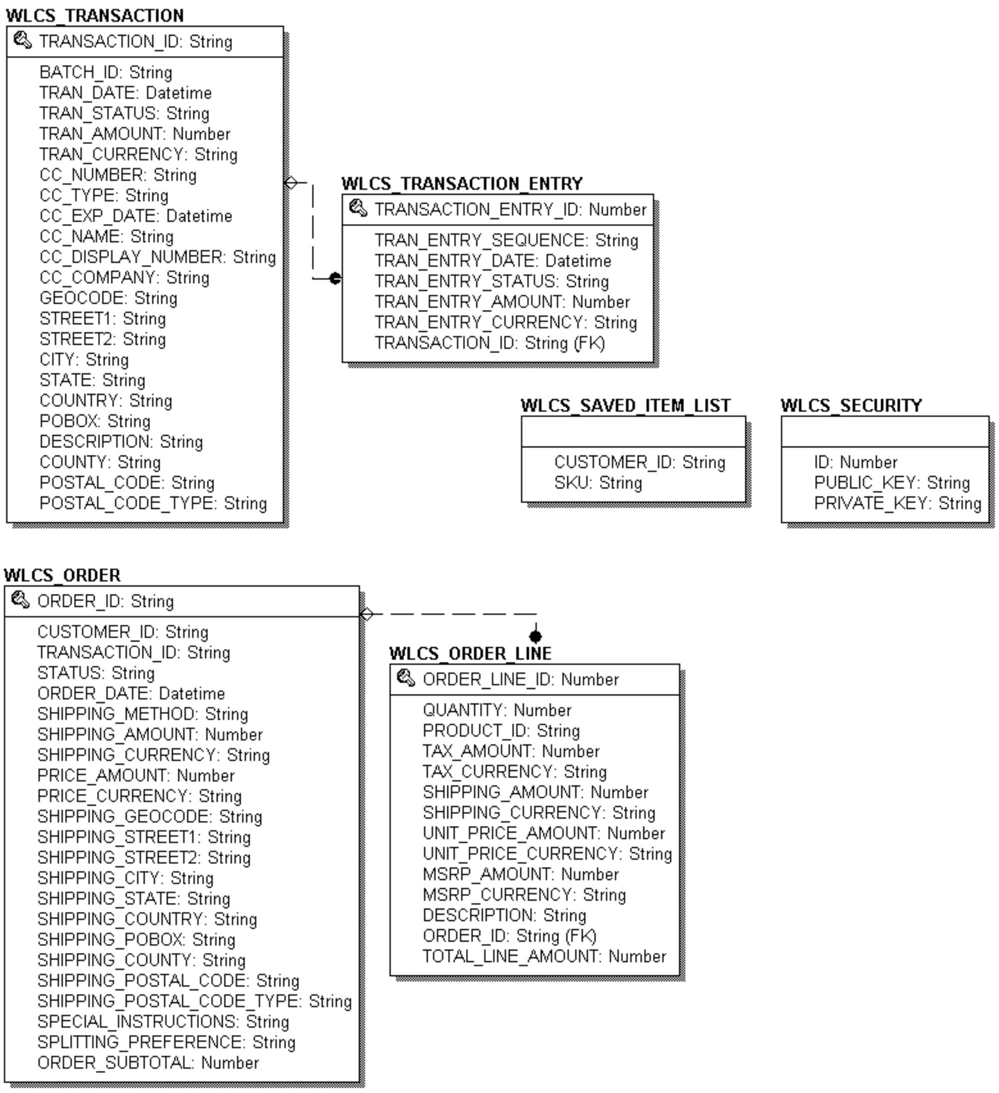
This topic includes the following sections:

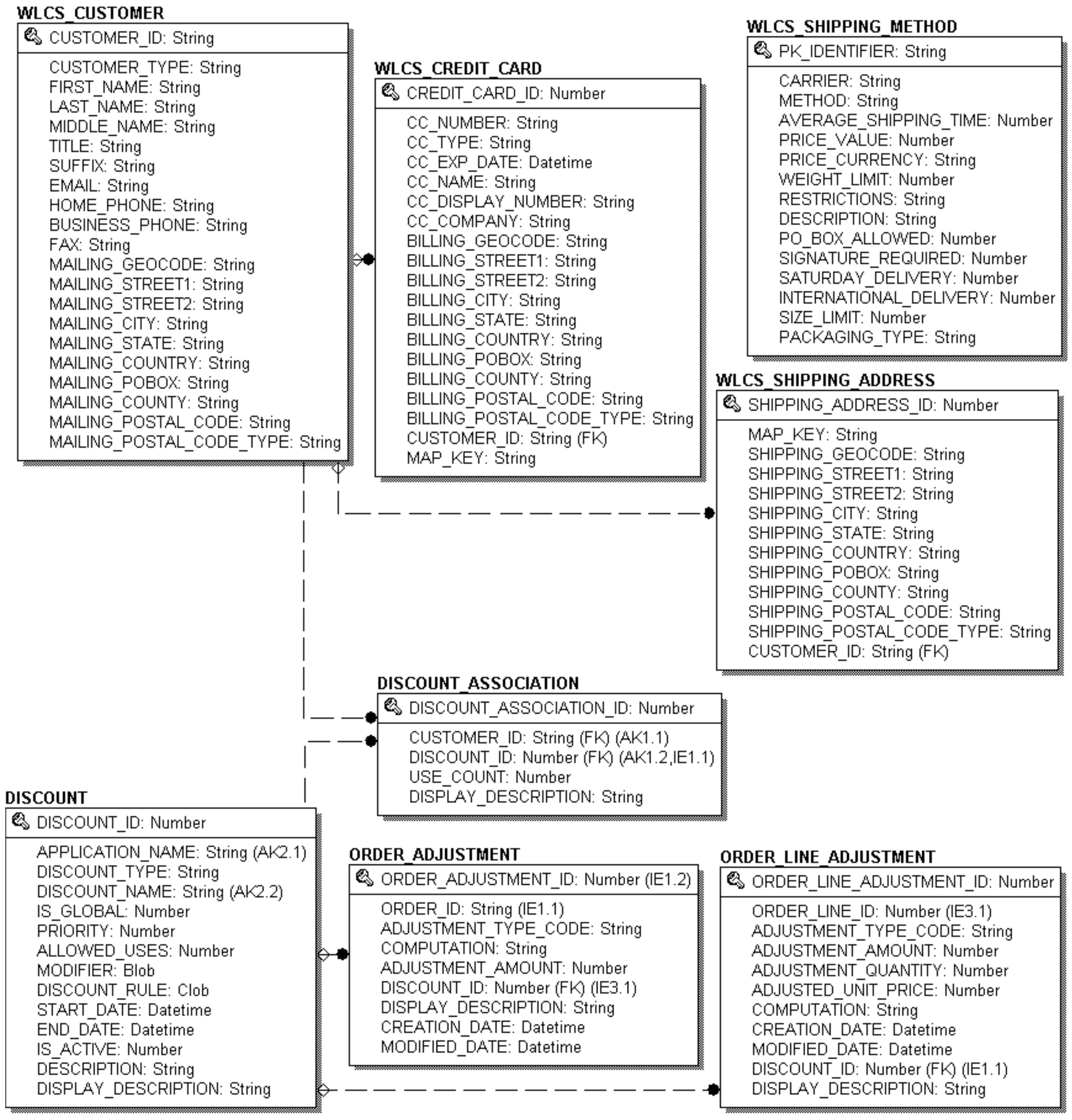
- The Entity-Relation Diagram
- List of Tables Comprising the Order Processing Schema
- The Order Processing Data Dictionary
- The SQL Scripts Used to Create the Database
- Defined Constraints

