



# BEA WebLogic Commerce Server Order Processing Package

BEA WebLogic Commerce Server 3.2  
Document Edition 3.2.3  
September 2001

## 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, ObjectBroker, TOP END, and Tuxedo are registered trademarks of BEA Systems, Inc. BEA Builder, BEA Connect, BEA Manager, BEA MessageQ, BEA Jolt, M3, eSolutions, eLink, WebLogic, WebLogic Enterprise, WebLogic Commerce Server, and WebLogic Personalization Server are trademarks of BEA Systems, Inc.

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

## Order Processing Package

| <b>Document Edition</b> | <b>Date</b>    | <b>Software Version</b>      |
|-------------------------|----------------|------------------------------|
| 3.2.3                   | September 2001 | WebLogic Commerce Server 3.2 |

---

# Contents

## About This Document

|                                 |      |
|---------------------------------|------|
| What You Need to Know .....     | xii  |
| e-docs Web Site .....           | xii  |
| How to Print the Document ..... | xiii |
| Related Information .....       | xiii |
| Contact Us! .....               | xiv  |
| Documentation Conventions ..... | xv   |

## 1. Overview of the Order Processing Package

|   |     |
|---|-----|
| What Is the Order Processing Package? ..... | 1-2 |
| High-level Architecture .....               | 1-3 |
| Development Roles .....                     | 1-5 |
| Next Steps .....                            | 1-5 |

## 2. The Order Processing Database Schema

|   |      |
|---|------|
| The Entity-Relation Diagram .....               | 2-2  |
| The WLCS_CUSTOMER Database Table .....          | 2-4  |
| The WLCS_SHIPPING_ADDRESS Database Table .....  | 2-6  |
| The WLCS_CREDIT_CARD Database Table .....       | 2-8  |
| The WLCS_TRANSACTION Database Table .....       | 2-10 |
| The WLCS_TRANSACTION_ENTRY Database Table ..... | 2-12 |
| The WLCS_SAVED_ITEM_LIST Database Table .....   | 2-13 |
| The WLCS_ORDER Database Table .....             | 2-14 |
| The WLCS_ORDER_LINE Database Table .....        | 2-16 |
| The WLCS_SHIPPING_METHOD Database Table .....   | 2-17 |
| The WLCS_SECURITY Database Table .....          | 2-19 |
| The SQL Files and Defined Constraints .....     | 2-20 |

---

### 3. Shopping Cart Management Services

|  |      |
|--|------|
| JavaServer Pages (JSPs) .....                                      | 3-2  |
| shoppingcart.jsp Template.....                                     | 3-2  |
| Sample Browser View.....   | 3-3  |
| Location in the WebLogic Commerce Server Directory Structure ..... | 3-6  |
| Tag Library Imports .....  | 3-6  |
| Java Package Imports .....   | 3-7  |
| Location in Default Webflow.....                                   | 3-7  |
| Included JSP Templates .....                                       | 3-8  |
| Events .....   | 3-8  |
| Dynamic Data Display .....   | 3-9  |
| Form Field Specification .....                                     | 3-13 |
| Input Processors.....  | 3-14 |
| DeleteProductItemFromShoppingCartIP .....                          | 3-14 |
| EmptyShoppingCartIP.....   | 3-15 |
| InitShoppingCartIP.....  | 3-15 |
| UpdateShoppingCartQuantitiesIP .....                               | 3-16 |
| UpdateSkuIP.....   | 3-17 |
| Pipeline Components .....  | 3-18 |
| DeleteProductItemFromSavedListPC .....                             | 3-18 |
| MoveProductItemToSavedListPC.....                                  | 3-19 |
| MoveProductItemToShoppingCartPC .....                              | 3-20 |
| RefreshSavedListPC.....  | 3-21 |

### 4. Shipping Services

|  |     |
|--|-----|
| JavaServer Pages (JSPs) .....                                      | 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 .....   | 4-5 |
| Location in Default Webflow.....                                   | 4-5 |
| Included JSP Templates .....                                       | 4-6 |
| Events .....   | 4-6 |
| Dynamic Data Display .....   | 4-6 |

---

|   |      |
|---|------|
| Form Field Specification.....                                   | 4-8  |
| selectaddress.jsp Template .....                                | 4-10 |
| Sample Browser View .....                                       | 4-10 |
| Location in the WebLogic Commerce Server Directory Structure... | 4-12 |
| Tag Library Imports .....                                       | 4-12 |
| Java Package Imports.....                                       | 4-13 |
| Location in Default Webflow .....                               | 4-13 |
| Included JSP Templates .....                                    | 4-13 |
| Events.....   | 4-14 |
| Dynamic Data Display .....                                      | 4-15 |
| Form Field Specification.....                                   | 4-18 |
| addaddress.jsp Template .....                                   | 4-19 |
| Sample Browser View .....                                       | 4-19 |
| Location in the WebLogic Commerce Server Directory Structure... | 4-20 |
| Tag Library Imports .....                                       | 4-21 |
| Java Package Imports.....                                       | 4-21 |
| Location in Default Webflow .....                               | 4-21 |
| Included JSP Templates .....                                    | 4-22 |
| Events.....   | 4-22 |
| Dynamic Data Display .....                                      | 4-23 |
| Form Field Specification.....                                   | 4-23 |
| Input Processors.....   | 4-25 |
| InitShippingMethodListIP.....                                   | 4-25 |
| UpdateShippingAddressIP .....                                   | 4-26 |
| ValidateAddressIP.....  | 4-27 |
| ValidateShippingInfoIP.....                                     | 4-28 |
| Pipeline Components .....                                       | 4-29 |
| AddShippingAddressPC.....                                       | 4-29 |
| CalculateShippingPC .....                                       | 4-30 |
| DeleteShippingAddressPC.....                                    | 4-31 |

## 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..... | 5-4  |
| Tag Library Imports .....   | 5-4  |
| Java Package Imports .....  | 5-5  |
| Location in Default Webflow.....                                  | 5-5  |
| Included JSP Templates .....                                      | 5-5  |
| Events .....  | 5-6  |
| Dynamic Data Display .....  | 5-6  |
| Form Field Specification .....                                    | 5-8  |
| Input Processors.....   | 5-10 |
| DecideShippingAddressPageIP .....                                 | 5-10 |
| UpdateShippingAddressIP .....                                     | 5-11 |
| Pipeline Components .....   | 5-12 |
| TaxCalculateLineLevelPC .....                                     | 5-12 |
| TaxCalculateAndCommitLineLevelPC .....                            | 5-13 |
| TaxVerifyShippingAddressPC .....                                  | 5-13 |
| Integration with TAXWARE.....                                     | 5-15 |
| Important TAXWARE Considerations .....                            | 5-15 |
| TAXWARE Installation .....  | 5-16 |
| Installation Directory Structure .....                            | 5-17 |
| Testing the TAXWARE Installation .....                            | 5-19 |
| Changing the TAXWARE Directory Structure .....                    | 5-20 |
| TAXWARE Configuration and Deployment .....                        | 5-21 |
| Addresses and Taxation .....                                      | 5-21 |
| TAXWARE-Specific Properties .....                                 | 5-24 |
| Run-Time Configuration .....                                      | 5-29 |
| Tax Codes and the Product Catalog .....                           | 5-32 |
| Updating TAXWARE Tax Data .....                                   | 5-32 |
| TAXWARE Checklist.....  | 5-33 |
| Viewing Debugging Information in TAXWARE .....                    | 5-33 |
| Removing Tax Calculations .....                                   | 5-34 |
| Modifying the Pipeline Properties File .....                      | 5-34 |
| Modifying the Webflow Properties File.....                        | 5-35 |
| What if I Don't Want to Use TAXWARE to Calculate My Taxes? .....  | 5-38 |

---

## 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..... | 6-4  |
| Tag Library Imports .....   | 6-4  |
| Java Package Imports.....   | 6-5  |
| Location in Default Webflow .....                                 | 6-5  |
| Included JSP Templates .....                                      | 6-5  |
| Events.....   | 6-6  |
| Dynamic Data Display .....  | 6-6  |
| Form Field Specification.....                                     | 6-7  |
| paymentnewcc.jsp Template .....                                   | 6-8  |
| Sample Browser View .....   | 6-8  |
| Location in the WebLogic Commerce Server Directory Structure..... | 6-9  |
| Tag Library Imports .....   | 6-10 |
| Java Package Imports.....   | 6-10 |
| Location in Default Webflow .....                                 | 6-10 |
| Included JSP Templates .....                                      | 6-11 |
| Events.....   | 6-11 |
| Dynamic Data Display .....  | 6-11 |
| Form Field Specification.....                                     | 6-11 |
| paymenteditcc.jsp Template.....                                   | 6-14 |
| Sample Browser View .....   | 6-14 |
| Location in the WebLogic Commerce Server Directory Structure...   | 6-15 |
| Tag Library Imports .....   | 6-15 |
| Java Package Imports.....   | 6-16 |
| Location in Default Webflow .....                                 | 6-16 |
| Included JSP Templates .....                                      | 6-17 |
| Events.....   | 6-17 |
| Dynamic Data Display .....  | 6-17 |
| Form Field Specification.....                                     | 6-19 |
| Input Processors.....   | 6-22 |
| PaymentAuthorizationIP .....                                      | 6-22 |
| UpdatePaymentInfoIP .....   | 6-23 |

---

|  |      |
|--|------|
| Pipeline Components .....  | 6-24 |
| PaymentAuthorizationHostPC .....   | 6-24 |
| PaymentAuthorizationTerminalPC .....                                       | 6-26 |
| Integration with CyberCash .....   | 6-28 |
| Configuration Activities for Using CyberCash .....                         | 6-29 |
| Payment Models .....   | 6-31 |
| How Do I Switch Between the Two Payment Models? .....                      | 6-33 |
| What if I Don't Want to Use CyberCash for Credit Card Processing? .....    | 6-35 |
| Credit Card Security Service .....   | 6-40 |
| Encryption/Decryption Implementation .....                                 | 6-41 |
| Customizable Security Settings .....                                       | 6-41 |
| Methods for Supplying the Private Key Encryption Password .....            | 6-44 |
| Specifying the Password in weblogiccommerce.properties (Default) .....     | 6-44 |
| Specifying the Password at Server Startup Using the Console .....          | 6-46 |
| Specifying the Password After Server Startup Using a Secure Web Form ..... | 6-46 |
| Important Notes About Supplying Your Password .....                        | 6-48 |
| What if I Want to Change My Password? .....                                | 6-48 |

## 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  |
| Included JSP Templates .....                                       | 7-6  |
| Events .....   | 7-6  |
| Dynamic Data Display .....   | 7-7  |
| Form Field Specification .....                                     | 7-13 |
| confirmorder.jsp Template .....                                    | 7-14 |
| Sample Browser View .....  | 7-14 |
| Location in the WebLogic Commerce Server Directory Structure ..... | 7-16 |



---

|                                   |      |
|-----------------------------------|------|
| Tag Library Imports .....         | 7-16 |
| Java Package Imports.....         | 7-16 |
| Location in Default Webflow ..... | 7-17 |
| Included JSP Templates .....      | 7-17 |
| Events.....                       | 7-17 |
| Dynamic Data Display .....        | 7-17 |
| Form Field Specification.....     | 7-23 |
| Input Processors.....             | 7-24 |
| Pipeline Components.....          | 7-24 |
| CommitOrderPC.....                | 7-24 |
| ResetCheckoutPC.....              | 7-25 |

## 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<br>Schema .....                               | 8-10 |
| Adding Attributes Against WLCS_CUSTOMER, WLCS_ORDER,<br>WLCS_TRANSACTION and WLCS_SHIPPING_METHOD Tables.... | 8-11 |
| Adding Attributes Against the WLCS_ORDER_LINE Table.....   | 8-13 |
| Adding Attributes Against the WLCS_CREDIT_CARD and<br>WLCS_SHIPPING_ADDRESS Tables.....                      | 8-15 |
| Transaction Management .....   | 8-18 |

## 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  |
| Using the Payment Management Search Page .....                   | 9-12 |
| Searching for a Payment by Customer ID.....                      | 9-13 |

---

|  |      |
|--|------|
| Searching for a Payment by Status.....             | 9-14 |
| Authorizing, Capturing, and Settling Payments..... | 9-16 |
| Authorizing the Transaction.....                   | 9-16 |
| Capturing the Transaction.....                     | 9-18 |
| Settling the Transaction.....                      | 9-18 |

## Index

---

# About This Document

This document explains how to use the services available within the BEA WebLogic Commerce Server™ Processing package.

This document includes the following topics:

- Chapter 1, “Overview of the Order Processing Package,” which describes the high-level architecture of the package and provides introductory information about its services.
- Chapter 2, “The Order Processing Database Schema,” which describes the database tables used for order processing activities.
- 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 the Order Package data model.
- Chapter 9, “Using the Order and Payment Management Pages,” which describes how to find and manage customer orders and modify payment transactions.

---

## What You Need to Know

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 and Personalization Server 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.

## 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 WebLogic Commerce and Personalization Server documentation Home page on the e-docs Web site (and also on the documentation CD). You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access the PDFs, open the WebLogic Commerce and 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 and Personalization Server documents contain information that is relevant to using the Order Processing package and understanding how to customize or extend the provided services.

- [WebLogic Commerce Server Webflow and Pipeline Management](#)
- [WebLogic Commerce Server Registration and User Processing Package](#)
- [WebLogic Commerce Server Product Catalog Management](#)

---

## Contact Us!

Your feedback on the WebLogic Commerce and Personalization Server documentation is important to us. Send us e-mail at [docsupport@beasys.com](mailto:docsupport@beasys.com) if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the WebLogic Commerce and Personalization Server documentation.

In your e-mail message, please indicate that you are using the documentation for the WebLogic Commerce and Personalization Server 3.2 release.

