



BEA WebLogic Commerce Server

Guide to Processing Orders and Managing Purchases

BEA WebLogic Commerce Server 3.5
Document Edition 3.5.3
May 2003

Copyright

Copyright © 2001 BEA Systems, Inc. All Rights Reserved.

Restricted Rights Legend

This software and documentation is subject to and made available only pursuant to the terms of the BEA Systems License Agreement and may be used or copied only in accordance with the terms of that agreement. It is against the law to copy the software except as specifically allowed in the agreement. This document may not, in whole or in part, be copied photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from BEA Systems, Inc.

Use, duplication or disclosure by the U.S. Government is subject to restrictions set forth in the BEA Systems License Agreement and in subparagraph (c)(1) of the Commercial Computer Software-Restricted Rights Clause at FAR 52.227-19; subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, subparagraph (d) of the Commercial Computer Software--Licensing clause at NASA FAR supplement 16-52.227-86; or their equivalent.

Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. FURTHER, BEA Systems DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIAL IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

Trademarks or Service Marks

BEA, WebLogic, Tuxedo, and Jolt are registered trademarks of BEA Systems, Inc. How Business Becomes E-Business, BEA WebLogic E-Business Platform, BEA Builder, BEA Manager, BEA eLink, BEA WebLogic Commerce Server, BEA WebLogic Personalization Server, BEA WebLogic Process Integrator, BEA WebLogic Collaborate, BEA WebLogic Enterprise, and BEA WebLogic Server, E-Business Control Center, and BEA Campaign Manager for WebLogic are trademarks of BEA Systems, Inc.

All other product names may be trademarks of the respective companies with which they are associated.

Guide to Managing Purchases and Processing Orders

Document Edition	Part Number	Date	Software Version
3.5.3	N/A	May 2003	WebLogic Commerce Server 3.5

Contents

1. Overview of Managing Purchases and Processing Orders	
What Are Managing Purchases and Processing Orders Services.....	1-2
High-level Architecture.....	1-4
Development Roles.....	1-6
Next Steps.....	1-6
2. Discounts	
User and Global Discounts.....	2-1
Introduction to How Discounts Work.....	2-2
Discount Management Service.....	2-3
Definition Parameters.....	2-3
Saving Versus Deploying Discounts.....	2-5
Association Service.....	2-7
Price Service.....	2-7
Triggers and Targets Specifications.....	2-8
Two Examples of Using Triggers and Targets.....	2-9
Consumption Model.....	2-10
How Discounts Are Applied.....	2-11
Priority.....	2-11
How Discounts Are Calculated.....	2-12
Examples.....	2-13
Item Discounts.....	2-13
Form of Discount Rules.....	2-13
Order Rules.....	2-15
3. Shopping Cart Management Services	
JavaServer Pages (JSPs).....	3-2

Common JSP Template Elements	3-2
shoppingcart.jsp Template.....	3-4
Sample Browser View.....	3-5
Location in the WebLogic Commerce Server Directory Structure.....	3-8
Tag Library Imports	3-8
Java Package Imports	3-8
Location in Default Webflow.....	3-9
Events	3-9
Dynamic Data Display	3-11
Form Field Specification.....	3-14
Input Processors.....	3-15
DeleteProductItemFromShoppingCartIP	3-15
EmptyShoppingCartIP.....	3-16
InitShoppingCartIP.....	3-17
UpdateShoppingCartQuantitiesIP	3-17
UpdateSkuIP.....	3-18
Pipeline Components.....	3-19
DeleteProductItemFromSavedListPC	3-19
MoveProductItemToSavedListPC.....	3-20
MoveProductItemToShoppingCartPC	3-21
RefreshSavedListPC.....	3-22
PriceShoppingCartPC.....	3-22
AddToCartTrackerPC.....	3-23
RemoveFromCartTrackerPC.....	3-24
UpdateShoppingCartQuantitiesTrackerPC.....	3-24

4. Shipping Services

JavaServer Pages	4-2
shipping.jsp Template.....	4-2
Sample Browser View.....	4-2
Location in the WebLogic Commerce Server Directory Structure.....	4-4
Tag Library Imports	4-4
Java Package Imports	4-4
Location in Default Webflow.....	4-5
Events	4-5

Dynamic Data Display	4-6
Form Field Specification.....	4-8
selectaddress.jsp Template	4-9
Sample Browser View	4-9
Location in the WebLogic Commerce Server Directory Structure...	4-10
Tag Library Imports	4-10
Java Package Imports.....	4-11
Location in Default Webflow	4-11
Events.....	4-12
Dynamic Data Display	4-13
Form Field Specification.....	4-16
addaddress.jsp Template	4-17
Sample Browser View	4-17
Location in the WebLogic Commerce Server Directory Structure...	4-18
Tag Library Imports	4-19
Java Package Imports.....	4-19
Location in Default Webflow	4-19
Included JSP Templates	4-20
Events.....	4-20
Dynamic Data Display	4-21
Form Field Specification.....	4-21
Input Processors.....	4-23
InitShippingMethodListIP.....	4-23
UpdateShippingAddressIP	4-24
ValidateAddressIP.....	4-25
ValidateShippingInfoIP.....	4-26
Pipeline Components.....	4-27
AddShippingAddressPC.....	4-27
CalculateShippingPC	4-28
DeleteShippingAddressPC.....	4-29

5. Taxation Services

JavaServer Pages (JSPs)	5-2
selecttaxaddress.jsp Template	5-2
Sample Browser View	5-2

Location in the WebLogic Commerce Server Directory Structure.....	5-3
Tag Library Imports	5-3
Java Package Imports	5-4
Location in Default Webflow.....	5-4
Included JSP Templates	5-4
Events	5-5
Dynamic Data Display	5-5
Form Field Specification	5-7
Input Processors.....	5-9
DecideShippingAddressPageIP	5-9
UpdateShippingAddressIP	5-10
Pipeline Components	5-11
TaxCalculateLineLevelPC	5-11
TaxCalculateAndCommitLineLevelPC	5-12
TaxVerifyShippingAddressPC	5-12
Integration with TAXWARE.....	5-14
Important TAXWARE Considerations	5-14
TAXWARE Installation	5-15
Installation Directory Structure	5-16
Testing the TAXWARE Installation	5-18
Changing the TAXWARE Directory Structure	5-19
TAXWARE Configuration and Deployment	5-20
Addresses and Taxation	5-20
TAXWARE-Specific Properties	5-23
Run-Time Configuration	5-33
Configuring the HTTP Server for TAXWARE	5-36
Tax Codes and the Product Catalog	5-41
Updating TAXWARE Tax Data	5-41
TAXWARE Checklist.....	5-41
Viewing Debugging Information in TAXWARE	5-42
Removing Tax Calculations	5-42
Modifying the Pipeline Properties File	5-42
Modifying the Webflow Properties File.....	5-43
What if I Don't Want to Use TAXWARE to Calculate My Taxes?	5-45

6. Payment Services

JavaServer Pages (JSPs)	6-2
payment.jsp Template	6-2
Sample Browser View	6-2
Location in the WebLogic Commerce Server Directory Structure	6-3
Tag Library Imports	6-3
Java Package Imports	6-4
Location in Default Webflow	6-4
Included JSP Templates	6-4
Events	6-5
Dynamic Data Display	6-5
Form Field Specification	6-6
paymentnewcc.jsp Template	6-7
Sample Browser View	6-7
Location in the WebLogic Commerce Server Directory Structure	6-8
Tag Library Imports	6-9
Java Package Imports	6-9
Location in Default Webflow	6-9
Included JSP Templates	6-10
Events	6-10
Dynamic Data Display	6-10
Form Field Specification	6-10
paymenteditcc.jsp Template	6-12
Sample Browser View	6-12
Location in the WebLogic Commerce Server Directory Structure	6-13
Tag Library Imports	6-13
Java Package Imports	6-14
Location in Default Webflow	6-14
Included JSP Templates	6-15
Events	6-15
Dynamic Data Display	6-15
Form Field Specification	6-17
Input Processors	6-20
PaymentAuthorizationIP	6-20
UpdatePaymentInfoIP	6-21

Pipeline Components	6-22
PaymentAuthorizationHostPC	6-22
PaymentAuthorizationTerminalPC	6-24
Integration with CyberCash	6-26
Configuration Activities for Using CyberCash	6-27
Payment Models	6-29
How Do I Switch Between the Two Payment Models?	6-31
What if I Don't Want to Use CyberCash for Credit Card Processing?	6-33
Credit Card Security Service	6-39

7. Order Summary and Confirmation Services

JavaServer Pages (JSPs)	7-2
checkout.jsp Template	7-2
Sample Browser View	7-2
Location in the WebLogic Commerce Server Directory Structure	7-4
Tag Library Imports	7-4
Java Package Imports	7-5
Location in Default Webflow	7-5
Events	7-6
Dynamic Data Display	7-6
Form Field Specification	7-12
confirmorder.jsp Template	7-12
Sample Browser View	7-12
Location in the WebLogic Commerce Server Directory Structure	7-14
Tag Library Imports	7-14
Java Package Imports	7-15
Location in Default Webflow	7-15
Events	7-15
Dynamic Data Display	7-16
Form Field Specification	7-20
Input Processors	7-21
Pipeline Components	7-21
CommitOrderPC	7-21
ResetCheckoutPC	7-22
PurchaseTrackerPC	7-23

8. Extending the Data Model

Data Model Extensions.....	8-2
Persistence Architecture	8-3
Adding Run-Time Attributes to Customer Data	8-6
Adding Run-Time Attributes to Other Entities	8-8
Extending the Schema	8-9
Overview of Approach to Extending the WebLogic Commerce Server Schema.....	8-10
Adding Attributes Against the WLCS_CUSTOMER, WLCS_ORDER, WLCS_TRANSACTION and WLCS_SHIPPING_METHOD Tables....	8-12
Adding Attributes Against the WLCS_ORDER_LINE Table.....	8-13
Adding Attributes Against the WLCS_CREDIT_CARD and WLCS_SHIPPING_ADDRESS Tables.....	8-16
Transaction Management	8-19

9. Using the Order and Payment Management Pages

Starting the WebLogic Commerce Server Administration Tools	9-2
Using the Order Management Search Page.....	9-4
Searching for an Order by Customer ID	9-4
Searching for an Order by Order Identifier Number.....	9-6
Searching for an Order by Date Range	9-8
Updating Order Status	9-12
Changing Order Status	9-12
Tailoring Order Status to Your Business	9-13
Using the Payment Management Search Page	9-15
Searching for a Payment by Customer ID.....	9-16
Searching for a Payment by Status.....	9-17
Authorizing, Capturing, and Settling Payments.....	9-19
Authorizing the Transaction.....	9-19
Capturing the Transaction	9-21
Settling the Transaction	9-21

10. The Order Processing Database Schema

The Entity-Relation Diagram	10-1
List of Tables Comprising the Order Processing Schema.....	10-4

The Order Processing Data Dictionary	10-5
The WLCS_CREDIT_CARD Database Table	10-5
The WLCS_COUNTRY Database Table	10-7
The WLCS_CURRENCY Database Table	10-7
The WLCS_CUSTOMER Database Table	10-8
The DISCOUNT Database Table	10-9
The DISCOUNT_ASSOCIATION Database Table	10-11
The DISCOUNT_SET Database Table	10-11
The WLCS_ORDER Database Table	10-12
The WLCS_ORDER_LINE Database Table	10-14
The ORDER_ADJUSTMENT Database Table	10-15
The ORDER_LINE_ADJUSTMENT Database Table	10-16
The WLCS_SAVED_ITEM_LIST Database Table	10-17
The WLCS_SECURITY Database Table	10-17
The WLCS_SHIPPING_ADDRESS Database Table	10-18
The WLCS_SHIPPING_METHOD Database Table	10-19
The WLCS_TRANSACTION Database Table	10-21
The WLCS_TRANSACTION_ENTRY Database Table	10-23
The SQL Scripts Used to Create the Database	10-24
Cloudscape	10-24
Oracle	10-25
Defined Constraints	10-27

Index

About This Document

This document explains how to use the functionality within the BEA WebLogic Commerce Server™ Managing Purchases and Processing Orders 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 JSP templates, input processors, and Pipelines associated with the tax Web pages.
- Chapter 6, “Payment Services,” which describes the JSP templates, input processors, and Pipelines associated with the payment Web pages.
- 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 Managing Purchases and Processing Orders 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 commerce business engineer (CBE) 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, 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 Campaign Manager for WebLogic, WebLogic Commerce Server, and WebLogic Personalization Server 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.beasys.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 Campaign Manager for WebLogic, WebLogic Commerce Server, 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 Campaign Manager for WebLogic, WebLogic Commerce Server, 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 Commerce Server documents contain information that is relevant to using the Managing Purchases and Processing Orders 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 Campaign Manager for WebLogic, WebLogic Commerce Server, and WebLogic Personalization Server documentation is important to us. Send us e-mail at docsupport@beasys.com if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the Campaign Manager for WebLogic, WebLogic Commerce Server, and WebLogic Personalization Server documentation.

In your e-mail message, please indicate that you are using the documentation for the Campaign Manager for WebLogic, WebLogic Commerce Server, and WebLogic Personalization Server 3.5 release.

If you have any questions about this version of Campaign Manager for WebLogic, WebLogic Commerce Server, or WebLogic Personalization Server, or if you have problems installing and running Campaign Manager for WebLogic, WebLogic Commerce Server, or WebLogic Personalization Server, contact BEA Customer Support through BEA WebSUPPORT at www.beasys.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.
<i>italics</i>	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. <i>Examples:</i> <pre>#include <iostream.h> void main () the pointer psz chmod u+w * \tux\data\ap .doc tux.doc BITMAP float</pre>
monospace boldface text	Identifies significant words in code. <i>Example:</i> <pre>void commit ()</pre>
<i>monospace italic text</i>	Identifies variables in code. <i>Example:</i> <pre>String <i>expr</i></pre>
UPPERCASE TEXT	Indicates device names, environment variables, and logical operators. <i>Examples:</i> <pre>LPT1 SIGNON OR</pre>

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. <i>Example:</i> 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: <ul style="list-style-type: none"> ■ 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. <i>Example:</i> 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 Commerce Server provides out-of-the-box Managing Purchases and Processing Orders 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:

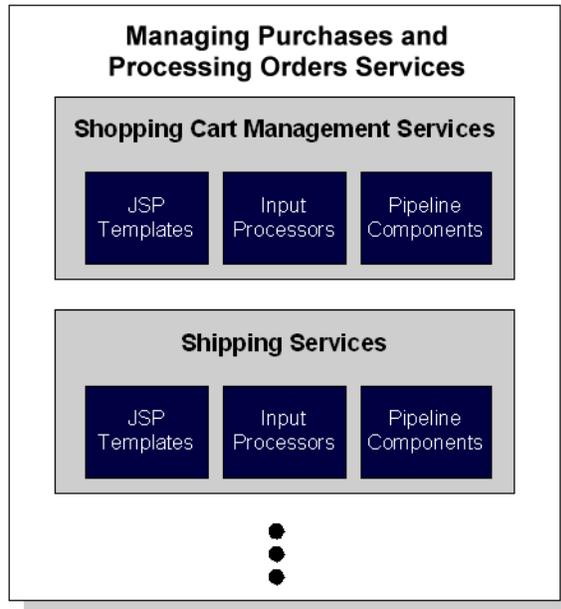
- What Are Managing Purchases and Processing Orders Services
- High-level Architecture
- Development Roles
- Next Steps

What Are Managing Purchases and Processing Orders Services

Managing Purchases and Processing Orders 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 Managing Purchases and Processing Orders services.

Figure 1-1 Structure of Managing Purchases and Processing Orders 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 Commerce Server 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*.

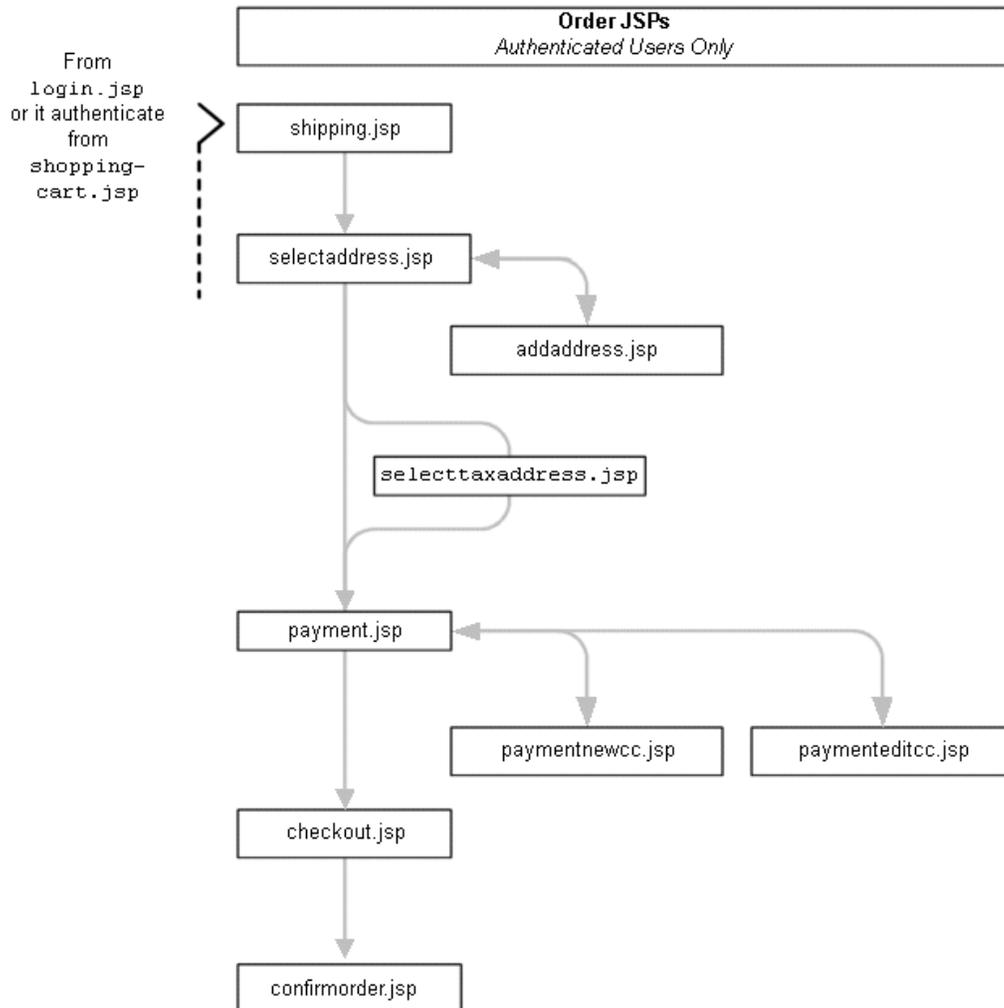
High-level Architecture

Managing Purchases and Processing Orders 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 Managing Purchases and Processing Orders 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 Managing Purchases and Processing Orders services (including the JSP templates, input processors, and Pipeline components associated

1 *Overview of Managing Purchases and Processing Orders*

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 commerce 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 Commerce Server and WebLogic Personalization Server with Portal Framework 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 Managing Purchases and Processing Orders 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”

1 *Overview of Managing Purchases and Processing Orders*

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 *Using the E-Business Control Center*.

This topic includes the following sections:

- User and Global Discounts
- Introduction to How Discounts Work
- Discount Management Service
- Association Service
- Price Service
- Triggers and Targets Specifications
- Examples

User and Global 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 user discounts. Discounts available to all customers are called global discounts.

Note: In the BEA E-Business Control Center, user discounts are referred to as campaign discounts and global discounts are referred to as stand-alone discounts.

If your product license is only for the BEA WebLogic Personalization Server with Portal Framework™, discounts will not be available. If you have the BEA Campaign Manager for WebLogic™, you can use both user and global discounts. With only the BEA WebLogic Commerce Server™, you can use global discounts, but not user discounts unless a Java developer writes the association coding.

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.

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.

2 Discounts

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.
- **Global Display Description**—applies only to global discounts. This feature is available so that JSP developers can show a description of the discount to customers.

Note: For user 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.

For more information, see “Saving Versus Deploying Discounts” on page 2-13.

Saving Versus Deploying Discounts

Saving and deploying discounts is usually done in the E-Business Control Center. It is important to understand the difference between saving and deploying discounts. When a discount is saved, the Discount Management service stores it in an XML document and the discount is not available to pricing operations. To put a discount into action, it must be deployed.

A discount is deployed via a method call to the Discount Management service. Deploying a discount essentially involves the Discount system parsing the XML and populating a database table. After a discount is deployed, only the active/deactive flag, the priority of the discount, and end date for the discount may be modified; all other values are immutable. This restriction maintains the integrity in the Price service.

The XML document is actually a collection of discounts. XML is used to save discounts because XML documents are easily transferred from a development environment to a production environment.

BEA provides a utility to upload and download discount XML and to deploy discounts. This utility, called `discountUtil`, is most helpful for migrating discounts between servers. You can find the `discountUtil` file at the following location, where `WL_COMMERCE_HOME` is the directory in which you installed WebLogic Commerce Server:

```
%WL_COMMERCE_HOME%\bin\win\dscountUtil.bat (Windows)
$WL_COMMERCE_HOME/bin/unix/sdscountUtil.sh (UNIX)
```

Notes: To use the utility, WebLogic Commerce Server must be running.

The discount system is currently implemented to use a single discount set called `DefaultDiscountSet`.

Use the following syntax to specify the commands, options, and set:

```
discountUtil [<options>] <command> <setname>
```

The utility has the following commands:

- `add <set name>`
Adds the discount set XML to the discount management service if it doesn't already exist. If the set already exists, an error is reported. Use the `update` command to replace the existing XML with a new XML file.
- `update <set name>`
Replaces the XML for an existing discount set.

2 Discounts

- `retrieve <set name>`
Retrieves the XML for the named discount set.
- `deploy <set name>`
Deploys the named discount set.
- `load <set name>`
Adds or updates the XML, then deploys the set.

The utility also has several options for specifying which server to talk to and the names of input and output files for the add/update and retrieve commands respectively. To see a list of all available options, run the utility with the help options. For example:

```
discountUtil.sh -h
```

The following options are most commonly necessary:

- `-i <file name>`
Specifies the input XML file for the add, update and load commands.
- `-o <file name>`
Specifies the output XML file to write to when using the retrieve command.
- `-host <hostname>`
Specifies the host on which the WLCS server is running.
- `-port <port number>`
Specifies the port on which the WLCS server is listening for normal requests.

Deleting a Discount

You can delete a deployed discount only when it is not referred to by orders or discount associations. The Discount Management service enforces this constraint. Deleting a deployed discount definition does not delete the discount from the XML document. You can modify the XML document and then redeploy the discount using the E-Business Control Center.

Association Service

The Campaign service uses the Association service to link discounts with particular customers. Campaigns provide 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. Global 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.

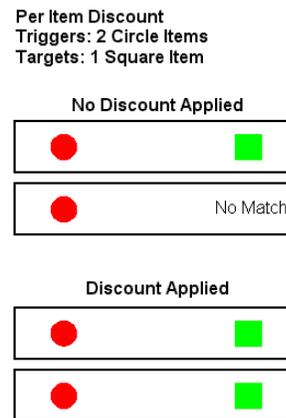
Triggers and Targets Specifications

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.

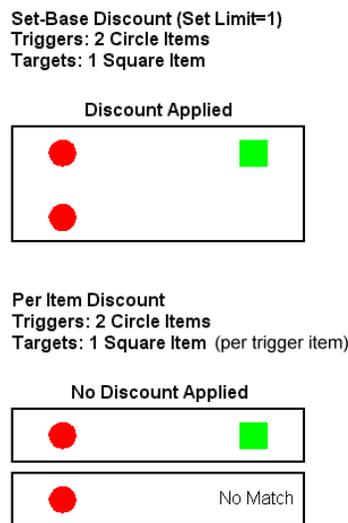
In a per item discount, each individual trigger item must be *paired* with items designated by the target specification. Figure 2-1 shows this relationship. Notice that in both cases the triggers and the targets is the same.

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 2-21. 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 global 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 user 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:
$$\text{newPrice} = \text{oldPrice}(1 - \text{value}), \text{ where } 0.0 \leq \text{value} \leq 1.0 \text{ and value is a property of the discount definition. For example, } \$90 = \$100(1 - .1).$$
- **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:
$$\text{newPrice} = \text{oldPrice} - \text{value}, \text{ where value is any non-negative monetary value. For example, } \$45 = \$50 - \$5.$$
- **Fixed Price Discount**—A discount where the price is reduced to a particular price. The calculator applies the following formula:
$$\text{newPrice} = \text{value}, \text{ where value is any non-negative monetary value. For example, } \$12 = \$12, \text{ where the original price was } \$15.$$

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

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

2 Discounts

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 \geq \$50, apply a 10% discount to qualifying order.

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

2 *Discounts*

Rule: For order subtotal \geq \$50 AND order subtotal \leq \$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 \geq \$100, apply a 10% discount to shipping.

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

Rule: For order subtotal \geq \$100 OR order subtotal \leq \$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.

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

3 Shopping Cart Management Services

- PriceShoppingCartPC
- AddToCartTrackerPC
- RemoveFromCartTrackerPC
- UpdateShoppingCartQuantitiesTrackerPC

JavaServer Pages (JSPs)

The Managing Purchases and Processing Orders 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 Commerce Server 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. 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 the WebLogic Commerce Server. 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 Commerce Server. The import call is:

3 Shopping Cart Management Services

```
<%@ 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 Commerce Server. 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.

3 Shopping Cart Management Services

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

The screenshot displays a web page for a shopping cart. At the top, there is a navigation bar with the BEA logo, links for 'About Current Template: shoppingcart.jsp', 'Template Index', and 'Administration', and a red banner for 'BEA WebLogic Commerce Server 3.5 Commerce Templates'. Below the navigation bar is a promotional banner for 'Routers!' with a 'Click here to see our full line of powerful Routers!' link. A secondary navigation bar contains links for 'Home', 'Search', 'View Cart', and 'Log in'. The main content area is divided into a left sidebar and a central cart section. The sidebar includes a 'Quick Look-up' search box, a 'Don't Forget' advertisement for extension cords, and a note about catalog data from TPN Register. The central cart section is titled 'Shopping Cart' and contains a table with columns for Quantity, Item, List Price, Our Price, You Save, and Subtotal. A single item is listed: 'drill-9-10144' with a quantity of 4, a list price of \$79.95, an our price of \$62.95, a savings of \$68.00, and a subtotal of \$251.80. A 'Remove' button is next to the item. Below the table, the total is shown as '\$251.80 (before shipping and taxes)'. There are buttons for 'Empty cart', 'Check out >', 'Continue shopping', and 'Check out >', along with an 'Update' button and a note about additional discounts.

Quick Look-up:
Enter keywords

Don't Forget
Extension Cords!
100% click here

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.

Quantity	Item	List Price	Our Price	You Save	Subtotal	
<input type="text" value="4"/>	drill-9-10144	\$ 79.95	\$ 62.95	\$ 68.00	\$ 251.80	<input type="button" value="Remove"/>
					Total \$ 251.80	
						(before shipping and taxes)

You may qualify for additional discounts! Please [log in](#).

Press this button to if you changed any quantities.

Main Content Area

Copyright © 1999-2001,
[BEA Systems Inc.](#)

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

The screenshot shows a shopping cart interface. At the top, there is a navigation bar with links for Home, Search, View Cart, and Logout. A sidebar on the left contains a welcome message, a 'View History' section with links for Orders and Payments, and a search box. The main content area displays a 'Shopping Cart' table with one item: 'drill-9-10505' with a quantity of 1, a list price of \$119.95, an our price of \$101.95, a you save of \$18.00, and a subtotal of \$101.95. The total is \$101.95. The page also includes a 'Routers!' advertisement and a 'See Our Large Selection of Saws Here!' advertisement.

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.

3 Shopping Cart Management Services

Location in the WebLogic Commerce Server Directory Structure

You can find the `shoppingcart.jsp` template file at the following location, where `WL_COMMERCE_HOME` is the directory in which you installed WebLogic Commerce Server:

```
%WL_COMMERCE_HOME%\config\wlcsDomain\applications\wlcsApp\wlcs\commerce\shoppingcart.jsp (Windows)
$WL_COMMERCE_HOME/config/wlcsDomain/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="pipeline.tld" prefix="pipeline" %>
<%@ 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 following directory for the WebLogic Commerce Server Web application:

```
%WL_COMMERCE_HOME%\config\wlcsDomain\applications\wlcsApp\wlcs\commerce\wlcs\WEB-INF (Windows)
$WL_COMMERCE_HOME/config/wlcsDomain/applications/wlcsApp/wlcs/commerce/WEB-INF (UNIX)
```

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.webflow.*" %>
<%@ page import="com.beasys.commerce.axiom.units.*" %>
<%@ page import="com.beasys.commerce.foundation.pipeline.*" %>
```

```
<%@ page import="com.beasys.commerce.ebusiness.shoppingcart.*" %>
<%@ page import="com.bea.commerce.ebusiness.price.service.DiscountPresentation"
%>
<%@ page import="com.bea.commerce.ebusiness.price.quote.OrderAdjustment" %>
<%@ page import="com.bea.commerce.ebusiness.price.quote.AdjustmentDetail" %>
```

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.

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.

3 Shopping Cart Management Services

Table 3-1 shoppingcart.jsp Events

Event	Webflow Response(s)
--	InitShoppingCartIP
--	RefreshSavedList
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, as they are defined in the `pipeline.properties` file. 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 `getPipelineProperty` 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
<code>PipelineSessionConstants</code> <code>.SAVED_SHOPPING_CART</code>	<code>com.beasys.commerce.ebusiness</code> <code>.shoppingcart.ShoppingCart</code>	The saved shopping cart (source of the saved items).
<code>PipelineSessionConstants</code> <code>.SHOPPING_CART</code>	<code>com.beasys.commerce.ebusiness</code> <code>.shoppingcart.ShoppingCart</code>	The currently active shopping cart.

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

3 Shopping Cart Management Services

Listing 3-1 Retrieving Shopping Cart Attributes

```
<pipeline:getPipelineProperty
  propertyName="<%=PipelineSessionConstants.SHOPPING_CART%>"
  returnName="shoppingCart"
  returnType="com.beasys.commerce.ebusiness.shoppingcart.ShoppingCart"/>

<pipeline:getPipelineProperty
  propertyName="<%=PipelineSessionConstants.SAVED_SHOPPING_CART%>"
  returnName="savedShoppingCart"
  returnType="com.beasys.commerce.ebusiness.shoppingcart.ShoppingCart"/>
```

Note: For more information on the `getPipelineProperty` 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
<code>getShoppingCartLineCollection()</code>	A collection of the individual lines in the shopping cart (that is, <code>ShoppingCartLine</code>).
<code>getTotal</code>	In this instance, the total tax specified by the <code>OrderConstants.LINE_TAX</code> 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.
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

```
<wl:repeat set="<%shoppingCart.getShoppingCartLineCollection().iterator()%>"
id="shoppingCartLine" type="ShoppingCartLine" count="100000">
<tr>
  <td>
    <%=shoppingCartLine.getProductItem().getName()%>
  </td>
  <td align="right">
    <input type="text" name="NewQuantity_<%=shoppingCartLine.getProductItem().
getKey().getIdentifier()%>"
value="<%=quantityFormat.format(shoppingCartLine.getQuantity())%>"
size="9">
  </td>
  <td align="right">
    <%=shoppingCartLine.getProductItem().getMsrp().getCurrency()%>
    <%=priceFormat.format(shoppingCartLine.getProductItem().getMsrp().
getValue())%>
  </td>
</tr>
```

3 Shopping Cart Management Services

```
<td align="center">
<input type="submit" value="Delete" onclick="submitForm('shoppingCartForm',
'button(deleteItemFromShoppingCart)', '<%=shoppingCartLine.getProductItem()
.getKey().getIdentifier()%>')">
</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

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 (<code>shoppingcart.jsp</code>), used by the Webflow.
<code>HttpRequestConstants.CATALOG_ITEM_SKU</code>	Hidden	SKU of the item that the event is to operate on.
<code>NewQuantity_<SKU></code> Where <code><SKU></code> 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	<code>com.beasys.commerce.ebusiness.shoppingcart.webflow.DeleteProductItemFromShoppingCartIP</code>
Description	Removes the item from the shopping cart.
Required HttpServletRequest Parameters	<code>HttpRequestConstants.CATALOG_ITEM_SKU</code>
Required Pipeline Session Attributes	<code>PipelineSessionConstants.SHOPPING_CART</code> <code>PipelineSessionConstants.CATALOG_ITEM</code>
Updated Pipeline Session Attributes	<code>PipelineSessionConstants.SHOPPING_CART</code>
Removed Pipeline Session Attributes	None
Validation	None

3 Shopping Cart Management Services

Exceptions	<code>ProcessingException</code> , thrown if the required request parameters or required Pipeline session attributes are not available.
-------------------	---

EmptyShoppingCartIP

Class Name	<code>com.beasys.commerce.ebusiness.shoppingcart.webflow.EmptyShoppingCartIP</code>
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	<code>PipelineSessionConstants.SHOPPING_CART</code> <code>PipelineSessionConstants.UPDATED_QUANTITY_DELTAS</code> <code>PipelineSessionConstants.UPDATED_PRODUCT_ITEMS</code>
Removed Pipeline Session Attributes	None
Validation	None
Exceptions	None

InitShoppingCartIP

Class Name	com.beasys.commerce.ebusiness.shoppingcart.webflow. InitShoppingCartIP
Description	Initializes the active shopping cart prior to loading the <code>shoppingcart.jsp</code> 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	com.beasys.commerce.ebusiness.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.