The Entity-Relation Diagram

Figure 10-1 shows the logical Entity-Relation diagram for the WebLogic Portal order and discount tables in the WebLogic Portal database. See the subsequent sections in this chapter for information about the data type syntax.

Figure 10-1 Entity-Relation Diagram for the Order and Discount Tables





Explanations for the columns in each table are provided in the remainder of this topic.

List of Tables Comprising the Order Processing Schema

The Commerce services order management system is comprised of the following tables:

- The DISCOUNT Database Table
- The DISCOUNT_ASSOCIATION Database Table
- The ORDER_ADJUSTMENT Database Table
- The ORDER_LINE_ADJUSTMENT Database Table
- The WLCS_CREDIT_CARD Database Table
- The WLCS_CUSTOMER Database Table
- The WLCS_ORDER Database Table
- The WLCS_ORDER_LINE Database Table
- The WLCS_SAVED_ITEM_LIST Database Table
- The WLCS_SECURITY Database Table
- The WLCS_SHIPPING_ADDRESS Database Table
- The WLCS_SHIPPING_METHOD Database Table
- The WLCS_TRANSACTION Database Table
- The WLCS_TRANSACTION_ENTRY Database Table

The Order Processing Data Dictionary

In this section, the schema tables are arranged alphabetically as a data dictionary.

Note: Even though the following documentation references “foreign keys” to various tables, these constraints do not currently exist in this release of Commerce services. However, they will be in place in future versions of Commerce services and we want you to be aware of these relationships now.

The DISCOUNT Database Table

Table 10-1 describes the metadata for the Commerce services DISCOUNT table. This table stores stores one or more discount records for every `DISCOUNT_SET` record.

See the section “” on page 10-25 for information about the constraint defined for this table.

The Primary Key is `DISCOUNT_ID`.

Table 10-1 DISCOUNT

Column Name	Data Type	Null Value	Description and Recommendations
DISCOUNT_ID	NUMBER(15)	NOT NULL	PK—a unique, system-generated number to be used as the record ID.
APPLICATION_NAME	VARCHAR(100)	NOT NULL	FK—foreign key to the DISCOUNT_SET table.
DISCOUNT_TYPE	VARCHAR(10)	NOT NULL	The type of discount offered. It is used for an *order* or for an *order line item*.
DISCOUNT_NAME	VARCHAR(254)	NOT NULL	The name of the discount.
IS_GLOBAL	NUMBER(1)	NOT NULL	A flag showing whether or not this discount can be used globally.
PRIORITY	NUMBER(3)	NOT NULL	The level of priority this discount has over other discounts.

Table 10-1 DISCOUNT (Continued)

Column Name	Data Type	Null Value	Description and Recommendations
`ALLOWED_USERS`	`NUMBER(10)`	`NOT NULL`	The number of times the discount may be used.
`MODIFIER`	`VARCHAR(254)`	`NOT NULL`	Describes the actual discount to be applied. This is XML.
`DISCOUNT_RULE`	`CLOB`	`NOT NULL`	The method used to select items for discount. This is XML.
`START_DATE`	`DATE`	`NOT NULL`	The starting date and time of the discount
`END_DATE`	`DATE`	`NOT NULL`	The ending date and time of the discount.
`IS_ACTIVE`	`NUMBER(1)`	`NOT NULL`	A flag that determines whether the discount is active or not. Active=1, Not active=0
`DESCRIPTION`	`VARCHAR(254)`	`NULL`	The discount description.
`DISPLAY_DESCRIPTION`	`VARCHAR(254)`	`NULL`	The discount description used for display purposes only.

The DISCOUNT_ASSOCIATION Database Table

Table 10-2 describes the metadata for the Commerce services DISCOUNT_ASSOCIATION table. This table associates each customer with a discount and maintains information regarding the times the customer has used each discount.

See the section “” on page 10-25 for information about the constraint defined for this table.

The Primary Key is `DISCOUNT_ASSOCIATION_ID`.