If you have any questions about this version of WebLogic Commerce and Personalization Server, or if you have problems installing and running WebLogic Commerce and Personalization Server, contact BEA Customer Support through BEA WebSUPPORT at [www.beasys.com](http://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   |
|--------------------------------|--|
| <b>boldface text</b>           | 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.<br><i>Examples:</i><br><pre>#include &lt;iostream.h&gt; void main ( ) the pointer psz chmod u+w * \tux\data\ap .doc tux.doc BITMAP float</pre> |
| <b>monospace boldface text</b> | Identifies significant words in code.<br><i>Example:</i><br><pre>void <b>commit</b> ( )</pre>  |
| <i>monospace italic text</i>   | Identifies variables in code.<br><i>Example:</i><br><pre>String <i>expr</i></pre>  |
| UPPERCASE TEXT                 | Indicates device names, environment variables, and logical operators.<br><i>Examples:</i><br><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.<br><i>Example:</i><br>buildobjclient [-v] [-o name ] [-f file-list]...<br>[-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"> <li>■ That an argument can be repeated several times in a command line</li> <li>■ That the statement omits additional optional arguments</li> <li>■ That you can enter additional parameters, values, or other information</li> </ul> The ellipsis itself should never be typed.<br><i>Example:</i><br>buildobjclient [-v] [-o name ] [-f file-list]...<br>[-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 the Order Processing Package

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 product provides you with an Order Processing package. This package contains default implementations for the most common e-business order-related services (such as shopping cart management, taxation, payment, and so on). Designed to be used out-of-the-box, the Order Processing package 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 the Order Processing package, and introduces you to the types of services that are available.

This topic includes the following sections:

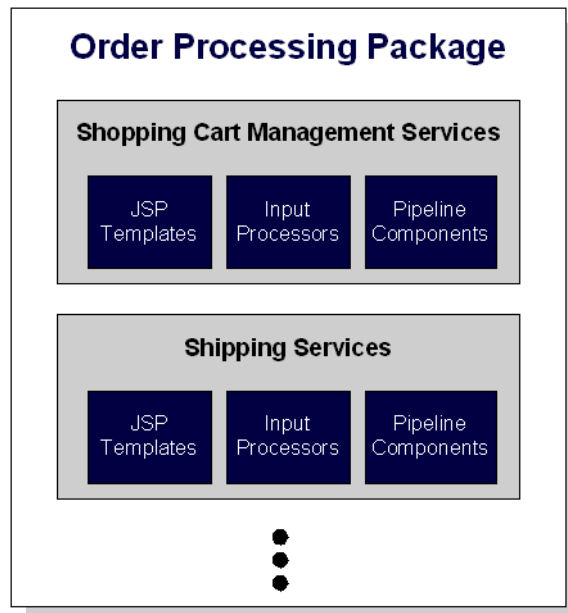
- What Is the Order Processing Package?
- High-level Architecture
- Development Roles
- Next Steps

## What Is the Order Processing Package?

The Order Processing package is a collection of services used to facilitate the online ordering process. There are services for shipping, payment, and so on. Together, the services in the Order Processing package 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 in the package 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 create your own input processors and Pipeline components to incorporate into the Order Processing package.

**Figure 1-1 Structure of the Order Processing Package**



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 [WebLogic Commerce Server WebFlow and Pipeline Management](#).

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 [WebLogic Commerce Server Product Catalog Management](#). For information on services related to registration and user processing, see [WebLogic Commerce Server Registration and User Processing Package](#).

## High-level Architecture

The Order Processing package 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 the Order Processing package work together in the default Webflow. It is also important that you understand how this package works in conjunction with JSP templates in the Registration and User Processing package.

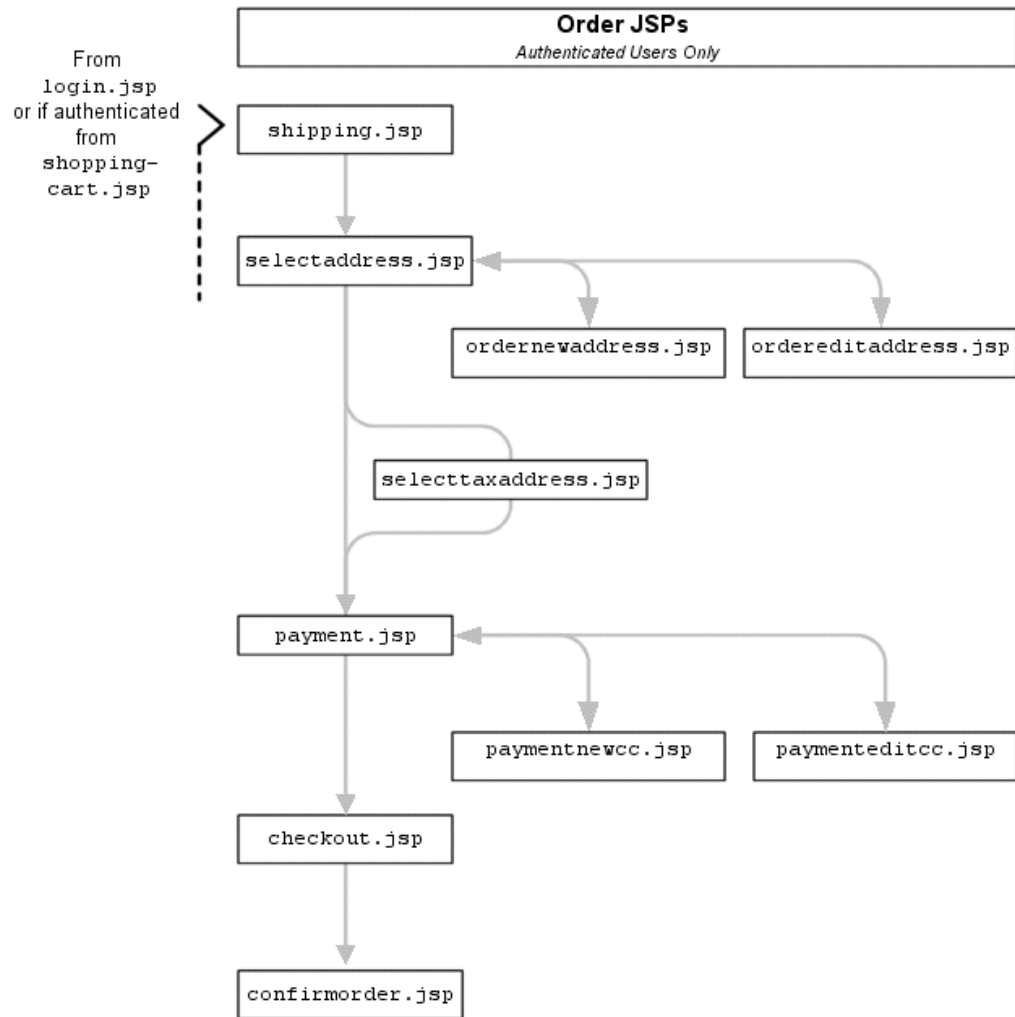
- For more information about the default Webflow, see [WebLogic Commerce Server Webflow and Pipeline Management](#).
- For more information about the Registration and User Processing package, see [WebLogic Commerce Server Registration and User Processing Package](#).

Figure 1-2 shows the ways in which your customer might move through the JSP templates in the Order Processing package. It also shows where the Registration and User Processing package 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).

# 1 Overview of the Order Processing Package

---

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 the services in the Order Processing package (including the JSP templates, input processors, and Pipeline components associated with them) is provided in this document. This includes detailed information about the database schema, for those advanced programmers who want to take their e-business site to the next level.

## Development Roles

This document is intended for the following audiences:

- The 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 and Personalization Server 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 the Order Processing package 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"

# **1** *Overview of the Order Processing Package*

---

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

# 2 The Order Processing Database Schema

This topic describes the database schema for the WebLogic Commerce Server Order Processing package. Understanding this schema will be helpful to those who may be customizing or extending the technologies provided in the product.

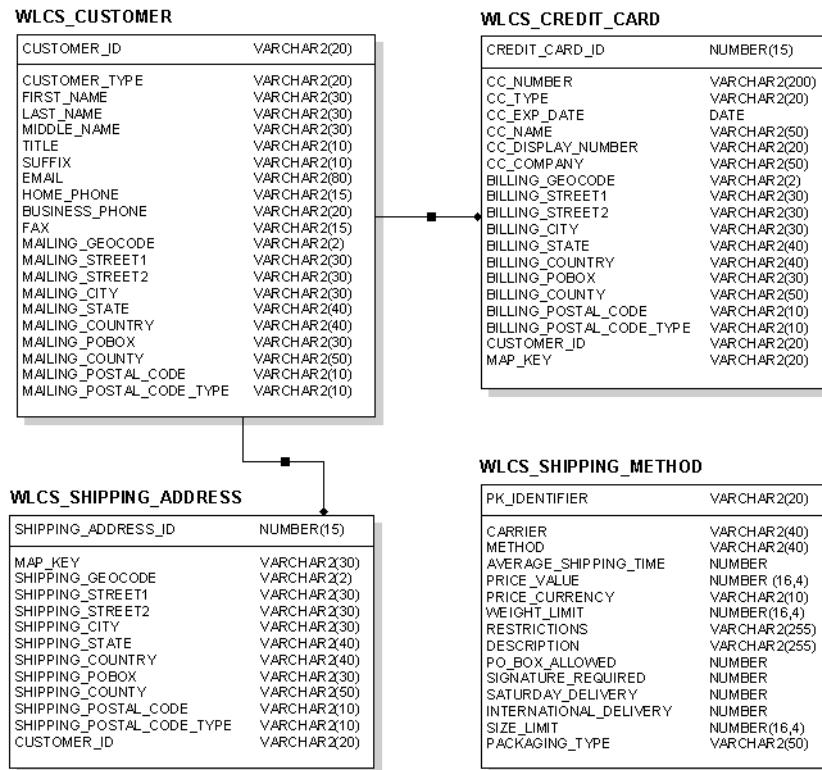
This topic includes the following sections:

- The Entity-Relation Diagram
- The WLCS\_CUSTOMER Database Table
- The WLCS\_SHIPPING\_ADDRESS Database Table
- The WLCS\_TRANSACTION Database Table
- The WLCS\_TRANSACTION\_ENTRY Database Table
- The WLCS\_SAVED\_ITEM\_LIST Database Table
- The WLCS\_ORDER Database Table
- The WLCS\_ORDER\_LINE Database Table
- The WLCS\_SHIPPING\_METHOD Database Table
- The WLCS\_SECURITY Database Table
- The SQL Files and Defined Constraints

# The Entity-Relation Diagram

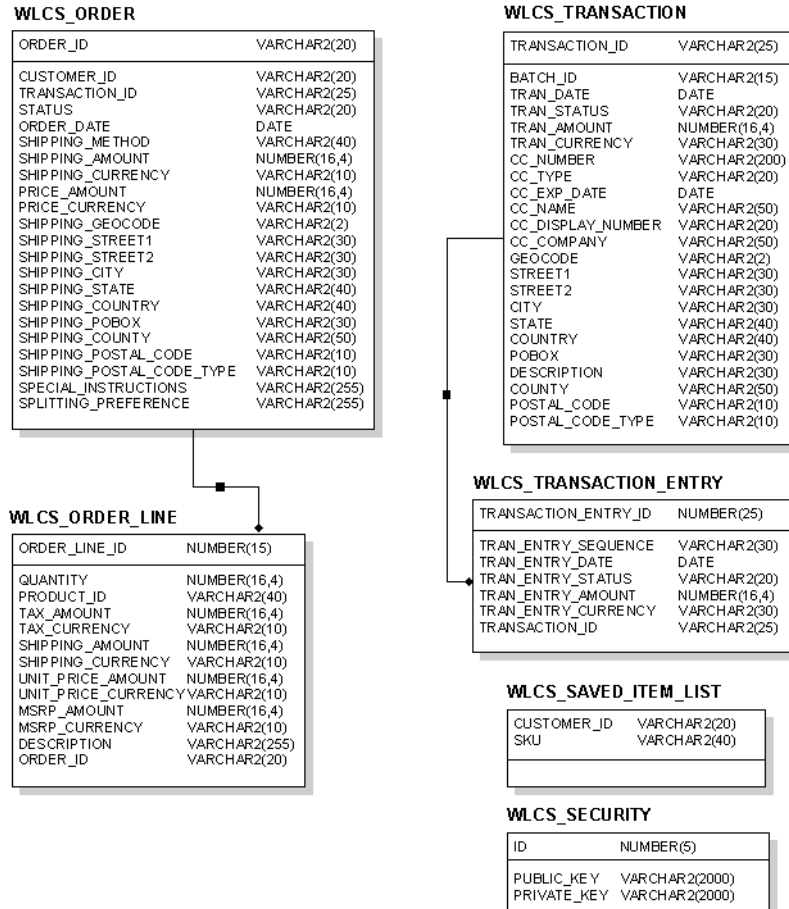
Figure 2-1 shows the Entity-Relation diagram for the WebLogic Commerce Server order processing database.

**Figure 2-1 Entity-Relation Diagram for the Order Processing Database**





## The Entity-Relation Diagram



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

## The WLCS\_CUSTOMER Database Table

Table 2-1 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.