3 Shopping Cart Management Services

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	com.beasys.commerce.ebusiness.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	<code>com.beasys.commerce.ebusiness.shoppingcart.pipeline.DeleteProductItemFromSavedListPC</code>
Description	Removes the item from the saved list and updates the <code>WLCS_SAVED_ITEM_LIST</code> table in the database.
Required Pipeline Session Attributes	<code>PipelineSessionConstants.CATALOG_ITEM_SKU</code> <code>PipelineSessionConstants.SAVED_SHOPPING_CART</code> <code>PipelineSessionConstants.USER_NAME</code>
Updated Pipeline Session Attributes	<code>PipelineSessionConstants.SAVED_SHOPPING_CART</code>
Removed Pipeline Session Attributes	None
Type	Session bean
JNDI Name	<code>com.beasys.commerce.ebusiness.shoppingcart.pipeline.DeleteProductItemFromSavedListPC</code>
Exceptions	<code>PipelineFatalException</code> , thrown if the required Pipeline session attributes are not available.

MoveProductItemToSavedListPC

Class Name	<code>com.beasys.commerce.ebusiness.shoppingcart.pipeline.MoveProductItemToSavedListPC</code>
Description	Removes the item from the shopping cart, adds it to the saved list, and then updates the <code>WLCS_SAVED_ITEM_LIST</code> table in the database.
Required Pipeline Session Attributes	<code>PipelineSessionConstants.CATALOG_ITEM_SKU</code> <code>PipelineSessionConstants.SAVED_SHOPPING_CART</code> <code>PipelineSessionConstants.SHOPPING_CART</code> <code>PipelineSessionConstants.USER_NAME</code>
Updated Pipeline Session Attributes	<code>PipelineSessionConstants.SAVED_SHOPPING_CART</code> <code>PipelineSessionConstants.SHOPPING_CART</code> <code>PipelineSessionConstants.CATALOG_ITEM</code> <code>PipelineSessionConstants.QUANTITY</code>
Removed Pipeline Session Attributes	None
Type	Session bean
JNDI Name	<code>com.beasys.commerce.ebusiness.shoppingcart.pipeline.MoveProductItemToSavedListPC</code>
Exceptions	<code>PipelineFatalException</code> , thrown if the required Pipeline session attributes are not available.

MoveProductItemToShoppingCartPC

Class Name	<code>com.beasys.commerce.ebusiness.shoppingcart.pipeline.MoveProductItemToShoppingCartPC</code>
Description	Removes the item from the saved list, adds it to the shopping cart with a quantity of 1, and then updates the <code>WLCS_SAVED_ITEM_LIST</code> table in the database.
Required Pipeline Session Attributes	<code>PipelineSessionConstants.CATALOG_ITEM_SKU</code> <code>PipelineSessionConstants.SAVED_SHOPPING_CART</code> <code>PipelineSessionConstants.SHOPPING_CART</code> <code>PipelineSessionConstants.USER_NAME</code>
Updated Pipeline Session Attributes	<code>PipelineSessionConstants.SAVED_SHOPPING_CART</code> <code>PipelineSessionConstants.SHOPPING_CART</code> <code>PipelineSessionConstants.CATALOG_ITEM</code>
Removed Pipeline Session Attributes	None
Type	Session bean
JNDI Name	<code>com.beasys.commerce.ebusiness.shoppingcart.pipeline.MoveProductItemToShoppingCartPC</code>
Exceptions	<code>PipelineFatalException</code> , thrown if the required Pipeline session attributes are not available.

3 Shopping Cart Management Services

RefreshSavedListPC

Class Name	<code>com.beasys.commerce.ebusiness.shoppingcart.pipeline.RefreshSavedListPC</code>
Description	Queries the <code>WLCS_SAVED_ITEM_LIST</code> 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	<code>PipelineSessionConstants.USER_NAME</code>
Updated Pipeline Session Attributes	<code>PipelineSessionConstants.SAVED_SHOPPING_CART</code>
Removed Pipeline Session Attributes	None
Type	Session bean
JNDI Name	<code>com.beasys.commerce.ebusiness.shoppingcart.pipeline.RefreshSavedListPC</code>
Exceptions	<code>PipelineFatalException</code> , thrown if the required Pipeline session attributes are not available.

PriceShoppingCartPC

Class Name	<code>com.beasys.commerce.ebusiness.shoppingcart.pipeline.PriceShoppingCartPC</code>
Description	Invokes the Pricing Service to compute the line totals, discounts, shopping cart total and shopping cart discounts
Required Pipeline Session Attributes	<code>PipelineSessionConstants.SHOPPING_CART</code> <code>PipelineSessionConstants.USER_NAME</code>

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

AddToCartTrackerPC

Class Name	com.bea.commerce.ebusiness.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 <i>Guide to Events and Behavior Tracking</i> .
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	<code>com.bea.commerce.ebusiness.tracking.pipeline.RemoveFromCartTrackerPC</code>
Description	Fires a <code>RemoveFromCartEvent</code> describing which item was just added to the cart. For more information about this event, see Event Details in the <i>Guide to Events and Behavior Tracking</i> .
Required Pipeline Session Attributes	<code>PipelineSessionConstants.CATALOG_ITEM</code> <code>PipelineSessionConstants.HTTP_SESSION_ID</code> <code>PipelineSessionConstants.USER_NAME</code> <code>PipelineSessionConstants.STOREFRONT</code> <code>PipelineSessionConstants.CUSTOM_REQUEST</code>
Updated Pipeline Session Attributes	None
Removed Pipeline Session Attributes	None
Type	Java object
JNDI Name	None
Exceptions	None

UpdateShoppingCartQuantitiesTrackerPC

Class Name	<code>com.bea.commerce.ebusiness.tracking.pipeline.UpdateShoppingCartQuantitiesTrackerPC</code>
Description	For each shopping cart line, if more items in the line were selected, fires an <code>AddToCartEvent</code> ; if fewer items in that line were selected, fires a <code>RemoveFromCartEvent</code> ; if the number of items in that line is the same as before, no event is fired.

Pipeline Components

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

3 *Shopping Cart Management Services*

4 Shipping Services

In Managing Purchases and Processing Orders 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 Managing Purchases and Processing Orders 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 Commerce ServerWeb 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 annotatted regions follow the figure.

Figure 4-1 Annotated shipping.jsp Template

The screenshot shows a web page titled "Shipping" with a navigation bar at the top. The navigation bar includes the BEA logo, "About Current Template: shipping.jsp", "Template Index", "Administration", and "BEA WebLogic Commerce Server 3.5 Commerce Templates". Below the navigation bar is a banner for "Routers!" with a "Home" link. The main content area is titled "Shipping" and contains three numbered regions:

- Region 1:** "How do you want this order shipped?" with radio buttons for "Second Day Air" (selected) and "Standard Shipping - 4 to 7 days".
- Region 2:** "Ship all at once, or 'as available'?" with radio buttons for "Please wait until the entire order is ready before shipping." (selected) and "Please ship items 1 ordered as they become available (you may incur additional shipping charges)."
- Region 3:** "Special Instructions" with a text input field.

Navigation buttons for "< Back" and "Continue >" are located at the bottom of the form. A sidebar on the left contains links for "Welcome Demo Customer", "View Profile", "Logout", "View History", "Orders", and "Payments". A "Don't Forget" banner for "Extension Cards!" is also present. The footer includes "Copyright © 1999-2001, BEA Systems Inc." and a "Built On" logo.

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.

4 Shipping Services

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

Location in the WebLogic Commerce Server Directory Structure

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

```
%WL_COMMERCE_HOME%\config\wlcsDomain\applications\wlcsApp\wlcs\commerce\order\shipping.jsp (Windows)
$WL_COMMERCE_HOME/config/wlcsDomain/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="pipeline.tld" prefix="pipeline"%>
```

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 following directory for the WebLogic Commerce Server Web application:

```
%WL_COMMERCE_HOME%\config\wlcsDomain\applications\wlcsApp\wlcs\WEB-INF (Windows)
$WL_COMMERCE_HOME/config/wlcsDomain/applications/wlcsApp/wlcs/WEB-INF (UNIX)
```

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="com.beasys.commerce.foundation.pipeline.*" %>
<%@ page import="com.beasys.commerce.ebusiness.shipping.*" %>
<%@ page import="com.beasys.commerce.webflow.*" %>
```

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`).

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)
<code>button(back)</code>	No business logic required. Loads <code>shoppingcart.jsp</code> .
<code>button(continue)</code>	<code>ValidateShippingInfoIP</code> .

4 Shipping Services

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 `getPipelineProperty` 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
<code>PipelineSessionConstants</code> <code>.SHIPPING_METHOD_LIST</code>	List of <code>com.beasys.commerce.ebusiness</code> <code>.shipping.ShippingMethodValue</code>	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

```
<pipeline:getPipelineProperty  
  propertyName="<%PipelineSessionConstants.SHIPPING_METHOD_LIST%">  
  returnName="shippingMethodList"  
  returnType="java.util.List"/>
```

Note: For more information on the `getPipelineProperty` 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

```

<table>
<tr>
  <td colspan=2>
    <b>Select Shipping Method</b>
  </td>
</tr>
<wl:repeat set="<%=shippingMethodList%>" id="shippingMethodValue"
type="ShippingMethodValue" count="100"
<tr>
  <td>
    <input type="radio" name="<%HttpRequestConstants.SHIPPING_METHOD%>"
      value="<%=shippingMethodValue.identifier%>" />
  </td>
  <td>
    <%=shippingMethodValue.description%>
  </td>
</tr>
</wl:repeat>
</table>

```

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

4 Shipping Services

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 (<code>shipping.jsp</code>), used by the Webflow.
<code>HttpRequestConstants.SHIPPING_METHOD</code>	Radio button	Identifies the shipping method the customer selects.
<code>HttpRequestConstants.SPECIAL_INSTRUCTIONS</code>	Textbox	Any special instructions the customer specifies.
<code>HttpRequestConstants.SPLITTING_PREFERENCE</code>	Radio button	String representing the splitting preference the customer selects.

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 %>`) 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

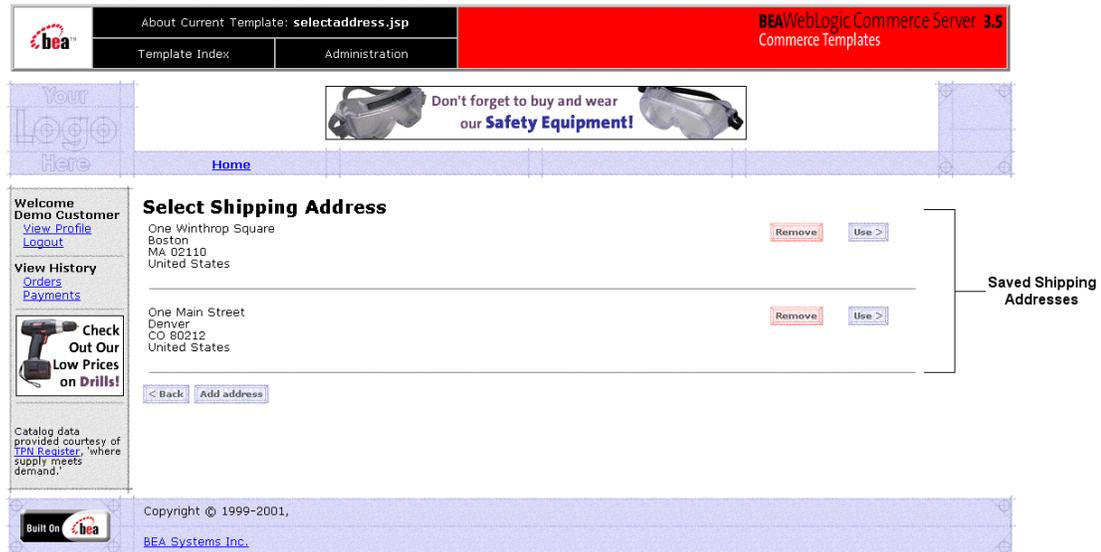
Figure 4-2 shows an annotated version of the `selectaddress.jsp` template. The Select Shipping Address region contains dynamically displayed data of the customer's saved shipping addresses. This is accomplished using a combination of WebLogic Server and WebLogic Personalization Server with Portal Framework 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.

4 Shipping Services

Figure 4-2 Annotated selectaddress.jsp Template



Location in the WebLogic Commerce Server Directory Structure

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

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

Tag Library Imports

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

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

4-10 Guide to Managing Purchases and Processing Orders

Note: For more information on the WebLogic Server JSP tags or the WebLogic Personalization Server with Portal Framework 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 following directory for the WebLogic Personalization Server with Portal Framework Web application:

```
%WL_COMMERCE_HOME%\config\wlcsDomain\applications\wlcsApp\wlcs\WEB-INF (Windows)
$WL_COMMERCE_HOME/config/wlcsDomain/applications/wlcsApp/wlcs/WEB-INF (UNIX)
```

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.foundation.pipeline*" %>
<%@ page import="com.beasys.commerce.axiom.contact.*" %>
<%@ page import="com.beasys.commerce.ebusiness.shipping.*" %>
<%@ page import="com.beasys.commerce.ebusiness.customer.*" %>
<%@ page import="com.beasys.commerce.webflow.*" %>
```

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`).

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

4 Shipping Services

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)
<code>button(back)</code>	No business logic required. Loads <code>shipping.jsp</code> .
<code>button(addNewShippingAddress)</code>	No business logic required. Loads <code>addaddress.jsp</code> .
<code>button(deleteShippingAddress)</code>	<code>UpdateAddressKeyIP</code> <code>DeleteShippingAddress</code>
<code>button(useShippingAddress)</code>	<code>UpdateShippingAddressIP</code> <code>TaxVerifyShippingAddress</code> <code>CalculateShippingCost</code> <code>TaxCalculateLineLevel</code>

Table 4-6 briefly describes each of the Pipelines from Table 4-5, as they are defined in the `pipeline.properties` file. For more information about individual Pipeline components, see “Pipeline Components” on page 4-27.

Table 4-6 Select Shipping Address Pipelines

Pipeline	Description
<code>TaxVerifyShippingAddress</code>	Contains <code>TaxVerifyShippingAddressPC</code> and is not transactional.
<code>CalculateShippingCost</code>	Contains <code>CalculateShippingCostPC</code> and is not transactional.
<code>TaxCalculateLineLevel</code>	Contains <code>TaxCalculateLineLevelPC</code> and is not transactional.

Table 4-6 Select Shipping Address Pipelines

Pipeline	Description
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 Personalization Server with Portal Framework'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 propertyName="shippingAddressMap"
  id="shippingAddressMap" />
```

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

4 Shipping Services

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 Personalization Server with Portal Framework'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
<code>getStreet1()</code>	The first line of the customer's street address.
<code>getStreet2()</code>	The second line of the customer's street address.
<code>getCity()</code>	The city in the customer's address.
<code>getState()</code>	The state in the customer's address.
<code>getPostalCode()</code>	The zip/postal code in the customer's address.
<code>getCountry()</code>	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%" >
      <div class="commentary">
        <a href="<%=WebflowJSPHelper.createWebflowURL(pageContext,
          "selectaddress.jsp", "button(deleteShippingAddress)","&" +
          HttpRequestConstants.ADDRESS_KEY + "=" + addressKey, true)%>">
          " border="0">
          </a>
        </div>
      </td>

    <td align="left" valign="top" width="5%" >
      <div class="commentary">
        <a href="<%=WebflowJSPHelper.createWebflowURL(pageContext,
          "selectaddress.jsp", "button(useShippingAddress)","&" +
          HttpRequestConstants.ADDRESS_KEY + "=" + addressKey, true)%>">
          " border="0">
          </a>
        </div>
      </td>
  </tr>
</table>

<tr>
  <td colspan="3">

```

4 *Shipping Services*

```
        <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 import calls in `addaddress.jsp` are:

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

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

4 Shipping Services

Figure 4-3 Annotated addaddress.jsp Template

BEA WebLogic Commerce Server 3.5
Commerce Templates

Store your stuff in our
Storage Boxes !

[Home](#)

Welcome Demo Customer
[View Profile](#)
[Logout](#)

View History
[Orders](#)
[Payments](#)

Don't Forget
Extension Cords!
click here

Catalog data provided courtesy of [TPH Register](#), where supply meets demand.

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)

Copyright © 1999-2001,
BEA Systems Inc.

Location in the WebLogic Commerce Server Directory Structure

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

```
%WL_COMMERCE_HOME%\config\wlcsDomain\applications\wlcsApp\wlcs\commerce\order\addaddress.jsp (Windows)  
$WL_COMMERCE_HOME/config/wlcsDomain/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="pipeline.tld" prefix="pipeline"%>
<%@ 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 following directory for the WebLogic Commerce Server Web application:

```
%WL_COMMERCE_HOME%\config\wlcsDomain\applications\wlcsApp\wlcs\WEB-INF (Windows)
$WL_COMMERCE_HOME/config/wlcsDomain/applications/wlcsApp/wlcs/WEB-INF (UNIX)
```

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

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.

Included JSP Templates

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

- `innerheader.jsp`, which creates the top banner.
- `states.jsp`, which contains a list of states that are displayed when the customer is prompted to enter an address.
- `countries.jsp`, which contains a list of countries that are displayed when the customer is prompted to enter an address.
- `innerfooter.jsp`, which creates a horizontal footer at the bottom of the page, and also includes the `innerrightside.jsp` template. `innerrightside.jsp` describes (for the benefit of you and your development team) the name of the current template and links to its *About* information.

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)
<code>button(back)</code>	No business logic required. Loads <code>selectaddress.jsp</code> .
<code>button(addNewShippingAddress)</code>	<code>ValidateAddressIP</code> <code>AddShippingAddress</code>

Table 4-9 briefly describes each of the Pipelines from Table 4-8, as they are defined in the `pipeline.properties` file. For more information about individual Pipeline components, see “Pipeline Components” on page 4-27.

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 (<code>addaddress.jsp</code>), used by the Webflow.
<code>HttpRequestConstants.CUSTOMER_SHIPPING_ADDRESS1</code>	Textbox	The first line of the shipping street address.

4 Shipping Services

Table 4-10 addaddress.jsp Form Fields

Parameter Name	Type	Description
<code>HttpRequestConstants.CUSTOMER_SHIPPING_ADDRESS2</code>	Textbox	The second line of the shipping street address.
<code>HttpRequestConstants.CUSTOMER_SHIPPING_CITY</code>	Textbox	The city in the shipping address.
<code>HttpRequestConstants.CUSTOMER_SHIPPING_STATE</code>	Textbox	The state in the shipping address.
<code>HttpRequestConstants.CUSTOMER_SHIPPING_ZIPCODE</code>	Textbox	The zip/postal code in the shipping address.
<code>HttpRequestConstants.CUSTOMER_SHIPPING_COUNTRY</code>	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	<code>com.beasys.commerce.ebusiness.shipping.webflow. InitShippingMethodListIP</code>
Description	Obtains a list of all shipping methods from the database and populates the Pipeline session with a list of <code>ShippingMethodValue</code> 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	<code>PipelineSessionConstants.SHOPPING_CART</code>
Updated Pipeline Session Attributes	<code>PipelineSessionConstants.SHIPPING_METHOD_LIST</code>
Removed Pipeline Session Attributes	None
Validation	None
Exceptions	<code>ProcessingException</code> , thrown if no shopping cart exists.

UpdateShippingAddressIP

Class Name	com.beasys.commerce.ebusiness.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	<code>com.beasys.commerce.ebusiness.shipping.webflow.ValidateAddressIP</code>
Description	Validates the address and places it in the Pipeline session.
Required HttpServletRequest Parameters	<code>HttpRequestConstants.CUSTOMER_SHIPPING_ADDRESS1</code> <code>HttpRequestConstants.CUSTOMER_SHIPPING_ADDRESS2</code> <code>HttpRequestConstants.CUSTOMER_SHIPPING_CITY</code> <code>HttpRequestConstants.CUSTOMER_SHIPPING_STATE</code> <code>HttpRequestConstants.CUSTOMER_SHIPPING_ZIPCODE</code> <code>HttpRequestConstants.CUSTOMER_SHIPPING_COUNTRY</code>
Required Pipeline Session Attributes	None
Updated Pipeline Session Attributes	<code>PipelineSessionConstants.ADDRESS</code>
Removed Pipeline Session Attributes	None
Validation	Verifies that the required fields contain values.
Exceptions	<code>ProcessingException</code> , thrown if the required request parameters or required Pipeline session attributes are not available.

ValidateShippingInfoIP

Class Name	<code>com.beasys.commerce.ebusiness.shipping.webflow.ValidateShippingInfoIP</code>
Description	Places the shipping method, splitting preference, and special instructions into the Pipeline session.
Required HttpServletRequest Parameters	<code>HttpRequestConstants.SHIPPING_METHOD</code> <code>HttpRequestConstants.SPLITTING_PREFERENCE</code> <code>HttpRequestConstants.SPECIAL_INSTRUCTIONS</code> <code>HttpRequestConstants.SPLITTING_PREFERENCE_CODE</code>
Required Pipeline Session Attributes	None
Updated Pipeline Session Attributes	<code>PipelineSessionConstants.SHIPPING_METHOD</code> <code>PipelineSessionConstants.SPLITTING_PREFERENCE</code> <code>PipelineSessionConstants.SPECIAL_INSTRUCTIONS</code> <code>PipelineSessionConstants.SPLITTING_PREFERENCE_CODE</code>
Removed Pipeline Session Attributes	None
Validation	Verifies that the required fields contain values.
Exceptions	<code>ProcessingException</code> , 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 “Taxation Services” on page 5-1.

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

AddShippingAddressPC

Class Name	<code>com.beasys.commerce.ebusiness.shipping.pipeline.AddShippingAddressPC</code>
Description	Adds the address to the list of customer shipping addresses stored for the customer.
Required Pipeline Session Attributes	<code>PipelineSessionConstants.ADDRESS</code> <code>PipelineSessionConstants.ADDRESS_KEY</code>
Updated Pipeline Session Attributes	None
Removed Pipeline Session Attributes	None
Type	Java object
JNDI Name	None
Exceptions	<code>PipelineFatalException</code> , thrown when the Pipeline component cannot update the address information in the database.

CalculateShippingPC

Class Name	<code>com.beasys.commerce.ebusiness.shipping.pipeline. CalculateShippingPC</code>
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	<code>PipelineSessionConstants.SHOPPING_CART</code> <code>PipelineSessionConstants.SHIPPING_METHOD</code>
Updated Pipeline Session Attributes	<code>PipelineSessionConstants.SHOPPING_CART</code>
Removed Pipeline Session Attributes	None
Type	Java object
JNDI Name	None
Exceptions	<code>PipelineFatalException</code> , thrown if the required request parameters or required Pipeline session attributes are not available.

DeleteShippingAddressPC

Class Name	<code>com.beasys.commerce.ebusiness.shipping.pipeline.DeleteShippingAddressPC</code>
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	<code>PipelineSessionConstants.ADDRESS_KEY</code>
Updated Pipeline Session Attributes	None
Removed Pipeline Session Attributes	None
Type	Java object
JNDI Name	None
Exceptions	<code>PipelineFatalException</code> , thrown when the Pipeline component cannot update the shipping address information in the database.

4 *Shipping Services*

5 Taxation Services

The taxation functionality provided in the Managing Purchases and Processing Orders services are used to calculate the taxes associated with your customer's order. They enable you to determine the accurate tax rates imposed on the sale or use of each item at the state, country, city, and district levels by interfacing with TAXWARE International, Inc. products. This topic describes the Taxation Service in detail.

This topic includes the following sections:

- JavaServer Pages (JSPs)
 - selecttaxaddress.jsp Template
- Input Processors
 - DecideShippingAddressPageIP
 - UpdateShippingAddressIP
- Pipeline Components
 - TaxCalculateLineLevelPC
 - TaxCalculateAndCommitLineLevelPC
 - TaxVerifyShippingAddressPC
- Integration with TAXWARE
 - Important TAXWARE Considerations
 - TAXWARE Installation
 - TAXWARE Configuration and Deployment
 - Removing Tax Calculations
 - What if I Don't Want to Use TAXWARE to Calculate My Taxes?

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 Commerce Server Web application and a listing of their locations in the directory structure, see the *E-Commerce JSP Template Summary*.

selecttaxaddress.jsp Template

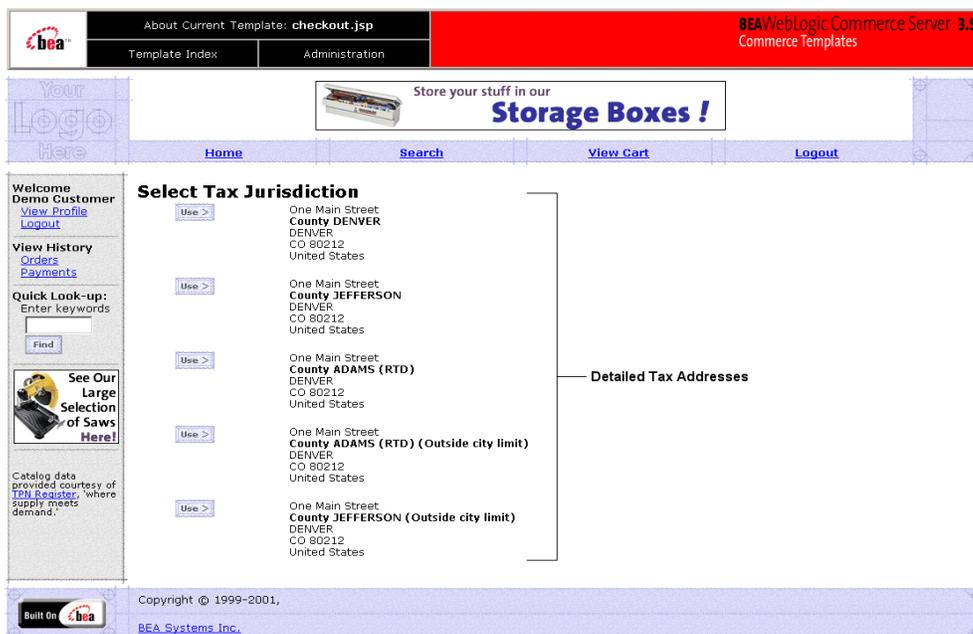
In cases where a customer provides a shipping address that does not resolve to a unique GeoCode (a TAXWARE code used to determine taxes based on jurisdiction), the `selecttaxaddress.jsp` template (shown in Figure 5-1) allows the customer to select from a list of more specific shipping addresses.