Table 10-2 DISCOUNT_ASSOCIATION

Column Name	Data Type	Null Value	Description and Recommendations
DISCOUNT_ASSOCIATION_ID	NUMBER(15)	NOT NULL	PK—a unique, system-generated number to be used as the record ID.
CUSTOMER_ID	VARCHAR(20)	NOT NULL	FK—foreign key to the DISCOUNT_SET table.
DISCOUNT_ID	NUMBER(15)	NOT NULL	FK—foreign key to the DISCOUNT_SET table.
USE_COUNT	NUMBER(10)	NOT NULL	The number of times the discount has been used.
DISPLAY_DESCRIPTION	VARCHAR(254)	NULL	The discount description used for display purposes only.

The ORDER_ADJUSTMENT Database Table

Table 10-3 describes the metadata for the Commerce services ORDER_ADJUSTMENT table. This table is used to maintain information about a discount taken at the order level (for example, $20.00 off any order between 1/1/02 and 1/31/02.)

See the section “” on page 10-25 for information about the constraint defined for this table.

The Primary Key is `ORDER_ADJUSTMENT_ID`.

Table 10-3 ORDER_ADJUSTMENT

Column Name	Data Type	Null Value	Description and Recommendations
ORDER_ADJUSTMENT_ID	NUMBER(15)	NOT NULL	PK—a unique, system-generated number to be used as the record ID.
ORDER_ID	VARCHAR(20)	NOT NULL	FK—foreign key to the DISCOUNT_SET table.

Table 10-3 ORDER_ADJUSTMENT (Continued)

Column Name	Data Type	Null Value	Description and Recommendations
`ADJUSTMENT_TYPE`	`VARCHAR(20)`	`NULL`	The type of adjustment being made to the order line item (e.g., order line discount, shipping discount, etc.)
`COMPUTATION`	`VARCHAR(254)`	`NOT NULL`	The number of times the discount has been used.
`ADJUSTMENT_AMOUNT`	`NUMBER(16,4)`	`NOT NULL`	The discount description used for display purposes only.
`DISCOUNT_ID`	`NUMBER(15)`	`NULL`	FK—foreign key to the DISCOUNT table.
`DISPLAY_DESCRIPTION`	`VARCHAR(254)`	`NULL`	The description used for display purposes only. Depending on the nature of the discount, the `DISPLAY_DESCRIPTION` is generated from either the Discount service or Campaign service.
`CREATION_DATE`	`DATE`	`NOT NULL`	The date and time the order adjustment was created.
`MODIFIED_DATE`	`DATE`	`NULL`	The date and time the order adjustment record was last modified.

The ORDER_LINE_ADJUSTMENT Database Table

Table 10-4 describes the metadata for the Commerce services ORDER_LINE_ADJUSTMENT table. This table is used to maintain information about a discount taken at the order line item level (for example, 10% off SKU “Power Drill”).

See the section “” on page 10-25 for information about the constraint defined for this table.

The Primary Key is `ORDER_LINE_ADJUSTMENT_ID`.

Table 10-4 ORDER_LINE_ADJUSTMENT Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
ORDER_LINE_ADJUSTMENT_ID	NUMBER(15)	NOT NULL	PK—a unique, system-generated number to be used as the record ID.
ORDER_LINE_ID	NUMBER(15)	NOT NULL	A unique identifier for each line in a customer's shopping cart. This field is the table's primary key and cannot be NULL. All other fields in the WLCS_ORDERLINE table can be NULL.
ADJUSTMENT_TYPE	VARCHAR(20)	NULL	The type of adjustment being made to the order line item (e.g., order line discount, shipping discount, etc.)
ADJUSTMENT_AMOUNT	NUMBER(16,4)	NOT NULL	The dollar amount of the adjustment.
ADJUSTMENT_QUANTITY	NUMBER(16,4)	NOT NULL	The quantity amount for the adjustment.
ADJUSTED_UNIT_PRICE	NUMBER(16,4)	NOT NULL	The adjusted unit price of the specific line item.
COMPUTATION	VARCHAR(254)	NOT NULL	The computation for determining ADJUSTED_UNIT_PRICE.
CREATION_DATE	DATE	NOT NULL	The date and time the adjustment record was created.
MODIFIED_DATE	DATE	NULL	The date and time the adjustment record was last modified.
DISCOUNT_ID	NUMBER(15)	NULL	FK—a foreign key to the discount used from the DISCOUNT table.
DISPLAY_DESCRIPTION	VARCHAR(254)	NULL	The adjustment description used for display purposes.

The WLCS_CREDIT_CARD Database Table

Table 10-5 describes the metadata for the Commerce services WLCS_CREDIT_CARD table. This table is used to store information related to a customer's credit card(s) in the order processing database.

See the section "" on page 10-25 for information about the constraint defined for this table.

The Primary Key is `CREDIT_CARD_ID`.

Table 10-5 WLCS_CREDIT_CARD Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
`CREDIT_CARD_ID`	`NUMBER(15)`	`NOT NULL`	A unique identifier for the credit card. This field is the table's primary key and cannot be `NULL`. All other fields in the `WLCS_CREDIT_CARD` table can be `NULL`.
`CC_NUMBER`	`VARCHAR(200)`	`NULL`	The customer's credit card number. This is encrypted if `is.encryption.enable` is set to `true` in the `weblogiccommerce.properties` file.
`CC_TYPE`	`VARCHAR(20)`	`NULL`	The customer's credit card type, such as VISA or MasterCard.
`CC_EXP_DATE`	`DATE`	`NULL`	The expiration date on the customer's credit card.
`CC_NAME`	`VARCHAR(50)`	`NULL`	The credit card holder's name.
`CC_DISPLAY_NUMBER`	`VARCHAR(20)`	`NULL`	The version of the credit card number that is displayed (all Xs except last 4-digits).
`CC_COMPANY`	`VARCHAR(50)`	`NULL`	The name of the credit card company.