**Table 2-1 WLCS\_CUSTOMER Table Metadata**

| Column Name     | Data Type       | Description and Recommendations  |
|-----------------|-----------------|--|
| CUSTOMER_ID     | VARCHAR2 ( 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   | VARCHAR2 ( 20 ) | A label for the customer (such as preferred, standard, or business).   |
| FIRST_NAME      | VARCHAR2 ( 30 ) | The customer's first name.   |
| LAST_NAME       | VARCHAR2 ( 30 ) | The customer's last name.  |
| MIDDLE_NAME     | VARCHAR2 ( 30 ) | The customer's middle name.  |
| TITLE           | VARCHAR2 ( 10 ) | The customer's preferred title (Mr., Mrs., Ms.).   |
| SUFFIX          | VARCHAR2 ( 10 ) | The customer's preferred suffix (Jr., Sr.).  |
| EMAIL           | VARCHAR2 ( 80 ) | The customer's email address.  |
| HOME_PHONE      | VARCHAR2 ( 15 ) | The customer's home phone number.  |
| BUSINESS_PHONE  | VARCHAR2 ( 20 ) | The customer's business phone number.  |
| FAX             | VARCHAR2 ( 15 ) | The customer's fax number.   |
| MAILING_GEOCODE | VARCHAR2 ( 2 )  | The code used by the TAXWARE system to identify taxes for the order based on jurisdiction.   |
| MAILING_STREET1 | VARCHAR2 ( 30 ) | The first line in the customer's street address.   |
| MAILING_STREET2 | VARCHAR2 ( 30 ) | The second line in the customer's street address.  |

**Table 2-1 WLCS\_CUSTOMER Table Metadata (Continued)**

| <b>Column Name</b>           | <b>Data Type</b> | <b>Description and Recommendations</b>  |
|------------------------------|------------------|---|
| MAILING_CITY                 | VARCHAR2(30)     | The city in the customer's address.   |
| MAILING_STATE                | VARCHAR2(40)     | The state in the customer's address.  |
| MAILING_COUNTRY              | VARCHAR2(40)     | The country in the customer's address.  |
| MAILING_POBOX                | VARCHAR2(30)     | The post office box in the customer's address.  |
| MAILING_COUNTY               | VARCHAR2(50)     | The county in the customer's address.   |
| MAILING_POSTAL_CODE          | VARCHAR2(10)     | The postal (zip) code in the customer's address.  |
| MAILING_POSTAL_CODE<br>_TYPE | VARCHAR2(10)     | Format or type of postal code, generally determined by country (such as zip code in the United States). |

## The WLCS\_SHIPPING\_ADDRESS Database Table

Table 2-2 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.

**Table 2-2 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         | VARCHAR2 ( 20 ) | A unique identifier for the customer.  |
| MAP_KEY             | VARCHAR2 ( 30 ) | Key that maps multiple shipping addresses with a single customer.  |
| SHIPPING_GEOCODE    | VARCHAR2 ( 2 )  | The code used by the TAXWARE system to identify taxes for the order based on jurisdiction.   |
| SHIPPING_STREET1    | VARCHAR2 ( 30 ) | The first line in the customer's shipping address.   |
| SHIPPING_STREET2    | VARCHAR2 ( 30 ) | The second line in the customer's shipping address.  |
| SHIPPING_CITY       | VARCHAR2 ( 30 ) | The city in the customer's shipping address.   |
| SHIPPING_STATE      | VARCHAR2 ( 40 ) | The state in the customer's shipping address.  |
| SHIPPING_COUNTRY    | VARCHAR2 ( 40 ) | The country in the customer's shipping address.  |
| SHIPPING_POBOX      | VARCHAR2 ( 30 ) | The post office box in the customer's shipping address.  |

*The WLCS\_SHIPPING\_ADDRESS Database Table*

**Table 2-2 WLCS\_SHIPPING\_ADDRESS Table Metadata (Continued)**

| <b>Column Name</b>            | <b>Data Type</b> | <b>Description and Recommendations</b>  |
|-------------------------------|------------------|---|
| SHIPPING_COUNTY               | VARCHAR2(50)     | The county in the customer's shipping address.  |
| SHIPPING_POSTAL_CODE          | VARCHAR2(10)     | The postal (zip) code in the customer's shipping address.   |
| SHIPPING_POSTAL_CODE<br>_TYPE | VARCHAR2(10)     | Format or type of postal code, generally determined by country (such as zip code in the United States). |

## The WLCS\_CREDIT\_CARD Database Table

Table 2-3 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.

**Table 2-3 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.                 |
| CUSTOMER_ID       | VARCHAR2 ( 20 )  | A unique identifier for the customer.  |
| MAP_KEY           | VARCHAR2 ( 20 )  | Key that maps multiple credit cards with a single customer.  |
| CC_NUMBER         | VARCHAR2 ( 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           | VARCHAR2 ( 20 )  | The customer's credit card type (VISA, MasterCard, and so on).   |
| CC_EXP_DATE       | DATE             | The expiration date on the customer's credit card.   |
| CC_NAME           | VARCHAR2 ( 50 )  | The credit card holder's name.   |
| CC_DISPLAY_NUMBER | VARCHAR2 ( 20 )  | The version of the credit card number that is displayed (all Xs except last 4-digits).   |
| CC_COMPANY        | VARCHAR2 ( 50 )  | The name of the credit card company.   |
| BILLING_GEOCODE   | VARCHAR2 ( 2 )   | The code used by the TAXWARE system to identify taxes for the order based on jurisdiction.   |

The WLCS\_CREDIT\_CARD Database Table

**Table 2-3 WLCS\_CREDIT\_CARD Table Metadata (Continued)**

| <b>Column Name</b>           | <b>Data Type</b> | <b>Description and Recommendations</b>  |
|------------------------------|------------------|---|
| BILLING_STREET1              | VARCHAR2(30)     | The first line in the customer's billing address.   |
| BILLING_STREET2              | VARCHAR2(30)     | The second line in the customer's billing address.  |
| BILLING_CITY                 | VARCHAR2(30)     | The city in the customer's billing address.   |
| BILLING_STATE                | VARCHAR2(40)     | The state in the customer's billing address.  |
| BILLING_COUNTRY              | VARCHAR2(40)     | The country in the customer's billing address.  |
| BILLING_POBOX                | VARCHAR2(30)     | The post office box in the customer's billing address.  |
| BILLING_COUNTY               | VARCHAR2(50)     | The county in the customer's billing address.   |
| BILLING_POSTAL_CODE          | VARCHAR2(10)     | The postal (zip) code in the customer's billing address.  |
| BILLING_POSTAL_CODE<br>_TYPE | VARCHAR2(10)     | Format or type of postal code, generally determined by country (such as zip code in the United States). |

## The WLCS\_TRANSACTION Database Table

Table 2-4 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.

**Table 2-4 WLCS\_TRANSACTION Table Metadata**

| Column Name    | Data Type         | Description and Recommendations   |
|----------------|-------------------|---|
| TRANSACTION_ID | VARCHAR2 ( 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       | VARCHAR2 ( 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 (date on which the transaction was first started).  |
| TRAN_STATUS    | VARCHAR2 ( 20 )   | The current status of the transaction (Settled, Authorized, MarkedForSettle, PendingSettle, Retry, 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  | VARCHAR2 ( 30 )   | The currency of the transaction.  |
| CC_NUMBER      | VARCHAR2 ( 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        | VARCHAR2 ( 20 )   | The customer's credit card type (VISA, MasterCard, and so on).  |



The WLCS\_TRANSACTION Database Table

**Table 2-4 WLCS\_TRANSACTION Table Metadata (Continued)**

| <b>Column Name</b> | <b>Data Type</b> | <b>Description and Recommendations</b>  |
|--------------------|------------------|---|
| CC_EXP_DATE        | DATE             | The expiration date on the customer's credit card.  |
| CC_NAME            | VARCHAR2(50)     | The credit card holder's name.  |
| CC_DISPLAY_NUMBER  | VARCHAR2(20)     | The version of the credit card number that is displayed (all Xs except last 4-digits).                  |
| CC_COMPANY         | VARCHAR2(50)     | The name of the credit card company.  |
| GEOCODE            | VARCHAR2(2)      | The code used by the TAXWARE system to identify taxes for the order based on jurisdiction.              |
| STREET1            | VARCHAR2(30)     | The first line in the customer's street address.  |
| STREET2            | VARCHAR2(30)     | The second line in the customer's street address.   |
| CITY               | VARCHAR2(30)     | The city in the customer's address.   |
| STATE              | VARCHAR2(40)     | The state in the customer's address.  |
| COUNTRY            | VARCHAR2(40)     | The country in the customer's address.  |
| POBOX              | VARCHAR2(30)     | The post office box in the customer's address.  |
| COUNTY             | VARCHAR2(50)     | The county in the customer's address.   |
| POSTAL_CODE        | VARCHAR2(10)     | The postal (zip) code in the customer's address.  |
| POSTAL_CODE_TYPE   | VARCHAR2(10)     | Format or type of postal code, generally determined by country (such as zip code in the United States). |
| DESCRIPTION        | VARCHAR2(30)     | Any additional data. Can be NULL.   |

## The WLCS\_TRANSACTION\_ENTRY Database Table

Table 2-5 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.

**Table 2-5 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  | VARCHAR2 ( 30 )   | Represents the running count per transaction.  |
| TRAN_ENTRY_DATE      | DATE              | The date of the log entry.   |
| TRAN_ENTRY_STATUS    | VARCHAR2 ( 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  | VARCHAR2 ( 30 )   | The currency of the transaction.   |
| TRANSACTION_ID       | VARCHAR2 ( 25 )   | A unique identifier for the transaction.   |

## The WLCS\_SAVED\_ITEM\_LIST Database Table

Table 2-6 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 2-6 WLCS\_SAVED\_ITEM\_LIST Table Metadata**

| <b>Column Name</b> | <b>Data Type</b> | <b>Description and Recommendations</b>                                     |
|--------------------|------------------|--|
| CUSTOMER_ID        | VARCHAR2 ( 20 )  | A unique identifier for the customer.                                      |
| SKU                | VARCHAR2 ( 40 )  | A unique identifier (the "Stock Keeping Unit," or SKU) for a product item. |

## The WLCS\_ORDER Database Table

Table 2-7 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.

**Table 2-7 WLCS\_ORDER Table Metadata**

| Column Name       | Data Type         | Description and Recommendations  |
|-------------------|-------------------|--|
| ORDER_ID          | VARCHAR2 ( 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       | VARCHAR2 ( 20 )   | A unique identifier for the customer.  |
| TRANSACTION_ID    | VARCHAR2 ( 25 )   | A unique identifier for the transaction.   |
| STATUS            | VARCHAR2 ( 20 )   | The status of the order.   |
| ORDER_DATE        | DATE              | The date the order was placed.   |
| SHIPPING_METHOD   | VARCHAR2 ( 40 )   | The method by which the order is to be shipped.  |
| SHIPPING_AMOUNT   | NUMBER ( 16 , 4 ) | The shipping amount for the order.   |
| SHIPPING_CURRENCY | VARCHAR2 ( 10 )   | The currency associated with the shipping amount.  |
| PRICE_AMOUNT      | NUMBER ( 16 , 4 ) | The price of the order.  |
| PRICE_CURRENCY    | VARCHAR2 ( 10 )   | The currency associated with the price.  |
| SHIPPING_GEOGODE  | VARCHAR2 ( 2 )    | The code used by the TAXWARE system to identify taxes for the order based on jurisdiction.   |

**Table 2-7 WLCS\_ORDER Table Metadata (Continued)**

| <b>Column Name</b>            | <b>Data Type</b> | <b>Description and Recommendations</b>  |
|-------------------------------|------------------|---|
| SHIPPING_STREET1              | VARCHAR2(30)     | The first line in the customer's shipping address.  |
| SHIPPING_STREET2              | VARCHAR2(30)     | The second line in the customer's shipping address.   |
| SHIPPING_CITY                 | VARCHAR2(30)     | The city in the customer's shipping address.  |
| SHIPPING_STATE                | VARCHAR2(40)     | The state in the customer's shipping address.   |
| SHIPPING_COUNTRY              | VARCHAR2(40)     | The country in the customer's shipping address.   |
| SHIPPING_POBOX                | VARCHAR2(30)     | The post office box in the customer's shipping address.   |
| SHIPPING_COUNTY               | VARCHAR2(50)     | The county in the customer's shipping address.  |
| SHIPPING_POSTAL_CODE          | VARCHAR2(10)     | The postal (zip) code in the customer's shipping address.   |
| SHIPPING_POSTAL_CODE<br>_TYPE | VARCHAR2(10)     | Format or type of postal code, generally determined by country (such as zip code in the United States). |
| SPECIAL_INSTRUCTIONS          | VARCHAR2(255)    | Any special shipping instructions associated with the order.  |
| SPLITTING_PREFERENCE          | VARCHAR2(255)    | The splitting preferences for the customer's order.   |

## The WLCS\_ORDER\_LINE Database Table

Table 2-8 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.

**Table 2-8 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          | VARCHAR2 ( 40 )   | An identification number for the item in the shopping cart.  |
| TAX_AMOUNT          | NUMBER ( 16 , 4 ) | The tax amount for the order.  |
| TAX_CURRENCY        | VARCHAR2 ( 10 )   | The currency associated with the tax amount.   |
| SHIPPING_AMOUNT     | NUMBER ( 16 , 4 ) | The shipping amount for the order.   |
| SHIPPING_CURRENCY   | VARCHAR2 ( 10 )   | The currency associated with the shipping amount.  |
| UNIT_PRICE_AMOUNT   | NUMBER ( 16 , 4 ) | The unit price amount for the item.  |
| UNIT_PRICE_CURRENCY | VARCHAR2 ( 10 )   | The currency associated with the unit price.   |
| MSRP_AMOUNT         | NUMBER ( 16 , 4 ) | The MSRP amount for the item.  |
| MSRP_CURRENCY       | VARCHAR2 ( 10 )   | The currency associated with the MSRP amount.  |
| DESCRIPTION         | VARCHAR2 ( 255 )  | The name of the item that is part of the order.  |
| ORDER_ID            | VARCHAR2 ( 20 )   | A unique identifier for the order.   |

# The WLCS\_SHIPPING\_METHOD Database Table

Table 2-9 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.