Sample Browser View

Figure 5-1 shows an annotated version of 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 5-1 Annotated selecttaxaddress.jsp Template



Location in the WebLogic Commerce Server Directory Structure

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

```
%WL_COMMERCE_HOME%\server\webapps\wlcs\commerce\order\
selecttaxaddress.jsp (Windows)
$WL_COMMERCE_HOME/server/webapps/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="weblogic.tld" prefix="wl" %>
<%@ taglib uri="pipeline.tld" prefix="pipeline" %>
```

5 Taxation Services

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 following directory for the WebLogic Commerce Server Web application:

```
%WL_COMMERCE_HOME%\server\webapps\wlcs\WEB-INF (Windows)
$WL_COMMERCE_HOME/server/webapps/wlcs/WEB-INF (UNIX)
```

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.foundation.pipeline.*" %>
<%@ page import="com.beasys.commerce.axiom.contact.*" %>
<%@ page import="com.beasys.commerce.ebusiness.shipping.*" %>
<%@ page import="com.beasys.commerce.webflow.*" %>
```

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`).

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:

- `innerheader.jsp`, which creates the top banner.
- `innerfooter.jsp`, which creates a horizontal footer at the bottom of the page, and also includes the `rightside.jsp` template. `rightside.jsp` describes (for the benefit of you and your development team) the name of the current template and links to its *About* information.

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)
<code>button(use)</code>	<code>UpdateTaxShippingAddressIP</code>

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 `getPipelineProperty` JSP tag retrieves the `VERIZIP_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
<code>PipelineSessionConstants.VERIZIP_SHIPPING_ADDRESSES</code>	List of <code>com.beasys.commerce.axiom.contact.Address</code>	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

```
<pipeline:getPipelineProperty
  propertyName="<%=PipelineSessionConstants.VERAZIP_SHIPPING_ADDRESSES%>"
  returnName="addressesObject" returnType="java.lang.Object"/>
```

Note: For more information on the `getPipelineProperty` 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
<code>getStreet1()</code>	The first line of the street in the shipping address.
<code>getStreet2()</code>	The second line of the street in the shipping address.
<code>getCity()</code>	The city in the shipping address.
<code>getCounty()</code>	The county in the shipping address.
<code>getState()</code>	The state in the shipping address.
<code>getPostalCode()</code>	The zip/postal code in the shipping address.
<code>getCountry()</code>	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> Tags and Accessor Methods in selecttaxaddress.jsp

```

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

<table>
  <tr>
    <td><b>County</b></td>
    <td><%=address.getCounty()%><br>
      <%=address.getCity()%><br>
      <%=address.getState()%><br>
      <%=address.getPostalCode()%><br>
      <%=address.getCountry()%>
    </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 selectataxaddress.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 (<code>selecttaxaddress.jsp</code>), used by the Webflow.

5 Taxation Services

Table 5-4 selectataxaddress.jsp Form Fields (Continued)

Parameter Name	Type	Description
<code>PipelineSessionConstants.TAX_SHIPPING_ADDRESS</code>	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	com.beasys.commerce.ebusiness.tax.webflow. DecideShippingAddressPageIP
Description	Makes the decision about whether to display <code>selecttaxaddress.jsp</code> based on the number of address variations returned from the TAXWARE VERAZIP service. If a single address is found, this input processor updates the shipping address, returns successfully, and allows the Webflow to proceed to <code>payment.jsp</code> . Otherwise, this input processor redirects the Webflow to <code>selecttaxaddress.jsp</code> .
Required HttpServletRequest Parameters	None
Required Pipeline Session Attributes	PipelineSessionConstants.SHIPPING_ADDRESS PipelineSessionConstants.VERIZIP_SHIPPING_ADDRESSES
Updated Pipeline Session Attributes	PipelineSessionConstants.SHIPPING_ADDRESS (in the case of a single address)
Removed Pipeline Session Attributes	None
Validation	None
Exceptions	MultipleAddressFoundException, thrown if the VERAZIP service returns more than one address.

UpdateShippingAddressIP

Class Name	com.beasys.commerce.ebusiness.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.VERIZIP_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	com.beasys.commerce.ebusiness.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	com.beasys.commerce.ebusiness.tax.pipeline. TaxCalculateAndCommitLineLevelPC
Description	Calculates the tax and provides line-level information about the taxability of an item. The results are logged to the TAXWARE 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	com.beasys.commerce.ebusiness.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.VERAZIP_SHIPPING_ADDRESSES

Pipeline Components

Removed Pipeline Session Attributes	None
Type	Java class
JNDI Name	None
Exceptions	<code>TaxSystemException</code> , thrown if processing could not occur due to system level problems (for example, some data files are missing or there is an installation problem in TAXWARE). <code>TaxUserException</code> , thrown if processing could not occur due to invalid user input.

Integration with TAXWARE

To ensure that the Taxation Services properly determine taxes for the items in your product catalog, the WebLogic Commerce Server product integrates with the following TAXWARE International Inc.'s commercial tax products.

- The SALES/USE Tax System is a TAXWARE product, which calculates the sales, use, and customer's use tax based on jurisdictions in the United States and Canada. Monthly updates of tax rates ensure the SALES/USE Tax System is kept up-to-date.
- The VERAZIP System is a TAXWARE product, which verifies addresses for tax purposes. Such verification ensures that the address is detailed enough for the SALES/USE Tax System to determine the correct tax.
- The Universal Tax Link (UTL) System is a TAXWARE product that can be used as a common application program interface for different modules of the tax system (that is, SALES/USE, VERAZIP, and so on).
- The WORLDTAX System is a TAXWARE product, which calculates and reports Value Added Tax (VAT), Goods and Services Tax (GST), sales tax, and consumption tax in many countries. BEA has tested France, Germany, Italy, South Korea, Spain, and the United Kingdom for accuracy with the WebLogic Commerce Server. For information about testing other countries supported by WORLDTAX, see "Adding Countries to the WebLogic Commerce Server."

Note: For more information about TAXWARE International, Inc. and TAXWARE products, visit the company's Web site at <http://www.taxware.com>.

Important TAXWARE Considerations

The following are important factors regarding the WebLogic Commerce Server product's integration with TAXWARE that should be considered prior to launching your e-business Web site:

- *What WebLogic Commerce Server Provides:* The WebLogic Commerce Server product ships with evaluation tax data from January 2001 to demonstrate the Taxation Service functionality. It does not include the TAXWARE utilities

required to upload new tax data, nor does it include the tools that allow you to run audit reports. Therefore, you will need to obtain and install these components by contacting TAXWARE International, Inc. prior to using the Taxation Service in a production environment.

- *About Tax Data Updates:* Due to changes in tax laws, TAXWARE data becomes obsolete with time. To calculate correct taxes for your customers' orders, you will need to obtain current tax data from TAXWARE International, Inc. This update process is required approximately 15 times per year, and TAXWARE makes new tax data available approximately one month in advance. For more information about tax data updates, visit TAXWARE International, Inc.'s Support and Updates Web site at <http://www.taxware.com/zsupport/support.htm>.
- *Domestic and International Taxes:* The TAXWARE products included in the WebLogic Commerce Server product handle tax calculations for the United States, Canada, France, Germany, Italy, South Korea, Spain, and the United Kingdom.
- *Tax Calculation Policies:* Tax computation is a complex subject. Your development team should not make decisions about the company's tax policies; rather, you should consult with an attorney in your Legal Department for policies regarding the use of tax software in your Web-based applications.

TAXWARE Installation

TAXWARE International's SALES/USE, VERAZIP, Universal Tax Link (UTL), and WORLDTAX systems are shipped with the WebLogic Commerce Server product to provide out-of-the-box TAXWARE functionality. The WebLogic Commerce Server's installation program will install these TAXWARE products along with the WebLogic Commerce Server, and will also uninstall them during uninstallation of the WebLogic Commerce Server.

The versions of the TAXWARE products installed with the WebLogic Commerce Server product are as follows:

- SALES/USE Tax System, release 3.2.0
- VERAZIP System, release 3.2.0
- Universal Tax Link, release 2.1

- WORLDTAX System, release 2.4

Note: Utilization of WebLogic Commerce Server in the connection to and operation of third-party software, services and applications including, but not limited to, Cybercash credit card services and TAXWARE tax calculation services, is entirely at the user's risk. BEA Systems, Inc. disclaims all liability and responsibility for the operation, accuracy and results of such software, services and applications.

Installation Directory Structure

The TAXWARE product files installed with the WebLogic Commerce Server product are organized into particular directories based on the system platform. This section describes the directory structures for both the Windows and UNIX installations of the TAXWARE products.

Windows

All TAXWARE audit files, DLLs, and preloaded data files needed for Win32 installation reside in subdirectories beneath `WL_COMMERCE_HOME\eval\win32\Taxware`, except for Java classes, which reside in `WL_COMMERCE_HOME\eval\common\Taxware`. `WL_COMMERCE_HOME` is the directory in which you installed WebLogic Commerce Server.

Table 5-5 lists the subdirectories where you would find these TAXWARE files.

Table 5-5 Location of TAXWARE Files (Windows)

Subdirectory	Description
<code>\audit</code>	Contains audit files for all tax transactions.
<code>\bin</code>	Contains DLLs for SALES/USE, VERIZIP, and UTL, including <code>avptax.dll</code> , <code>avpzip.dll</code> , <code>taxcommon.dll</code> , and <code>taxcommon0.dll</code> .
<code>\common\Taxware\classes</code>	Contains Java classes for UTL, including <code>taxmain.class</code> and <code>taxcommon.class</code> .

Table 5-5 Location of TAXWARE Files (Windows)

Subdirectory	Description
\data	Contains preloaded data files for SALES/USE, VERAZIP, and WORLDTAX such as INDATA (which includes all run-time, test and parameter, tax master, product sequential, and update files) and OUTDATA (which includes all generated data files when tax data is loaded or updated).
\temp	Contains temporary files generated by TAXWARE while processing a transaction.

Additionally, the `WL_COMMERCE_HOME\eval\win32\Taxware` directory (where `WL_COMMERCE_HOME` is where you installed the WebLogic Commerce Server) contains the following three `ini` files:

- `avptax.ini`, which describes the input, output, audit and temporary directory path environment variables used by the TAXWARE SALES/USE System.
- `avpzip.ini`, which describes the input, output, audit, and temporary directory path environment variables used by the TAXWARE VERAZIP System.
- `taxware.ini`, which describes the input, output, audit, and temporary directory path environment variables used by the TAXWARE WORLDTAX System.

Notes: The WebLogic Commerce Server product’s installation program automatically moves these files from the `WL_COMMERCE_HOME` directory to the `C:\Winnt` directory.

For more information about the `ini` files, see “Run-Time Configuration” on page 5-33.

UNIX

All TAXWARE audit files, shared objects, and preloaded data files needed for UNIX installation reside in subdirectories beneath

`WL_COMMERCE_HOME/eval/solaris2/Taxware`, except for Java classes, which reside in `WL_COMMERCE_HOME/eval/common/Taxware`. `WL_COMMERCE_HOME` is the directory in which you installed WebLogic Commerce Server.

Table 5-6 lists the subdirectories where you would find these TAXWARE files.

Table 5-6 Location of TAXWARE Files (UNIX)

Subdirectory	Description
/audit	Contains audit files for all tax transactions.
/common/Taxware /classes	Contains UTL Java classes, including <code>taxmain.class</code> and <code>taxcommon.class</code> .
/data	Contains preloaded data files for SALES/USE, VERAZIP, and WORLDTAX such as INDATA (which includes all run-time, test and parameter, tax master, product sequential, and update files) and OUTDATA (which includes all generated data files when tax data is loaded or updated).
/lib	Contains shared objects, including <code>libsalesusetax.so</code> , <code>libstep.so</code> , <code>libtaxcommon.so</code> , <code>libtaxcommono.so</code> , and <code>libverazip.so</code> .
/temp	Contains temporary files generated by TAXWARE while processing a transaction.

Testing the TAXWARE Installation

You can test the installation of the WebLogic Commerce Server-provided TAXWARE products on both Windows and UNIX platforms using some predefined test scripts. Refer to the appropriate section for details.

Windows

To run the test scripts in a Windows environment, follow these steps:

1. From a DOS prompt, set up the home directory for WebLogic Commerce Server by typing: `SET WL_COMMERCE_HOME=<directory_where_you_installed_WebLogic_Commerce_Server>`.
2. Navigate to the `WL_COMMERCE_HOME\eval\win32\Taxware` directory, where `WL_COMMERCE_HOME` is the directory in which you installed WebLogic Commerce Server.
3. To test the SALES/USE component of TAXWARE, type `runsample.bat commonsu.in`.

4. To test the VERAZIP component of TAXWARE, type `runsample.bat vzip.in`. The result should be a long line that begins with: `00000542No I/O Error`.
5. Check that output string has the expected completion code.

Note: Refer to the TAXWARE SALES/USE and VERAZIP product documentation for more details about the output string fields and their values.

UNIX

To test installation of TAXWARE in a UNIX environment, follow these steps:

1. From a command window, set up the home directory for WebLogic Commerce Server by typing: `SET WL_COMMERCE_HOME=<directory_where_you_installed_WebLogic_Commerce_Server>`.
2. Navigate to the `WL_COMMERCE_HOME/eval/solaris2/Taxware` directory, where `WL_COMMERCE_HOME` is the directory in which you installed WebLogic Commerce Server.
3. To test the SALES/USE component of TAXWARE, type `runsample.sh commonsu.in`.
4. To test the VERAZIP component of TAXWARE, type `runsample.sh vzip.in`. The result should be a long line that begins with: `00000542No I/O Error`.
5. Check that output string has the expected completion code.

Note: Refer to the TAXWARE SALES/USE and VERAZIP product documentation for more details about the output string fields and their values.

Changing the TAXWARE Directory Structure

TAXWARE products are integrated with the WebLogic Commerce Server product through the Java Native Interface (JNI). This means that a specially prepared shared object or DLL must be made available for loading during server startup. The WebLogic Commerce Server ships with a working version of TAXWARE, complete with the correct DLLs and sample data files. If your organization has purchased TAXWARE products and installed these files in a different location, you must point

the WebLogic Commerce Server product's Taxation Services to the correct directories. For more information about changing the TAXWARE directory structure, see "Run-Time Configuration" on page 5-33.

TAXWARE Configuration and Deployment

The correct calculation of taxes requires that a number of important pieces of information come together. The bulk of the information needed to calculate taxes is stored in the data structures provided by TAXWARE, and can be loaded using TAXWARE utilities. Additional tax information (from the product catalog, ship to address, and so on) is made available to the WebLogic Commerce Server product via our programmatic interface (API). The information that cannot be obtained from the data structures or specified using the API must be configured using property files.

This section describes all of the configuration and deployment issues that you need to consider when using TAXWARE products. The information described here focuses on the configuration properties in the `weblogiccommerce.properties` file, which enables tax calculations.

Addresses and Taxation

In many cases, the proper calculation of taxes requires that you specify a number of addresses, including the location where the order is accepted, where the order originated, where the order shipped from, and where the title is exchanged.

Note: For a detailed explanation of the tax implications associated with these addresses, you will need to consult with TAXWARE International, Inc. and the attorneys in your organization's Legal Department.

For Canadian postal codes, the official format is one space between the first three and last three characters. For TAXWARE, omit the space.

The Pipeline components that ship with the WebLogic Commerce Server product support specifying a single location of these addresses for each instance of the WebLogic Commerce Server. This information is specified and read from the `tax` section of `weblogiccommerce.properties` file, located in `WL_COMMERCE_HOME`. `WL_COMMERCE_HOME` is the directory in which you installed the WebLogic Commerce Server.

For each of the relevant address fields (street, city, state, and so on), there is a separate line in the properties file (see Listing 5-3). The information that you are required to specify is described in the following list:

- `taxware.shipFrom.country`: The ISO 3166 two or three character abbreviation for the country from which orders are shipped. This property is required.
- `taxware.shipFrom.state`: The state or province from where goods are shipped. This value must be mappable from the `province.properties` file using either the `name` or the `abbr` of the province. This property is not required for addresses outside the United States and Canada.
- `taxware.shipFrom.city`: The city from where goods are shipped. This is a required property.
- `taxware.shipFrom.zip`: The ZIP code or postal code from where goods are shipped. This is a required property. In the United States and Canada, the value is checked using the VERAZIP module within the TAXWARE software. Other postal codes are not currently verified.
- `taxware.shipFrom.geoCode`: The TAXWARE code that specifies an address that covers multiple taxing jurisdictions. It is only used for U.S. and Canadian addresses. This property is optional. However, if the WebLogic Commerce Server cannot resolve the `shipFrom` address to a single GeoCode using VERAZIP, it throws a fatal exception. The multiple address information that is returned from VERAZIP is logged to the error log.
- `taxware.orderOrigin`: These properties are exactly the same as the `shipFrom` properties. They are used when the order is taken at a different location than from which the order is shipped. To determine if an order origin address exists, the WebLogic Commerce Server checks for the existence of the `taxware.orderOrigin.country` property. If it doesn't exist, the order origin properties are not used and the WebLogic Commerce Server defaults to the `shipFrom` properties.
- `taxware.titlePassage`: A property that lets TAXWARE determine the point of title passage. This property is optional and is only used in the United States and Canada. If it is set to `shipFrom` (case insensitive), it indicates that legal transfer occurred at the point of origin. If set to `shipTo`, it indicates that legal transfer occurred at the point of delivery.

Listing 5-3 Specifying Addresses in the `weblogiccommerce.properties` File

```
#####  
# ShipFrom Address  
# -----  
# ShipFrom Address is address from where goods are shipped  
# Please review Taxware documentation when setting these properties  
#  
taxware.shipFrom.country=US  
taxware.shipFrom.state=TX  
taxware.shipFrom.city=Round Rock  
taxware.shipFrom.zip=78682  
taxware.shipFrom.county=WILLIAMSON  
taxware.shipFrom.geoCode=00  
#####  
# Order Origin Address  
#-----  
# Order Origin is the address where orders are Originated  
# Please review Taxware documentation when setting these properties  
#  
taxware.orderOrigin.countycode=000  
taxware.orderOrigin.state=MA  
taxware.orderOrigin.city=SALEM  
taxware.orderOrigin.zip=01970  
taxware.orderOrigin.geocode=00  
taxware.orderOrigin.country=USA
```

The point of title passage may be defaulted to be either the ship from or the ship to address. The most common case is to use the `shipfrom` address. Changing this involves replacing the title passage line by uncommenting one line and replacing it with the other, as shown in Listing 5-4.

Listing 5-4 Specifying Point of Title Passage in the `weblogiccommerce.properties` File

```
#####  
# Point of title passage  
# -----  
  
# Location at which legal title has transferred to purchaser  
  
#taxware.titlePassage=shipto  
taxware.titlePassage=shipfrom
```

Note: It is possible to modify the tax calculation Pipeline component to obtain the Address and Taxation properties from a source other than the `weblogiccommerce.properties` file. Alternative sources may be input from the customer or from a pre-existing inventory or product delivery system. Obtaining the addresses from alternative sources may require prompting the customer for an address, or obtaining the address from your other systems on a per-order basis. Regardless of the method used to obtain the addresses, the addresses must be placed in the Pipeline session, and set in the `TaxParameters` object prior to calculating tax.

TAXWARE-Specific Properties

Because TAXWARE is an external product, there are some properties specific to TAXWARE that must also be configured in the `weblogiccommerce.properties` file. This section describes each of these properties in detail.

The TAXWARE specific parameters are all prepended by the string `taxware`. The general format of the `weblogiccommerce.properties` file is `key=value`, where `key` tells the WebLogic Commerce Server which parameter is being configured and `value` is what is set by the end user. Any amount of white space may separate the key from the equals sign and from the value. This file is read using `java.util.Properties.load(InputStream)` and subsequently key is case sensitive.

5 Taxation Services

Specifying a Currency

It is important that the ISO 4217 currency code be provided to TAXWARE products. In the shipped WebLogic Commerce Server product, the currency field in the shopping cart lines have been defaulted or are empty. It is therefore necessary for you to specify at least one currency for use in calculating tax in the `weblogiccommerce.properties` file, as shown in Listing 5-5. This currency will be used for all tax calculation amounts, and enables future localization of tax calculations.

Listing 5-5 Specifying Currency in the `weblogiccommerce.properties` File

```
#####  
# Currency for Tax Calculation  
# -----  
taxware.currency = USD
```

Specifying Your Company's ID and Business Location

When you configure TAXWARE in order to calculate taxes, you need to provide identification information for your company. Because it is possible for multiple corporate entities to share a set of TAXWARE configuration files, your `companyId` must be specified with each request to TAXWARE. This property is the identifier for your company as configured in your TAXWARE deployment. The demonstration configuration uses `companyId` as the default for this property, so it must be changed for a production environment. Listing 5-6 shows a sample configuration.

You may also need to specify your business location for WORLD TAX reporting. If the seller registration number is not set, WORLD TAX uses the business location and company ID to look up the seller registration number. This mapping is set up using TAXWARE tools. If neither the business location nor the seller registration number are set, WORLD TAX cannot properly calculate the tax. If the seller registration is not set, you must modify this property in the `weblogiccommerce.properties` file, as shown in Listing 5-6.

Listing 5-6 Specifying Company ID and Business Location in the weblogiccommerce.properties File

```
#####
#-----
# User Defined company identification to access information
# for tax calculating and reporting

taxware.companyId=BEA Systems
taxware.businessLocation=London Office
```

Specifying Your Seller Registration Number

If you have accounts outside of the U.S. or Canada, you may need to set the registration number of the seller. This parameter works with the `businessLocation` parameter. Please see “Specifying Your Company’s ID and Business Location” on page 5-24. If you need to specify the Seller Registration Number, you must modify this property in the `weblogiccommerce.properties` file. For an example, see Listing 5-7. Table 5-7 lists the Registration Number Formats for the countries supported by the WebLogic Commerce Server.

Table 5-7 Registration Number Formats

Country	Country Code	Example	Format
France	FR	12345678901 X1123456789 1X123456789 XX123456789	Eleven characters in a block. The first or second, or the first and second can be any letter except I or O.
Germany	DE	123456789	Nine numbers in a block. Always starts with 1, 2, or 8.
Great Britain	GB	123456789 123 4567 89	Nine numbers in a block; or three numbers, four numbers, and two numbers, separated by spaces.
Italy	IT	12345678901	Eleven numbers in a block.
Japan	JP		No specific format is required.

Table 5-7 Registration Number Formats (Continued)

Country	Country Code	Example	Format
South Korea	KR		No specific format is required.
Spain	ES	12345678X X1234567X	Nine characters in a block. Includes one or two letters, either last or first and last.

Listing 5-7 Specifying Seller Registration Number in the weblogiccommerce.properties File

```
#####
# Seller registration number is used outside the US and Canada
# It's format is country specific. Please see the WorldTax
# docs for more information
#
taxware.sellerRegistrationNumber=123 4567 89
```

Specifying Your Tax Type

Depending on the nature of your business, you may need to select the type of taxes you want to calculate. The WebLogic Commerce Server product defaults to calculating sales tax for hard and soft goods. The SALES/USE module also supports calculation of taxes for usage, commercial usage, rental, and services. If your organization requires any of these other models, you will need to modify this property in the `weblogiccommerce.properties` file, as shown in Listing 5-8.

Listing 5-8 Specifying TaxType in the weblogiccommerce.properties File

```
#####
# TaxType
#-----
# Type of tax to be calculated

#taxware.taxType=use
#taxware.taxType=rental
#taxware.taxType=consumeruse
#taxware.taxType=services
taxware.taxType=sales
```

Note: The tax calculation Pipeline components that ship with WebLogic Commerce Server only allow you to choose one tax type. If your organization requires multiple tax types, you will need to modify the appropriate Pipeline component(s) (TaxCalculateLineLevelPC, TaxCalculateAndCommitLineLevelPC, and TaxVerifyShippingAddressPC) to specify this to the Taxation Service via the *taxType* parameters.

Specifying Calculation of Jurisdiction

Setting the *TaxSelParm* property (shown in Listing 5-9) will indicate to the TAXWARE product whether or not you must fully calculate jurisdiction. If you set this option to 2, TAXWARE will not determine the jurisdiction. If you do not need to determine jurisdiction, you may also remove the *shipFrom* and *orderOrigin* address properties from the *weblogiccommerce.properties* file, as they will not be required (see Listing 5-3).

Listing 5-9 Specifying Jurisdiction Calculations in the weblogiccommerce.properties File

```
#####
# TaxSelParm
#-----
# TaxSelParm to decide jurisdiction while calculating
# if value is 2 Calculate tax only
# if value is 3 Determine jurisdiction and calculate taxes
```

5 Taxation Services

```
#taxSelParm=2  
taxSelParm=3
```

Note: Setting the `taxSelParm` property is a business decision that will require input from your Legal Department and TAXWARE International, Inc.

Adding Countries to the WebLogic Commerce Server

As previously mentioned, the WebLogic Commerce Server supports nine countries, including the U.S. and Canada. BEA has tested only these countries for accuracy. However, WORLDTAX supports many more countries. If you have the capability to test the countries supported by WORLDTAX but not the WebLogic Commerce Server, this section provides the necessary information for adding countries.

To add countries, two files need to be updated to the out-of-the-box installation of WebLogic Commerce Server. The first file is
`%WL_COMMERCE_HOME%\server\webapps\wlcs\commerce\includes\countries.jsp` (Windows) or
`$WL_COMMERCE_HOME/server/webapps/wlcs/commerce/includes/countries.jsp` (UNIX).

These files have entries like those shown in Listing 5-10.

Listing 5-10 Sample countries.jsp File Entries

```
<option value=canada>Canada  
<option value="United Kingdom">United Kingdom  
<option value="France">France  
<option value="Spain">Spain  
<option value="Italy">Italy
```

The second file is
`$WL_COMMERCE_HOME/classes/com/beasys/commerce/util/country.properties` (UNIX) or
`%WL_COMMERCE_HOME%\classes\com\beasys\commerce\util\country.properties` (Windows). These files contains entries like those shown in Listing 5-11.

Listing 5-11 Sample country.properties File Entries

```
country1.iso3166Code=840
country1.twoCharAbbr=US
country1.threeCharAbbr=USA
country1.englishLabel=United States
country1.iso4217CurrencyCode=USD
country1.currencyEnglishLabel=United States Dollar
```

The `country.properties` file contains information about a particular country and is referenced once at startup.

The `country.properties` file contain a list of countries. Each country *stanza* is prepended with the word `country` and a number from 0 to 10000. You can skip numbers and there is not a requirement that the numbers be in order. All fields displayed in Listing 5-11 are required. The following list describes these fields in detail:

- `iso3166Code`: The three digit numeric code for the country as assigned by the ISO 3166 standard. A list of these codes can be found on the Internet or in the TAXWARE documentation.
- `twoCharAbbr`: The two character ISO 3166 abbreviation.
- `threeCharAbbr`: The three character ISO 3166 abbreviation.
- `englishLabel`: The label that is assigned to this country, in English.
- `iso4217CurrencyCode`: The three character currency code as assigned by the ISO 4217 standard. A list of these codes can be found on the Internet or in the TAXWARE documentation.
- `currencyEnglishLabel`: The default English label for the currency.

To add an additional country you must add it in both `countries.jsp` and `country.properties`.

Note: The *value* in the option HTML tag in `countries.jsp` must match one of `iso3166Code`, `twoCharAbbr`, `threeCharAbbr`, or `englishLabel` in the properties file for the country to be added. An unknown country in the JSP file throws an exception.

Adding and Modifying Provinces and States

The WebLogic Commerce Server uses two files to generate province information. For this discussion, a state is considered a province. The first file is

```
$WL_COMMERCE_HOME/server/webapps/wlcs/commerce/includes/states.jsp (UNIX) or  
%WL_COMMERCE_HOME%\server\webapps\wlcs\commerce\includes\states.jsp (Windows). These files have entries like those shown in Listing 5-12.
```

Listing 5-12 Sample states.jsp File Entries

```
<option>WV  
<option>WY  
<option>Alberta  
<option>British Columbia
```

The second file is country dependent. The WebLogic Commerce Server ships with two files, one for the U.S. and one for Canada. The U.S. file is

```
%WL_COMMERCE_HOME%\classes\com\beasys\commerce\util\us.province.properties (Windows) or  
$WL_COMMERCE_HOME/classes/com/beasys/commerce/util/us.province.properties (UNIX).
```

In general, these files have the name format of the lowercase, two-character country code followed by the `.province.properties` suffix. The `province.properties` files contains entries that look like those in Listing 5-13.

Listing 5-13 Sample province.properties File Entries

```
province7.name=Colorado  
province7.abbr=CO  
  
province8.name=Connecticut  
province8.abbr=CT  
  
province9.name=Delaware  
province9.abbr=DE
```

```
province10.name=District of Columbia  
province10.abbr=DC
```

```
province11.name=Federated States of Micronesia  
province11.abbr=FM
```

The `province.properties` file contains a list of the specified country's provinces. Each province *stanza* is prepended with the word `province` and a number from 0 to 10000. You can skip numbers and there is not a requirement that the numbers be in order. All fields displayed in Listing 5-12 are required. The following list describes these fields in detail:

- `name`: The label that is applied to the province.
- `abbr`: An abbreviation for the province.

It is important to note that there is no requirement that you use all of the provinces of a country. For example, in the United States a supplier may ship only to addresses within the continental United States. In this case, the supplier needs only to modify the `states.jsp` and `us.province.properties` to include only the 48 contiguous states.