Table 10-5 WLCS_CREDIT_CARD Table Metadata (Continued)

Column Name	Data Type	Null Value	Description and Recommendations
BILLING_GEOCODE	VARCHAR(2)	NULL	The code used by the TAXWARE system to identify taxes for the order based on jurisdiction.
BILLING_STREET1	VARCHAR(30)	NULL	The first line in the customer's billing address.
BILLING_STREET2	VARCHAR(30)	NULL	The second line in the customer's billing address.
BILLING_CITY	VARCHAR(30)	NULL	The city in the customer's billing address.
BILLING_STATE	VARCHAR(40)	NULL	The state in the customer's billing address.
BILLING_COUNTRY	VARCHAR(40)	NULL	The country in the customer's billing address.
BILLING_POBOX	VARCHAR(30)	NULL	The post office box in the customer's billing address.
BILLING_COUNTY	VARCHAR(50)	NULL	The county in the customer's billing address.
BILLING_POSTAL_CODE	VARCHAR(10)	NULL	The postal (ZIP) code in the customer's billing address.
BILLING_POSTAL_CODE_TYPE	VARCHAR(10)	NULL	Format or type of postal code, generally determined by country (such as ZIP code in the United States).
CUSTOMER_ID	VARCHAR(20)	NULL	A unique identifier for the customer.
MAP_KEY	VARCHAR(60)	NULL	Key that maps multiple credit cards with a single customer.

The WLCS_CUSTOMER Database Table

Table 10-6 describes the metadata for the Commerce services WLCS_CUSTOMER table. This table is used to store information about the customer in the order processing database.

The Primary Key is CUSTOMER_ID.

Table 10-6 WLCS_CUSTOMER Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
CUSTOMER_ID	VARCHAR(20)	NOT NULL	A unique identifier for the customer. This field is the table's primary key and cannot be NULL. All other fields in the WLCS_CUSTOMER table can be NULL.
CUSTOMER_TYPE	VARCHAR(20)	NULL	A label for the customer (such as preferred, standard, or business).
FIRST_NAME	VARCHAR(30)	NULL	The customer's first name.
LAST_NAME	VARCHAR(30)	NULL	The customer's last name.
MIDDLE_NAME	VARCHAR(30)	NULL	The customer's middle name.
TITLE	VARCHAR(10)	NULL	The customer's preferred title, such as Mr., Mrs., or Ms.
SUFFIX	VARCHAR(10)	NULL	The customer's preferred suffix, such as Jr.or Sr.
EMAIL	VARCHAR(80)	NULL	The customer's email address.
HOME_PHONE	VARCHAR(15)	NULL	The customer's home phone number.
BUSINESS_PHONE	VARCHAR(20)	NULL	The customer's business phone number.
FAX	VARCHAR(15)	NULL	The customer's fax number.
MAILING_GEOCODE	VARCHAR(2)	NULL	The code used by the TAXWARE system to identify taxes for the order based on jurisdiction.

Table 10-6 WLCS_CUSTOMER Table Metadata (Continued)

Column Name	Data Type	Null Value	Description and Recommendations
`MAILING_STREET1`	`VARCHAR(30)`	`NULL`	The first line in the customer's street address.
`MAILING_STREET2`	`VARCHAR(30)`	`NULL`	The second line in the customer's street address.
`MAILING_CITY`	`VARCHAR(30)`	`NULL`	The city in the customer's address.
`MAILING_STATE`	`VARCHAR(40)`	`NULL`	The state in the customer's address.
`MAILING_COUNTRY`	`VARCHAR(40)`	`NULL`	The country in the customer's address.
`MAILING_POBOX`	`VARCHAR(30)`	`NULL`	The post office box in the customer's address.
`MAILING_COUNTY`	`VARCHAR(50)`	`NULL`	The county in the customer's address.
`MAILING_POSTAL_CODE`	`VARCHAR(10)`	`NULL`	The postal (ZIP) code in the customer's address.
`MAILING_POSTAL_CODE_TYPE`	`VARCHAR(10)`	`NULL`	Format or type of postal code, generally determined by country (such as ZIP code in the United States).

The WLCS_ORDER Database Table

Table 10-7 describes the metadata for the Commerce services WLCS_ORDER table. This table is used to store information about a customer's specific order in the order processing database.

Note: The Commerce services product does not populate the SHIPPING_AMOUNT, SHIPPING_CURRENCY, PRICE_AMOUNT, or PRICE_CURRENCY columns.

The Primary Key is `ORDER_ID`.

Table 10-7 WLCS_ORDER Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
ORDER_ID	VARCHAR(20)	NOT NULL	A unique identifier for the order. This field is the table's primary key and cannot be NULL. All other fields in the WLCS_ORDER table can be NULL.
CUSTOMER_ID	VARCHAR(20)	NULL	A unique identifier for the customer.
TRANSACTION_ID	VARCHAR(25)	NULL	A unique identifier for the transaction.
STATUS	VARCHAR(20)	NULL	The status of the order.
ORDER_DATE	DATE	NULL	The date the order was placed.
SHIPPING_METHOD	VARCHAR(40)	NULL	The method by which the order is to be shipped.
SHIPPING_AMOUNT	NUMBER(16,4)	NULL	The shipping amount for the order.
SHIPPING_CURRENCY	VARCHAR(10)	NULL	The currency associated with the shipping amount.
PRICE_AMOUNT	NUMBER(16,4)	NULL	The price of the order.
PRICE_CURRENCY	VARCHAR(10)	NULL	The currency associated with the price.
SHIPPING_GEOGODE	VARCHAR(2)	NULL	The code used by the TAXWARE system to identify taxes for the order based on jurisdiction.
SHIPPING_STREET1	VARCHAR(30)	NULL	The first line in the customer's shipping address.
SHIPPING_STREET2	VARCHAR(30)	NULL	The second line in the customer's shipping address.
SHIPPING_CITY	VARCHAR(30)	NULL	The city in the customer's shipping address.