**Table 2-9 WLCS\_SHIPPING\_METHOD Table Metadata**

| Column Name           | Data Type     | Description and Recommendations  |
|-----------------------|---------------|--|
| PK_IDENTIFIER         | VARCHAR2(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               | VARCHAR2(40)  | The carrier being used to ship the order (such as UPS, FedEx, and so on).  |
| METHOD                | VARCHAR2(40)  | The method by which the order is to be shipped (such as air, 2nd day air, parcel post, and so on).   |
| 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        | VARCHAR2(10)  | The currency associated with the PRICE_VALUE column (such as dollars, pounds, lira, and so on).  |
| WEIGHT_LIMIT          | NUMBER(16,4)  | The weight limit for the shipment.   |
| RESTRICTIONS          | VARCHAR2(255) | Any restrictions associated with the shipment.   |
| DESCRIPTION           | VARCHAR2(255) | 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.  |

## 2 The Order Processing Database Schema

---

**Table 2-9 WLCS\_SHIPPING\_METHOD Table Metadata (Continued)**

| <b>Column Name</b>     | <b>Data Type</b>  | <b>Description and Recommendations</b>   |
|------------------------|-------------------|--|
| 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         | VARCHAR2 ( 50 )   | The packaging type for the shipment.   |



---

## The WLCS\_SECURITY Database Table

Table 2-10 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 2-10 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  | VARCHAR2(2000) | The public key to be used for encryption/decryption of credit cards.                            |
| PRIVATE_KEY | VARCHAR2(2000) | The private key to be used for encryption/decryption of credit cards.                           |

## The SQL Files and Defined Constraints

The WebLogic Commerce Server product provides two SQL files to create the Cloudscape and Oracle versions of the order processing database. The SQL files are in the `WL_COMMERCE_HOME\db\<database-vendor>\wlcs\` directories.

`WL_COMMERCE_HOME` is the directory in which you installed the WebLogic Commerce Server software, and the `<database-vendor>` directory is either `cloudscape` or `oracle`. The files are:

- `create-order-cloudscape.sql`
- `create-order-oracle.sql`

You can run the `create-*` procedure for the desired database vendor type by invoking one of the following procedures in the `WL_COMMERCE_HOME\db\` directory:

- `create-all-cloudscape.bat` (Windows) or `create-all-cloudscape.sh` (UNIX)
- `create-all-oracle.bat` (Windows) or `create-all-oracle.sh` (UNIX)

**Note:** You can also create just the WebLogic Commerce Server or WebLogic Personalization Server specific databases. Simply substitute `wlcs` or `wlps` for `all` in the procedures shown above.

In each `create-order-*` SQL file, the database tables described earlier in this chapter are created. In addition, the SQL files define constraints. Table 2-11 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 2-11 Constraints Defined on Order Database Tables**

| <b>Table Name</b>      | <b>Constraints as Defined in create-order-oracle.sql</b>   |
|------------------------|--|
| 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:<br/>CONSTRAINT CUSTOMER_FK REFERENCES<br/>WLCS_CUSTOMER(CUSTOMER_ID) ON DELETE CASCADE</p>                      |
| 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:<br/>CONSTRAINT CUSTOMER_CREDIT_CARD_FK REFERENCES<br/>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:<br/>CONSTRAINT WLCS_TRANSACTION_FK REFERENCES<br/>WLCS_TRANSACTION(TRANSACTION_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:<br/>CONSTRAINT ORDER_FK REFERENCES<br/>WLCS_ORDER(ORDER_ID) ON DELETE CASCADE</p>                                   |

## **2** *The Order Processing Database Schema*

---

# 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

## JavaServer Pages (JSPs)

The Order Processing package 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 [Summary of JSP Templates](#) document.

### shoppingcart.jsp Template

The `shoppingcart.jsp` template (shown in Figure 3-1 and Figure 3-2) 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 3-1 and Figure 3-2 show annotated versions of the `shoppingcart.jsp` template; the first shows the page for a customer who has not logged in, the second shows the page for a customer who has logged in. The dashed lines and numbers in the diagram are not part of the template; they are referenced in the explanation that follows the screen shot.

### 3 Shopping Cart Management Services

Figure 3-1 Annotated shoppingcart.jsp Template - Customer Not Logged In

The screenshot shows the BEA WebLogic Commerce Server shopping cart interface. At the top, there is a navigation bar with the BEA logo, the text "BEA WebLogic Commerce Server 3.2", and links for "Home", "View Profile", "Search", "View Cart", "Administration", and "Template Index". Below the navigation bar, the page title is "Shopping Cart". A message reads: "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." There are two buttons: "Empty cart" and "Check out >".

The main content area is titled "Shopping Cart" and contains a table with the following data:

| Quantity | Item                            | List Price | Our Price | You Save | Subtotal              |        |
|----------|---------------------------------|------------|-----------|----------|-----------------------|--------|
| 1        | <a href="#">grinder-9-27723</a> | USD79.95   | USD76.95  | USD3.00  | USD76.95              | Delete |
|          |                                 |            |           |          | <b>Total USD76.95</b> |        |

Below the table, there is a message: "Press this button to Update if you changed any quantities." and an "Update" button. At the bottom of the main content area, there are two buttons: "Continue shopping" and "Check out >".

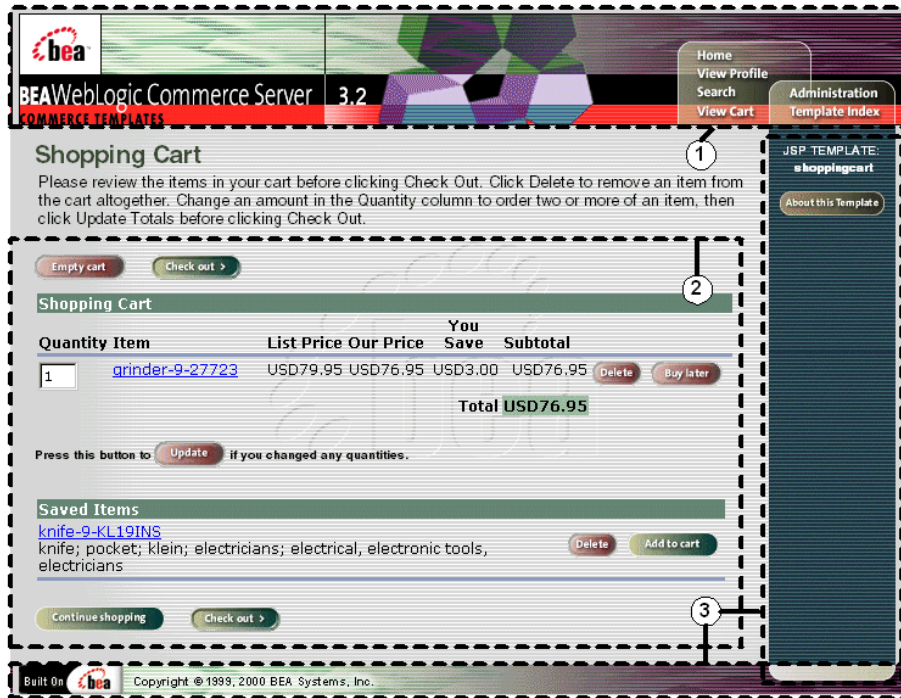
On the right side of the page, there is a sidebar with the text "JSP TEMPLATE: shoppingcart" and a link "About this Template".

At the bottom of the page, there is a footer with the text "Built On BEA Copyright © 1998, 2000 BEA Systems, Inc." and the BEA logo.

Annotations are present: a circled '1' points to the introductory text; a circled '2' points to the table; and a circled '3' points to the footer.



Figure 3-2 Annotated shoppingcart.jsp Template - Customer Logged In



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

1. The page header (top banner) is created from an import of the `header2.jsp` template. This is standard across many of the JSP templates provided by the WebLogic Commerce Server. The import call is:
 

```
<%@ include file="/commerce/includes/header2.jsp" %>
```
2. This region is the main content area for the page, which contains both dynamically generated data and static content. The dynamic content on `shoppingcart.jsp` is generated using WebLogic Server and Pipeline JSP tags that obtain and display the contents of both the active shopping cart and Saved Item list. For the `shoppingcart.jsp` template, the form posts include Delete, Buy Later, and Add to Cart (all per item), and Empty Cart, Check Out, Update, and Continue Shopping.

### 3 Shopping Cart Management Services

---

3. The `shoppingcart.jsp` template's content in region 3 contains the included `footer2.jsp` template. The include call in `shoppingcart.jsp` is:

```
<%@ include file="/commerce/includes/footer2.jsp" %>
```

`footer2.jsp` consists of the horizontal footer at the bottom of the page, plus the right-side vertical column that describes (for the benefit of you and your development team) the name of the current template and links to its *About* information. In the `footer2.jsp` file, the right-side vertical column is an include file:

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

#### 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%\server\webapps\wlcs\commerce\shoppingcart.jsp  
(Windows)
```