Note: The value in the option `HTML` tag in the `states.jsp` must match either the `name` or `abbr` in the properties file for the province to be added. An unknown province in the JSP file will throw an exception.

Configuration Parameters

The `weblogiccommerce.properties` holds many configuration parameters that configure the operation of the WebLogic Commerce Server's interaction with TAXWARE. It's important to understand how these fields are used in order to configure them correctly.

The general format of this file is `key = value`, where *key* tells WebLogic Commerce Server which parameter is being configured and *value* is that set by the end user. Any number of spaces may separate the key from the equals sign and from the value. This file is read using `java.util.Properties.load(InputStream)` and therefore the key is case sensitive.

The TAXWARE specific parameters are all prepended by the string `taxware`. The following lists describes each value in detail:

5 Taxation Services

- `taxware.currency`: Set this to the three character ISO 4217 code of the currency you want to have all calculations and TAXWARE reports done in. This is a required parameter.
- `taxware.companyId`: This parameter tells TAXWARE your company ID. This is a required parameter.
- `taxware.businessLocation`: This parameter tells WORLDTAX your business location. It is optional, but for WORLDTAX reporting it may be required. If the seller registration number is not set, WORLDTAX uses the company ID and the business location to look up the seller registration number. This mapping is set up using TAWARE tools. If neither the business location nor the seller registration number are set, WORLDTAX cannot properly calculate the tax.
- `taxware.sellerRegistrationNumber`: This is an optional parameter that specifies the registration number of the seller. For this parameters interaction with `businessLocation`, see the preceding bullet item in this list.
- `taxware.shipFrom.country`: The ISO 3166 two or three character abbreviation for the country from which orders are shipped from. This parameter is required.
- `taxware.shipFrom.state`: The state or province from where goods are shipped. This value must be mappable from the `province.properties` file using either name or abbr. This parameter is not required for addresses outside the United States and Canada.
- `taxware.shipFrom.city`: The city from where goods are shipped. This is a required parameter.
- `taxware.shipFrom.zip`: The ZIP code or postal code from where goods are shipped. This is a required parameter. In the United States and Canada, the value is checked using the VERAZIP module within the TAXWARE software. Other postal codes are not currently verified.
- `taxware.shipFrom.geoCode`: This TAXWARE code narrows down an address that covers multiple taxing jurisdictions. It is only used for United States and Canadian addresses. This parameter is optional. However, if the WebLogic Commerce Server can't resolve the address to a single geo code using VERAZIP, it throws a fatal exception. The multiple address information that is returned from VERAZIP is logged to the error log.
- `taxware.orderOrigin`: These parameters are exactly the same as the `shipFrom` parameters but exist in the event that the location from which the

order is taken is different from the location from where the order is shipped. The WebLogic Commerce Server checks for the existence of the `taxware.orderOrigin.country` parameter to determine if an order origin address exists. If this parameter is missing, no order origin parameters are used. If the order origin is not set, the WebLogic Commerce Server defaults the order origin to the ship from address.

- `taxware.titlePassage`: A string that lets TAXWARE determine the point of title passage. This parameter is optional and is used only in the United States and Canada. If set to `shipFrom` (case insensitive), it says that legal transfer has occurred at the point of origin. If set to `shipTo`, it says that legal transfer occurred at the destination. For more information, refer to the TAXWARE SALES/USE documentation.
- `taxware.taxType`: A string that indicates what type of tax should be calculated in SALES/USE. This string must be one of (case insensitive) the following: `sales`, `use`, `rental`, `consumeruse`, or `services`. The default is `sales`.
- `taxware.debug.tax`: If this parameter exists, regardless of its value, it turns on debugging output for the SALES/USE and WORLD TAX components.
- `taxware.debug.verazip`: If this parameter exists, regardless of its value, it turns on debugging output for the VERAZIP component.

Run-Time Configuration

TAXWARE products are integrated with the WebLogic Commerce Server product through the Java Native Interface (JNI). This means that a specially prepared shared object or DLL must be made available for loading during server startup. Additionally, there are a number of files containing the address verification data and tax tables that are accessed at run time. The WebLogic Commerce Server ships with a working version of TAXWARE, complete with the correct DLLs and sample data files. If you have installed TAXWARE in a different location, you must change the location from which these files are loaded. The differences between the default WebLogic Commerce Server and the sample TAXWARE directory structure are shown in Table 5-8.

Table 5-8 Differences in WebLogic Commerce Server and TAXWARE Directory Structures

Default WebLogic Commerce Server Structure	Sample TAXWARE Structure
Subdirectories:	Subdirectories:
\data	\indata
\audit	\outdata
\temp	\audit
\bin	\temp
	\bin

On Windows systems, pointing to the correct file locations is accomplished by making the following changes:

- In the `set-environment.bat` file, change the `WLCS_CLASSPATH` environmental variable to the directory where the TAXWARE Java Class files reside.
- In the `StartCommerce.bat` file, change the `PATH` environment variable in `StartCommerce.bat` to the directory where the TAXWARE DLL files reside.
- In the `avptax.ini`, `avpzip.ini`, and `taxware.ini` files, change the location of the address verification data and tax tables. These files are located in the `winnt` directory. For an example, see Listing 5-16.

Note: For these changes to take effect, you need to restart your server.

The default WebLogic Commerce Server run-time configuration is shown in Listing 5-14.

Listing 5-14 WebLogic Commerce Server Run-Time Configuration on Windows Systems

```
REM ---- Add WebLogic, CyberCash, and Taxware bin directories to the path ----
SETLOCAL
SET
PATH=%PATH%;%WEBLOGIC_HOME%\bin;%WL_COMMERCE_HOME%\eval\win32\CyberCash\bin;%WL_COMMERCE_HOME%\eval\win32\Taxware\bin
```

On UNIX systems, pointing to the correct file locations is accomplished by making the following changes in the file `bin/unix/set-environment.sh`:

1. Set the environment variable `TAXWARE_HOME` to point to the location of your TAXWARE installation. The default WebLogic Commerce Server run-time configuration is shown in Listing 5-15.
2. Set the TAXWARE-specific environment variables to the correct data directories. For an example, see Listing 5-16.
3. Check the environment variable `WLCS_CLASSPATH` to make sure it includes the directory in which `taxcommon.class` lives.
4. Verify that the environment variable for your TAXWARE shared libraries (`.so` or `.sl` files) are correct. For example, under Solaris, the default environment variable `LD_LIBRARY_PATH` includes `$TAXWARE_HOME/lib`. It might change to `$TAXWARE_HOME/utl` or similar depending on your TAXWARE installation.

Notes: The actual variable name varies depending on the type of UNIX platform.

For these changes to take effect, you need to restart your server.

Listing 5-15 The WebLogic Commerce Server Run-Time Configuration on UNIX Systems

```
#----- WLCS Taxware Environment variables -----
TAXWARE_HOME=$WL_COMMERCE_HOME/eval/solaris2/Taxware

#----- Taxware and CyberCash shared objects
LD_LIBRARY_PATH=$TAXWARE_HOME/lib:$WL_COMMERCE_HOME/eval/solaris2
/CyberCash/lib:$JDK_HOME/jre/lib/sparc
export LD_LIBRARY_PATH
```

Listing 5-16 TAXWARE Environment Variables on UNIX Systems (Sample TAXWARE Installation)

```
#-----Taxware Environment variables -----
```

5 Taxation Services

```
TAXWARE_HOME=$WL_COMMERCE_HOME/eval/solaris2/Taxware
AVPIN=$TAXWARE_HOME/indata
export AVPIN
AVPOUT=$TAXWARE_HOME/outdata
export AVPOUT
AVPTEMP=$TAXWARE_HOME/temp
export AVPTEMP
AVPAUDIT=$TAXWARE_HOME/audit
export AVPAUDIT

STEPIN=$TAXWARE_HOME/indata
export STEPIN
STEPOUT=$TAXWARE_HOME/outdata
export STEPOUT
STEPTEMP=$TAXWARE_HOME/temp
export STEPTEMP

ZIPIN=$TAXWARE_HOME/indata
export ZIPIN
ZIPOUT=$TAXWARE_HOME/outdata
export ZIPOUT
ZIPTEMP=$TAXWARE_HOME/temp
export ZIPTEMP

BT_SHARE=N
export BT_SHARE
```

Notes: The use of these directories is described in more detail in the TAXWARE product documentation.

The most important of these directories is the AVPAUDIT directory. This is where the audit information used by TAXWARE to generate tax reports is stored. You will need to establish a process for your production environment whereby a given server is taken offline while the audit files are copied and replaced. The details of this process will depend largely on whether or not you deploy TAXWARE in a cluster.

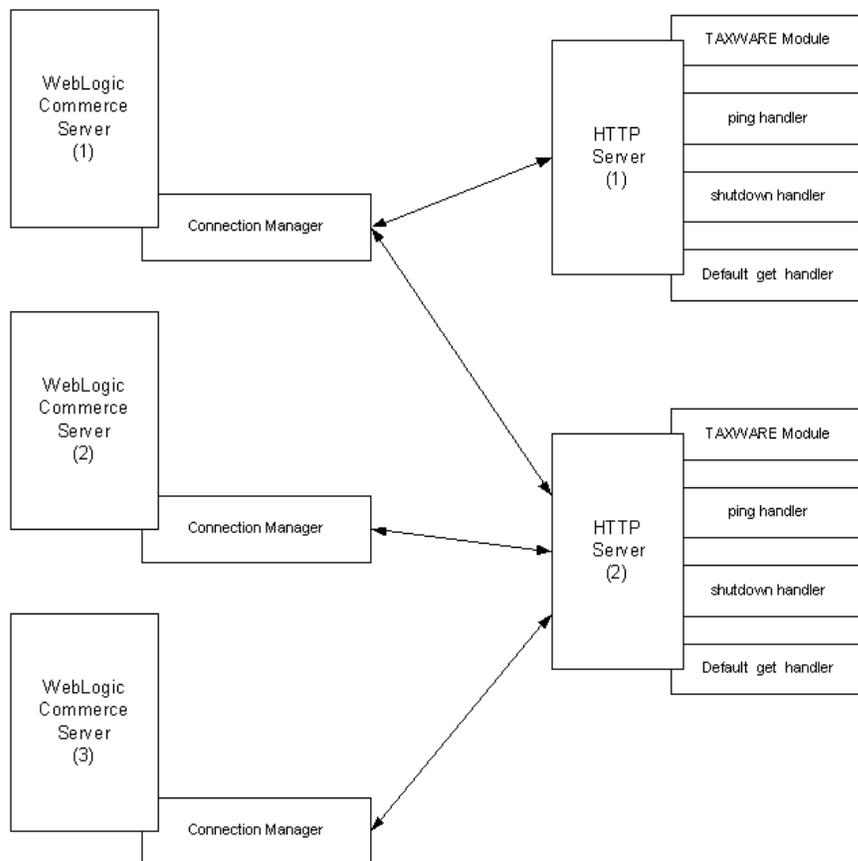
Configuring the HTTP Server for TAXWARE

The TAXWARE service uses one or more separate Java Virtual Machines (JVM) to run an HTTP server. This HTTP server handles TAXWARE requests and returns the TAXWARE data.

Implementing and Configuring the HTTP Server

The TAXWARE service is implemented as a generic multithreaded HTTP server. The server dynamically loads Java classes for servicing a particular type of request. Additionally, so that a sufficient number of file descriptors are available to TAXWARE, it implements a *governor* that limits the number of simultaneous connections. For one possible implementation, see Figure 5-2.

Figure 5-2 A Possible TAXWARE Configuration



A TAXWARE specific handler is part of the standard distribution. Other handlers include a *ping* handler, a handler to serve statistics, a shutdown handler, and a generic *get* handler. Only the TAXWARE and the ping handlers are required.

Server Configuration

Two configuration files are needed for the HTTP server. One for the server side and one for the client side. Both configuration files are Java properties files. The server configuration file is discussed in this section and the client configuration file is discussed in “Connection Manager” on page 5-39.

The server configuration file, named `SimpleHTTPServer.properties`, is in `%WLCS_HOME%\classes\com\beasys\commerce\netservice\http\server` (Windows) or `$WLCS_HOME/classes/com/beasys/commerce/netservice/http/server` (UNIX) and has the following parameters:

- `main.portNumber`: Specifies the port on which the server listens. Under UNIX, if this port is less than or equal to 1024, the server is required to run as the *root* user. The server itself does not require any special permissions to run. However, the server may load a handler that requires a particular permission. This is a required parameter.
- `main.maxThreads`: This parameter tells the server the maximum number of threads that can be active at any one time. It is the *governor* referred to above. Each connection to the server spawns a thread, which means that the number of threads equals the number of socket file descriptors in use. Under Solaris, this parameter defaults to 200.

Note: Except for Solaris, no operating system has limits set by the server. However, limits are still imposed by operating systems. If this parameter doesn't exist, regardless of operating system, a warning is printed. If this number is set too high, you run the risk of hitting a file descriptor limit.

- `postHandler.URI.<name>`: Specifies a handler for an HTTP post, which specifies the URI (Universal Resource Identifier) of the post request. A normal HTTP post request looks like:

```
POST /uri HTTP/1.1
```

where `/uri` is the Universal Resource Identifier for the requested resource.

The server uses the URI to determine which handler is used for the requested resource. For example, the TAXWARE handler provided with WebLogic Commerce Server looks like:

```
postHandler.URI./taxware=com.beasys.commerce.ebusiness.tax.t  
axware.TaxwareNetServiceHandler
```

This means that for post requests asking for `/taxware` as the URI, the server loads and calls the class `com.beasys.commerce.ebusiness.tax.taxware.TaxwareNetServiceHandler` to service the request.

When the server cannot find another handler for a requested URI, it uses a default handler, which is a special handler of type `"*"`. No requirement exists for a default handler; however, if it doesn't exist and a URI comes in without a handler, the client receives a 400 Bad Request message.

- `getHandler.URI.<name>`: This parameter works just like the post handlers. There is a single get handler that must exist. It is the ping handler and is configured as followed:

```
getHandler.URI./ping=com.beasys.commerce.net.service.http.server.PingHandler
```

Additionally, by default, two other get handlers are included: the Shutdown handler and the Statistics handler. The Shutdown handler allows remote shutdown of a server. The Statistics handler returns status information about the server to the requesting client. Although these handlers are useful, you are not required to have them installed.

Note: The default get handler acts as a web server. It returns files in the same way as a normal web server. This handler is useful for testing but it should probably not be deployed in a production environment.

- `getHandler.URI.*.serverRoot`: Specifies the directory name under which any get requests are serviced from the default get handler. For example, if this parameter is set to `/usr/local`, a request of the form

```
http://hostname:portnum/file_name.html
```

reads the file `/usr/local/file_name.html` and sends it back to the client. If your configuration does not have a generic get handler, this parameter is not required.

Connection Manager

Clients talk to the server using a special connection manager. The file for the connection manager, named `HTTPConnectionManager.properties`, is in `%WLCS_HOME%\classes\com\beasys\commerce\net.service\http\client`

(Windows) or
`$WLCS_HOME/classes/com/beasys/commerce/netservice/http/client.`
(UNIX).

The connection manager is configured by two parameters, which are discussed in the following list:

- `connectionManager.pingFrequency`: Specifies, in milliseconds, the amount of time that is allowed to elapse between pings to a server. This parameter defaults to 10000 (10 seconds). A zero value indicates that the connection is pinged each time it is used. A value less than zero means never ping. On a reliable network with a reliable server, this value can be set to a high value such as 60000 (10 minutes). For less reliable networks and servers, set the parameter lower. A ping from the client to the server sends 46 bytes and gets back 43 bytes.
- `connectionManager.connectionType.taxware`: Allows you to specify the host or hosts used for a particular type of connection. Because connections are specified as a particular type, to get a valid connection, clients only have to ask a connection manager for the type of connection that they are interested in. Currently, the only type of connection that exists is the *taxware* connection type. Each service is specified in the following form:

```
host:port:connection_count, host:port:connection_count
```

where

- `host` is the host name or IP address of a server that can handle requests of the named connection type.
- `port` is the port number on which the server listens for connections.
- `connection_count` specifies the maximum number of simultaneous connections that can be made to this host from the client. This parameter is optional. If the `connection_count` parameter is not set, it defaults to 10.

Note: Regardless of how many connections the client wants to make, the server still controls the maximum number of connections that it can handle.

There can be any number of service specifications for a particular type. Separate each triplet by a comma. When the connection manager needs to get a connection, it looks into the pool of available connections and returns the one that was used least recently. Consequently, the load is spread out across all available connections.

Tax Codes and the Product Catalog

Another important factor in the calculation of taxes is that the items in your product catalog must have properly assigned tax codes. Specifically, the tax codes assigned to items in your product catalog must match the tax codes configured in TAXWARE. Ensuring this match involves either manually updating the tax codes using the product catalog administration tool, or creating bulk loading scripts.

Note: To obtain the appropriate tax codes for your product items, refer to the TAXWARE product documentation.

Updating TAXWARE Tax Data

As previously described, TAXWARE periodically provides updates to the tax data used in tax calculations. This update process is handled by TAXWARE tools, for which TAXWARE International, Inc. provides the installation and usage procedures. However, you will need to establish a process for your production environment to handle the server being taken offline and the tax data files updated. This procedure will depend largely on whether or not you deploy TAXWARE in a cluster.

TAXWARE Checklist

Based on the information described in this section, you should be able to configure and deploy the TAXWARE products. The following checklist will help ensure that you have followed all the necessary steps for accurate tax calculations.

- Install and license the TAXWARE components that are not included in the WebLogic Commerce Server product.
- Determine the `shipFrom` address.
- Determine the `orderOrigin` address.
- Determine if the `titlePassage` should be `ShipFrom` or `ShipTo`.
- Record the `companyId` that has been assigned to your organization.
- Determine the `taxType` you will be using.
- Update these values in the `weblogiccommerce.properties` file, located in `WL_COMMERCE_HOME`, where `WL_COMMERCE_HOME` is the directory where you installed WebLogic Commerce Server.

- Ensure that the TAXWARE directories (see “Run-Time Configuration” on page 5-33) are set properly.
- Establish a process by which tax data is periodically updated.
- Establish a process by which tax audit files are archived.

Viewing Debugging Information in TAXWARE

Use the `taxware.debug.tax` property to help you debug TAXWARE. When this property exists, regardless of its value, it turns on debugging output for the SALES/USE and WORLDTAX components.

Listing 5-17 Enable `taxware.debug.tax`

```
# Debug Sale/Use and Worldtax  
#taxware.debug.tax=bug
```

Removing Tax Calculations

This section describes the process by which you might remove Taxation Services from your customized Web application. Removing these tax calculation entails modifying the Pipeline and Webflow properties files to bypass the Taxation Services currently provided in the Order Pipeline.

Modifying the Pipeline Properties File

To remove the Taxation Services from the Pipeline, follow these steps:

1. Copy the `WL_COMMERCE_HOME/pipeline.properties` file to `WL_COMMERCE_HOME/pipeline.properties.stock`, where `WL_COMMERCE_HOME` is the directory in which you installed WebLogic Commerce Server. This is done in case you want to revert back to the original file content.
2. Open the `pipeline.properties` file and locate the `CommitOrder` Pipeline, as shown in Listing 5-18.

Listing 5-18 Default CommitOrder Pipeline

```
# CommitOrder  
  
CommitOrder.componentList=CommitOrderPC, AuthorizePaymentPC,  
TaxCalculateAndCommitLineLevelPC  
CommitOrder.isTransactional=true
```

3. Remove the `TaxCalculateAndCommitLineLevelPC` Pipeline component from the first line of the `CommitOrder` Pipeline definition, so the `CommitOrder` Pipeline is as shown in Listing 5-19.

Listing 5-19 CommitOrder Pipeline Without Tax Pipeline Component

```
# CommitOrder  
  
CommitOrder.componentList=CommitOrderPC, AuthorizePaymentPC  
CommitOrder.isTransactional=true
```

4. Locate and remove (or comment out) all lines that reference the following Pipeline components:
 - `TaxVerifyShippingAddressPC`
 - `TaxCalculateLineLevelPC`
5. Save the modified file. You do not need to restart the server to view your changes if you have set the `pipeline.hotdeploy.enable` property to `true` in the `weblogiccommerce.properties` file.

Modifying the Webflow Properties File

1. Copy the `WL_COMMERCE_HOME/webflow.properties` file to `WL_COMMERCE_HOME/webflow.properties.stock`, where `WL_COMMERCE_HOME` is the directory in which you installed WebLogic Commerce Server. This is done in case you want to revert back to the original file content.

5 Taxation Services

2. Locate the Select Shipping Address Page section of the `webflow.properties` file. In the default configuration, the `TaxVerifyShippingAddress` Pipeline is invoked on successful execution of the `UpdateShippingAddressIP` input processor.

Listing 5-20 Default Shipping Address Page in the `webflow.properties` File

```
...  
SelectShippingAddress_UpdateShippingAddress.inputprocessor.  
success=TaxVerifyShippingAddress.pipeline  
...
```

3. Replace the `TaxVerifyShippingAddress.pipeline` with `CalculateShippingCost.pipeline`.

Listing 5-21 Shipping Address Page Without Tax Pipeline

```
...  
SelectShippingAddress_UpdateShippingAddress.inputprocessor.  
success=CalculateShippingCost.pipeline  
...
```

4. Locate the success path for the `CalculateShippingCost` Pipeline in the `webflow.properties` file.

Listing 5-22 Default Success Path for `CalculateShippingCost` Pipeline

```
...  
CalculateShippingCost.pipeline.success=TaxCalculateLineLevel.  
pipeline  
...
```

5. Replace the `TaxCalculateLineLevel` Pipeline with `PriceOrder.pipeline`, so the success path for the `CalculateShippingCost` Pipeline is as shown in Listing 5-23.

Listing 5-23 Success Path for CalculateShippingCost Pipeline Without Tax Pipeline

```
...  
CalculateShippingCost.pipeline.success=PriceOrder.pipeline  
...
```

6. Locate and remove (or comment out) all lines in the `webflow.properties` file that reference the following:
 - The JSP file `selecttaxaddress.jsp`.
 - The input processors `DecideShippingAddressPageIP` and `UpdateTaxShippingAddressIP`.
7. Save the modified file. You do not need to restart the server to view your changes if you have set the `webflow.hotdeploy.enable` property to `true` in the `weblogiccommerce.properties` file.

What if I Don't Want to Use TAXWARE to Calculate My Taxes?

Although the WebLogic Commerce Server product utilizes products from TAXWARE International, Inc. to calculate taxes, you may choose to use another provider of tax services. If you do not wish to use TAXWARE, you will need to remove TAXWARE from the Pipeline (see “Removing Tax Calculations” on page 5-42), write new Pipeline components to handle tax calculations using the new tax provider, and integrate these Pipeline components into the Webflow/Pipeline infrastructure.

5 *Taxation Services*

Note: The existing TAXWARE Pipeline components are delivered as source and provide an excellent starting point for anyone wanting to use another provider of tax services. The integration point for tax calculations is the `Tax` attribute of the `ShoppingCartLine`, for which you can use the `set()` and `get()` methods to set the tax for each line in a customer's shopping cart. For more information, see the *Javadoc*.

6 Payment Services

Managing Purchases and Processing Orders services also contains a Payment Service, which specifies how payment for an order is authorized and settled. Currently the Payment Service allows credit card payments to be made using the CyberCash, Inc. service. However, the JSP templates, input processors, and Pipeline components allow different services to be integrated. This topic describes the Payment Services in detail.

This topic includes the following sections:

- JavaServer Pages (JSPs)
 - payment.jsp Template
 - paymentnewcc.jsp Template
 - paymenteditcc.jsp Template
- Input Processors
 - PaymentAuthorizationIP
 - UpdatePaymentInfoIP
- Pipeline Components
 - PaymentAuthorizationHostPC
 - PaymentAuthorizationTerminalPC
- Integration with CyberCash
 - Configuration Activities for Using CyberCash
 - What if I Don't Want to Use CyberCash for Credit Card Processing?
- Credit Card Security Service

JavaServer Pages (JSPs)

A primary goal of Managing Purchases and Processing Orders services' is to allow you to quickly establish a fully-functioning e-commerce site. To this end, the Payment Service provides you with a JavaServer 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 Commerce Server 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 6-1) 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*.

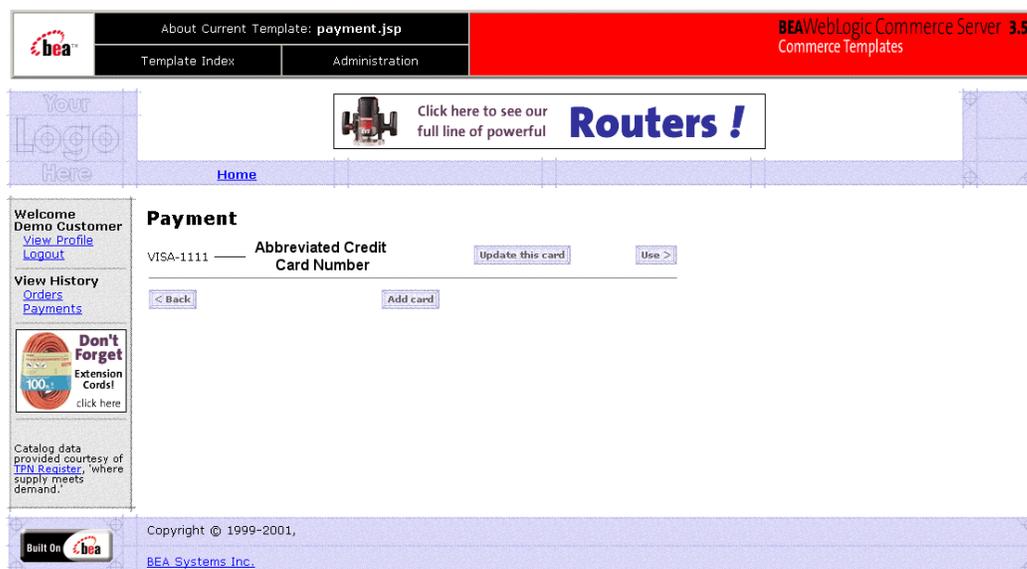
Customers can also choose to update the information associated with this credit card by clicking the Update This Card button. If your customer wants to use a credit card they have never used on your e-commerce site before, the customer can click the Add Card button to add it to the list (using the `paymentnewcc.jsp`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

Sample Browser View

Figure 6-1 shows an annotated version of the `payment.jsp` template. The Payment region uses a combination of the WebLogic Server and WebLogic Personalization Server 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 6-1 Annotated payment.jsp Template



Location in the WebLogic Commerce Server Directory Structure

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

```
%WL_COMMERCE_HOME%\server\webapps\wlcs\commerce\order\payment.jsp
(Windows)
```

```
$WL_COMMERCE_HOME/server/webapps/wlcs/commerce/order/payment.jsp
(UNIX)
```

Tag Library Imports

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

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

6 Payment Services

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

These files reside in the following directory for the WebLogic Commerce Server Web application:

```
%WL_COMMERCE_HOME%\server\webapps\wlcs\WEB-INF (Windows)
$WL_COMMERCE_HOME/server/webapps/wlcs/WEB-INF (UNIX)
```

Java Package Imports

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

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

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:

- `innerheader.jsp`, which creates the top banner.

- `innerfooter.jsp`, which creates a horizontal footer at the bottom of the page, and also includes the `rightside.jsp` template. `rightside.jsp` describes (for the benefit of you and your development team) the name of the current template and links to its *About* information.

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)
<code>button(addNewCreditCard)</code>	No business logic required. Loads <code>paymentnewcc.jsp</code> .
<code>button(continue)</code>	<code>AuthorizePaymentIP</code>
<code>button(updatePaymentInfo)</code>	No business logic required. Loads <code>paymenteditcc.jsp</code> .