Table 10-7 WLCS_ORDER Table Metadata (Continued)

Column Name	Data Type	Null Value	Description and Recommendations
SHIPPING_STATE	VARCHAR(40)	NULL	The state in the customer's shipping address.
SHIPPING_COUNTRY	VARCHAR(40)	NULL	The country in the customer's shipping address.
SHIPPING_POBOX	VARCHAR(30)	NULL	The post office box in the customer's shipping address.
SHIPPING_COUNTY	VARCHAR(50)	NULL	The county in the customer's shipping address.
SHIPPING_POSTAL_CODE	VARCHAR(10)	NULL	The postal (ZIP) code in the customer's shipping address.
SHIPPING_POSTAL_CODE_TYPE	VARCHAR(10)	NULL	Format or type of postal code, generally determined by country, such as ZIP code in the United States.
SPECIAL_INSTRUCTIONS	VARCHAR(254)	NULL	Any special shipping instructions associated with the order.
SPLITTING_PREFERENCE	VARCHAR(254)	NULL	The splitting preferences for the customer's order.
ORDER_SUBTOTAL	NUMBER(16,4)	NULL	The sum of all the `TOTAL_LINE_AMOUNT` columns in the `WLCS_ORDER_LINE` table for that specific order.

The WLCS_ORDER_LINE Database Table

Table 10-8 describes the metadata for the Commerce services WLCS_ORDER_LINE table. This table is used to store information about each line of a customer's shopping cart in the order processing database.

See the section "" on page 10-25 for information about the constraint defined for this table.

The Primary Key is `ORDER_LINE_ID`.

Table 10-8 WLCS_ORDER_LINE Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
`ORDER_LINE_ID`	`NUMBER(15)`	`NOT NULL`	A unique identifier for each line in a customer's shopping cart. This field is the table's primary key and cannot be `NULL`. All other fields in the `WLCS_ORDERLINE` table can be `NULL`.
`QUANTITY`	`NUMBER(16,4)`	`NULL`	The quantity of the item in the shopping cart.
`PRODUCT_ID`	`VARCHAR(40)`	`NULL`	An identification number for the item in the shopping cart.
`TAX_AMOUNT`	`NUMBER(16,4)`	`NULL`	The tax amount for the order.
`TAX_CURRENCY`	`VARCHAR(10)`	`NULL`	The currency associated with the tax amount.
`SHIPPING_AMOUNT`	`NUMBER(16,4)`	`NULL`	The shipping amount for the order.
`SHIPPING_CURRENCY`	`VARCHAR(10)`	`NULL`	The currency associated with the shipping amount.
`UNIT_PRICE_AMOUNT`	`NUMBER(16,4)`	`NULL`	The unit price amount for the item.
`UNIT_PRICE_CURRENCY`	`VARCHAR(10)`	`NULL`	The currency associated with the unit price.
`MSRP_AMOUNT`	`NUMBER(16,4)`	`NULL`	The MSRP amount for the item.
`MSRP_CURRENCY`	`VARCHAR(10)`	`NULL`	The currency associated with the MSRP amount.
`DESCRIPTION`	`VARCHAR(254)`	`NULL`	The name of the item that is part of the order.
`ORDER_ID`	`VARCHAR(20)`	`NULL`	A unique identifier for the order.
`TOTAL_LINE_AMOUNT`	`NUMBER(16,4)`	`NULL`	The total discounted price for the line item. `UNIT_PRICE_AMOUNT` (less any discount) times the `QUANTITY`.

The WLCS_SAVED_ITEM_LIST Database Table

Table 10-9 describes the metadata for the Commerce services WLCS_SAVED_ITEM_LIST table. This table is used to store information about the customer's saved shopping cart items in the order processing database.

There is no Primary Key.

Table 10-9 WLCS_SAVED_ITEM_LIST Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
`CUSTOMER_ID`	`VARCHAR(20)`	`NULL`	A unique identifier for the customer.
`SKU`	`VARCHAR(40)`	`NULL`	A unique identifier (the Stock Keeping Unit or SKU) for a product item.

The WLCS_SECURITY Database Table

Table 10-10 describes the metadata for the Commerce services WLCS_SECURITY table. This table is used to persist public and private keys for encryption and decryption purposes in the order processing database. This table is meant for internal use by the Commerce services product.

There is no Primary Key.

Table 10-10 WLCS_SECURITY Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
`ID`	`NUMBER(5)`	`NULL`	A unique identifier for the key pair. This field is the table's primary key and cannot be `NULL`.
`PUBLIC_KEY`	`VARCHAR(2000)`	`NULL`	The public key to be used for encryption/decryption of credit cards.
`PRIVATE_KEY`	`VARCHAR(2000)`	`NULL`	The private key to be used for encryption/decryption of credit cards.

The WLCS_SHIPPING_ADDRESS Database Table

Table 10-11 describes the metadata for the Commerce services WLCS_SHIPPING_ADDRESS table. This table is used to store information related to a customer's shipping address(es) in the order processing database.

See the section "" on page 10-25 for information about the constraint defined for this table.

The Primary Key is `SHIPPING_ADDRESS_ID`.

Table 10-11 WLCS_SHIPPING_ADDRESS Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
SHIPPING_ADDRESS_ID	NUMBER(15)	NOT NULL	A unique identifier for the shipping address. This field is the table's primary key and cannot be `NULL`. All other fields in the `WLCS_SHIPPING_ADDRESS` table can be `NULL`.
MAP_KEY	VARCHAR(60)	NULL	Key that maps multiple shipping addresses with a single customer.
SHIPPING_GEOCODE	VARCHAR(2)	NULL	The code used by the TAXWARE system to identify taxes for the order based on jurisdiction.
SHIPPING_STREET1	VARCHAR(30)	NULL	The first line in the customer's shipping address.
SHIPPING_STREET2	VARCHAR(30)	NULL	The second line in the customer's shipping address.
SHIPPING_CITY	VARCHAR(30)	NULL	The city in the customer's shipping address.
SHIPPING_STATE	VARCHAR(40)	NULL	The state in the customer's shipping address.
SHIPPING_COUNTRY	VARCHAR(40)	NULL	The country in the customer's shipping address.