```
$WL_COMMERCE_HOME/server/webapps/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" %>
```

**Note:** For more information on the WebLogic Server JSP tags, see [“JSP Tag Reference”](#) in the *WebLogic Personalization Server* documentation. For more information about the Pipeline JSP tags, see [WebLogic Commerce Server Webflow and Pipeline Management](#).

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

## 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 [WebLogic Commerce Server Registration and User Processing Package](#).

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 Totals, 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 the Order Processing Package” on page 1-1.

### 3 Shopping Cart Management Services

---

#### Included JSP Templates

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

- `header2.jsp`, which creates the top banner.
- `footer2.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

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

**Table 3-1** `shoppingcart.jsp` Events

| Event   | Webflow Response(s)                           |
|---|---|
| --  | InitShoppingCartIP                            |
| --  | RefreshSavedList                              |
| <code>button(checkout)</code>                     | InitShippingMethodListIP                      |
| <code>button(deleteItemFromShoppingCart)</code>   | DeleteProductItemFromShoppingCartIP           |
| <code>button(deleteItemFromSavedList)</code>      | UpdateSkuIP<br>DeleteProductItemFromSavedList |
| <code>button(emptyShoppingCart)</code>            | EmptyShoppingCartIP                           |
| <code>button(moveItemToSavedList)</code>          | UpdateSkuIP<br>MoveProductItemToSavedList     |
| <code>button(moveItemToShoppingCart)</code>       | UpdateSkuIP<br>MoveProductItemToShoppingCart  |
| <code>button(updateShoppingCartQuantities)</code> | 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 3-18.

**Table 3-2 Shopping Cart Pipelines**

| Pipeline                                    | Description  |
|---|--|
| <code>RefreshSavedList</code>               | Contains <code>RefreshSavedListPC</code> and is not transactional.           |
| <code>DeleteProductItemFromSavedList</code> | Contains <code>DeleteProductItemFromSavedListPC</code> and is transactional. |
| <code>MoveProductItemToSavedList</code>     | Contains <code>MoveProductItemToSavedListPC</code> and is transactional.     |
| <code>MoveProductItemToShoppingCart</code>  | Contains <code>MoveProductItemToShoppingCartPC</code> 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 *WebLogic Commerce Server Product Catalog Management* documentation.

## 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.

### 3 Shopping Cart Management Services

---

**Table 3-3 shoppingcart.jsp Pipeline Session Attributes**

| Attribute  | Type  | Description   |
|--|---|---|
| PipelineSessionConstants<br>.SAVED_SHOPPING_CART | com.beasys.commerce.ebusiness<br>.shoppingcart.ShoppingCart | The saved shopping cart<br>(source of the saved items). |
| PipelineSessionConstants<br>.SHOPPING_CART       | com.beasys.commerce.ebusiness<br>.shoppingcart.ShoppingCart | The currently active<br>shopping cart.                  |

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

#### **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 [WebLogic Commerce Server Webflow and Pipeline Management](#).

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> ). |

### 3-10 Order Processing Package

**Table 3-4 ShoppingCart Accessor Methods/Attributes (Continued)**

| Method/Attribute                     | Description   |
|--------------------------------------|---|
| <code>getTotal(int totalType)</code> | <p>The total amount specified by the <code>totalType</code> parameter. Valid parameters include:</p> <p><code>ShoppingCartConstants.LINE_UNIT_PRICE_TIMES_Q<br/>UANTITY</code><br/> <code>ShoppingCartConstants.LINE_SHIPPING</code><br/> <code>ShoppingCartConstants.LINE_TAX</code></p> <p><b>Note:</b> The <code>getTotal()</code> method also allows you to combine different total types. For more information, see the <a href="#">Javadoc</a>.</p> |

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   |
|--|---|
| <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>getLineTotal(int totalType)</code> | <p>The total amount specified by the <code>totalType</code> parameter. Valid parameters include:</p> <p><code>ShoppingCartConstants.LINE_UNIT_PRICE_T<br/>IMES_QUANTITY</code><br/> <code>ShoppingCartConstants.LINE_SHIPPING</code><br/> <code>ShoppingCartConstants.LINE_TAX</code></p> <p><b>Note:</b> The <code>getLineTotal()</code> method also allows you to combine different total types. For more information, see the <a href="#">Javadoc</a>.</p> |

### 3 Shopping Cart Management Services

---

Listing 3-2 illustrates how these accessor methods/attributes are used within Java scriptlets.

**Note:** The `ProductItem` object is described in the [WebLogic Commerce Server Product Catalog Management](#) document.

#### 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>

  <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 *WebLogic Personalization Server* documentation.



## 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_&lt;SKU&gt;</code><br>Where <code>&lt;SKU&gt;</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

|   |   |
|---|---|
| <b>Class Name</b>                                     | <code>com.beasys.commerce.ebusiness.shoppingcart.webflow.DeleteProductItemFromShoppingCartIP</code>                                     |
| <b>Description</b>                                    | Removes the item from the shopping cart.  |
| <b>Required<br/>HttpServletRequest<br/>Parameters</b> | <code>HttpRequestConstants.CATALOG_ITEM_SKU</code>  |
| <b>Required Pipeline<br/>Session Attributes</b>       | <code>PipelineSessionConstants.SHOPPING_CART</code>   |
| <b>Updated Pipeline<br/>Session Attributes</b>        | <code>PipelineSessionConstants.SHOPPING_CART</code>   |
| <b>Removed Pipeline<br/>Session Attributes</b>        | None  |
| <b>Validation</b>                                     | None  |
| <b>Exceptions</b>                                     | <code>ProcessingException</code> , thrown if the required request parameters or required Pipeline session attributes are not available. |

## EmptyShoppingCartIP

|   |  |
|---|--|
| <b>Class Name</b>                                     | <code>com.beasys.commerce.ebusiness.shoppingcart.webflow.EmptyShoppingCartIP</code>                    |
| <b>Description</b>                                    | Creates a new shopping cart and stores it in the Pipeline session. The old shopping cart is discarded. |
| <b>Required<br/>HttpServletRequest<br/>Parameters</b> | None   |
| <b>Required Pipeline<br/>Session Attributes</b>       | None   |
| <b>Updated Pipeline<br/>Session Attributes</b>        | <code>PipelineSessionConstants.SHOPPING_CART</code>  |
| <b>Removed Pipeline<br/>Session Attributes</b>        | None   |
| <b>Validation</b>                                     | None   |
| <b>Exceptions</b>                                     | None   |

## InitShoppingCartIP

|   |   |
|---|---|
| <b>Class Name</b>                                     | <code>com.beasys.commerce.ebusiness.shoppingcart.webflow.InitShoppingCartIP</code>  |
| <b>Description</b>                                    | 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. |
| <b>Required<br/>HttpServletRequest<br/>Parameters</b> | None  |

### 3 Shopping Cart Management Services

---

|   |  |
|---|--|
| <b>Required Pipeline Session Attributes</b> | None                                   |
| <b>Updated Pipeline Session Attributes</b>  | PipelineSessionConstants.SHOPPING_CART |
| <b>Removed Pipeline Session Attributes</b>  | None                                   |
| <b>Validation</b>                           | None                                   |
| <b>Exceptions</b>                           | None                                   |

## UpdateShoppingCartQuantitiesIP

|   |  |
|---|--|
| <b>Class Name</b>                             | com.beasys.commerce.ebusiness.shoppingcart.webflow.<br>UpdateShoppingCartQuantitiesIP  |
| <b>Description</b>                            | 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. |
| <b>Required HTTPServletRequest Parameters</b> | NewQuantity_<SKU><br>Where <SKU> is replaced with the SKU of the item on the shopping cart line.   |
| <b>Required Pipeline Session Attributes</b>   | PipelineSessionConstants.SHOPPING_CART   |
| <b>Updated Pipeline Session Attributes</b>    | PipelineSessionConstants.SHOPPING_CART   |
| <b>Removed Pipeline Session Attributes</b>    | None   |
| <b>Validation</b>                             | Verifies that the quantity fields only contain positive integers.  |
| <b>Exceptions</b>                             | ProcessingException, thrown if the required request parameters or required Pipeline session attributes are not available.  |

---

## UpdateSkuIP

|   |   |
|---|---|
| <b>Class Name</b>                                     | <code>com.beasys.commerce.ebusiness.shoppingcart.webflow.UpdateSkuIP</code>                     |
| <b>Description</b>                                    | Reads the SKU from the HTTP request and places it into the Pipeline session.                    |
| <b>Required<br/>HttpServletRequest<br/>Parameters</b> | <code>HttpRequestConstants.CATALOG_ITEM_SKU</code>  |
| <b>Required Pipeline<br/>Session Attributes</b>       | None  |
| <b>Updated Pipeline<br/>Session Attributes</b>        | <code>PipelineSessionConstants.CATALOG_ITEM_SKU</code>  |
| <b>Removed Pipeline<br/>Session Attributes</b>        | None  |
| <b>Validation</b>                                     | None  |
| <b>Exceptions</b>                                     | <code>ProcessingException</code> , 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 *WebLogic Commerce Server Product Catalog Management* documentation.

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

### DeleteProductItemFromSavedListPC

|   |  |
|---|--|
| <b>Class Name</b>                           | <code>com.beasys.commerce.ebusiness.shoppingcart.pipeline.DeleteProductItemFromSavedListPC</code>  |
| <b>Description</b>                          | Removes the item from the saved list and updates the <code>WLCS_SAVED_ITEM_LIST</code> table in the database.  |
| <b>Required Pipeline Session Attributes</b> | <code>PipelineSessionConstants.CATALOG_ITEM_SKU</code><br><code>PipelineSessionConstants.SAVED_SHOPPING_CART</code><br><code>PipelineSessionConstants.USER_NAME</code> |
| <b>Updated Pipeline Session Attributes</b>  | <code>PipelineSessionConstants.SAVED_SHOPPING_CART</code>  |
| <b>Removed Pipeline Session Attributes</b>  | None   |
| <b>Type</b>                                 | Session bean   |
| <b>JNDI Name</b>                            | <code>com.beasys.commerce.ebusiness.shoppingcart.pipeline.DeleteProductItemFromSavedListPC</code>  |

---

|                   |   |
|-------------------|---|
| <b>Exceptions</b> | PipelineFatalException, thrown if the required Pipeline session attributes are not available. |
|-------------------|---|

---

## MoveProductItemToSavedListPC

---

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

---

## MoveProductItemToShoppingCartPC

|   |   |
|---|---|
| <b>Class Name</b>                           | <code>com.beasys.commerce.ebusiness.shoppingcart.pipeline.MoveProductItemToShoppingCartPC</code>  |
| <b>Description</b>                          | 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.  |
| <b>Required Pipeline Session Attributes</b> | <code>PipelineSessionConstants.CATALOG_ITEM_SKU</code><br><code>PipelineSessionConstants.SAVED_SHOPPING_CART</code><br><code>PipelineSessionConstants.SHOPPING_CART</code><br><code>PipelineSessionConstants.USER_NAME</code> |
| <b>Updated Pipeline Session Attributes</b>  | <code>PipelineSessionConstants.SAVED_SHOPPING_CART</code><br><code>PipelineSessionConstants.SHOPPING_CART</code>  |
| <b>Removed Pipeline Session Attributes</b>  | None  |
| <b>Type</b>                                 | Session bean  |
| <b>JNDI Name</b>                            | <code>com.beasys.commerce.ebusiness.shoppingcart.pipeline.MoveProductItemToShoppingCartPC</code>  |
| <b>Exceptions</b>                           | <code>PipelineFatalException</code> , thrown if the required Pipeline session attributes are not available.   |



---

## RefreshSavedListPC

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

### **3** *Shopping Cart Management Services*

---

**3-22** Order Processing Package

# 4 Shipping Services

The Order Processing package's 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 (JSPs)
  - shipping.jsp Template
  - selectaddress.jsp Template
  - addaddress.jsp Template
- Input Processors
  - InitShippingMethodListIP
  - UpdateShippingAddressIP
  - ValidateAddressIP
  - ValidateShippingInfoIP
- Pipeline Components
  - AddShippingAddressPC
  - CalculateShippingPC
  - DeleteShippingAddressPC

# JavaServer Pages (JSPs)

The Order Processing package's Shipping 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 Server Web application and a listing of their locations in the directory structure, see the [Summary of JSP Templates](#) document.

## shipping.jsp Template

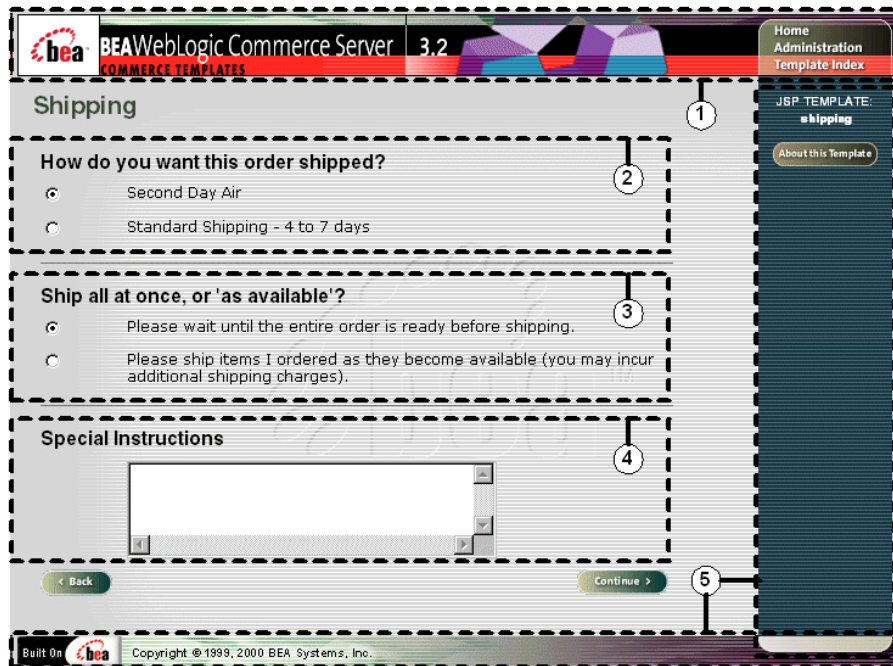
The `shipping.jsp` template (shown in Figure 4-1) allows the customer to select and input shipping details for the order. Shipping details include the shipping method (such as standard, second day air, and so on), shipping preference (all at once or as items become available) and any special shipping 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. The dashed lines and numbers in the diagram are not part of the template; they are referenced in the explanation that follows the screen shot.

Figure 4-1 Annotated shipping.jsp Template



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

1. The header (top banner) for inner pages is created from an import of the `innerheader.jsp` template. This is standard across many of the second-level JSP templates provided by WebLogic Commerce Server. The import call is:
 

```
<%@ include file="/commerce/includes/innerheader.jsp" %>
```
2. 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 3 and 4, the form then posts the customer's selected shipping method.

## 4 Shipping Services

---

3. 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 2 and 4, the form then posts the customer's selected splitting preference.
4. 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 2 and 3, the form then posts any special instructions the customer specifies.
5. The `shipping.jsp` template's content in region 5 of Figure 4-1 contains the included `innerfooter.jsp` template. The include call in `shipping.jsp` is:

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

innerfooter.jsp consists of the horizontal footer at the bottom of the page,
plus the right-side vertical column that describes (for the benefit of you and your
development team) the name of the current template and links to its About
information. In the innerfooter.jsp file, the right-side vertical column is an
include file:

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

### 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%\server\webapps\wlcs\commerce\order\
shipping.jsp (Windows)
$WL_COMMERCE_HOME/server/webapps/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 *WebLogic Personalization Server* documentation. For more information about the Pipeline JSP tags, see [WebLogic Personalization Server Webflow and Pipeline Management](#).

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 `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.axiom.units.*" %>
<%@ 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 [WebLogic Commerce Server Registration and User Processing Package](#).

For more information about the default Webflow, see “Overview of the Order Processing Package” on page 1-1.

## 4 Shipping Services

---

### Included JSP Templates

The following JSP templates are included in the `shipping.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 `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 `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> .                             |

### 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                                |
|---|---|--|
| PipelineSessionConstants<br>.SHIPPING_METHOD_LIST | List of<br>com.beasys.commerce.ebusiness<br>.shipping.ShippingMethodValue | The list of available shipping<br>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 [WebLogic Commerce Server Webflow and Pipeline Management](#).

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                                  |
|--------------------------|--|
| <code>description</code> | A description of the shipping method.        |
| <code>identifier</code>  | Key in the database for the shipping method. |

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

## 4 Shipping Services

---

### 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 *WebLogic Personalization Server* documentation.

### 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>HttpServletRequest.SHIPPING_METHOD</code>      | Radio button | Identifies the shipping method the customer selects.   |
| <code>HttpServletRequest.SPECIAL_INSTRUCTIONS</code> | Textbox      | Any special instructions the customer specifies.   |
| <code>HttpServletRequest.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 `<%= HttpServletRequest.SPLITTING_PREFERENCE %>`) for use in the JSP.

### selectaddress.jsp Template

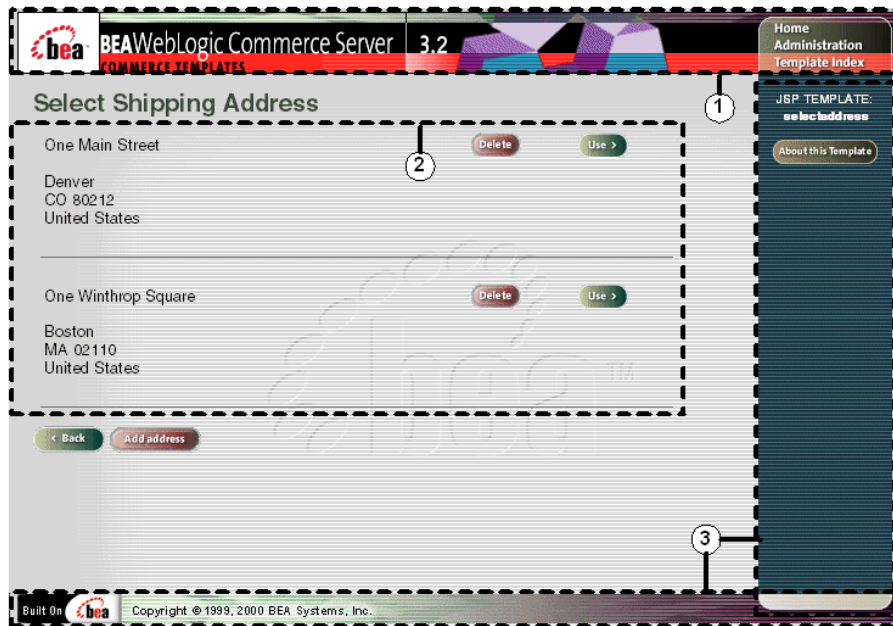
The `selectaddress.jsp` template (shown in Figure 4-2) displays a list of shipping addresses that have previously been associated with the customer. If the customer clicks the Use button associated with a particular address, that address will be used as the shipping address and the customer will continue 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 dashed lines and numbers in the diagram are not part of the template; they are referenced in the explanation that follows the screen shot.

Figure 4-2 Annotated selectaddress.jsp Template



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

1. The header (top banner) for inner pages is created from an import of the `innerheader.jsp` template. This is standard across many of the second-level JSP templates provided by *WebLogic Commerce Server*. The import call is:
 

```
<%@ include file="/commerce/includes/innerheader.jsp" %>
```
2. This 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 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.

**Note:** 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 4-19.

## 4 Shipping Services

---

3. The `selectaddress.jsp` template's content in region 3 contains the included `innerfooter.jsp` template. The include call in `selectaddress.jsp` is:

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

`innerfooter.jsp` consists of the horizontal footer at the bottom of the page, plus the right-side vertical column that describes (for the benefit of you and your development team) the name of the current template and links to its *About* information. In the `innerfooter.jsp` file, the right-side vertical column is an include file:

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

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

### Tag Library Imports

The `selectaddress.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"%>
```

**Note:** For more information on the WebLogic Server JSP tags or the WebLogic Personalization Server JSP tags, see [“JSP Tag Reference”](#) in the *WebLogic Personalization Server* documentation. For more information about the Pipeline JSP tags, see [WebLogic Commerce Server Webflow and Pipeline Management](#).