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 Personalization Server 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 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>
  <td><%=creditCard%></td>
</tr>
</wl:repeat>
</table>
```

Note: For more information on the WebLogic Server JSP tags or the WebLogic Personalization Server 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

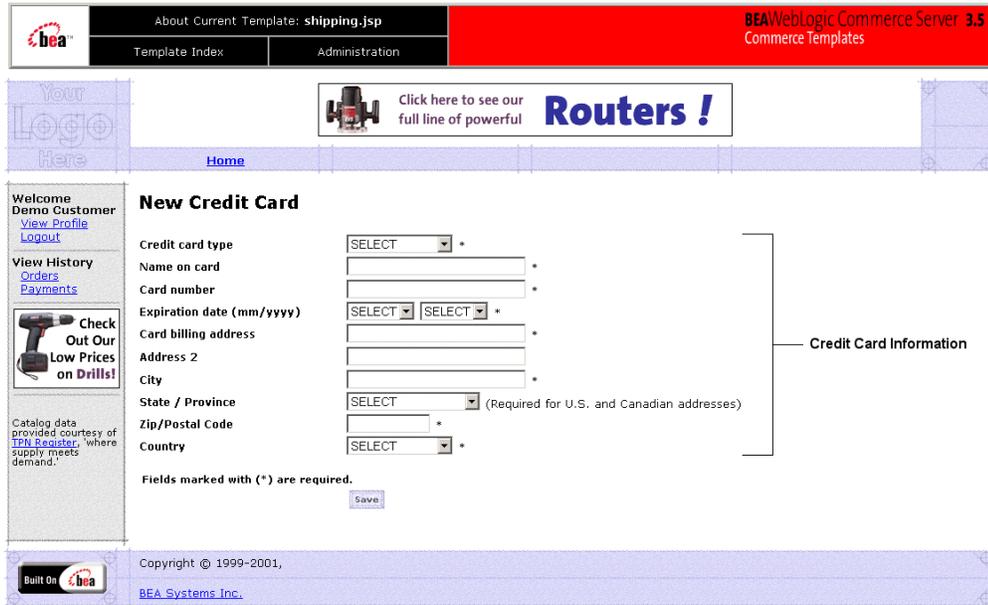
The `paymentnewcc.jsp` template (shown in Figure 6-2) allows customers to enter information about a new credit card, which will be added to their profile. This information includes the credit card type (VISA, MasterCard, and so on), the name on the card, the card number, the card expiration date (month 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 6-2 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 6-2 Annotated paymentnewcc.jsp Template



Location in the WebLogic Commerce Server Directory Structure

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

```
%WL_COMMERCE_HOME%\server\webapps\wlcs\commerce\order\
paymentnewcc.jsp (Windows)
$WL_COMMERCE_HOME/server/webapps/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" %>
<%@ taglib uri="pipeline.tld" prefix="pipeline" %>
```

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 following directory for the WebLogic Commerce Server Web application:

```
%WL_COMMERCE_HOME%\server\webapps\wlcs\WEB-INF (Windows)
$WL_COMMERCE_HOME/server/webapps/wlcs/WEB-INF (UNIX)
```

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

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.

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:

- `innerheader.jsp`, which creates the top banner.
- `innerfooter.jsp`, which creates a horizontal footer at the bottom of the page, and also includes the `rightside.jsp` template. `rightside.jsp` describes (for the benefit of you and your development team) the name of the current template and links to its *About* information.
- `newcctemplate.jsp`, 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)
<code>button(save)</code>	<code>UpdatePaymentInfoIP</code>

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.

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 `<%= HttpServletRequest.Constants.CUSTOMER_CREDIT_CARD_COUNTRY %>`) for use in the JSP.

paymenteditcc.jsp Template

The `paymenteditcc.jsp` template (shown in Figure 6-3) 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 6-3 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.jsp" %>
```

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

Figure 6-3 Annotated paymenteditcc.jsp Template

Location in the WebLogic Commerce Server Directory Structure

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

```
%WL_COMMERCE_HOME%\server\webapps\wlcs\commerce\order\
paymenteditcc.jsp (Windows)
$WL_COMMERCE_HOME/server/webapps/wlcs/commerce/order/
paymenteditcc.jsp (UNIX)
```

Tag Library Imports

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

6 Payment Services

```
<%@ taglib uri="webflow.tld" prefix="webflow" %>
<%@ taglib uri="pipeline.tld" prefix="pipeline" %>
<%@ 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 Personalization Server JSP tags, see “JSP Tag Reference” in the *Guide to Building Personalized Applications*.

These files reside in the following directory for the WebLogic Commerce Server Web application:

```
%WL_COMMERCE_HOME%\server\webapps\wlcs\WEB-INF (Windows)
$WL_COMMERCE_HOME/server/webapps/wlcs/WEB-INF (UNIX)
```

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

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.

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:

- `innerheader.jsp`, which creates the top banner.
- `innerfooter.jsp`, which creates a horizontal footer at the bottom of the page, and also includes the `rightside.jsp` template. `rightside.jsp` describes (for the benefit of you and your development team) the name of the current template and links to its *About* information.
- `editcctemplate.jsp`, 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)
<code>button(save)</code>	<code>UpdatePaymentInfoIP</code>

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 the WebLogic Personalization Server’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 Personalization Server's 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 propertyName="creditCardsMap"
id="creditCardsMapObject" />

<%
Map creditCardsMap = (Map) creditCardsMapObject;
String creditCardKey =
    request.getParameter(HttpRequestConstants.CREDITCARD_KEY);
CreditCard defaultCreditCard = null;
defaultCreditCard = (CreditCard)
creditCardsMap.get(creditCardKey);
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
<code>getType()</code>	The credit card type (VISA, MasterCard, AMEX, and so on).
<code>getName()</code>	The credit card holder's name.
<code>getDisplayNumber()</code>	The credit card number for display (12 Xs and last 4 digits).
<code>getNumber()</code>	The credit card number.
<code>getExpirationDate()</code>	The credit card's expiration date.

Table 6-6 billingAddress Accessor Methods/Attributes

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

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.
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.

Input Processors

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

PaymentAuthorizationIP

Class Name	<code>com.beasys.commerce.ebusiness.payment.webflow.PaymentAuthorizationIP</code>
Description	Retrieves the shopping cart from the Pipeline session, the <code>CreditCardMapKey</code> 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	<code>HttpRequestConstants.CREDITCARD_KEY</code>
Required Pipeline Session Attributes	<code>PipelineSessionConstants.SHOPPING_CART</code>
Updated Pipeline Session Attributes	<code>PipelineSessionConstants.PAYMENT_CREDIT_CARD</code> <code>PipelineSessionConstants.PAYMENT_AUTHORIZATION_AMOUNT</code>
Removed Pipeline Session Attributes	None
Validation	Verifies that the credit card key is valid and that it references an existing credit card.
Exceptions	<code>ProcessingException</code> , thrown for invalid types of <code>CREDITCARD_KEY</code> , <code>PAYMENT_CREDIT_CARD</code> , or <code>SHOPPING_CART</code> . Also thrown if these attributes are not available.

UpdatePaymentInfoIP

Class Name	<code>com.beasys.commerce.ebusiness.customer.webflow.UpdatePaymentInfoIP</code>
Description	Processes the customer's input from <code>paymentnewcc.jsp</code> and <code>paymenteditcc.jsp</code> . Retrieves the customer name from the Pipeline session, creates a new <code>CustomerValue</code> object, and sets it in the Pipeline session.
Required HttpServletRequest Parameters	<code>HttpRequestConstants.CUSTOMER_CREDITCARD_TYPE</code> <code>HttpRequestConstants.CUSTOMER_CREDITCARD HOLDER</code> <code>HttpRequestConstants.CUSTOMER_CREDITCARD_NUMBER</code> <code>HttpRequestConstants.CUSTOMER_CREDITCARD_MONTH</code> <code>HttpRequestConstants.CUSTOMER_CREDITCARD_YEAR</code> <code>HttpRequestConstants.CUSTOMER_CREDITCARD_ADDRESS1</code> <code>HttpRequestConstants.CUSTOMER_CREDITCARD_ADDRESS2</code> <code>HttpRequestConstants.CUSTOMER_CREDITCARD_CITY</code> <code>HttpRequestConstants.CUSTOMER_CREDITCARD_STATE</code> <code>HttpRequestConstants.CUSTOMER_CREDITCARD_ZIPCODE</code> <code>HttpRequestConstants.CUSTOMER_CREDITCARD_COUNTRY</code>
Required Pipeline Session Attributes	<code>PipelineSessionConstants.USER_NAME</code>
Updated Pipeline Session Attributes	<code>PipelineSessionConstants.CUSTOMER</code>
Removed Pipeline Session Attributes	None
Validation	Verifies that the required fields contain values.
Exceptions	<code>InvalidInputException</code> , thrown if invalid credit card information is obtained from the <code>HttpServletRequest</code> .

Pipeline Components

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

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

PaymentAuthorizationHostPC

Class Name	<code>com.beasys.commerce.ebusiness.payment.pipeline.PaymentAuthorizationHostPC</code>
Description	Authorizes a given credit card for a specified amount. Used for host-based payment models, shown in the <code>weblogiccommerce.properties</code> file as: <code>HOST_AUTH_CAPTURE</code> <code>HOST_AUTH_CAPTURE_AVS</code> <code>HOST_POST_AUTH_CAPTURE</code> <code>HOST_POST_AUTH_CAPTURE_AVS</code>
Required Pipeline Session Attributes	<code>PipelineSessionConstants.PAYMENT_CREDIT_CARD</code> <code>PipelineSessionConstants.PAYMENT_AUTHORIZATION_AMOUNT</code> <code>PipelineSessionConstants.ORDER_HANDLE (Request scope)</code>
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.

PaymentAuthorizationTerminalPC

Class Name	<code>com.beasys.commerce.ebusiness.payment.pipeline.PaymentAuthorizationTerminalPC</code>
Description	Authorizes a given credit card for a specified amount. Used for terminal-based payment models, shown in the <code>weblogiccommerce.properties</code> file as: <code>AUTO_MARK_AUTO_SETTLE</code> <code>AUTO_MARK_AUTO_SETTLE_AVS</code> <code>AUTO_MARK_MANUAL_SETTLE</code> <code>AUTO_MARK_MANUAL_SETTLE_AVS</code> <code>MANUAL_MARK_AUTO_SETTLE</code> <code>MANUAL_MARK_AUTO_SETTLE_AVS</code> <code>MANUAL_MARK_MANUAL_SETTLE</code> <code>MANUAL_MARK_MANUAL_SETTLE_AVS</code>
Required Pipeline Session Attributes	<code>PipelineSessionConstants.PAYMENT_CREDIT_CARD</code> <code>PipelineSessionConstants.PAYMENT_AUTHORIZATION_AMOUNT</code> <code>PipelineSessionConstants.ORDER_HANDLE (Request scope)</code>
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).

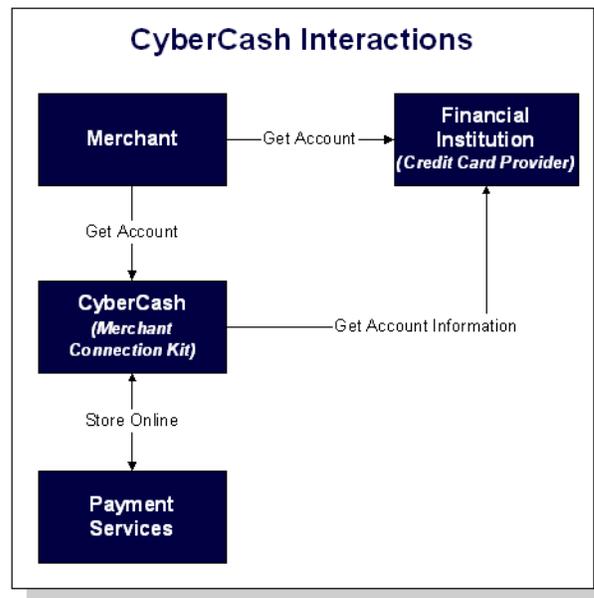
`PipelineNonFatalException`, thrown when the external payment service is unavailable. The transaction is recorded for retry.

`PipelineFatalException`, thrown when there is a configuration error, a general service error, or a system-level exception from a back-end component.

Integration with CyberCash

Part of the functionality provided by the Payment Services is their ability to interact with CyberCash, a service which allows you to accept credit cards from customers over the Internet. However, to run CyberCash with Payment Services, you will need to perform a number of configuration activities so that CyberCash, your financial institution (credit card provider), and the Payment Services can work together as shown in Figure 6-4.

Figure 6-4 CyberCash Interactions Diagram



Note: For more information about CyberCash, Inc. and their payment solutions, see <http://www.cybercash.com>.

Configuration Activities for Using CyberCash

The following is a list of the configuration activities you must perform in order to use CyberCash with Payment Services:

1. Obtain an account from a financial institution that provides credit card processing services. At this time, you will receive a payment model.

Note: For more information about the possible payment models, see “Payment Models” on page 6-29.

2. Using the account information from your financial institution, register and apply for a merchant bank account with CyberCash at <http://amps.cybercash.com/>. Once you install the Merchant Connection Kit (MCK) from CyberCash on your machine, you can create a merchant account. As part of this process, you will also create a configuration file.
3. In the `weblogiccommerce.properties` file (located in the `WL_COMMERCE_HOME` directory, where `WL_COMMERCE_HOME` is the directory in which you installed WebLogic Commerce Server), use the `CyberCashConfigFile` property to specify the location of the CyberCash configuration file on your system, as shown in Listing 6-6.

Note: Be sure to carefully read the instructions in the `weblogiccommerce.properties` file under the Payment Services heading prior to making any changes.

Listing 6-6 Setting the CyberCashConfigFile Property

```
#####
# Properties required for the payment component
#####

#
# This property defers payment authorization to the administration tools.
# If set to true, all payment service authorization calls are disabled
# and payment transactions are persisted in a RETRY state. Payments must
# then be reauthorized through the payment administration tool.
#
commerce.payment.defer.authorization=true
```

6 Payment Services

```
#
# CyberCash configuration files contain CyberCash-specific data, such as a
# merchant-id and merchant hash secret. The specific properties in the
# configuration files depend upon the payment model assigned to a merchant by
# his/her financial institution. The two files declared below are example files
# and are provided for demonstration purposes ONLY. MERCHANTS MUST ACQUIRE A
# CYBERCASH CONFIGURATION FILE FROM CYBERCASH. These will be furnished by
# CyberCash as part of the merchant agreement. Once a merchant has a CyberCash
# configuration file, the property below must be replaced with the location of
# the configuration file.
#
# Example: CyberCashConfigFile=c:/merchang/config/file/location/merchant_conf
#
# This file may be used for testing terminal based payment models.
CyberCashConfigFile=@BEA_WEBLOGIC_COMMERCE_SERVER_HOME@/eval/common/CyberCash/
conf/merchant_conf-terminal
#
# This file may be used for testing host based payment models.
CyberCashConfigFile=@BEA_WEBLOGIC_COMMERCE_SERVER_HOME@/eval/common/CyberCash/
conf/merchant_conf-host
```

Note: Single front slashes (or double back slashes) are required in this location specification.

4. If you want to perform real-time authorization, you must set the `commerce.payment.defer.authorization` property in the `weblogiccommerce.properties` file to `false`. Otherwise, set it to `true` for offline authorization using the Payment Management Administration Tool.

Note: For instructions on how to use the Payment Management Administration Tool, see Chapter 9, “Using the Order and Payment Management Pages.”

5. In the `weblogiccommerce.properties` file, use the `PaymentModel` property to specify the payment model you received from your financial institution, as shown in Listing 6-7.

Listing 6-7 Setting the PaymentModel Property

```
#
# Properties below represent the different payment models provided # by CyberCash.
#
```

```
# Terminal based models
PaymentModel=AUTO_MARK_AUTO_SETTLE
# PaymentModel=AUTO_MARK_AUTO_SETTLE_AVS
# PaymentModel=AUTO_MARK_MANUAL_SETTLE
# PaymentModel=AUTO_MARK_MANUAL_SETTLE_AVS
# PaymentModel=MANUAL_MARK_AUTO_SETTLE
# PaymentModel=MANUAL_MARK_AUTO_SETTLE_AVS
# PaymentModel=MANUAL_MARK_MANUAL_SETTLE
# PaymentModel=MANUAL_MARK_MANUAL_SETTLE_AVS

# Host based models
#PaymentModel=HOST_AUTHCAPTURE
#PaymentModel=HOST_AUTHCAPTURE_AVS
#PaymentModel=HOST_AUTH_POSTAUTH
#PaymentModel=HOST_AUTH_POSTAUTH_AVS
```

6. Be sure to save your changes to the `weblogiccommerce.properties` file, and restart the server.

Note: Detailed documentation for CyberCash, Inc. products can be found online at <http://www.cybercash.com/cashregister/docs/>.

Payment Models

There are two types of payment models: terminal-based and host-based. The difference between these payment models is where the transaction batch is stored. For a host-based model, the transaction batch is stored on the host network rather than on the local system at the merchant's site. Settlement typically occurs sometime at the end of the day, and the merchant is not required to do anything to initiate the settlement process.

For a terminal-based model, the transaction batch is stored as data files on the local system at the merchant's site. Merchants must initiate the settlement process at the end of each day in order for the funds to be transferred to the merchant's bank account.

Table 6-8 describes each of the terminal-based payment models that may be assigned by your financial institution. Table 6-9 describes each of the host-based payment models that may be assigned.

Table 6-8 Terminal-based Payment Models

Payment Model	Description
AUTO_MARK_AUTO_SETTLE	This payment model is used for soft goods. Settlement occurs as soon as authorization is complete, because it is assumed that soft goods are shipped at the time of purchase.
AUTO_MARK_MANUAL_SETTLE	This payment model is used in cases where goods have been shipped at authorization but the merchant requests that funds should be transferred at a later date.
MANUAL_MARK_AUTO_SETTLE	This payment model allows merchants to indicate that the goods have been shipped, at which point settlement is done automatically.
MANUAL_MARK_MANUAL_SETTLE	This is the most flexible payment model in that it allows merchants to specify when goods are shipped and when funds should be transferred. The mark process allows the merchant to specify that the goods have been shipped. The settlement process allows the merchant to indicate that funds may be transferred.

Note: Each of the terminal-based payment models may be suffixed by `_AVS`. This suffix indicates that merchants are also required to send an address. The WebLogic Commerce Server product always sends this address for verification purposes.

Table 6-9 Host-based Payment Models

Payment Model	Description
HOST_AUTH_CAPTURE	This payment model is used for services, sale of digital goods, or physical goods shipped within 24 hours of when the order is placed. In this case, the merchant only needs to get an authorization for the purchase amount. The capture of the authorization into the batch and the settlement of the transaction are done for the merchant by the processor at the time of authorization.
HOST_POST_AUTH_CAPTURE	When the merchant fulfills orders more than one day after receiving them, the merchant must authorize and capture transactions separately. In this payment model, authorization is performed at the time the consumer wants to make the purchase. Capture is performed when the merchant ships the order. The processor handles settlement of the batched transactions at certain times of the day.

Note: Each of the host-based payment models may be suffixed by `_AVS`. This suffix indicates that merchants are also required to send an address. The WebLogic Commerce Server product always sends this address for verification purposes.

How Do I Switch Between the Two Payment Models?

If you decide to use the terminal-based payment model, your Web application must use the `PaymentAuthorizationTerminalPC` Pipeline component. If you decide to use the host-based payment model, your Web application must use the `PaymentAuthorizationHostPC` Pipeline component instead.

To change the Pipeline component to reflect the payment model, follow these steps:

1. Start a simple text editor like Notepad.
2. Open the `weblogiccommerce.properties` file, which can be found in `WL_COMMERCE_HOME`, where `WL_COMMERCE_HOME` is the top-level directory where you installed WebLogic Commerce Server

6 *Payment Services*

3. Set the Payment Model property (refer to Listing 6-7 for more details), and save the `weblogiccommerce.properties` file.
4. Open the default Pipeline properties file, which can be found in `WL_COMMERCE_HOME/pipeline.properties`, where `WL_COMMERCE_HOME` is the top-level directory where you installed WebLogic Commerce Server.
5. In the `AuthorizePaymentPC` Pipeline component definition (set to use the `PaymentAuthorizationTerminalPC` Pipeline component by default), change the `className`, `jndiName`, and `isEJBSessionBean` properties to reflect those associated with the other Pipeline component.

Note: For more information about the properties associated with the `PaymentAuthorizationTerminalPC` and `PaymentAuthorizationHostPC` Pipeline components, see “Pipeline Components” on page 6-22.

6. Save the modified file. You do not need to restart the server to view your changes if you have set the `pipeline.hotdeploy.enable` property to `true` in the `weblogiccommerce.properties` file.

What if I Don't Want to Use CyberCash for Credit Card Processing?

The WebLogic Commerce Server product provides you with a CyberCash-based implementation of a Payment Service. However, you may want to use a service provider other than CyberCash. Use of a different provider requires that you implement a payment authorization Pipeline component that is specific to the provider of your choice.

Note: It is expected that a Java/EJB programmer (or someone with similar technical knowledge and abilities) will develop new Pipeline components.

To implement a new Pipeline component for a Payment Service provider other than CyberCash, complete the following steps:

1. Create a new Pipeline component that extends `CommercePipelineComponent`, as shown in Listing 6-8.

Listing 6-8 Creating a New Pipeline Component

```
/ java imports

import java.rmi.RemoteException;
import java.sql.Date;
import java.sql.Connection;

// javax imports
import javax.ejb.*;

// com.beasys imports
import com.beasys.commerce.ebusiness.payment.*;
import com.beasys.commerce.ebusiness.order.*;
import com.beasys.commerce.ebusiness.security.*;
import com.beasys.commerce.axiom.contact.*;
import com.beasys.commerce.axiom.units.*;
import com.beasys.commerce.axiom.util.helper.*;
import com.beasys.commerce.webflow.*;
import com.beasys.commerce.foundation.*;
import com.beasys.commerce.foundation.exception.*;
import com.beasys.commerce.foundation.pipeline.*;
import com.beasys.commerce.util.*;
```

6 Payment Services

```
/**
 * This <code>PipelineComponent</code> authorizes a credit card
 * for a purchase of a given amount using a payment service other
 * than CyberCash. This class is a concrete extension of the
 * <code>CommercePipelineComponent</code> abstract base class.
 *
 * PipelineSession input attributes:
 *   PipelineSessionConstants.PAYMENT_CREDIT_CARD
 *   PipelineSessionConstants.PAYMENT_AUTHORIZATION_AMOUNT
 *   PipelineSessionConstants.ORDER_HANDLE
 */

public class MyPaymentAuthorizationPC extends
CommercePipelineComponent

{
```

2. Implement the `process()` method (as declared in the `PipelineComponent` interface) in the new Pipeline component, as shown in Listing 6-9.

Listing 6-9 Implementing the `process()` Method

```
/**
 * Authorize a credit card for a purchase amount.
 *
 * @param pipelineSession The current PipelineSession
 * @throws PipelineFatalException on fatal error
 * @throws PipelineNonFatalException on non-fatal error
 * @throws RemoteException on remote error
 */

public PipelineSession process(PipelineSession pipelineSession)
throws PipelineFatalException, PipelineNonFatalException, RemoteException {

    //
    // Get the order, credit card, and authorization amount from
    // the PipelineSession.
    //

    CreditCard card = (CreditCard)pipelineSession.
getAttribute(PipelineSessionConstants.PAYMENT_CREDIT_CARD);

    Price amount = (Price)pipelineSession.getAttribute
(PipelineSessionConstants.PAYMENT_AUTHORIZATION_AMOUNT);
```

```
Handle orderHandle = (Handle)pipelineSession.getAttribute
(PipelineSessionConstants.ORDER_HANDLE, PipelineConstants.REQUEST_SCOPE);

Order order = (Order)(orderHandle.getEJBObject());

//Create a Transaction ID
//This can be done with any persistent number generator.
//Every transaction ID must be unique.
//Look at
//http://edocs.beasys.com/wlcs/docs32/javadoc/wlps/com/beasys/commerce/util/Seq
uencer.html
//for information on the Sequencer interface.

com.beasys.commerce.util.Sequencer mySequencer =
    com.beasys.commerce.util.SequencerFactory.createSequencer
    ("PaymentTransactionIDSequence");
mySequencer.setCacheSize(10);    //optional

Connection myConnection = getConnection();
long myTransactionID = 0;

try {

    myTransactionID = mySequencer.getNext(myConnection);
} catch (java.sql.SQLException sqlException) {

    //Add the appropriate exception handling logic.

}

//
// Decrypt the credit card using the Decryptor service.
//

String creditCardNumber = null;
try {

    DecryptorHome home = (DecryptorHome)JNDIHelper.getHome(
        "com.beasys.commerce.ebusiness.security.Decryptor");
    Decryptor decryptor = home.create();
    creditCardNumber = decryptor.decrypt(card.getNumber());

} catch (Exception e) {

    // Add the appropriate exception handling logic.
```

6 Payment Services

```
        // This will depend on your payment service requirements.
    }

    //
    // Invoke the credit card service authorization method using
    // the order, credit card, and authorization amount.
    //
    // Throw an appropriate exception for authorization error
    // condition(s).
    //

    Logger.getInstance().info("In MyPaymentAuthorizationPC:
        calling payment service.");

    < Insert credit card service authorization code here >

    //
    // Immediately nullify the decrypted number.
    //

    creditCardNumber = null;

    //
    // If the authorization was successful, create a
    // PaymentTransaction entity EJB for the transaction.
    // Use the transaction ID returned by the credit card
    // service as the primary key.

    PaymentTransaction paymentTransaction = null;
    try {
        PaymentTransactionHome home = (PaymentTransactionHome)JNDIHelper.
            getHome("com.beasys.commerce.ebusiness.payment.PaymentTransaction");
        PaymentTransactionPk pk =
            new PaymentTransactionPk(Long.toString(myTransactionID));
        paymentTransaction = home.create(pk);
    } catch (Exception e){

        // Add the appropriate exception handling logic.
        // This will depend on your payment service requirements.

    }

    //
    // Set the PaymentTransaction date, credit card, and amount.
    //

    paymentTransaction.setTransactionDate(new Date(System.
        currentTimeMillis()));
```

```
paymentTransaction.setCreditCard(card);
paymentTransaction.setTransactionAmount(amount);

//
// Add a TransactionEntry to the PaymentTransaction and
// mark the PaymentTransaction with the appropriate status.
// In this example, we assume that the payment transaction
// was successfully authorized.
//

TransactionEntry entry = TransactionEntryHome.create();
entry.setIdentifier(Long.toString(myTransactionID));
entry.setEntryDate(new Date(System.currentTimeMillis()));
entry.setTransactionAmount(amount);

try {

    paymentTransaction.authorize();

} catch (IllegalWorkflowTransitionException e){

    // Add the appropriate exception handling logic.
    // This will depend on your payment service requirements.

}

paymentTransaction.addTransactionEntry(entry);

//
// Add a reference to the PaymentTransaction to the order.
//

order.setPaymentTransaction(paymentTransaction);
return pipelineSession;

}
}
```

As shown in Listing 6-9, the credit card, authorization amount, and order is first extracted from the supplied `PipelineSession`. Next, the Decryptor security service is used to decrypt the encrypted credit card number. After obtaining all the information necessary to authorize a payment, you must next call your Payment Service provider authorization routine using any of the collected data necessary. Finally, after completing the authorization, a payment transaction is recorded using the `PaymentTransaction` entity EJB. The

6 Payment Services

PaymentTransaction entity EJB records the date, amount, credit card, and status (in this case, authorized) associated with the payment. It also keeps an audit trail of payment transaction modifications via a collection of TransactionEntry objects. Each TransactionEntry object stores a date, identifier, and amount.

3. Compile the new Pipeline component. Make sure to include any Payment Service provider classes that the new Pipeline component uses in your classpath.
4. Configure the `pipeline.properties` file to use your new Pipeline component. To do this, locate the following line in the `pipeline.properties` file:

```
AuthorizePaymentPC.className=com.beasys.commerce.ebusiness.  
payment.pipeline.PaymentAuthorizationTerminalPC
```

Then, modify the `AuthorizePaymentPC` Pipeline component definition to use your new Pipeline component as follows:

```
AuthorizePaymentPC.className=MyPaymentAuthorizationPC
```

5. Restart the WebLogic Commerce Server. Make sure to include the new Pipeline component as well as any Payment Service provider classes used by the Pipeline component in your classpath.

You should now be able to authorize payments using the new Payment Service PipelineComponent.

Note: If you replace the existing Payment Authorization Pipeline component, you must administer payments using tools supplied by your Payment Service provider and NOT the administrative Payment Management pages. The administrative Payment Management pages should only be used for CyberCash-based payment administration. For more information about the administrative Payment Management pages, see Chapter 9, “Using the Order and Payment Management Pages.”

Credit Card Security Service

All credit card information your customers provide is considered sensitive and is encrypted for security purposes. This information is decrypted only when absolutely necessary during specific payment processing activities (authorization). For example, on the order confirmation JSP template (`confirmorder.jsp`), only the last 4 digits of a customer's credit card are displayed.

Notes: For more information about credit card security, see “Credit Card Security Service” in the *Security Guide*.

For information about other BEA security features, see the *Security Guide*.

6 *Payment Services*

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 Commerce Server 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

BEA WebLogic Commerce Server 3.5
Commerce Templates

Don't forget to buy and wear our **Safety Equipment!**

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: xxxxxxxxxxxxxx8484

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

< Back Complete purchase >

Copyright © 1999-2001, BEA Systems Inc.

7 Order Summary and Confirmation Services

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

1. The Final Checkout Review region uses a combination of the WebLogic Personalization Server 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 the WebLogic Personalization Server 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 Commerce Server Directory Structure

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

```
%WL_COMMERCE_HOME%\config\wlcsDomain\applications\wlcs\commerce\order\  
checkout.jsp (Windows)  
$WL_COMMERCE_HOME/config/wlcsDomain/applications/wlcs/commerce/order/  
checkout.jsp (UNIX)
```

Tag Library Imports

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

```
<%@ taglib uri="weblogic.tld" prefix="wl" %>  
<%@ taglib uri="pipeline.tld" prefix="pipeline" %>  
<%@ 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 Personalization 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 following directory for the WebLogic Commerce Server Web application:

```
%WL_COMMERCE_HOME%\config\wlcsDomain\applications\wlcs\WEB-INF
(Windows)
$WL_COMMERCE_HOME/config/wlcsDomain/applications/wlcs/WEB-INF
(UNIX)
```

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.webflow.*" %>
<%@ page import="com.beasys.commerce.axiom.units.*" %>
<%@ page import="com.beasys.commerce.axiom.contact.*" %>
<%@ page import="com.beasys.commerce.ebusiness.shoppingcart.*" %>
<%@ page import="com.beasys.commerce.foundation.pipeline.*" %>
<%@ page import="com.bea.commerce.ebusiness.price.service.DiscountPresentation"
%>
<%@ page import="com.bea.commerce.ebusiness.price.quote.OrderAdjustment" %>
<%@ page import="com.bea.commerce.ebusiness.price.quote.AdjustmentDetail" %>
<%@ page import="com.bea.commerce.ebusiness.price.quote.AdjustmentType" %>
```

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.

7 Order Summary and Confirmation Services

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)
<code>button(back)</code>	No business logic required. Loads <code>payment.jsp</code> .
<code>button(purchase)</code>	<code>CommitOrder</code>

Table 7-2 briefly describes each of the Pipelines from Table 7-1, as they are defined in the `pipeline.properties` file. For more information about individual Pipeline components, see “Pipeline Components” on page 7-21.

Table 7-2 Checkout Review Pipelines

Pipeline	Description
<code>CommitOrder</code>	Contains <code>CommitOrderPC</code> , <code>AuthorizePaymentPC</code> , <code>CalculateTaxLineLevelCommitPC</code> , and is transactional.
<code>PurchaseTracker</code>	Contains <code>PurchaseTrackerPC</code> , <code>ResetCheckoutPC</code> , 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 Personalization Server 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:getProfile
  profileKey="<%=request.getRemoteUser()%>
  profileType="WLCS_Customer" />
```

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

Next, the `getPipelineProperty` 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	com.beasys.commerce.ebusiness .shipping.shippingMethodValue	Identifies the shipping method the customer selected.
PipelineSessionConstants. SHOPPING_CART	com.beasys.commerce.ebusiness .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	com.bea.commerce.ebusiness .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.