Table 10-11 WLCS_SHIPPING_ADDRESS Table Metadata (Continued)

Column Name	Data Type	Null Value	Description and Recommendations
SHIPPING_POBOX	VARCHAR(30)	NULL	The post office box in the customer's shipping address.
SHIPPING_COUNTY	VARCHAR(50)	NULL	The county in the customer's shipping address.
SHIPPING_POSTAL_CODE	VARCHAR(10)	NULL	The postal (zip) code in the customer's shipping address.
SHIPPING_POSTAL_CODE_TYPE	VARCHAR(10)	NULL	Format or type of postal code, generally determined by country, such as ZIP code in the United States.
CUSTOMER_ID	VARCHAR(20)	NULL	A unique identifier for the customer.

The WLCS_SHIPPING_METHOD Database Table

Table 10-12 describes the metadata for the Commerce services WLCS_SHIPPING_METHOD table. This table is used to store information about the shipping method in the order processing database.

The Primary Key is `PK_IDENTIFIER`.

Table 10-12 WLCS_SHIPPING_METHOD Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
PK_IDENTIFIER	VARCHAR(20)	NOT NULL	A unique identifier for the shipping method. This field is the table's primary key and cannot be `NULL`. All other fields in the `WLCS_SHIPPING_METHOD` table can be `NULL`.
CARRIER	VARCHAR(40)	NULL	The carrier being used to ship the order, such as UPS or FedEx.

Table 10-12 WLCS_SHIPPING_METHOD Table Metadata (Continued)

Column Name	Data Type	Null Value	Description and Recommendations
METHOD	VARCHAR(40)	NULL	The method by which the order is to be shipped, such as Air, 2nd Day Air, or Parcel Post.
AVERAGE_SHIPPING_TIME	NUMBER	NULL	The average number of days it will take the order to arrive.
PRICE_VALUE	NUMBER(16,4)	NULL	The amount it will cost to ship the order.
PRICE_CURRENCY	VARCHAR(10)	NULL	The currency associated with the PRICE_VALUE column, such as dollars, pounds, or lira.
WEIGHT_LIMIT	NUMBER(16,4)	NULL	The weight limit for the shipment.
RESTRICTIONS	VARCHAR(254)	NULL	Any restrictions associated with the shipment.
DESCRIPTION	VARCHAR(254)	NULL	A description of the shipping method, such as FedEx Overnight or Standard.
PO_BOX_ALLOWED	NUMBER	NULL	Specifies whether or not the shipment can be left at a post office box.
SIGNATURE_REQUIRED	NUMBER	NULL	Specifies whether or not a signature is required upon receipt of the shipment.
SATURDAY_DELIVERY	NUMBER	NULL	Specifies whether or not the shipment can be delivered on Saturday.
INTERNATIONAL_DELIVERY	NUMBER	NULL	Specifies whether or not international delivery is an option.
SIZE_LIMIT	NUMBER(16,4)	NULL	The size limit for the shipment.
PACKAGING_TYPE	VARCHAR(50)	NULL	The packaging type for the shipment.

The WLCS_TRANSACTION Database Table

Table 10-13 describes the metadata for the Commerce services WLCS_TRANSACTION table. This table is used to store data for every payment transaction in the order processing database.

See the section “” on page 10-25 for information about the constraint defined for this table.

The Primary Key is `TRANSACTION_ID`.

Table 10-13 WLCS_TRANSACTION Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
TRANSACTION_ID	VARCHAR(25)	NOT NULL	A unique identifier for the transaction. This field is the table’s primary key and cannot be `NULL`. All other fields in the `WLCS_` `TRANSACTION` table can be `NULL`.
BATCH_ID	VARCHAR(15)	NULL	A unique identifier of a batch submitted for settlement, as returned by the Payment Web service. This field need not be populated for other external payment services.
TRAN_DATE	DATE	NULL	The date of the transaction (that is, date on which the transaction was first started).
TRAN_STATUS	VARCHAR(20)	NULL	The current status of the transaction (Settled, Authorized, MarkedForSettle, PendingSettle, Retry, or Settled).
TRAN_AMOUNT	NUMBER(16,4)	NULL	The most recent amount applied to the transaction. MarkForSettle amounts can be different from the authorization amount.
TRAN_CURRENCY	VARCHAR(30)	NULL	The currency of the transaction.

Table 10-13 WLCS_TRANSACTION Table Metadata (Continued)

Column Name	Data Type	Null Value	Description and Recommendations
CC_NUMBER	VARCHAR(200)	NULL	The customer's credit card number. This is encrypted if is.encryption.enable is set to true in the weblogiccommerce.properties file.
CC_TYPE	VARCHAR(20)	NULL	The customer's credit card type, such as VISA or MasterCard.
CC_EXP_DATE	DATE	NULL	The expiration date on the customer's credit card.
CC_NAME	VARCHAR(50)	NULL	The credit card holder's name.
CC_DISPLAY_NUMBER	VARCHAR(20)	NULL	The version of the credit card number that is displayed (displays all Xs except last 4-digits).
CC_COMPANY	VARCHAR(50)	NULL	The name of the credit card company.
GEOCODE	VARCHAR(2)	NULL	The code used by the TAXWARE system to identify taxes for the order based on jurisdiction.
STREET1	VARCHAR(30)	NULL	The first line in the customer's street address.
STREET2	VARCHAR(30)	NULL	The second line in the customer's street address.
CITY	VARCHAR(30)	NULL	The city in the customer's address.
STATE	VARCHAR(40)	NULL	The state in the customer's address.
COUNTRY	VARCHAR(40)	NULL	The country in the customer's address.
POBOX	VARCHAR(30)	NULL	The post office box in the customer's address.
DESCRIPTION	VARCHAR(30)	NULL	Any additional data. Can be NULL.
COUNTY	VARCHAR(50)	NULL	The county in the customer's address.