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 `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 the Order Processing Package” on page 1-1.

## Included JSP Templates

The following JSP templates are included in the `selectaddress.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 `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.

## 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

| <b>Event</b>                               | <b>Web Flow Response(s)</b>   |
|--|---|
| <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><br><code>DeleteShippingAddress</code>   |
| <code>button(useShippingAddress)</code>    | <code>UpdateShippingAddressIP</code><br><code>TaxVerifyShippingAddress</code><br><code>CalculateShippingCost</code><br><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 4-29.

**Table 4-6** Select Shipping Address Pipelines

| <b>Pipeline</b>                       | <b>Description</b>   |
|---------------------------------------|--|
| <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'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's JSP tags, see [“JSP Tag Reference”](#) in the *WebLogic Personalization Server* documentation.

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>getCounty()</code>     | The county 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>
  <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 dashed lines and numbers in the diagram are not part of the template; they are referenced in the explanation that follows the screen shot.

**Figure 4-3** Annotated `addaddress.jsp` Template

The screenshot displays the `addaddress.jsp` template in a browser. At the top, the header shows the BEA WebLogic Commerce Server 3.2 logo and navigation links for Home, Administration, and Template Index. The main content area is titled "Add Shipping Address" and contains a form with the following fields:

- Street address (required, marked with \*)
- Address 2
- City (required, marked with \*)
- State (required, marked with \*)
- Zip/Postal Code (required, marked with \*)
- Country (required, marked with \*)

Below the form, a note states "Fields marked with (\*) are required." and there are "Back" and "Save" buttons. On the right side, a sidebar contains a "JSP TEMPLATE:" section with a link to "addaddress" and an "About this Template" button. Annotations 1, 2, and 3 are placed on the page to highlight the form area, the form fields, and the sidebar, respectively.

At the bottom of the page, the footer includes the BEA logo and the text "Copyright © 1999, 2000 BEA Systems, Inc."

## 4 Shipping Services

---

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

1. The header (top banner) for inner pages is created from an import of the `innerheader.jsp` template. This is standard across many of the second-level JSP templates provided by WebLogic Commerce Server. The import call is:

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

2. This 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" %>
```

3. The `addaddress.jsp` template's content in region 3 contains the included `innerfooter.jsp` template. The include call in `addaddress.jsp` is:

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

`innerfooter.jsp` consists of the horizontal footer at the bottom of the page, plus the right-side vertical column that describes (for the benefit of you and your development team) the name of the current template and links to its *About* information. In the `innerfooter.jsp` file, the right-side vertical column is an include file:

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

### 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%\server\webapps\wlcs\commerce\order\
addaddress.jsp (Windows)
$WL_COMMERCE_HOME/server/webapps/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 [WebLogic Commerce Server Webflow and Pipeline Management](#).

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 `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 the Order Processing Package” 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><br><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 4-29.



**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 *WebLogic Commerce Server Registration and User Processing Package* documentation.

## 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>HttpServletRequest.CUSTOMER_SHIPPING_ADDRESS2</code> | Textbox | The second line of the shipping street address. |
| <code>HttpServletRequest.CUSTOMER_SHIPPING_CITY</code>     | Textbox | The city in the shipping address.               |
| <code>HttpServletRequest.CUSTOMER_SHIPPING_STATE</code>    | Textbox | The state in the shipping address.              |
| <code>HttpServletRequest.CUSTOMER_SHIPPING_ZIPCODE</code>  | Textbox | The zip/postal code in the shipping address.    |
| <code>HttpServletRequest.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 `<%= HttpServletRequest.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

|   |   |
|---|---|
| <b>Class Name</b>                                     | <code>com.beasys.commerce.ebusiness.shipping.webflow.<br/>InitShippingMethodListIP</code>   |
| <b>Description</b>                                    | 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. |
| <b>Required<br/>HttpServletRequest<br/>Parameters</b> | None  |
| <b>Required Pipeline<br/>Session Attributes</b>       | None  |
| <b>Updated Pipeline<br/>Session Attributes</b>        | <code>PipelineSessionConstants.SHIPPING_METHOD_LIST</code>  |
| <b>Removed Pipeline<br/>Session Attributes</b>        | None  |
| <b>Validation</b>                                     | None  |
| <b>Exceptions</b>                                     | <code>ProcessingException</code> , thrown if the input processor cannot read the shipping method information from the database.   |

## UpdateShippingAddressIP

|   |   |
|---|---|
| <b>Class Name</b>                                     | <code>com.beasys.commerce.ebusiness.shipping.webflow.<br/>UpdateShippingAddressIP</code>                  |
| <b>Description</b>                                    | Updates the shipping address attribute in the Pipeline session based on the address the customer selects. |
| <b>Required<br/>HttpServletRequest<br/>Parameters</b> | <code>HTTPRequestConstants.ADDRESS_KEY</code>   |
| <b>Required Pipeline<br/>Session Attributes</b>       | None  |
| <b>Updated Pipeline<br/>Session Attributes</b>        | <code>PipelineSessionConstants.SHIPPING_ADDRESS</code>  |
| <b>Removed Pipeline<br/>Session Attributes</b>        | None  |
| <b>Validation</b>                                     | None  |
| <b>Exceptions</b>                                     | None  |

---

## ValidateAddressIP

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

---

# ValidateShippingInfoIP

|   |   |
|---|---|
| <b>Class Name</b>                                     | <code>com.beasys.commerce.ebusiness.shipping.webflow.ValidateShippingInfoIP</code>  |
| <b>Description</b>                                    | Places the shipping method, splitting preference, and special instructions into the Pipeline session.   |
| <b>Required<br/>HttpServletRequest<br/>Parameters</b> | <code>HttpRequestConstants.SHIPPING_METHOD</code><br><code>HttpRequestConstants.SPLITTING_PREFERENCE</code><br><code>HttpRequestConstants.SPECIAL_INSTRUCTIONS</code>             |
| <b>Required Pipeline<br/>Session Attributes</b>       | None  |
| <b>Updated Pipeline<br/>Session Attributes</b>        | <code>PipelineSessionConstants.SHIPPING_METHOD</code><br><code>PipelineSessionConstants.SPLITTING_PREFERENCE</code><br><code>PipelineSessionConstants.SPECIAL_INSTRUCTIONS</code> |
| <b>Removed Pipeline<br/>Session Attributes</b>        | None  |
| <b>Validation</b>                                     | Verifies that the required fields contain values.   |
| <b>Exceptions</b>                                     | <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

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

# CalculateShippingPC

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



---

## DeleteShippingAddressPC

|   |  |
|---|--|
| <b>Class Name</b>                           | <code>com.beasys.commerce.ebusiness.shipping.pipeline.DeleteShippingAddressPC</code>   |
| <b>Description</b>                          | Uses the address key in the Pipeline session to locate the correct customer shipping address, then removes it from the list.             |
| <b>Required Pipeline Session Attributes</b> | <code>PipelineSessionConstants.ADDRESS_KEY</code>  |
| <b>Updated Pipeline Session Attributes</b>  | None   |
| <b>Removed Pipeline Session Attributes</b>  | None   |
| <b>Type</b>                                 | Java object  |
| <b>JNDI Name</b>                            | None   |
| <b>Exceptions</b>                           | <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 Services provided in the Order Processing package 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 Order Processing package's 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 [Summary of JSP Templates](#) documentation.

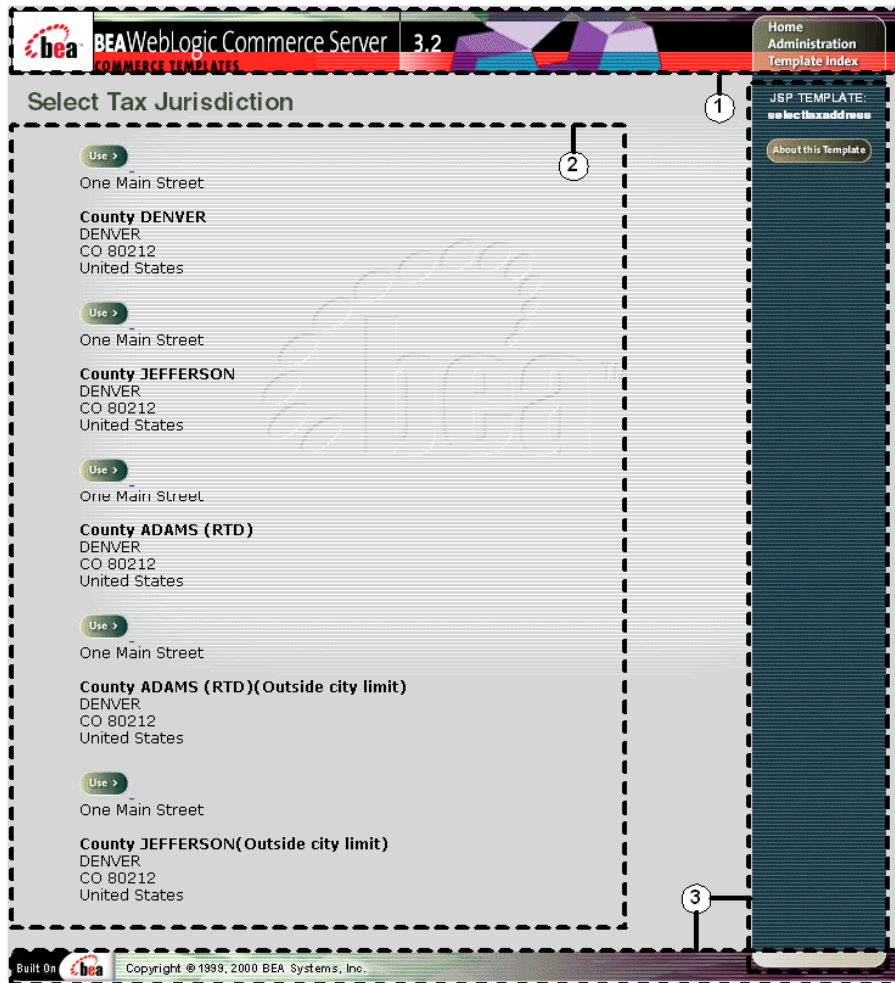
### selecttaxaddress.jsp Template

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 dashed lines and numbers in the diagram are not part of the template; they are referenced in the explanation that follows the screen shot.

Figure 5-1 Annotated selecttaxaddress.jsp Template



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

1. The page header (top banner) is created from an import of the `innerheader.jsp` template. This is standard across many of the JSP templates provided by WebLogic Commerce Server. The import call is:

## 5 Taxation Services

---

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

2. Region 2 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.
3. The `selecttaxaddress.jsp` template's content in region 3 contains the included `innerfooter.jsp` template. The include call in `selecttaxaddress.jsp` is:

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

`innerfooter.jsp` consists of the horizontal footer at the bottom of the page, plus the right-side vertical column that describes (for the benefit of you and your development team) the name of the current template and links to its *About* information. In the `innerfooter.jsp` file, the right-side vertical column is an include file:

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

### 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" %>
```

**Note:** For more information on the WebLogic Server JSP tags, see [“JSP Tag Reference”](#) in the *WebLogic Personalization Server* documentation. For more information about the Pipeline JSP tags, see [WebLogic Commerce Server Webflow and Pipeline Management](#).

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 the Order Processing Package” 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

## 5 Taxation Services

---

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<br><code>com.beasys.commerce.axiom<br/>.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 [WebLogic Commerce Server Webflow and Pipeline Management](#).

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 *WebLogic Personalization Server* documentation.