7 Order Summary and Confirmation Services

Listing 7-2 Retrieving Check Out Attributes

```
<pipeline:getPipelineProperty
  propertyName="<%=PipelineSessionConstants.SHOPPING_CART%>"
  returnName="shoppingCart"
  returnType="com.beasys.commerce.ebusiness.shoppingcart.ShoppingCart" />

<pipeline:getPipelineProperty
  propertyName="<%=PipelineSessionConstants.SHIPPING_ADDRESS%>"
  returnName="shippingAddress"
  returnType="com.beasys.commerce.axiom.contact.Address" />
```

Note: For more information on the `getPipelineProperty` 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

```
<table>
  <tr>
    <td>
      <um:getPropertyAsString propertyName="firstName" />
      <um:getPropertyAsString propertyName="lastName" />
    </td>
  </tr>
</table>
```

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

The data stored within the Pipeline session attributes (retrieved using the `getPipelineProperty` 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
<code>getStreet1()</code>	The first line in the customer's street address.
<code>getStreet2()</code>	The second line in the customer's street address.
<code>getCity()</code>	The city in the customer's address.
<code>getState()</code>	The state in the customer's address.
<code>getPostalCode()</code>	The zip/postal code in the customer's address.
<code>getCountry()</code>	The country in the customer's address.

Table 7-5 ShoppingCart Accessor Methods/Attributes

Method/Attribute	Description
<code>getShoppingCartLineCollection()</code>	The individual lines in the shopping cart (that is, <code>ShoppingCartLine</code>).
<code>getTotal(int totalType)</code>	The total amount specified by the <code>totalType</code> parameter. The relevant parameter is: <code>ShoppingCartConstants.LINE_TAX</code> Note: The <code>getTotal()</code> 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.

7 Order Summary and Confirmation Services

Table 7-6 ShoppingCartLine Accessor Methods/Attributes

Method/Attribute	Description
<code>getQuantity()</code>	The quantity of the item.
<code>getProductItem()</code>	The product item in the shopping cart line.
<code>getUnitPrice()</code>	The current price for the item at the time it was added to the shopping cart. May be different from MSRP.
<code>getBaseTotal()</code>	The total before discounts.
<code>getDiscountPresentations()</code>	Returns an array list of <code>DiscountPresentation</code> 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 nowrap valign="top">
    <div class="tabletext">
      <code>%=shoppingCartLine.getProductItem().getKey().getIdentifier()%>
    </div>
  </td>
  <td valign="top">
    <div class="tabletext">
      <code>%=shoppingCartLine.getProductItem().getName()%>
    </div>
  </td>
  <td align="center" valign="top">
    <div class="tabletext">
      <code>%=WebflowJSPHelper.quantityFormat(shoppingCartLine.getQuantity()) %>
    </div>
  </td>
</tr>
```

```

<td align="right" nowrap valign="top">
  <div class="tabletext">
    <%=shoppingCartLine.getUnitPrice().getCurrency()%>
    <%=WebflowJSPHelper.priceFormat(shoppingCartLine.getUnitPrice().
      getValue())%>
  </div>
</td>
</tr><tr>

  <td colspan="5"><hr size="1"></td>

</tr>
</wl:repeat>

<tr>
  <td colspan="4" align="right">
    <div class="tabletext">Shipping & handling</div>
  </td>

  <td align="right" nowrap>
    <% Money shipping=shoppingCart.getTotal(ShoppingCartConstants.LINE_SHIPPING);
    %>

    <div class="tabletext">
      <%=shipping.getCurrency()%>
      <%=WebflowJSPHelper.priceFormat(shipping.getValue())%>
    </div>
  </td>

</tr><tr>

  <td colspan="4" align="right">
    <div class="tabletext">Total tax</div>
  </td>

  <td align="right" nowrap>
    <% Money tax=shoppingCart.getTotal(ShoppingCartConstants.LINE_TAX); %>
    <div class="tabletext">
      <%=tax.getCurrency()%>
      <%=WebflowJSPHelper.priceFormat(tax.getValue())%>
    </div>
  </td>

</tr><tr>

  <td colspan="4" align="right">
    <div class="tabletext"><b>Total due</b></div>
  </td>

```

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

The screenshot shows a web page for confirming an order. At the top, there is a navigation bar with links for 'Home', 'Search', 'View Cart', and 'Logout'. A red banner at the top right reads 'BEA WebLogic Commerce Server 3.5 Commerce Templates'. Below the navigation bar, there is a promotional banner for 'Routers!' with a 'Click here to see our full line of powerful Routers!' link. The main content area is titled 'Confirm Order' and includes a 'Welcome Demo Customer' message with links for 'View Profile' and 'Logout'. A 'Quick Look-up' section has a search box and a 'Find' button. A 'See Our Large Selection of Saws Here!' banner is also present. The order confirmation details include:

- Order Confirmation # 1** (Annotation 1)
- Will be billed to card:** xxxxxxxxxxxxx8484
- Will be shipped to:** Demo Customer, One Main Street, DENVER, CO-80212, United States
- Shipping Preferences:** Second Day Air, Ship all at once (Annotation 2)
- Order Summary Table:**

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

 (Annotation 3)

 A 'Continue shopping' button is located below the table. A note at the bottom states: '**' indicates discounts or adjustments associated with a particular item ID.

7 Order Summary and Confirmation Services

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 Personalization Server 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 Personalization Server 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 Commerce Server Directory Structure

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

```
%WL_COMMERCE_HOME%\server\webapps\wlcs\commerce\order\  
confirmorder.jsp (Windows)  
$WL_COMMERCE_HOME/server/webapps/wlcs/commerce/order/  
confirmorder.jsp (UNIX)
```

Tag Library Imports

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

```
<%@ taglib uri="weblogic.tld" prefix="wl" %>  
<%@ taglib uri="pipeline.tld" prefix="pipeline" %>  
<%@ 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 Personalization 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 following directory for the WebLogic Commerce Server Web application:

```
%WL_COMMERCE_HOME%\config\wlcsDomain\applications\wlcs\WEB-INF
(Windows)
$WL_COMMERCE_HOME/config/wlcsDomain/applications/wlcs/WEB-INF
(UNIX)
```

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.webflow.PipelineSessionConstants" %>
<%@ page import="com.beasys.commerce.axiom.units.*" %>
<%@ page import="com.beasys.commerce.axiom.contact.*" %>
<%@ page import="com.beasys.commerce.ebusiness.order.*" %>
<%@ page import="com.beasys.commerce.ebusiness.shipping.*" %>
<%@ page import="com.beasys.commerce.foundation.pipeline.*" %>
<%@ page import="com.beasys.commerce.webflow.*" %>
<%@ page import="com.beasys.commerce.ebusiness.catalog.*" %>
```

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.

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 Personalization Server 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 Personalization Server JSP tags, see “JSP Tag Reference” in the *Guide to Building Personalized Applications*.

Next, the `getPipelineProperty` 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
<code>PipelineSessionConstants.ORDER_VALUE</code>	List of <code>com.beasys.commerce.ebusiness.order.OrderValue</code>	List of the orders available for the customer.
<code>PipelineSessionConstants.SHIPPING_METHOD</code>	<code>com.beasys.commerce.ebusiness.shipping.ShippingMethodValue</code>	The method being used to ship the order.
<code>PipelineSessionConstants.CREDIT_CARD_KEY</code>	<code>java.lang.String</code>	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

```
<pipeline:getPipelineProperty
  propertyName="<%=PipelineSessionConstants.ORDER_VALUE%"
  returnName="orderValue"
  returnType="OrderValue"
  attributeScope="<%=PipelineConstants.REQUEST_SCOPE%" />

<pipeline:getPipelineProperty
  propertyName="<%=PipelineSessionConstants.SHIPPING_METHOD%"
  returnName="shippingMethodValue"
  returnType="com.beasys.commerce.ebusiness.shipping.ShippingMethodValue" />
```

Note: For more information on the `getPipelineProperty` 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

```
<table>
  <tr>
    <td>
      <um:getPropertyAsString propertyName="firstName" />
      <um:getPropertyAsString propertyName="lastName" />
    </td>
  </tr>
</table>
```

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

The data stored within the Pipeline session attributes (retrieved using the `getPipelineProperty` 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`.

7 Order Summary and Confirmation Services

Table 7-8 Address Accessor Methods/Attributes

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

Table 7-9 ShippingMethodValue Accessor Methods/Attributes

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

Table 7-10 OrderValue Accessor Methods/Attributes

Method/Attribute	Description
<code>createdDate</code>	The date the customer's order was created.
<code>identifier</code>	Key in the database for the order.
<code>getTotal(int totalType)</code>	The total amount specified by the <code>totalType</code> parameter. The relevant parameter is <code>OrderConstants.LINE_TAX</code> Note: The <code>getTotal()</code> method also allows you to combine different total types. For more information, see the Javadoc.
<code>orderLines</code>	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
<code>getProductIdentifier()</code>	The name (identifier) for the shopping cart item.
<code>getDescription()</code>	A description of the shopping cart item.
<code>getQuantity()</code>	The quantity of the shopping cart item.
<code>getUnitPrice()</code>	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">
    <div class="tabletext">
      <%=orderLine.getProductIdentifier()%>
    </div>
  </td>

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

7 Order Summary and Confirmation Services

```
</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">
    <%=orderLine.getUnitPrice().getCurrency()%>
    <%= WebflowJSPHelper.priceFormat(orderLine.getUnitPrice().getValue())%>
  </div>
</td>
</tr>
```

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	<code>com.beasys.commerce.ebusiness.order.pipeline.CommitOrderPC</code>
Description	Reads all the information about a customer's order from the Pipeline session and creates an <code>Order</code> entity bean. This is committed to the database in the <code>WLCS_ORDER</code> and <code>WLCS_ORDER_LINE</code> tables. The <code>OrderValue</code> object for the order is then stored in the Pipeline session.

7 Order Summary and Confirmation Services

Required Pipeline Session Attributes	<code>PipelineSessionConstants.USER_NAME</code> <code>PipelineSessionConstants.SHOPPING_CART</code> <code>PipelineSessionConstants.SPLITTING_PREFERENCE</code> <code>PipelineSessionConstants.SPECIAL_INSTRUCTIONS</code> <code>PipelineSessionConstants.ORDER_CONFIRMATION_NUMBER</code> <code>PipelineSessionConstants.SHIPPING_ADDRESS</code> <code>PipelineSessionConstants.ORDER_ADJUSTMENTS</code> <code>PipelineSessionConstants.SHIPPING_METHOD</code> <code>PipelineSessionConstants.DISCOUNT_IDS</code> <code>PipelineSessionConstants.GLOBAL_DISCOUNTS_IDS</code>
Updated Pipeline Session Attributes	<code>PipelineSessionConstants.ORDER_HANDLE</code> (Request scope) <code>PipelineSessionConstants.ORDER_VALUE</code> (Request scope) <code>PipelineSessionConstants.ORDER_SHIPPING_METHOD</code> (Request scope) <code>PipelineSessionConstants.PAYMENT_AUTHORIZATION_ACCOUNT</code>
Removed Pipeline Session Attributes	<code>PipelineSessionConstants.SHIPPING_METHOD</code>
Type	Java object
JNDI Name	None
Exceptions	<code>PipelineFatalException</code> , thrown when the required Pipeline session attributes are not available or if the shopping cart is empty.

ResetCheckoutPC

Class Name	<code>com.beasys.commerce.ebusiness.order.pipeline.ResetCheckoutPC</code>
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	com.bea.commerce.ebusiness.tracking.pipeline.PurchaseTrackerPC
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 <i>Guide to Events and Behavior Tracking</i> .
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

7 *Order Summary and Confirmation Services*

Type	Java Object
JNDI Name	None
Exceptions	None

8 Extending the Data Model

This chapter explains how to extend Managing Purchases and Processing Orders 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 Commerce Server 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 Managing Purchases and Processing Orders services are two core components of the WebLogic Commerce Server 3.5. 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 the WebLogic Commerce Server to provide more sophisticated and specialized commerce scenarios to meet your business needs. The WebLogic Commerce Server infrastructure supports use-case driven extensibility in the form of the Webflow and Pipelines. This infrastructure provides you with three forms of extensibility:

- You can rapidly modify the existing use-case flows by changing the Webflow and Pipeline configurations.
- You can customize use-cases by adding new input processors and Pipelines.
- You can implement new use-cases by defining new Webflows and Pipelines 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 WebLogic Commerce Server data model and schema. For example, you may want to customize the checkout process of your commerce site to collect a promotion code or gift coupon data, and 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 for the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline* of the WebLogic Commerce Server. While this chapter does not guarantee automatic compatibility of such extensions with future releases of the WebLogic Commerce Server, the approaches discussed in this chapter try to minimize potential problems, by leveraging the WebFlow/Pipeline infrastructure.

This chapter addresses the following:

- The WebLogic Commerce Server 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 WebLogic Commerce Server services (such as extensions for building a product catalog). You can periodically check the WebLogic Commerce Server documentation for future updates on how to extend other services.

Persistence Architecture

Before we go into the approaches for extending the WebLogic Commerce Server schema, consider the persistence architecture of WebLogic Commerce Server 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.

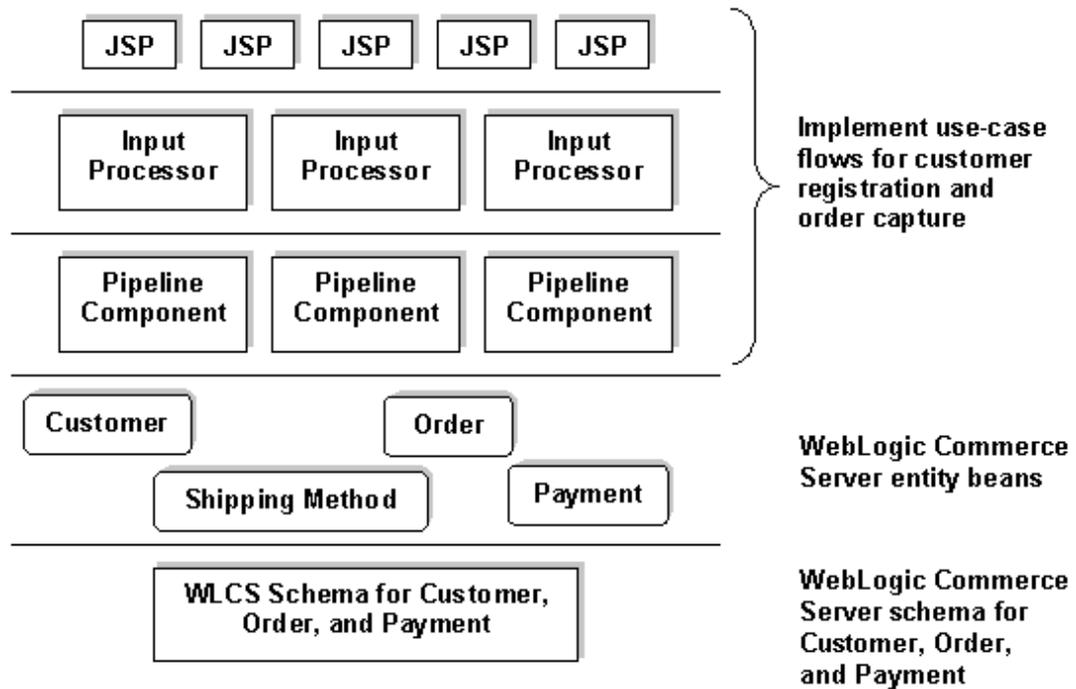


Figure 8-1 Persistence Architecture for Registering Customers and Managing Customer Services and Managing Purchases and Processing Orders Services

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

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

For persistence, these entities use the WebLogic Commerce Server tables discussed in the Chapter 10, “The Order Processing Database Schema.”

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

Table	Description
Entity: <code>com.beasys.commerce.ebusiness.customer.Customer</code>	
WLCS_CUSTOMER	Customer description
WLCS_CREDIT_CARD	Credit cards
WLCS_SHIPPING_ADDRESS	Shipping address
Entity: <code>com.beasys.commerce.ebusiness.order.Order</code>	
WLCS_ORDER	Order description
WLCS_ORDER_LINES	Order lines
Entity: <code>com.beasys.commerce.ebusiness.payment.PaymentTransaction</code>	
WLCS_TRANSACTION	CyberCash transaction description
WLCS_TRANSACTION_ENTRY	CyberCash transaction entries
Entity: <code>com.beasys.commerce.ebusiness.shipping.ShippingMethod</code>	
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 Commerce Server, run-time attributes can be added on these entities without having to change the underlying database schema.

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

The Customer entity of the WebLogic Commerce Server is a component that relies on the Unified User Profile (UUP) technology of the WebLogic Commerce Server. A UUP for customer data allows the abstraction of a customer to be seamlessly integrated into the WebLogic Personalization Server. Apart from personalization, this approach allows you to use the user management tools of the WebLogic Personalization Server to administer customer data, and maps the customer identity into a WebLogic Personalization Server-administered groups and the RDMBS security realm. For more information on unified user profiles, see “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 `com.beasys.commerce.ebusiness.customer.pipeline` package. To add attributes to the customer data, the WebLogic Commerce Server Registration Package provides an abstract Pipeline component `com.beasys.commerce.ebusiness.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 Commerce Server specifies the customer entity as a UUP of type "WLCS_Customer." The `<um:getPropertyAsString>` tag is one of the user management tags to extract user attributes in JSP pages. For more documentation on user management tags, 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. The WebLogic Commerce Server persists run-time attributes in tables that are internal to WebLogic Commerce Server. Consequently, you cannot execute SQL level operations on such data.

Adding Run-Time Attributes to Other Entities

For the entities in Managing Purchases and Processing Orders services such as `com.beasys.commerce.ebusiness.shipping.Order`, `com.beasys.commerce.ebusiness.shipping.PaymentTransaction`, and `com.beasys.commerce.ebusiness.shipping.ShippingMethod`, there exists a similar mechanism for adding run-time attributes. All the entities in the Managing Purchases and Processing Orders 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 WebLogic Commerce Server schema:

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

Both these drivers manifest in the following:

- Modifying (or sometimes adding) the templates to render and/or collect additional data from the user interface.
- Modifying the WebFlow to change the flow of user interaction.
- Extending the WebLogic Commerce Server schema.

Note: Almost all the data in the Managing Purchases and Processing Orders 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 WebLogic Commerce Server and your back-end applications may depend on this data. Any extension to the schema of the Managing Purchases and Processing Orders 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 Commerce Server 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 Commerce Server Schema

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

Figure 8-2 Extending the Data Model

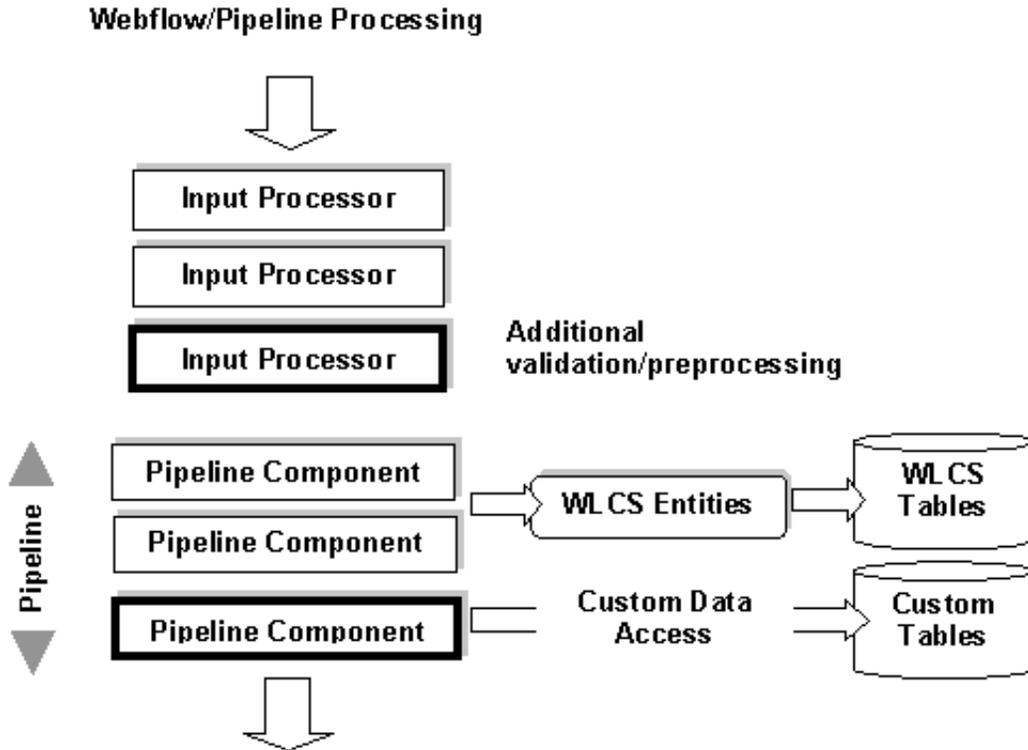


Figure 8-2 demonstrates how a given WebFlow/Pipeline processing can be modified to process additional data, without modifying existing input processors and Pipeline components. In Figure 8-2, the blocks with heavy borders are new input processors and Pipeline components inserted to process the additional data. While the WebLogic Commerce Server Pipeline components manage the WebLogic Commerce Server data via the WebLogic Commerce Server entities, the new Pipeline component in the Pipeline may directly access the data via plain JDBC, or indirectly via another layer of custom entity beans. Alternatively, the new Pipeline component may also delegate this data access to legacy data access mechanisms.

As we shall discuss in a later section, depending on whether the additional data should be processed 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 WebLogic Commerce Server tables for such attributes, we recommend against doing so, as it could lead to compatibility issues and potential name collision issues with future releases of WebLogic Commerce Server.

Step 2: Modify corresponding JSP templates.

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

Step 3: Implement new input processor.

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

Step 4: Include the new input processor.

Modify the `webFlow.properties` to include the new input processor.

Step 5: Implement a new Pipeline component.

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

Step 6: Obtain a database connection.

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

Step 7: Include the new Pipeline component.

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

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

Adding Attributes Against the `WLCS_ORDER_LINE` Table

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

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

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

Step 1: Design a new table.

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

Step 2: Modify the corresponding JSP template.

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

Step 3: Implement a new input processor.

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

Step 4: Include the new input processor.

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

Step 5: Implement a new Pipeline component.

Implement a new Pipeline component to extract the additional data from the Pipeline session, and write to the new tables. However, since the primary key for the `WLCS_ORDER_LINE` table is internal to the WebLogic Commerce Server, 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, orderId);
    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 performed, 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>
    com.beasys.commerce.ebusiness.order.Order
  </ejb-name>
  <persistence-descriptor>
    <is-modified-method-name>
      isModified
    </is-modified-method-name>
    <delay-updates-until-end-of-tx>
      false
    </delay-updates-until-end-of-tx>
  </persistence-descriptor>
  <reference-descriptor>
    .
  </reference-descriptor>
  <enable-call-by-reference>true</enable-call-by-
    reference>
  <jndi-name>
    com.beasys.commerce.ebusiness.order.Order
  </jndi-name>
</weblogic-enterprise-bean>
```

- Jar the contents of the temporary directory, and run the 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 `com.beasys.commerce.ebusiness.customer.webflow.UpdatePaymentInfoIP` to add the `mapKey` attribute to the `PipelineSession`. Similarly, in the case of shipping address, add the `ShippingAddressMapKey` attribute to the `PipelineSession` in the `com.beasys.commerce.ebusiness.customer.webflow.UpdateShippingInfoIP`.

Step 4: Implement new input processor.

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

Step 5: Implement new Pipeline component.

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

- Jar the contents of the temporary directory, and run the 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 the WebLogic Commerce Server.

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.

8 *Extending the Data Model*

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 Commerce Server.

The Order and Payment Management pages ship as part of the 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 Commerce Server 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
- Tailoring Order Status to Your Business
- 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 Commerce Server Administration Tools

Before you can use the Order and Payment Management pages, you need to start the server and load the WebLogic Commerce Server Administration Tools page in your Web browser.

To start the server on a Windows system, you can either:

- Run `StartCommerce.bat` from the command line in the `WL_COMMERCE_HOME` directory, where `WL_COMMERCE_HOME` is the directory where you installed the WebLogic Commerce Server.
- From the Start menu, select Programs → WebLogic Commerce Server 3.5 → Start WebLogic Commerce Server.

To start the server on a UNIX system, run `StartCommerce.sh` from the command line in the `WL_COMMERCE_HOME` directory, where `WL_COMMERCE_HOME` is the directory where you installed the WebLogic Commerce Server.

The Administration Tools page (shown in Figure 9-1) is an entry page into all of the available WebLogic Commerce Server Administration Tools. To load this page, use one of the following methods:

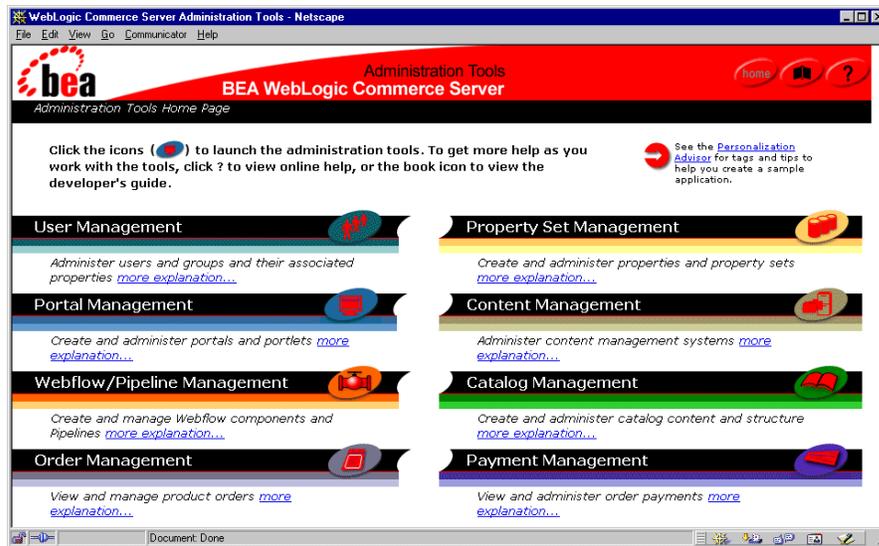
- Specify the URL for the page (<http://localhost:7501/tools/application/admin>) in your Web browser.

Starting the WebLogic Commerce Server Administration Tools

Note: If you need to perform an administrative task on another node in the cluster, also specify the machine such as `http://elvis:7501/tools/application/admin`.

- From the Start menu on a Windows system, select Programs → WebLogic Commerce Server 3.5 → Administration Tool.

Figure 9-1 WebLogic Commerce Server Administration Tools Page



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

Administration Tools home ?

BEA WebLogic Commerce Server

Order Management

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 :

search search search

back

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

The screenshot displays the BEA WebLogic Commerce Server interface. At the top, there is a navigation bar with the BEA logo, 'Administration Tools', and 'BEA WebLogic Commerce Server'. Below this, a sub-header reads 'Order Management' and 'Obtained '2' orders for customer democustomer'. The main section is titled 'Order Search' and includes instructions: 'Use this page to search for Orders. Click on order id to edit it.' There are three search filters: 'Search By Customer' with a 'Customer ID' input field and a 'search' button; 'Search By Order Identifier' with an 'Order ID' input field and a 'search' button; and 'Search By Date Range' with 'From' and 'To' input fields, each with a calendar icon, and a 'search' button. Below the filters is an 'Order List' table with the following data:

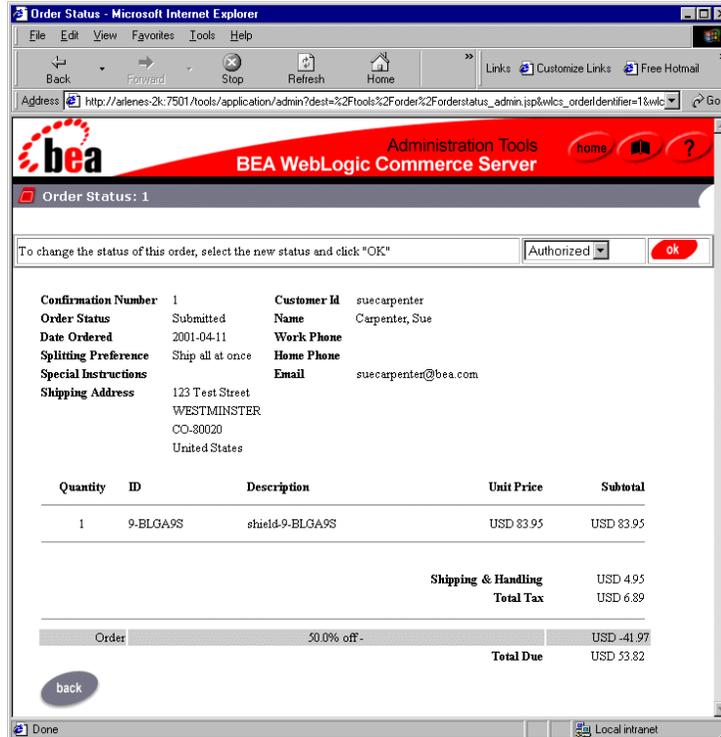
Identifier	Create Date	Price	Customer Id
1	2000-09-15	54.18	democustomer
2	2000-09-18	93.83	democustomer

At the bottom of the table, there is a 'back' button.