Table 10-13 WLCS_TRANSACTION Table Metadata (Continued)

Column Name	Data Type	Null Value	Description and Recommendations
POSTAL_CODE	VARCHAR(10)	NULL	The postal (ZIP) code in the customer's address.
POSTAL_CODE_TYPE	VARCHAR(10)	NULL	Format or type of postal code, generally determined by country, such as Zip code in the United States.

The WLCS_TRANSACTION_ENTRY Database Table

Table 10-14 describes the metadata for the Commerce services WLCS_TRANSACTION_ENTRY table. This table is used to store (log) the different states a payment transaction has passed through in the order processing database.

See "" on page 10-25 for information about the constraint defined for this table.

The Primary Key is `TRANSACTION_ENTRY_ID`.

Table 10-14 WLCS_TRANSACTION_ENTRY Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
TRANSACTION_ENTRY_ID	NUMBER(25)	NOT NULL	A unique identifier for the transaction entry. This field is the table's primary key and cannot be `NULL`. All other fields in the `WLCS_TRANSACTION_ENTRY` table can be `NULL`.
TRAN_ENTRY_SEQUENCE	VARCHAR(30)	NULL	Represents the running count per transaction.
TRAN_ENTRY_DATE	DATE	NULL	The date of the log entry.
TRAN_ENTRY_STATUS	VARCHAR(20)	NULL	The status of the transaction when this entry was made.
TRAN_ENTRY_AMOUNT	NUMBER(16,4)	NULL	The amount of the transaction when the log entry was made.
TRAN_ENTRY_CURRENCY	VARCHAR(30)	NULL	The currency of the transaction.

Table 10-14 WLCS_TRANSACTION_ENTRY Table Metadata (Continued)

Column Name	Data Type	Null Value	Description and Recommendations
`TRANSACTION_ID`	`VARCHAR(25)`	`NULL`	A unique identifier for the transaction.

The SQL Scripts Used to Create the Database

The database schemas for WebLogic Portal and WebLogic Personalization Server are all created by executing the `create_all` script for the target database environment.

Scripts

Regardless of your database, execute one of the following to generate the necessary database objects for the modules desired (WebLogic Portal, WebLogic Personalization Server, Commerce services, Campaign services and Sample Portal):

- `PORTAL_HOME\db\create_all.bat` (Windows)
- `PORTAL_HOME/db/create_all.sh` (UNIX)

The following are the various directories underneath

`WL_COMMERCE_HOME/db`

(as seen in a UNIX environment):

```
PORTAL_HOME/db/cloudscape/351
PORTAL_HOME/db/oracle/817
```

Note: In this documentation,`PORTAL_HOME` is used to designate the directory where the product is installed.

Each of the databases supported have the same number of scripts in each of their subdirectories. The scripts are listed and described in Table 10-15 below.

Table 10-15 The Scripts Supporting the Databases

Script Name	Description
`create_all.bat`	Windows script used to connect to the database and create the necessary database objects for the modules desired (e.g., WebLogic Portal, WebLogic Personalization Server, Commerce services, Campaign services and Sample Portal)
`create_all.sh`	Unix script used to connect to the database and create the necessary database objects for the modules desired (e.g., WebLogic Portal, WebLogic Personalization Server, Commerce services, Campaign services and Sample Portal)
`campaign_create_fkeys.sql`	SQL script used to create all foreign keys associated with the Campaign services.
`campaign_create_indexes.sql`	SQL script used to create all indexes associated with the Campaign services.
`campaign_create_tables.sql`	SQL script used to create all tables associated with the Campaign services.
`campaign_create_triggers.sql`	SQL script used to create all database triggers associated with the Campaign services.
`campaign_create_views.sql`	SQL script used to create all views associated with the Campaign services.
`campaign_drop_constraints.sql`	SQL script used to drop all constraints (other than foreign keys) associated with the Campaign services.
`campaign_drop_fkeys.sql`	SQL script used to drop all foreign key constraints associated with the Campaign services.
`campaign_drop_indexes.sql`	SQL script used to drop all indexes associated with the Campaign services.
`campaign_drop_tables.sql`	SQL script used to drop all tables associated with the Campaign services.
`campaign_drop_views.sql`	SQL script used to drop all views associated with the Campaign services.
`p13n_create_fkeys.sql`	SQL script used to create all foreign keys associated with the WebLogic Personalization Server.
`p13n_create_indexes.sql`	SQL script used to create all indexes associated with the WebLogic Personalization Server.

Table 10-15 The Scripts Supporting the Databases (Continued)

Script Name	Description
`p13n_create_tables.sql`	SQL script used to create all tables associated with the WebLogic Personalization Server.
`p13n_create_triggers.sql`	SQL script used to create all database triggers associated with the WebLogic Personalization Server.
`p13n_create_views.sql`	SQL script used to create all views associated with the WebLogic Personalization Server.
`p13n_drop_constraints.sql`	SQL script used to drop all constraints (other than foreign keys) associated with the WebLogic Personalization Server.
`p13n_drop_fkeys.sql`	SQL script used to drop all foreign key constraints associated with the WebLogic Personalization Server.
`p13n_drop_indexes.sql`	SQL script used to drop all indexes associated with the WebLogic Personalization Server.
`p13n_drop_tables.sql`	SQL script used to drop all tables associated with the WebLogic Personalization Server.
`p13n_drop_views.sql`	SQL script used to drop all views associated with the WebLogic Personalization Server.
`portal_create_fkeys.sql`	SQL script used to create all foreign keys associated with the WebLogic Portal.
`portal_create_indexes.sql`	SQL script used to create all indexes associated with the WebLogic Portal.
`portal_create_tables.sql`	SQL script used to create all tables associated with the WebLogic Portal.
`portal_create_triggers.sql`	SQL script used to create all database triggers associated with the WebLogic Portal.
`portal_create_views.sql`	SQL script used to create all views associated with the WebLogic Portal.
`portal_drop_constraints.sql`	SQL script used to drop all constraints (other than foreign keys) associated with the WebLogic Portal.
`portal_drop_fkeys.sql`	SQL script used to drop all foreign key constraints associated with the WebLogic Portal.
`portal_drop_indexes.sql`	SQL script used to drop all indexes associated with the WebLogic Portal.