## 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.            |

**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

|   |  |
|---|--|
| <b>Class Name</b>                                     | <code>com.beasys.commerce.ebusiness.tax.webflow.<br/>DecideShippingAddressPageIP</code>  |
| <b>Description</b>                                    | 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> . |
| <b>Required<br/>HttpServletRequest<br/>Parameters</b> | None   |
| <b>Required Pipeline<br/>Session Attributes</b>       | <code>PipelineSessionConstants.SHIPPING_ADDRESS</code><br><code>PipelineSessionConstants.VERIZIP_SHIPPING_ADDRESSES</code>   |
| <b>Updated Pipeline<br/>Session Attributes</b>        | <code>PipelineSessionConstants.SHIPPING_ADDRESS</code> (in the case of a single address)   |
| <b>Removed Pipeline<br/>Session Attributes</b>        | None   |
| <b>Validation</b>                                     | None   |
| <b>Exceptions</b>                                     | <code>MultipleAddressFoundException</code> , thrown if the VERAZIP service returns more than one address.  |

---

## UpdateShippingAddressIP

|   |   |
|---|---|
| <b>Class Name</b>                                     | com.beasys.commerce.ebusiness.shipping.webflow.<br>UpdateShippingAddressIP                                    |
| <b>Description</b>                                    | Updates the shipping address attribute in the Pipeline session based on the tax address the customer selects. |
| <b>Required<br/>HttpServletRequest<br/>Parameters</b> | HTTPRequestConstants.TAX_SHIPPING_ADDRESS   |
| <b>Required Pipeline<br/>Session Attributes</b>       | PipelineSessionConstants.SHIPPING_ADDRESS<br>PipelineSessionConstants.VERIZIP_SHIPPING_ADDRESSES              |
| <b>Updated Pipeline<br/>Session Attributes</b>        | PipelineSessionConstants.SHIPPING_ADDRESS   |
| <b>Removed Pipeline<br/>Session Attributes</b>        | None  |
| <b>Validation</b>                                     | None  |
| <b>Exceptions</b>                                     | 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

|   |   |
|---|---|
| <b>Class Name</b>                           | <code>com.beasys.commerce.ebusiness.tax.pipeline.TaxCalculateLineLevelPC</code>   |
| <b>Description</b>                          | 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. |
| <b>Required Pipeline Session Attributes</b> | <code>PipelineSessionConstants.SHOPPING_CART</code><br><code>PipelineSessionConstants.SHIPPING_ADDRESS</code>   |
| <b>Updated Pipeline Session Attributes</b>  | <code>PipelineSessionConstants.SHOPPING_CART</code>   |
| <b>Removed Pipeline Session Attributes</b>  | None  |
| <b>Type</b>                                 | Java class  |
| <b>JNDI Name</b>                            | None  |
| <b>Exceptions</b>                           | None  |

## TaxCalculateAndCommitLineLevelPC

|   |   |
|---|---|
| <b>Class Name</b>                           | com.beasys.commerce.ebusiness.tax.pipeline.<br>TaxCalculateAndCommitLineLevelPC   |
| <b>Description</b>                          | 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. |
| <b>Required Pipeline Session Attributes</b> | PipelineSessionConstants.SHOPPING_CART<br>PipelineSessionConstants.SHIPPING_ADDRESS   |
| <b>Updated Pipeline Session Attributes</b>  | PipelineSessionConstants.SHOPPING_CART  |
| <b>Removed Pipeline Session Attributes</b>  | None  |
| <b>Type</b>                                 | Java class  |
| <b>JNDI Name</b>                            | None  |
| <b>Exceptions</b>                           | None  |

## TaxVerifyShippingAddressPC

|   |  |
|---|--|
| <b>Class Name</b>                           | com.beasys.commerce.ebusiness.tax.pipeline.<br>TaxVerifyShippingAddressPC  |
| <b>Description</b>                          | Ensures that the shipping address is descriptive enough to properly calculate taxation for an order based on jurisdiction. |
| <b>Required Pipeline Session Attributes</b> | PipelineSessionConstants.SHIPPING_ADDRESS  |
| <b>Updated Pipeline Session Attributes</b>  | PipelineSessionConstants.VERAZIP_SHIPPING_ADDRESSES  |

## 5 Taxation Services

---

---

|  |  |
|--|--|
| <b>Removed Pipeline Session Attributes</b> | None   |
| <b>Type</b>                                | Java class   |
| <b>JNDI Name</b>                           | None   |
| <b>Exceptions</b>                          | <p><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).</p> <p><code>TaxUserException</code>, thrown if processing could not occur due to invalid user input.</p> |

---



## 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 "Viewing Debugging Information in TAXWARE."

**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 2000 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 within 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   |
|-------------------------|---|
| \audit                  | Contains audit files for all tax transactions.  |
| \bin                    | Contains DLLs for SALES/USE, VERIZIP, and UTL, including avptax.dll, avpzip.dll, taxcommon.dll, and taxcommon0.dll. |
| \common\Taxware\classes | Contains Java classes for UTL, including taxmain.class and taxcommon.class.   |

**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 5-29.

## 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 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<br>/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 WORLD TAX 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`.

## 5 Taxation Services

---

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 5-29.

## 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.

## 5 Taxation Services

---

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.

## 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.businessLocationCode=London Office
```

---

**Note:** For the evaluation version of TAXWARE provided with this release, you must set `taxware.companyId` to "Widgets R Us" and `taxware.businessLocationCode` to "Boulder Loc". TAXWARE will not calculate taxes for addresses outside the US and Canada if `companyId` and `businessLocationCode` are not set to these values.

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 `businessLocationCode` parameter. Please see "Specifying Your Company's ID and Business Location" on page 5-25. 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<br>X1123456789<br>1X123456789<br>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.   |

**Table 5-7 Registration Number Formats (Continued)**

| Country       | Country Code | Example                  | Format   |
|---------------|--------------|--------------------------|--|
| Great Britain | GB           | 123456789<br>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.  |
| South Korea   | KR           |                          | No specific format is required.  |
| Spain         | ES           | 12345678X<br>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 weblogcommerce.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 weblogcommerce.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
```

```
#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.

## 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 |
|--|--------------------------|
| <b>Subdirectories:</b>                     | <b>Subdirectories:</b>   |
| \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.

## 5 Taxation Services

---

- 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 5-12.

**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 5-10.

### Listing 5-10 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 `WLCS` run-time configuration is shown in Listing 5-11.
2. Set the `TAXWARE`-specific environment variables to the correct data directories. For an example, see Listing 5-12.
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 5-11 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 5-12 TAXWARE Environment Variables on UNIX Systems (Sample TAXWARE Installation)**

---

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

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
```

## 5 Taxation Services

---

```
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.

### 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 5-29) 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 WORLD TAX components.

### Listing 5-13 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 the Order Processing package's 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 5-14.

#### Listing 5-14 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 5-15.

#### Listing 5-15 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, as shown in Listing 5-16. Notice that in the default configuration, the `TaxVerifyShippingAddress` Pipeline is invoked on successful execution of the `UpdateShippingAddressIP` input processor.

### Listing 5-16 Default Shipping Address Page in the `webflow.properties` File

---

```
# Select Shipping Address Page
...
...

SelectShippingAddress_UpdateShippingAddress.inputprocessor.
success=TaxVerifyShippingAddress.pipeline

...
```

---

3. Replace the `TaxVerifyShippingAddress.pipeline` with `CalculateShippingCost.pipeline`, so the Select Shipping Address Page section in the `webflow.properties` file is as shown in Listing 5-17.

### Listing 5-17 Shipping Address Page Without Tax Pipeline

---

```
# Select Shipping Address Page
...
...

SelectShippingAddress_UpdateShippingAddress.inputprocessor.
success=CalculateShippingCost.pipeline

...
```

---

4. Locate the success path for the `CalculateShippingCost` Pipeline in the `webflow.properties` file, as shown in Listing 5-18.

**Listing 5-18 Default Success Path for CalculateShippingCost Pipeline**

---

```
# Decide to prompt selecttaxaddress.jsp on basis of number of
# addresses returned by verazip

...

CalculateShippingCost.pipeline.success=TaxCalculateLineLevel.
pipeline

...
```

---

5. Replace the `TaxCalculateLineLevel` Pipeline with `commerce/order/payment.jsp`, so the success path for the `CalculateShippingCost` Pipeline is as shown in Listing 5-19.

**Listing 5-19 Success Path for CalculateShippingCost Pipeline Without Tax Pipeline**

---

```
# Decide to prompt selecttaxaddress.jsp on basis of number of
# addresses returned by verazip

...

CalculateShippingCost.pipeline.success=commerce/order/payment.jsp

...
```

---

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 5-34), write new Pipeline components to handle tax calculations using the new tax provider, and integrate these Pipeline components into the Webflow/Pipeline infrastructure.

**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

The Order Processing package 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
  - Encryption/Decryption Implementation
  - Customizable Security Settings
  - Methods for Supplying the Private Key Encryption Password

## JavaServer Pages (JSPs)

A primary goal of the WebLogic Commerce Server's Order Processing package 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 [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.

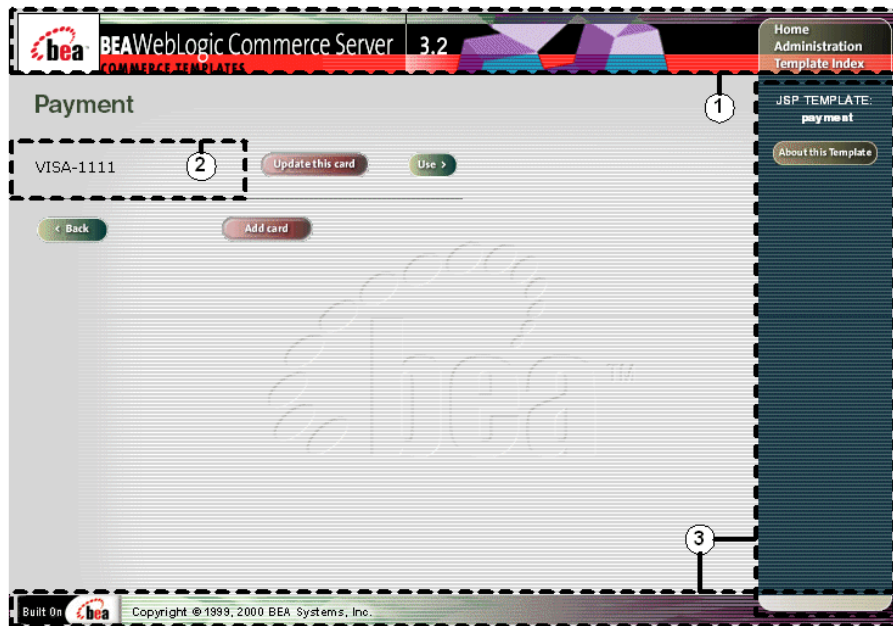
**Note:** For more information about user profiles, see “[Customer Profile Services](#)” in the *WebLogic Commerce Server Registration and User Processing Package* documentation.

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 dashed lines and numbers in the diagram are not part of the template; they are referenced in the explanation that follows the screen shot.

Figure 6-1 Annotated payment.jsp Template



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

1. The page header (top banner) is created from an import of the `innerheader.jsp` template. This is standard across many of the JSP templates provided by WebLogic Commerce Server. The import call is:

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

2. If available, region 2 uses a combination of the WebLogic Server and WebLogic Personalization Server JSP tags to obtain and display the customer's saved credit card(s).

3. The `payment.jsp` template's content in region 3 contains the included `innerfooter.jsp` template. The include call in `payment.jsp` is:

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

`innerfooter.jsp` consists of the horizontal footer at the bottom of the page, plus the right-side vertical column that describes (for the benefit of you and your

development team) the name of the current template and links to its *About* information. In the `innerfooter.jsp` file, the right-side vertical column is an include file:

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

### 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" %>
```

**Note:** For more information on the WebLogic Server JSP tags or the WebLogic Personalization Server JSP tags, see [“JSP Tag Reference”](#) in the *WebLogic Personalization Server* documentation.

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 the Order Processing Package” 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 *WebLogic Personalization Server* documentation.

## Form Field Specification

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

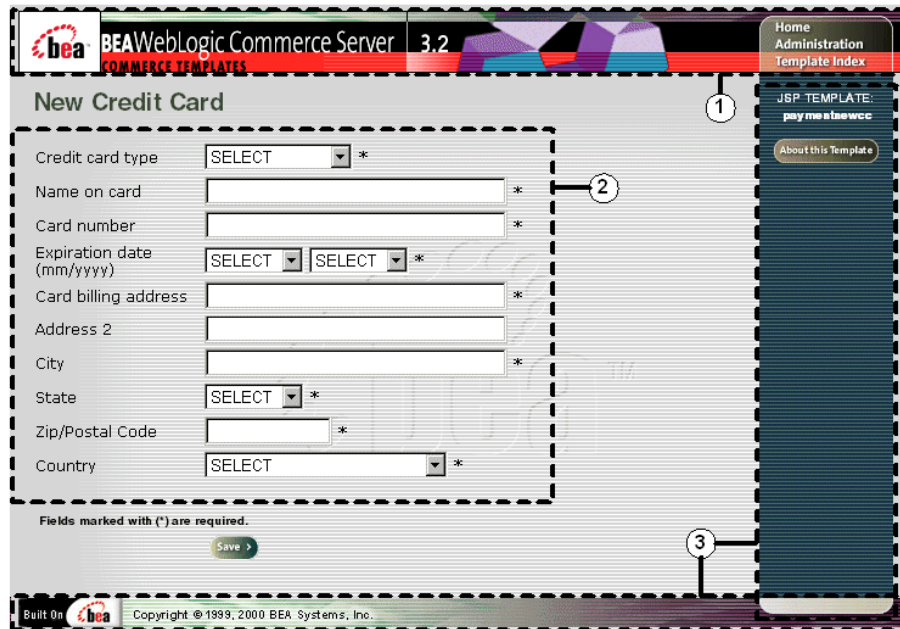
## paymentnewcc.jsp Template

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 dashed lines and numbers in the diagram are not part of the template; they are referenced in the explanation that follows the screen shot.