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.

9 Using the Order and Payment Management Pages

Figure 9-4 Sample Order Status Page



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

The screenshot displays the BEA WebLogic Commerce Server interface. At the top, there is a navigation bar with the BEA logo, 'Administration Tools', and 'BEA WebLogic Commerce Server'. Below this, a sub-header reads 'Order Management' and 'Obtained '1' orders for order 2'. The main section is titled 'Order Search' and includes instructions: 'Use this page to search for Orders. Click on order id to edit it.' There are three search filters: 'Search By Customer' with a 'Customer ID' input field and a 'search' button; 'Search By Order Identifier' with an 'Order ID' input field and a 'search' button; and 'Search By Date Range' with 'From' and 'To' input fields, each with a calendar icon, and a 'search' button. Below the filters is a table titled 'Order List' with the following data:

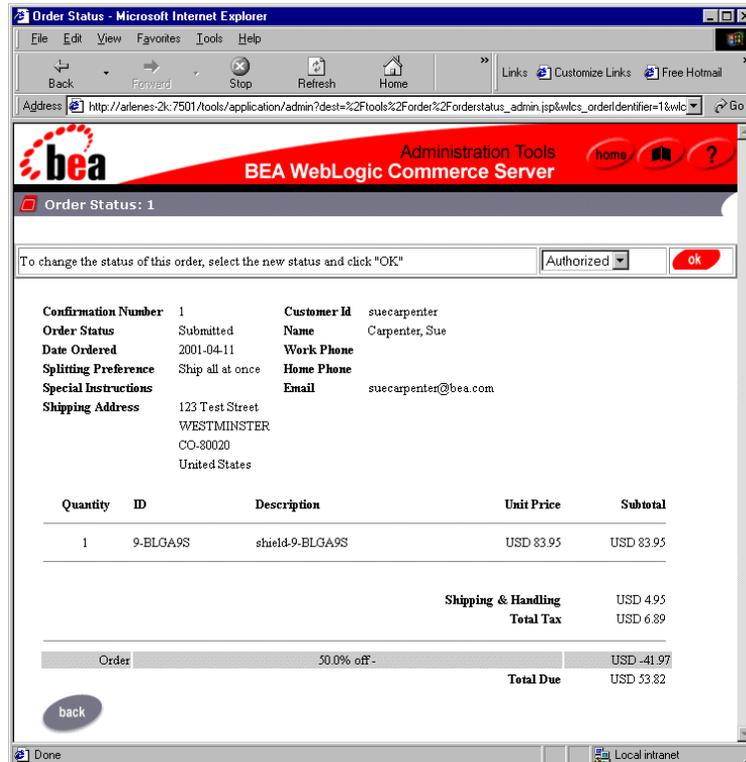
Identifier	Create Date	Price	Customer Id
2	2000-09-18	93.83	democustomer

At the bottom of the table, there is a 'back' button.

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.

9 Using the Order and Payment Management Pages

Figure 9-6 Sample Order Status Page

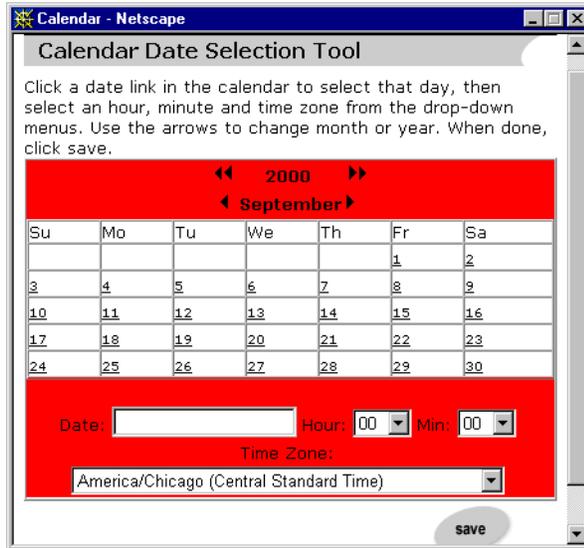


Click the Back button at the bottom of the Order Status page to return to the Order Management search/results page.

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



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

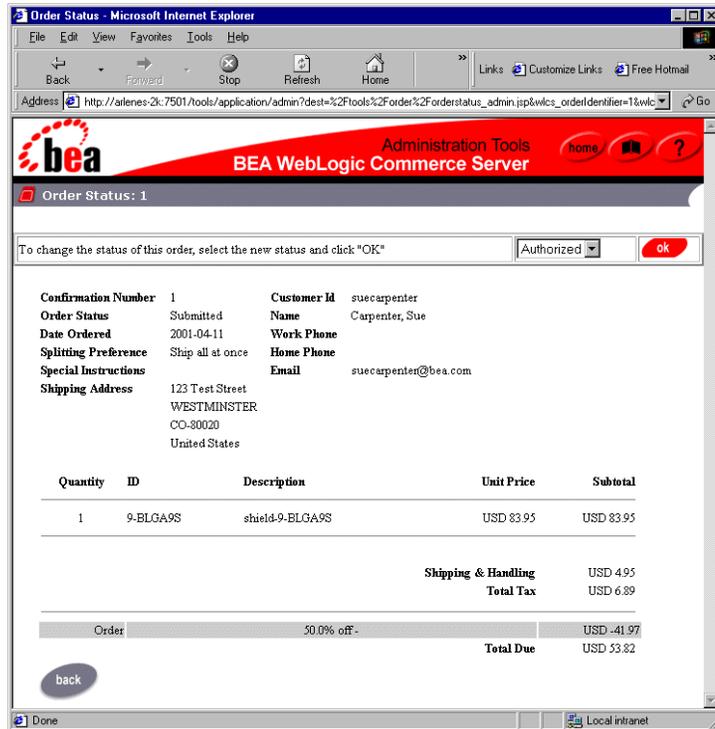
The screenshot displays the BEA WebLogic Commerce Server Administration Tools interface. At the top, there is a navigation bar with the BEA logo, the text "Administration Tools BEA WebLogic Commerce Server", and buttons for "home", a folder icon, and a question mark. Below this is a "Order Management" section with a sub-header "Order Search" and a message: "Obtained '1' orders between 2000-09-18 and 2000-09-21". The "Order Search" section includes three search criteria: "Search By Customer" with a "Customer ID" input field and a "search" button; "Search By Order Identifier" with an "Order ID" input field and a "search" button; and "Search By Date Range" with "From" and "To" input fields, each with a calendar icon, and a "search" button. Below the search options is an "Order List" table with the following data:

Identifier	Create Date	Price	Customer Id
2	2000-09-18	93.83	democustomer

At the bottom of the screenshot, there is a "back" button.

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



Click the Back button at the bottom of the Order Status page to return to the Order Management search/results page.

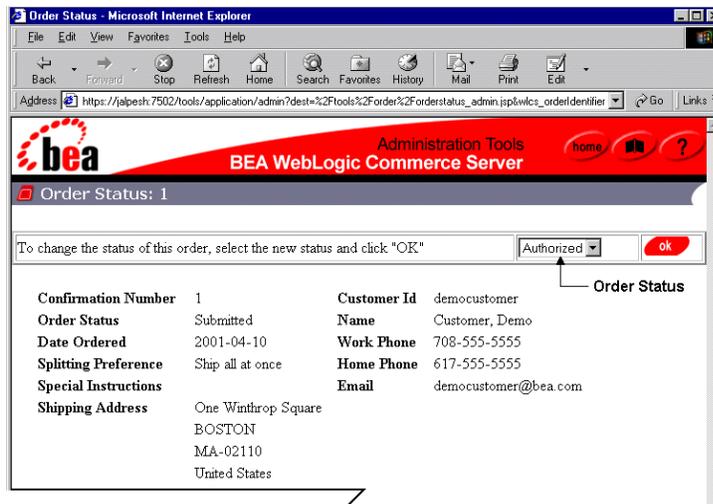
Updating Order Status

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. For information on how to customize order status for your business, see “Tailoring Order Status to Your Business” on page 9-13.

Figure 9-10 Sample Order Status Page



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

Tailoring Order Status to Your Business

Note: Before changing the Order Status list and the order status as described in this section, please read “Changing Order Status” on page 9-12.

To tailor the Order Status list and the order status sequence to fit your business, you need to edit the `orderstatus.xml` document. You can find the `orderstatus.xml` file in the following location, where `WL_COMMERCE_HOME` is the directory in which you installed the WebLogic Commerce Server:

`%WL_COMMERCE_HOME%\classes\orderstatus.xml` (Windows)
`$WL_COMMERCE_HOME/classes/orderstatus.xml` (UNIX)

The `stateMachine` shown in Listing 9-1, shows the items of order status. When a customer places an order that order’s status is set to the `firstState` in the `stateMachine`. You can edit the nodes to fit your business. For example, you might want to add `Pending` to the list. You can also change the sequence of the status states with the transition tags. For example, the sequence shown in Listing 9-1 is `Authorized → Shipped → Paid → Shipped`. You could change the sequence to `Authorized → Paid → Shipped`. A sequence ends when a state is not associated with a transition.

Note: Only one word is allowed for the name of the node.

Listing 9-1 State Machine Sample

```
<stateMachine>
```

9 Using the Order and Payment Management Pages

```
<firstState>
  <state>
    <stateName>Submitted</stateName>
    <transition>
      <event> Authorized </event>
    </transition>
    <transition>
      <event> Cancelled </event>
    </transition>
    <transition>
      <event> Rejected </event>
    </transition>
  </state>
</firstState>

<state>
...
  <stateName> Authorized </stateName>
  <transition>
    <event> Paid </event>
  </transition>
</state>
...

<state>
  <stateName> Shipped </stateName>
  <transition>
    <event> Paid </event>
  </transition>
</state>
...

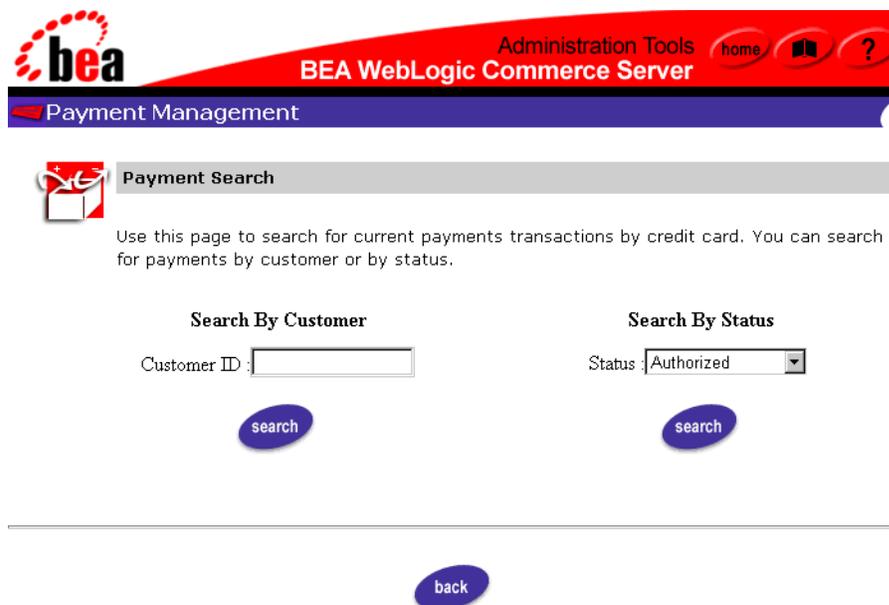
<state>
  <stateName> Paid </stateName>
</state>
...
</statemachine>
```

Note: For any changes that you make in the `orderstatus.xml` file to take effect, you need to restart your server.

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



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

The screenshot displays the BEA WebLogic Commerce Server Administration Tools interface. At the top, there is a red navigation bar with the BEA logo, "Administration Tools", and buttons for "home", a book icon, and a question mark. Below this is a blue bar labeled "Payment Management". A red message states: "Obtained '2' transactions for customer democustomer".

The main content area is titled "Payment Search" and includes a sub-header with a mail icon. Below the sub-header, there is a descriptive text: "Use this page to search for current payments transactions by credit card. You can search for payments by customer or by status."

There are two search sections:

- Search By Customer:** A text input field labeled "Customer ID:" followed by a blue "search" button.
- Search By Status:** A dropdown menu labeled "Status:" with "Authorized" selected, followed by a blue "search" button.

Below the search sections is a blue header for "Payment Transaction History". Underneath is a table with the following data:

Date	Transaction Id	Amount	Status	Credit Card	New Amount	
2000-09-15	1969038797504	54.18	MarkedForSettle	xxxxxxxxxxxx1111	<input type="text"/>	Settle
2000-09-18	2969291417253	93.83	Retry	xxxxxxxxxxxx1111		Authorize

At the bottom of the table area, there is a blue "back" button.

For a detailed explanation of the Payment Transaction History fields and further payment management activities, refer to “Authorizing, Capturing, and Settling Payments” on page 9-19.

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

The screenshot shows the BEA WebLogic Commerce Server Administration Tools interface. The top navigation bar includes 'home', a book icon, and a question mark icon. The main header is 'Payment Management' with a sub-header 'Obtained '1' transactions for status Retry'. Below this is a 'Payment Search' section with a brief description: 'Use this page to search for current payments transactions by credit card. You can search for payments by customer or by status.' There are two search options: 'Search By Customer' with a 'Customer ID' input field and a 'search' button, and 'Search By Status' with a 'Status' dropdown menu set to 'Authorized' and a 'search' button. Below the search options is a 'Payment Transaction History' table with the following data:

Date	Transaction Id	Amount	Status	Credit Card	New Amount
2000-09-18	2969291417253	93.83	Retry	xxxxxxxxxx1111	<input type="button" value="Authorize"/>

Below the table is a 'back' button.

For a detailed explanation of the Payment Transaction History fields and further payment management activities, refer to “Authorizing, Capturing, and Settling Payments” on page 9-19.

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

Administration Tools home ?

BEA WebLogic Commerce Server

Payment Management

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.

Search By Customer Search By Status

Customer ID : Status : Authorized

Payment Transaction History

Date	Transaction Id	Amount	Status	Credit Card	New Amount
2000-09-15	1969038797504	54.18	MarkedForSettle	xxxxxxxxxxxx1111	<input type="text"/> <input type="button" value="Settle"/>
2000-09-18	2969291417253	93.83	Retry	xxxxxxxxxxxx1111	<input type="button" value="Authorize"/>

Pressing this button will cause the WebLogic Commerce Server product to connect to the CyberCash (payment) server, 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.

Note: For more information about configuring the server to defer authorization of payments, see “Configuration Activities for Using CyberCash” on page 6-27.

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.

Note: For more information about the different payment models, see “Payment Models” on page 6-29.

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 Commerce Server 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 Commerce Server 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.

9 *Using the Order and Payment Management Pages*

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.

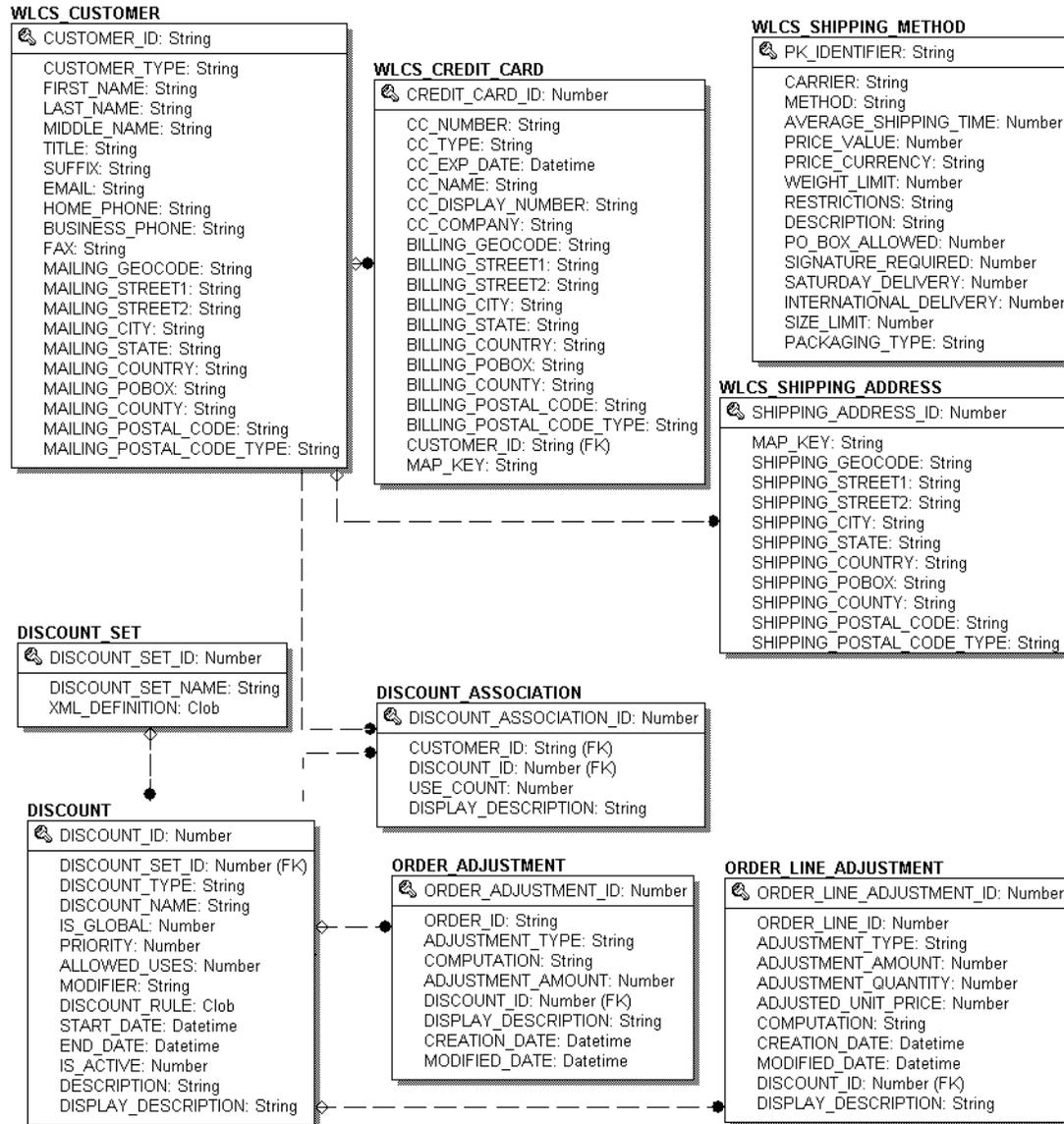
This topic includes the following sections:

- 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 Commerce Server order and discount tables in the Commerce database. See the subsequent sections in this chapter for information about the data type syntax.

The Entity-Relation Diagram



Explanations for the columns in each table are provided in the remainder of this topic.

List of Tables Comprising the Order Processing Schema

The WebLogic Commerce Server order management system is comprised of the following tables:

- The WLCS_COUNTRY Database Table
- The WLCS_CREDIT_CARD Database Table
- The WLCS_CURRENCY Database Table
- The WLCS_CUSTOMER Database Table
- The DISCOUNT Database Table
- The DISCOUNT_ASSOCIATION Database Table
- The DISCOUNT_SET Database Table
- The WLCS_ORDER Database Table
- The WLCS_ORDER_LINE Database Table
- The ORDER_ADJUSTMENT Database Table
- The ORDER_LINE_ADJUSTMENT 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 WebLogic Commerce Server. However, they will be in place in future versions of WebLogic Commerce Server and we want you to be aware of these relationships now.

The WLCS_CREDIT_CARD Database Table

Table 10-1 describes the metadata for the WebLogic Commerce Server WLCS_CREDIT_CARD table. This table is used to store information related to a customer’s credit card(s) in the order processing database.

The Primary Key is CREDIT_CARD_ID.

Table 10-1 WLCS_CREDIT_CARD Table Metadata

Column Name	Data Type	Description and Recommendations
CREDIT_CARD_ID	NUMBER (15)	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)	The customer’s credit card number. This is encrypted if <code>is. encryption. enable</code> is set to <code>true</code> in the <code>weblogiccommerce. properties</code> file.
CC_TYPE	VARCHAR (20)	The customer’s credit card type, such as VISA or MasterCard.
CC_EXP_DATE	DATE	The expiration date on the customer’s credit card.

10 The Order Processing Database Schema

Table 10-1 WLCS_CREDIT_CARD Table Metadata (Continued)

Column Name	Data Type	Description and Recommendations
CC_NAME	VARCHAR (50)	The credit card holder's name.
CC_DISPLAY_NUMBER	VARCHAR (20)	The version of the credit card number that is displayed (all Xs except last 4-digits).
CC_COMPANY	VARCHAR (50)	The name of the credit card company.
BILLING_GEOCODE	VARCHAR (2)	The code used by the TAXWARE system to identify taxes for the order based on jurisdiction.
BILLING_STREET1	VARCHAR (30)	The first line in the customer's billing address.
BILLING_STREET2	VARCHAR (30)	The second line in the customer's billing address.
BILLING_CITY	VARCHAR (30)	The city in the customer's billing address.
BILLING_STATE	VARCHAR (40)	The state in the customer's billing address.
BILLING_COUNTRY	VARCHAR (40)	The country in the customer's billing address.
BILLING_POBOX	VARCHAR (30)	The post office box in the customer's billing address.
BILLING_COUNTY	VARCHAR (50)	The county in the customer's billing address.
BILLING_POSTAL_CODE	VARCHAR (10)	The postal (ZIP) code in the customer's billing address.
BILLING_POSTAL_CODE_TYPE	VARCHAR (10)	Format or type of postal code, generally determined by country (such as ZIP code in the United States).
CUSTOMER_ID	VARCHAR (20)	A unique identifier for the customer.
MAP_KEY	VARCHAR (20)	Key that maps multiple credit cards with a single customer.

The WLCS_COUNTRY Database Table

Table 10-2 describes the metadata for the WebLogic Commerce Server WLCS_COUNTRY table. This is a reference table and contains pertinent information regarding each country around the world.

The primary key is COUNTRY_ID.

This table holds information that pertains to all of the various currencies used throughout the world.

Note: The WLCS_COUNTRY feature has not been implemented at this time and, therefore, this table is not being used/populated.

Table 10-2 WLCS_COUNTRY Table Metadata

Column Name	Data Type	Description and Recommendations
COUNTRY_ID	VARCHAR(3)	PK—a unique textual identifier associated with each country, such as a country code.
COUNTRY_ABBR2	VARCHAR(2)	A second textual identifier.
CURRENCY_ID	NUMERIC(3)	An ID for the form of currency used in transactions with citizens of this country.
COUNTRY_NAME	VARCHAR(50)	The formal name of the country.

The WLCS_CURRENCY Database Table

Table 10-3 describes the metadata for the WebLogic Commerce Server WLCS_CURRENCY table. This table holds info pertaining to all of the various currencies used throughout the world.

The primary key is CURRENCY_ID.

Note: The WLCS_CURRENCY feature has not been implemented at this time and, therefore, this table is not being used/populated.

10 The Order Processing Database Schema

Table 10-3 WLCS_CURRENCY Table Metadata

Column Name	Data Type	Description and Recommendations
CURRENCY_ID	NUMBER (3)	PK—a unique numeric ID.
CURRENCY_ABBR	VARCHAR (3)	A textual abbreviation for the currency.
CURRENCY_NAME	VARCHAR (50)	The name of the currency.

The WLCS_CUSTOMER Database Table

Table 10-4 describes the metadata for the WebLogic Commerce Server 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 10-4 WLCS_CUSTOMER Table Metadata

Column Name	Data Type	Description and Recommendations
CUSTOMER_ID	VARCHAR (20)	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)	A label for the customer (such as preferred, standard, or business).
FIRST_NAME	VARCHAR (30)	The customer's first name.
LAST_NAME	VARCHAR (30)	The customer's last name.
MIDDLE_NAME	VARCHAR (30)	The customer's middle name.
TITLE	VARCHAR (10)	The customer's preferred title, such as Mr., Mrs., or Ms.
SUFFIX	VARCHAR (10)	The customer's preferred suffix, such as Jr.or Sr.
EMAIL	VARCHAR (80)	The customer's email address.

Table 10-4 WLCS_CUSTOMER Table Metadata (Continued)

Column Name	Data Type	Description and Recommendations
HOME_PHONE	VARCHAR (15)	The customer's home phone number.
BUSINESS_PHONE	VARCHAR (20)	The customer's business phone number.
FAX	VARCHAR (15)	The customer's fax number.
MAILING_GEOCODE	VARCHAR (2)	The code used by the TAXWARE system to identify taxes for the order based on jurisdiction.
MAILING_STREET1	VARCHAR (30)	The first line in the customer's street address.
MAILING_STREET2	VARCHAR (30)	The second line in the customer's street address.
MAILING_CITY	VARCHAR (30)	The city in the customer's address.
MAILING_STATE	VARCHAR (40)	The state in the customer's address.
MAILING_COUNTRY	VARCHAR (40)	The country in the customer's address.
MAILING_POBOX	VARCHAR (30)	The post office box in the customer's address.
MAILING_COUNTY	VARCHAR (50)	The county in the customer's address.
MAILING_POSTAL_CODE	VARCHAR (10)	The postal (ZIP) code in the customer's address.
MAILING_POSTAL_CODE_TYPE	VARCHAR (10)	Format or type of postal code, generally determined by country (such as ZIP code in the United States).

The DISCOUNT Database Table

Table 10-5 describes the metadata for the WebLogic Commerce Server DISCOUNT table. This table stores stores one or more discount records for every DISCOUNT_SET record.

The Primary Key is DISCOUNT_ID.

10 The Order Processing Database Schema

Table 10-5 DISCOUNT

Column Name	Data Type	Description and Recommendations
DISCOUNT_ID	NUMBER (15)	PK—a unique, system-generated number to be used as the record ID.
DISCOUNT_SET_ID	NUMBER (15)	FK—foreign key to the DISCOUNT_SET table.
DISCOUNT_TYPE	VARCHAR (10)	The type of discount offered. It is used for an <i>order</i> or for an <i>order line item</i> .
DISCOUNT_NAME	VARCHAR (254)	The name of the discount.
IS_GLOBAL	NUMBER (1)	A flag showing whether or not this discount can be used globally.
PRIORITY	NUMBER (3)	The level of priority this discount has over other discounts.
ALLOWED_USERS	NUMBER (10)	The number of times the discount may be used.
MODIFIER	CLOB	Describes the actual discount to be applied. This is XML.
DISCOUNT_RULE	CLOB	The method used to select items for discount. This is XML.
START_DATE	DATE	The starting date and time of the discount
END_DATE	DATE	The ending date and time of the discount.
IS_ACTIVE	NUMBER (1)	A flag that determines whether the discount is active or not. Active=1, Not active=0
DESCRIPTION	VARCHAR (254)	The discount description.
DISPLAY_DESCRIPTION	VARCHAR (254)	The discount description used for display purposes only.

The DISCOUNT_ASSOCIATION Database Table

Table 10-6 describes the metadata for the WebLogic Commerce Server DISCOUNT_ASSOCIATION table. This table associates each customer with a discount and maintains information regarding the times the customer has used each discount.

The primary key is DISCOUNT_ASSOCIATION_ID.

Table 10-6 DISCOUNT_ASSOCIATION

Column Name	Data Type	Description and Recommendations
DISCOUNT_ASSOCIATION_ID	NUMBER (15)	PK—a unique, system-generated number to be used as the record ID.
CUSTOMER_ID	NUMBER (15)	FK—foreign key to the DISCOUNT_SET table.
DISCOUNT_ID	NUMBER (15)	FK—foreign key to the DISCOUNT_SET table.
USE_COUNT	NUMBER (10)	The number of times the discount has been used.
DISPLAY_DESCRIPTION	VARCHAR (254)	The discount description used for display purposes only.

The DISCOUNT_SET Database Table

Table 10-7 describes the metadata for the WebLogic Commerce Server DISCOUNT_SET table. This table is used to establish a group of discounts as a set.

The primary key is DISCOUNT_SET_ID.

Table 10-7 DISCOUNT_SET

Column Name	Data Type	Description and Recommendations
DISCOUNT_SET_ID	NUMBER (15)	PK—a unique, system-generated number to be used as the record ID.

10 The Order Processing Database Schema

Table 10-7 DISCOUNT_SET (Continued)

Column Name	Data Type	Description and Recommendations
DISCOUNT_SET_NAME	VARCHAR (50)	The name of the discount set.
XML_DEFINITION	CLOB	This is XML.

The WLCS_ORDER Database Table

Table 10-8 describes the metadata for the WebLogic Commerce Server WLCS_ORDER table. This table is used to store information about a customer's specific order in the order processing database.

Note: The WebLogic Commerce Server product does not populate the SHIPPING_AMOUNT, SHIPPING_CURRENCY, PRICE_AMOUNT, or PRICE_CURRENCY columns.

The primary key is ORDER_ID.

Table 10-8 WLCS_ORDER Table Metadata

Column Name	Data Type	Description and Recommendations
ORDER_ID	VARCHAR (20)	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)	A unique identifier for the customer.
TRANSACTION_ID	VARCHAR (25)	A unique identifier for the transaction.
STATUS	VARCHAR (20)	The status of the order.
ORDER_DATE	DATE	The date the order was placed.
SHIPPING_METHOD	VARCHAR (40)	The method by which the order is to be shipped.
SHIPPING_AMOUNT	NUMBER (16 , 4)	The shipping amount for the order.
SHIPPING_CURRENCY	VARCHAR (10)	The currency associated with the shipping amount.