Table 10-15 The Scripts Supporting the Databases (Continued)

Script Name	Description
`portal_drop_tables.sql`	SQL script used to drop all tables associated with the WebLogic Portal.
`portal_drop_views.sql`	SQL script used to drop all views associated with the WebLogic Portal.
`sample_portal_create_fkeys.sql`	SQL script used to create all foreign keys associated with the Sample Portal.
`sample_portal_create_indexes.sql`	SQL script used to create all indexes associated with the Sample Portal.
`sample_portal_create_tables.sql`	SQL script used to create all tables associated with the Sample Portal.
`sample_portal_create_triggers.sql`	SQL script used to create all database triggers associated with the Sample Portal.
`sample_portal_create_views.sql`	SQL script used to create all views associated with the Sample Portal.
`sample_portal_drop_constraints.sql`	SQL script used to drop all constraints (other than foreign keys) associated with the Sample Portal.
`sample_portal_drop_fkeys.sql`	SQL script used to drop all foreign key constraints associated with the Sample Portal.
`sample_portal_drop_indexes.sql`	SQL script used to drop all indexes associated with the Sample Portal.
`sample_portal_drop_tables.sql`	SQL script used to drop all tables associated with the Sample Portal.
`sample_portal_drop_views.sql`	SQL script used to drop all views associated with the Sample Portal.
`wlcs_create_fkeys.sql`	SQL script used to create all foreign keys associated with the Commerce services.
`wlcs_create_indexes.sql`	SQL script used to create all indexes associated with the Commerce services.
`wlcs_create_tables.sql`	SQL script used to create all tables associated with the Commerce services.
`wlcs_create_triggers.sql`	SQL script used to create all database triggers associated with the Commerce services.

Table 10-15 The Scripts Supporting the Databases (Continued)

Script Name	Description
`wlcs_create_views.sql`	SQL script used to create all views associated with the Commerce services.
`wlcs_drop_constraints.sql`	SQL script used to drop all constraints (other than foreign keys) associated with the Commerce services.
`wlcs_drop_fkeys.sql`	SQL script used to drop all foreign key constraints associated with the Commerce services.
`wlcs_drop_indexes.sql`	SQL script used to drop all indexes associated with the Commerce services.
`wlcs_drop_tables.sql`	SQL script used to drop all tables associated with the Commerce services.
`wlcs_drop_views.sql`	SQL script used to drop all views associated with the Commerce services.

Defined Constraints

Various constraints are defined and used in the Order database schema. These constraints can be found in the following scripts:

`wlcs_create_fkeys.sql`—contains the Foreign Keys

`wlcs_create_tables.sql`—contains the Check Constraints

Table 10-16 Constraints Defined on Order Database Tables

Table Name	Constraints
DISCOUNT_ASSOCIATION	**Column**—CUSTOMER_ID **Constraint—**FK1_DISC_ASSOC **Constraint Type—**FOREIGN KEY Ensures that each CUSTOMER_ID references an existing WLCS_CUSTOMER via the CUSTOMER_ID column. **Column**—DISCOUNT_ID **Constraint—**FK2_DISC_ASSOC **Constraint Type—**FOREIGN KEY Ensures that each DISCOUNT_ID references an existing DISCOUNT via the DISCOUNT_ID column.
WLCS_CREDIT_CARD	**Column**—CUSTOMER_ID **Constraint—**FK1_CREDIT_CARD **Constraint Type—**FOREIGN KEY Ensures that each CUSTOMER_ID references an existing WLCS_CUSTOMER via the CUSTOMER_ID column
WLCS_ORDER_LINE	**Column**—ORDER_ID **Constraint—**FK1_ORDER_LINE **Constraint Type—**FOREIGN KEY Ensures that each ORDER_ID references an existing WLCS_ORDER via the ORDER_ID column.
ORDER_ADJUSTMENT	**Column**—DISCOUNT_ID **Constraint—**FK1_ORDER_ADJ **Constraint Type—**FOREIGN KEY Ensures that each DISCOUNT_ID references an existing DISCOUNT via the DISCOUNT_ID column.
ORDER_LINE_ADJUSTMENT	**Column**—DISCOUNT_ID **Constraint—**FK1_ORDER_L_ADJ **Constraint Type—**FOREIGN KEY Ensures that each DISCOUNT_ID references an existing DISCOUNT via the DISCOUNT_ID column
WLCS_SHIPPING_ADDRESS	**Column**—CUSTOMER_ID **Constraint—** FK1_SHIP_ADDR **Constraint Type—**FOREIGN KEY Ensures that each CUSTOMER_ID references an existing WLCS_CUSTOMER via the CUSTOMER_ID column.

Table 10-16 Constraints Defined on Order Database Tables (Continued)

Table Name	Constraints
`WLCS_TRANSACTION_ENTRY`	**Column**—TRANSACTION_ID **Constraint—**FK1_TRANS_ENTRY **Constraint Type—**FOREIGN KEY Ensures that each `TRANSACTION_ID` references an existing `WLCS_TRANSACTION` via the `TRANSACTION_ID` column.
`DISCOUNT`	**Column**—IS_GLOBAL **Constraint—** CC1_DISCOUNT **Constraint Type—**CHECK Ensures the value of the `IS_GLOBAL` column is either 0 (false) or 1 (true). **Column**—IS_ACTIVE **Constraint—** CC2_DISCOUNT **Constraint Type—**CHECK Ensures the value of the `IS_ACTIVE` column is either 0 (false) or 1 (true).

Index