Table 10-8 WLCS_ORDER Table Metadata (Continued)

Column Name	Data Type	Description and Recommendations
PRICE_AMOUNT	NUMBER (16 , 4)	The price of the order.
PRICE_CURRENCY	VARCHAR (10)	The currency associated with the price.
SHIPPING_GEOGODE	VARCHAR (2)	The code used by the TAXWARE system to identify taxes for the order based on jurisdiction.
SHIPPING_STREET1	VARCHAR (30)	The first line in the customer's shipping address.
SHIPPING_STREET2	VARCHAR (30)	The second line in the customer's shipping address.
SHIPPING_CITY	VARCHAR (30)	The city in the customer's shipping address.
SHIPPING_STATE	VARCHAR (40)	The state in the customer's shipping address.
SHIPPING_COUNTRY	VARCHAR (40)	The country in the customer's shipping address.
SHIPPING_POBOX	VARCHAR (30)	The post office box in the customer's shipping address.
SHIPPING_COUNTY	VARCHAR (50)	The county in the customer's shipping address.
SHIPPING_POSTAL_CODE	VARCHAR (10)	The postal (ZIP) code in the customer's shipping address.
SHIPPING_POSTAL_CODE_TYPE	VARCHAR (10)	Format or type of postal code, generally determined by country, such as ZIP code in the United States.
SPECIAL_INSTRUCTIONS	VARCHAR (254)	Any special shipping instructions associated with the order.
SPLITTING_PREFERENCE	VARCHAR (254)	The splitting preferences for the customer's order.
ORDER_SUBTOTAL	NUMBER (16 , 4)	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 10-9 describes the metadata for the WebLogic Commerce Server 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.

The Primary Key is ORDER_LINE_ID.

Table 10-9 WLCS_ORDER_LINE Table Metadata

Column Name	Data Type	Description and Recommendations
ORDER_LINE_ID	NUMBER (15)	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)	The quantity of the item in the shopping cart.
PRODUCT_ID	VARCHAR (40)	An identification number for the item in the shopping cart.
TAX_AMOUNT	NUMBER (16 , 4)	The tax amount for the order.
TAX_CURRENCY	VARCHAR (10)	The currency associated with the tax amount.
SHIPPING_AMOUNT	NUMBER (16 , 4)	The shipping amount for the order.
SHIPPING_CURRENCY	VARCHAR (10)	The currency associated with the shipping amount.
UNIT_PRICE_AMOUNT	NUMBER (16 , 4)	The unit price amount for the item.
UNIT_PRICE_CURRENCY	VARCHAR (10)	The currency associated with the unit price.
MSRP_AMOUNT	NUMBER (16 , 4)	The MSRP amount for the item.
MSRP_CURRENCY	VARCHAR (10)	The currency associated with the MSRP amount.
DESCRIPTION	VARCHAR (255)	The name of the item that is part of the order.
ORDER_ID	VARCHAR (20)	A unique identifier for the order.

The ORDER_ADJUSTMENT Database Table

Table 10-10 describes the metadata for the WebLogic Commerce Server 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 1/1/01 and 1/31/01.)

The Primary Key is ORDER_ADJUSTMENT_ID.

Table 10-10 ORDER_ADJUSTMENT

Column Name	Data Type	Description and Recommendations
ORDER_ADJUSTMENT_ID	NUMBER (15)	PK—a unique, system-generated number to be used as the record ID.
ORDER_ID	NUMBER (15)	FK—foreign key to the DISCOUNT_SET table.
ADJUSTMENT_TYPE	NUMBER (15)	FK—foreign key to the DISCOUNT_SET table.
COMPUTATION	NUMBER (10)	The number of times the discount has been used.
ADJUSTMENT_AMOUNT	VARCHAR (254)	The discount description used for display purposes only.
DISCOUNT_ID	NUMBER (15)	FK—foreign key to the DISCOUNT table.
DISPLAY_DESCRIPTION	VARCHAR (254)	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	The date and time the order adjustment was created.
MODIFIED_DATE	DATE	The date and time the order adjustment record was last modified.

The ORDER_LINE_ADJUSTMENT Database Table

Table 10-11 describes the metadata for the WebLogic Commerce Server 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”).

The Primary Key is ORDER_LINE_ADJUSTMENT_ID.

Table 10-11 ORDER_LINE_ADJUSTMENT Table Metadata

Column Name	Data Type	Description and Recommendations
ORDER_LINE_ADJUSTMENT_ID	NUMBER (15)	PK—a unique, system-generated number to be used as the record ID.
ORDER_LINE_ID	NUMBER (15)	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)	The type of adjustment (credit or debit.)
ADJUSTMENT_AMOUNT	NUMBER (16 , 4)	The dollar amount of the adjustment.
ADJUSTMENT_QUANTITY	NUMBER (16 , 4)	The quantity amount for the adjustment.
ADJUSTED_UNIT_PRICE	NUMBER (16 , 4)	The adjusted unit price of the specific line item.
COMPUTATION	VARCHAR (254)	The computation for determining ADJUSTED_UNIT_PRICE.
CREATION_DATE	DATE	The date and time the adjustment record was created.
MODIFIED_DATE	DATE	The date and time the adjustment record was last modified.
DISCOUNT_ID	NUMBER (15)	FK—a foreign key to the discount used from the DISCOUNT table.
DISPLAY_DESCRIPTION	VARCHAR (254)	The adjustment description used for display purposes.

The WLCS_SAVED_ITEM_LIST Database Table

Table 10-12 describes the metadata for the WebLogic Commerce Server 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.

Table 10-12 WLCS_SAVED_ITEM_LIST Table Metadata

Column Name	Data Type	Description and Recommendations
CUSTOMER_ID	VARCHAR(20)	A unique identifier for the customer.
SKU	VARCHAR(40)	A unique identifier (the Stock Keeping Unit or SKU) for a product item.

The WLCS_SECURITY Database Table

Table 10-13 describes the metadata for the WebLogic Commerce Server 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 WebLogic Commerce Server product.

Table 10-13 WLCS_SECURITY Table Metadata

Column Name	Data Type	Description and Recommendations
ID	NUMBER(2)	A unique identifier for the key pair. This field is the table's primary key and cannot be NULL.
PUBLIC_KEY	VARCHAR(2000)	The public key to be used for encryption/decryption of credit cards.
PRIVATE_KEY	VARCHAR(2000)	The private key to be used for encryption/decryption of credit cards.

The WLCS_SHIPPING_ADDRESS Database Table

Table 10-14 describes the metadata for the WebLogic Commerce Server WLCS_SHIPPING_ADDRESS table. This table is used to store information related to a customer's shipping address(es) in the order processing database.

The primary key is SHIPPING_ADDRESS_ID.

Table 10-14 WLCS_SHIPPING_ADDRESS Table Metadata

Column Name	Data Type	Description and Recommendations
SHIPPING_ADDRESS_ID	NUMBER (15)	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.
CUSTOMER_ID	VARCHAR (20)	A unique identifier for the customer.
MAP_KEY	VARCHAR (30)	Key that maps multiple shipping addresses with a single customer.
SHIPPING_GEOCODE	VARCHAR (2)	The code used by the TAXWARE system to identify taxes for the order based on jurisdiction.
SHIPPING_STREET1	VARCHAR (30)	The first line in the customer's shipping address.
SHIPPING_STREET2	VARCHAR (30)	The second line in the customer's shipping address.
SHIPPING_CITY	VARCHAR (30)	The city in the customer's shipping address.
SHIPPING_STATE	VARCHAR (40)	The state in the customer's shipping address.
SHIPPING_COUNTRY	VARCHAR (40)	The country in the customer's shipping address.
SHIPPING_POBOX	VARCHAR (30)	The post office box in the customer's shipping address.
SHIPPING_COUNTY	VARCHAR (50)	The county in the customer's shipping address.

Table 10-14 WLCS_SHIPPING_ADDRESS Table Metadata (Continued)

Column Name	Data Type	Description and Recommendations
SHIPPING_POSTAL_CODE	VARCHAR(10)	The postal (zip) code in the customer's shipping address.
SHIPPING_POSTAL_CODE_TYPE	VARCHAR(10)	Format or type of postal code, generally determined by country, such as ZIP code in the United States.

The WLCS_SHIPPING_METHOD Database Table

Table 10-15 describes the metadata for the WebLogic Commerce Server 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 10-15 WLCS_SHIPPING_METHOD Table Metadata

Column Name	Data Type	Description and Recommendations
PK_IDENTIFIER	VARCHAR(20)	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)	The carrier being used to ship the order, such as UPS or FedEx.
METHOD	VARCHAR(40)	The method by which the order is to be shipped, such as Air, 2nd Day Air, or Parcel Post.
AVERAGE_SHIPPING_TIME	NUMBER	The average number of days it will take the order to arrive.
PRICE_VALUE	NUMBER(16,4)	The amount it will cost to ship the order.
PRICE_CURRENCY	VARCHAR(10)	The currency associated with the PRICE_VALUE column, such as dollars, pounds, or lira.

10 The Order Processing Database Schema

Table 10-15 WLCS_SHIPPING_METHOD Table Metadata (Continued)

Column Name	Data Type	Description and Recommendations
WEIGHT_LIMIT	NUMBER (16 , 4)	The weight limit for the shipment.
RESTRICTIONS	VARCHAR (254)	Any restrictions associated with the shipment.
DESCRIPTION	VARCHAR (254)	A description of the shipping method, such as FedEx Overnight or Standard.
PO_BOX_ALLOWED	NUMBER	Specifies whether or not the shipment can be left at a post office box.
SIGNATURE_REQUIRED	NUMBER	Specifies whether or not a signature is required upon receipt of the shipment.
SATURDAY_DELIVERY	NUMBER	Specifies whether or not the shipment can be delivered on Saturday.
INTERNATIONAL_DELIVERY	NUMBER	Specifies whether or not international delivery is an option.
SIZE_LIMIT	NUMBER (16 , 4)	The size limit for the shipment.
PACKAGING_TYPE	VARCHAR (50)	The packaging type for the shipment.

The WLCS_TRANSACTION Database Table

Table 10-16 describes the metadata for the WebLogic Commerce Server WLCS_TRANSACTION table. This table is used to store data for every payment transaction in the order processing database.

The primary key is TRANSACTION_ID.

Table 10-16 WLCS_TRANSACTION Table Metadata

Column Name	Data Type	Description and Recommendations
TRANSACTION_ID	VARCHAR(25)	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)	A unique identifier of a batch submitted for settlement, as returned by CyberCash. This field need not be populated for other external payment services.
TRAN_DATE	DATE	The date of the transaction (that is, date on which the transaction was first started).
TRAN_STATUS	VARCHAR(20)	The current status of the transaction (Settled, Authorized, MarkedForSettle, PendingSettle, Retry, or Settled).
TRAN_AMOUNT	NUMBER(16,4)	The most recent amount applied to the transaction. MarkForSettle amounts can be different from the authorization amount.
TRAN_CURRENCY	VARCHAR(30)	The currency of the transaction.
CC_NUMBER	VARCHAR(200)	The customer's credit card number. This is encrypted if <code>is.encrypted.enable</code> is set to <code>true</code> in the <code>weblogiccommerce.properties</code> file.
CC_TYPE	VARCHAR(20)	The customer's credit card type, such as VISA or MasterCard.

10 The Order Processing Database Schema

Table 10-16 WLCS_TRANSACTION Table Metadata (Continued)

Column Name	Data Type	Description and Recommendations
CC_EXP_DATE	DATE	The expiration date on the customer's credit card.
CC_NAME	VARCHAR (50)	The credit card holder's name.
CC_DISPLAY_NUMBER	VARCHAR (20)	The version of the credit card number that is displayed (displays all Xs except last 4-digits).
CC_COMPANY	VARCHAR (50)	The name of the credit card company.
GEOCODE	VARCHAR (2)	The code used by the TAXWARE system to identify taxes for the order based on jurisdiction.
STREET1	VARCHAR (30)	The first line in the customer's street address.
STREET2	VARCHAR (30)	The second line in the customer's street address.
CITY	VARCHAR (30)	The city in the customer's address.
STATE	VARCHAR (40)	The state in the customer's address.
COUNTRY	VARCHAR (40)	The country in the customer's address.
POBOX	VARCHAR (30)	The post office box in the customer's address.
DESCRIPTION	VARCHAR (30)	Any additional data. Can be NULL.
COUNTY	VARCHAR (50)	The county in the customer's address.
POSTAL_CODE	VARCHAR (10)	The postal (ZIP) code in the customer's address.
POSTAL_CODE_TYPE	VARCHAR (10)	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 10-17 describes the metadata for the WebLogic Commerce Server 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.

The primary key is TRANSACTION_ENTRY_ID.

Table 10-17 WLCS_TRANSACTION_ENTRY Table Metadata

Column Name	Data Type	Description and Recommendations
TRANSACTION_ENTRY_ID	NUMBER (25)	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)	Represents the running count per transaction.
TRAN_ENTRY_DATE	DATE	The date of the log entry.
TRAN_ENTRY_STATUS	VARCHAR (20)	The status of the transaction when this entry was made.
TRAN_ENTRY_AMOUNT	NUMBER (16 , 4)	The amount of the transaction when the log entry was made.
TRAN_ENTRY_CURRENCY	VARCHAR (30)	The currency of the transaction.
TRANSACTION_ID	VARCHAR (25)	A unique identifier for the transaction.

The SQL Scripts Used to Create the Database

The database schemas for the WebLogic Personalization Server, WebLogic Commerce Server and Campaign Manager for WebLogic are all created by executing the `create_all` script for the target database environment.

Cloudscape

For Cloudscape, execute one of the following:

- `WL_COMMERCE_HOME\db\cloudscape\3.5.1\create_all.bat` (Windows)
- `WL_COMMERCE_HOME/db/cloudscape/3.5.1/create_all.sh` (UNIX)

Script Name	Description
<code>create_all.bat</code>	The execution of this script will create the WLPS, WLCS, and Campaign Manager database schema.
<code>create_all.sh</code>	The execution of this script will create the WLPS, WLCS, and Campaign Manager database schema.
<code>create_campaign.sql</code>	Creates the Campaign Manager specific database objects, such as tables, indexes, and constraints.
<code>create_common.sql</code>	Creates the database objects which are common to WLPS and WLCS.
<code>create_mail_ad.sql</code>	Creates all the database objects used by the mail messaging component.
<code>create_wlcs.sql</code>	Creates all the database objects for WLCS, including Catalog and Order Management.
<code>create_wlps.sql</code>	Creates all the database object for WLPS.
<code>drop_campaign.sql</code>	Drops all database objects associated with Campaign Manager.
<code>drop_common.sql</code>	Drops the database objects which are common between WLPS and WLCS.

The SQL Scripts Used to Create the Database

Script Name	Description
<code>drop_mail_ad.sql</code>	Drops the database objects used by the mail messaging component.
<code>drop_wlcs.sql</code>	Drops the database objects associated with WLCS.
<code>drop_wlps.sql</code>	Drops the database objects associated with WLPS.
<code>insert_common.sql</code>	Inserts core data into the common tables between WLPS and WLCS.
<code>insert_wlcs.sql</code>	Inserts core data into some of the WLCS tables.
<code>insert_wlcs_sample_catalog.sql</code>	Inserts sample data into the product catalog.
<code>insert_wlcs_sample_customer.sql</code>	Inserts sample customer information into WLCS tables.
<code>insert_wlcs_sample_data.sql</code>	Inserts sample data into various WLCS tables.
<code>insert_wlps.sql</code>	Inserts core data into WLPS tables.
<code>insert_wlps_sample_data.sql</code>	Inserts sample data into various WLPS tables.

Oracle

For Oracle, from the command line, move to the following directory:

```
WL_COMMERCE_HOME/db/oracle/8.1.6
```

After logging into SQL*Plus, simply execute the `create_all.sql` script, for example `@create_all`.

Script Name	Description
<code>create_campaign.sql</code>	Creates the Campaign Manager specific database objects, such as tables, indexes, and constraints.
<code>create_common.sql</code>	Creates the database objects which are common to WLPS and WLCS.

10 The Order Processing Database Schema

Script Name	Description
<code>create_mail_ad.sql</code>	Creates all the database objects used by the mail messaging component.
<code>create_wlcs.sql</code>	Creates all the database objects for WLCS, including Catalog and Order Management.
<code>create_wlps.sql</code>	Creates all the database object for WLPS.
<code>drop_campaign.sql</code>	Drops all database objects associated with Campaign Manager.
<code>drop_common.sql</code>	Drops the database objects which are common between WLPS and WLCS.
<code>drop_mail_ad.sql</code>	Drops the database objects used by the mail messaging component.
<code>drop_wlcs.sql</code>	Drops the database objects associated with WLCS.
<code>drop_wlps.sql</code>	Drops the database objects associated with WLPS.
<code>insert_common.sql</code>	Inserts core data into the common tables between WLPS and WLCS.
<code>insert_wlcs.sql</code>	Inserts core data into some of the WLCS tables.
<code>insert_wlcs_sample_catalog.sql</code>	Inserts sample data into the product catalog.
<code>insert_wlcs_sample_customer.sql</code>	Inserts sample customer information into WLCS tables.
<code>insert_wlcs_sample_data.sql</code>	Inserts sample data into various WLCS tables.
<code>insert_wlps.sql</code>	Inserts core data into WLPS tables.
<code>insert_wlps_sample_data.sql</code>	Inserts sample data into various WLPS tables.
<code>install_report.sql</code>	This script is used to summarize the database installation. Displays information such as the number of tables, indexes, and so on.
<code>statistics.sql</code>	This script is used in computing statistics on various database objects, such as tables and indexes, in an Oracle environment.

Defined Constraints

In each `create-order-*` SQL file, the database tables described earlier in this chapter are created. In addition, the SQL files define constraints. Table 10-18 shows the table name and describes the constraint(s) defined for it.

Note: The sample SQL statements shown in the table are from the `create-order-oracle.sql` file. The syntax is different for Cloudscape. Except where noted, the effect of each constraint is the same.

Table 10-18 Constraints Defined on Order Database Tables

Table Name	Constraints as Defined in <code>create-order-oracle.sql</code>
WLCS_COUNTRY	A referential integrity constraint is used to ensure the appropriate country ID (COUNTRY_ID) is being used with the order.
DISCOUNT	<p>A check constraint (DISCOUNT_IS_GLOBAL) is used on the IS_GLOBAL column to ensure that the value of the column is either a 0 (false) or 1 (true).</p> <p>A check constraint (DISCOUNT_IS_ACTIVE) is used on the IS_ACTIVE column to ensure that the value of the column is either a 0 (false) or 1 (true).</p> <p>A referential integrity constraint (FK1_DISCOUNT) ensures that the DISCOUNT_SET record exists before a DISCOUNT record can be inserted.</p>
DISCOUNT_ASSOCIATION	<p>A data integrity constraint exists so in the event that a customer record is deleted in WLCS_CUSTOMER, the constraint FK1_DISCOUNT_ASSOCIATION will ensure all DISCOUNT_ASSOCIATION records for that customer are deleted as well.</p> <p>A referential integrity constraint (FK2_DISCOUNT_ASSOCIATION) ensures that the discount exists before a DISCOUNT_ASSOCIATION record can be inserted.</p>

10 The Order Processing Database Schema

Table 10-18 Constraints Defined on Order Database Tables (Continued)

Table Name	Constraints as Defined in create-order-oracle.sql
WLCS_CREDIT_CARD	<p>If a customer is deleted from the database, the CUSTOMER_CREDIT_CARD_FK constraint causes all their associated credit cards to be deleted.</p> <p>The constraint for the schema in Oracle is: CONSTRAINT CUSTOMER_CREDIT_CARD_FK REFERENCES WLCS_CUSTOMER(CUSTOMER_ID) ON DELETE CASCADE</p>
WLCS_ORDER_LINE	<p>If an order is deleted from the database, the WLCS_ORDER_FK constraint causes all the associated order line items to be deleted.</p> <p>The constraint for the schema in Oracle is: CONSTRAINT ORDER_FK REFERENCES WLCS_ORDER(ORDER_ID) ON DELETE CASCADE</p>
ORDER_ADJUSTMENT	<p>A referential integrity constraint (FK1_ORDER_ADJUSTMENT) ensures that the DISCOUNT record exists before the ORDER_ADJUSTMENT record can be inserted.</p>
ORDER_LINE_ADJUSTMENT	<p>A referential integrity constraint (FK1_ORDER_LINE_ADJUSTMENT) ensures that the DISCOUNT record exists before the ORDER_LINE_ADJUSTMENT record can be inserted.</p>
WLCS_SHIPPING_ADDRESS	<p>If a customer is deleted from the database, the CUSTOMER_FK constraint causes all their associated shipping addresses to be deleted.</p> <p>The constraint for the schema in Oracle is: CONSTRAINT CUSTOMER_FK REFERENCES WLCS_CUSTOMER(CUSTOMER_ID) ON DELETE CASCADE</p>
WLCS_TRANSACTION_ENTRY	<p>If a transaction is deleted from the database, the WLCS_TRANSACTION_FK constraint causes all the associated transaction entries be deleted.</p> <p>The constraint for the schema in Oracle is: CONSTRAINT WLCS_TRANSACTION_FK REFERENCES WLCS_TRANSACTION(TRANSACTION_ID) ON DELETE CASCADE</p>

Index

A

- accessor method(s)
 - attributes of 3-12
 - ShoppingCart 3-12
 - ShoppingCartLine 3-13
- addaddress.jsp 4-17
- adding run-time attributes 8-6, 8-8
- Administration Tools page
 - loading 9-2
 - sample page 9-3
- association service 2-7
- attributes
 - accessor method 3-12
- authorizing transactions 9-19

B

- browsing the product catalog 1-3
- business logic 1-2, 3-9

C

- Calendar Date Selection Tool
 - about 9-8
- campaign dates and discount dates 2-3
- capturing transactions 9-21
- checkout process 3-4
- checkout.jsp 7-2
- comparison of per item and set-based
 - discounts 2-9
- confirmorder.jsp 7-12

- consumption model of discounts 2-10
- credit card
 - processing using CyberCash 6-26
 - security 6-39
- Customer ID
 - searching for a payment by 9-16
 - searching for an order by 9-1, 9-4
- customer support contact information 1-xiv
- CyberCash
 - about 6-26
 - configuring 6-27
 - integration with 6-26

D

- data dictionary 10-5
- database schema 1-6, 10-1
- database table
 - DISCOUNT 10-9
 - ORDER_LINE_ADJUSTMENT 10-16
 - WLCS_COUNTRY 10-7
 - WLCS_CREDIT_CARD 10-5
 - WLCS_CURRENCY 10-7
 - WLCS_CUSTOMER 10-8
 - WLCS_ORDER 10-12
 - WLCS_ORDER_LINE 10-14
 - WLCS_SAVED_ITEM_LIST 10-17
 - WLCS_SECURITY 10-17
 - WLCS_SHIPPING_ADDRESS 10-18
 - WLCS_SHIPPING_METHOD 10-19
 - WLCS_TRANSACTION 10-21

WLCS_TRANSACTION_ENTRY 10-23

date
 range
 searching for an order by 9-2, 9-8

default Webflow 1-4

DISCOUNT 10-9

discount
 deactivating 2-4
 definitions 2-3
 limits 2-3
 management service 2-3
 parameters 2-3
 priorities 2-4
 rules 2-13

discounts
 background information 2-1
 calculators 2-12
 consumption model 2-10
 delete deployed 2-6
 deployed 2-5
 deploying 2-5
 fixed off 2-12
 fixed price 2-12
 global 2-1
 percentage off 2-12
 priority 2-11
 saving and deploying 2-5
 user 2-1

discountUtil 2-5

documentation, where to find it 1-xii

dynamic data display
 checkout.jsp 7-6
 confirmorder.jsp 7-16
 payment.jsp 6-5
 paymenteditcc.jsp 6-15
 selectaddress.jsp 4-13
 selecttaxaddress.jsp 5-5
 shipping.jsp 4-6
 shoppingcart.jsp 3-11

E

entity beans 8-4

entity-relation diagram 10-1

event(s)
 addaddress.jsp 4-20
 checkout.jsp 7-6
 payment.jsp 6-5
 paymenteditcc.jsp 6-15
 paymentnewcc.jsp 6-10
 selecttaxaddress.jsp 5-5
 shipping.jsp 4-5
 shoppingcart.jsp 3-9

G

getShoppingCartLineCollection() 7-9

H

high-level architecture 1-4

I

inactivating a discount 2-4

input processors
 DecideShippingAddressPageIP 5-9
 DeleteProductItemFromShoppingCartIP 3-15
 EmptyShoppingCartIP 3-16
 InitShippingMethodListIP 4-23
 InitShoppingCartIP 3-17
 PaymentAuthorizationIP 6-20
 UpdatePaymentInfoIP 6-21
 UpdateShippingAddressIP 4-24, 5-10
 UpdateShoppingCartQuantitiesIP 3-17
 UpdateSkulIP 3-18
 ValidateAddressIP 4-25
 ValidateShippingInfoIP 4-26

J

- Java scriptlets 3-12
- JavaServer Page (JSP) templates
 - addaddress.jsp 4-17
 - checkout.jsp 7-2
 - confirmorder.jsp 7-12
- Managing Purchases and Processing Orders 1-2
 - Orders 1-2
- payment.jsp 6-2
- paymenteditcc.jsp 6-12
- paymentnewcc.jsp 6-7
- selecttaxaddress.jsp 5-2
- shoppingcart.jsp 3-4

JSP tags

- getPipelineProperty 3-11, 5-5
- getProfile 4-13, 6-5, 6-15
- getProperty 4-13, 6-6, 6-16

L

- license 2-2

M

- Managing Purchases and Processing Orders
 - about 1-2
 - structure of 1-3

O

- Order Identifier number
 - searching for an order by 9-2, 9-6
- Order List
 - about 9-5
- Order Management page
 - loading Administration Tools page 9-2
 - search page 9-4
- order processing data dictionary 10-5
- order processing schema 10-4
- Order Status page 9-5
- order(s)

- search

- by Customer ID 9-1, 9-4
 - by date range 9-2, 9-8
 - by Order Identifier number 9-2, 9-6

P

- payment models
 - host-based 6-29
 - switching between two 6-31
 - terminal-based 6-29
- Payment Transaction History
 - about 9-16
- payment(s)
 - search
 - by Customer ID 9-16
 - by status 9-17
 - status 9-19
- payment.jsp 6-2
- paymenteditcc.jsp 6-12
- paymentnewcc.jsp 6-7
- persistence architecture 8-3
- Pipeline components
 - AddShippingAddressPC 4-27
 - AddToCartTrackerPC 3-23
 - CalculateShippingPC 4-28
 - CommitOrderPC 7-21
 - DeleteProductItemFromSavedListPC 3-19
 - DeleteShippingAddressPC 4-29
 - MoveProductItemToSavedListPC 3-20
 - MoveProductItemToShippingCartPC 3-21
 - PaymentAuthorizationHostPC 6-22
 - PaymentAuthorizationTerminalPC 6-24
 - PriceShoppingCartPC 3-22
 - PurchaseTrackerPC 7-23
 - RefreshSavedListPC 3-22
 - RemoveFromCartTrackerPC 3-24
 - ResetOrderCheckoutPC 7-22
 - TaxCalculateAndCommitLineLevelPC

5-12
TaxCalculateLineLevelPC 5-11
TaxVerifyShippingAddressPC 5-12
UpdateShoppingCartQuantitiesTracker
PC 3-24
price service 2-7
pricing operation 2-11
printing product documentation 1-xiii
product license 2-2

R

registrating customers and managing
customer services 1-3
related information 1-xiii
retrieving Pipeline session attributes
shopping cart 3-12

S

schema extension 8-9
scriptlets, Java 3-12
search
order
by Customer ID 9-1, 9-4
by date range 9-2, 9-8
by Order Identifier number 9-2, 9-6
payment
by Customer ID 9-16
by status 9-17
security
credit card 6-39
selectaddress.jsp 4-9
selecttaxaddress.jsp 5-2
services, order-related 1-1
settling transactions 9-21
Shipping Services
about 4-1
addaddress.jsp template 4-17
selectaddress.jsp template 4-9
shipping.jsp template 4-2

shipping.jsp 4-2
shopping cart
managing 3-1
Shopping Cart Management Services 3-1
shoppingcart.jsp 3-4
SQL Scripts 10-24
StartCommerce.bat 9-2
StartCommerce.sh 9-2
starting the WebLogic Commerce Server
Administration Tools 9-1, 9-2
status
of payments
searching by 9-17
values for 9-19
support
technical 1-xiv

T

target items 2-2
Taxation Services
removing 5-42
TAXWARE
checklist 5-41
configuration and deployment 5-20
considerations 5-15
installing on UNIX 5-17
installing on Windows 5-16
integrating with 5-14
run-time configuration 5-33
SALES/USE Tax System 5-14
specific properties 5-23
tax codes and product catalog 5-41
to calculate taxes 5-14
Universal Tax Link (UTL) System 5-14
VERAZIP System 5-14
time zones 2-3
transactions
authorizing 9-19
capturing 9-21
managing 8-19

settling 9-21
trigger items 2-2

U

Unified User Profile (UUP) technology 8-6
uploading and downloading discount XML
documents 2-5
use-cases 8-2

W

ways to use discounts 2-1
Webflow
 modifying properties file 5-43
 property file 6-32
Webflow/Pipeline infrastructure 1-4
WebLogic Commerce Server
 Administration Tools
 starting 9-1, 9-2
WebLogic Commerce Server schema
 extension 8-9
WLCS_CREDIT_CARD 10-5
WLCS_CURRENCY 10-7
WLCS_CUSTOMER 10-8
WLCS_ORDER 10-12
WLCS_ORDER_LINE 10-14
WLCS_ORDER_LINE_ADJUSTMENT
 10-16
WLCS_SAVED_ITEM_LIST 10-17
WLCS_SECURITY 10-17
WLCS_SHIPPING_ADDRESS 10-18
WLCS_SHIPPING_METHOD 10-19
WLCS_TRANSACTION 10-21
WLCS_TRANSACTION_ENTRY 10-23

X

XML discount document 2-5