Figure 6-2 Annotated `paymentnewcc.jsp` Template





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

1. The page header (top banner) is created from an import of the `innerheader.jsp` template. This is standard across many of the JSP templates provided by WebLogic Commerce Server. The import call is:

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

2. Region 2 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" %>
```

3. The `paymentnewcc.jsp` template's content in region 3 contains the included `innerfooter.jsp` template. The include call in `paymentnewcc.jsp` is:

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

`innerfooter.jsp` consists of the horizontal footer at the bottom of the page, plus the right-side vertical column that describes (for the benefit of you and your development team) the name of the current template and links to its *About* information. In the `innerfooter.jsp` file, the right-side vertical column is an include file:

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

## 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 [WebLogic Commerce Server Webflow and Pipeline Management](#).

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 the Order Processing Package” 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 WebLogic Commerce Server *Registration and User Processing Package* documentation.

## 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

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

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

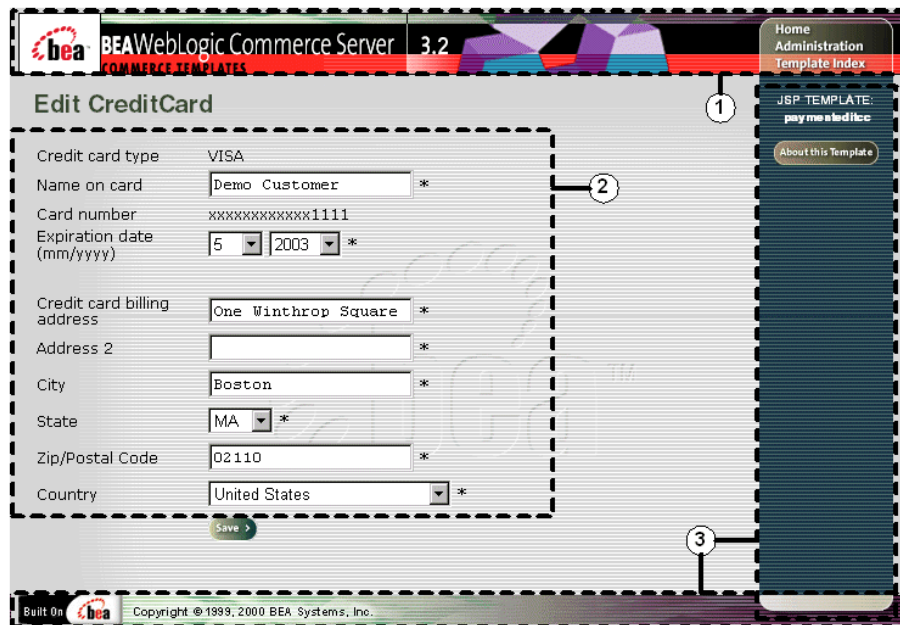
## paymenteditcc.jsp Template

The `paymenteditcc.jsp` template (shown in Figure 6-3) 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 dashed lines and numbers in the diagram are not part of the template; they are referenced in the explanation that follows the screen shot.

**Figure 6-3 Annotated paymenteditcc.jsp Template**



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

1. The page header (top banner) is created from an import of the `innerheader.jsp` template. This is standard across many of the JSP templates provided by WebLogic Commerce Server. The import call is:

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

2. Region 2 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" %>
```

3. The `paymenteditcc.jsp` template's content in region 3 contains the included `innerfooter.jsp` template. The include call in `paymenteditcc.jsp` is:

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

`innerfooter.jsp` consists of the horizontal footer at the bottom of the page, plus the right-side vertical column that describes (for the benefit of you and your development team) the name of the current template and links to its *About* information. In the `innerfooter.jsp` file, the right-side vertical column is an include file:

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

## 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 [WebLogic Commerce Server Webflow and Pipeline Management](#). For more information on the WebLogic Personalization Server JSP tags, see “[JSP Tag Reference](#)” in the *WebLogic Personalization Server* documentation.

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 the Order Processing Package](#)” 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 *WebLogic Commerce Server Registration and User Processing Package* documentation.

## 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 *WebLogic Personalization Server* documentation.

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**

| <b>Method/Attribute</b>          | <b>Description</b>  |
|----------------------------------|---|
| <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**

| <b>Method/Attribute</b>      | <b>Description</b>  |
|------------------------------|---|
| <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**

| <b>Parameter Name</b>                                 | <b>Type</b> | <b>Description</b>  |
|---|-------------|---|
| "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.<br>CUSTOMER_CREDITCARD_TYPE     | Listbox     | The type of the customer's credit card.   |
| HttpRequestConstants.<br>CUSTOMER_CREDITCARD_HOLDER   | Textbox     | The name on the credit card.  |
| HttpRequestConstants.<br>CUSTOMER_CREDITCARD_NUMBER   | Textbox     | The number of the customer's credit card.   |
| HttpRequestConstants.<br>CUSTOMER_CREDITCARD_MONTH    | Listbox     | The month of the customer's credit card expiration date.  |
| HttpRequestConstants.<br>CUSTOMER_CREDITCARD_YEAR     | Listbox     | The year of the customer's credit card expiration date.   |
| HttpRequestConstants.<br>CUSTOMER_CREDITCARD_ADDRESS1 | Textbox     | The first line in the customer's billing address.   |
| HttpRequestConstants.<br>CUSTOMER_CREDITCARD_ADDRESS2 | Textbox     | The second line in the customer's billing address.  |
| HttpRequestConstants.<br>CUSTOMER_CREDITCARD_CITY     | Textbox     | The city in the customer's billing address.   |
| HttpRequestConstants.<br>CUSTOMER_CREDITCARD_STATE    | Listbox     | The state in the customer's billing address.  |
| HttpRequestConstants.<br>CUSTOMER_CREDITCARD_ZIPCODE  | Textbox     | The zip/postal code in the customer's billing address.  |
| HttpRequestConstants.<br>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

---

|   |  |
|---|--|
| <b>Class Name</b>                                     | <code>com.beasys.commerce.ebusiness.payment.webflow.PaymentAuthorizationIP</code>  |
| <b>Description</b>                                    | 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. |
| <b>Required<br/>HttpServletRequest<br/>Parameters</b> | <code>HttpRequestConstants.CREDITCARD_KEY</code>   |
| <b>Required Pipeline<br/>Session Attributes</b>       | <code>PipelineSessionConstants.SHOPPING_CART</code>  |
| <b>Updated Pipeline<br/>Session Attributes</b>        | <code>PipelineSessionConstants.PAYMENT_CREDIT_CARD</code><br><code>PipelineSessionConstants.PAYMENT_AUTHORIZATION_AMOUNT</code>  |
| <b>Removed Pipeline<br/>Session Attributes</b>        | None   |
| <b>Validation</b>                                     | Verifies that the credit card key is valid and that it references an existing credit card.   |
| <b>Exceptions</b>                                     | <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

|   |  |
|---|--|
| <b>Class Name</b>                                     | <code>com.beasys.commerce.ebusiness.customer.webflow.UpdatePaymentInfoIP</code>  |
| <b>Description</b>                                    | 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.  |
| <b>Required<br/>HttpServletRequest<br/>Parameters</b> | <code>HttpRequestConstants.CUSTOMER_CREDITCARD_TYPE</code><br><code>HttpRequestConstants.CUSTOMER_CREDITCARD HOLDER</code><br><code>HttpRequestConstants.CUSTOMER_CREDITCARD_NUMBER</code><br><code>HttpRequestConstants.CUSTOMER_CREDITCARD_MONTH</code><br><code>HttpRequestConstants.CUSTOMER_CREDITCARD_YEAR</code><br><code>HttpRequestConstants.CUSTOMER_CREDITCARD_ADDRESS1</code><br><code>HttpRequestConstants.CUSTOMER_CREDITCARD_ADDRESS2</code><br><code>HttpRequestConstants.CUSTOMER_CREDITCARD_CITY</code><br><code>HttpRequestConstants.CUSTOMER_CREDITCARD_STATE</code><br><code>HttpRequestConstants.CUSTOMER_CREDITCARD_ZIPCODE</code><br><code>HttpRequestConstants.CUSTOMER_CREDITCARD_COUNTRY</code> |
| <b>Required Pipeline<br/>Session Attributes</b>       | <code>PipelineSessionConstants.USER_NAME</code>  |
| <b>Updated Pipeline<br/>Session Attributes</b>        | <code>PipelineSessionConstants.CUSTOMER</code>   |
| <b>Removed Pipeline<br/>Session Attributes</b>        | None   |
| <b>Validation</b>                                     | Verifies that the required fields contain values.  |
| <b>Exceptions</b>                                     | <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

|   |   |
|---|---|
| <b>Class Name</b>                           | <code>com.beasys.commerce.ebusiness.payment.pipeline.PaymentAuthorizationHostPC</code>  |
| <b>Description</b>                          | Authorizes a given credit card for a specified amount. Used for host-based payment models, shown in the <code>weblogiccommerce.properties</code> file as:<br><code>HOST_AUTH_CAPTURE</code><br><code>HOST_AUTH_CAPTURE_AVS</code><br><code>HOST_POST_AUTH_CAPTURE</code><br><code>HOST_POST_AUTH_CAPTURE_AVS</code> |
| <b>Required Pipeline Session Attributes</b> | <code>PipelineSessionConstants.PAYMENT_CREDIT_CARD</code><br><code>PipelineSessionConstants.PAYMENT_AUTHORIZATION_AMOUNT</code><br><code>PipelineSessionConstants.ORDER_HANDLE (Request scope)</code>   |
| <b>Updated Pipeline Session Attributes</b>  | None  |
| <b>Removed Pipeline Session Attributes</b>  | None  |
| <b>Type</b>                                 | Java object   |
| <b>JNDI Name</b>                            | 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

|   |  |
|---|--|
| <b>Class Name</b>                           | <code>com.beasys.commerce.ebusiness.payment.pipeline.PaymentAuthorizationTerminalPC</code>   |
| <b>Description</b>                          | <p>Authorizes a given credit card for a specified amount. Used for terminal-based payment models, shown in the <code>weblogiccommerce.properties</code> file as:</p> <p><code>AUTO_MARK_AUTO_SETTLE</code><br/> <code>AUTO_MARK_AUTO_SETTLE_AVS</code><br/> <code>AUTO_MARK_MANUAL_SETTLE</code><br/> <code>AUTO_MARK_MANUAL_SETTLE_AVS</code><br/> <code>MANUAL_MARK_AUTO_SETTLE</code><br/> <code>MANUAL_MARK_AUTO_SETTLE_AVS</code><br/> <code>MANUAL_MARK_MANUAL_SETTLE</code><br/> <code>MANUAL_MARK_MANUAL_SETTLE_AVS</code></p> |
| <b>Required Pipeline Session Attributes</b> | <p><code>PipelineSessionConstants.PAYMENT_CREDIT_CARD</code><br/> <code>PipelineSessionConstants.PAYMENT_AUTHORIZATION_AMOUNT</code><br/> <code>PipelineSessionConstants.ORDER_HANDLE (Request scope)</code></p>   |
| <b>Updated Pipeline Session Attributes</b>  | None   |
| <b>Removed Pipeline Session Attributes</b>  | None   |
| <b>Type</b>                                 | Java object  |
| <b>JNDI Name</b>                            | 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.

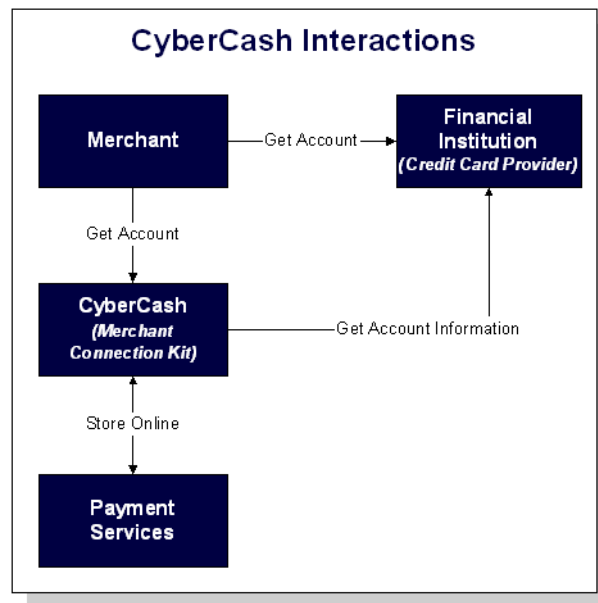
`PipelineFatalException`, thrown when there is a configuration error, a general service error, or a system-level exception from a back-end component.

---

## 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 the Order Processing package's 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 the Order Processing package's 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 6-31.

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**

| <b>Payment Model</b>      | <b>Description</b>  |
|---------------------------|---|
| 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 6-24.

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/Sequencer.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.
    // This will depend on your payment service requirements.

}

//
// Invoke the credit card service authorization method using
```

## 6 Payment Services

---

```
// 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 `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). On the order confirmation JSP template (`confirmorder.jsp`), for example, only the last 4 digits of a customer’s credit card are displayed.



## Encryption/Decryption Implementation

The WebLogic Commerce Server product's encryption mechanism is based upon RSA's public key infrastructure. A public key is used to encrypt a customer's credit card information, while a private key is used to decrypt it when required.

The public key is stored in the database for use by the `EncryptCreditCardPC` Pipeline component, while the private key is itself encrypted using a password you supply, and stored in the database.

When invoked from the Webflow, the `EncryptCreditCardPC` Pipeline component reads the customer-provided credit card information from the Pipeline session, encrypts it using the public key, and then places it back into the Pipeline session. This encrypted data is subsequently written to the database. Decryption is accomplished using a back-end component and the private key. Again, decryption is initiated only in stages of the ordering process where this data is absolutely necessary.

For more technical information about the Credit Card Security Service, please contact your BEA representative.

## Customizable Security Settings

Although the WebLogic Commerce Server product specifies default settings for the Credit Card Security Service, you can customize them. The security settings reside 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). These security settings are shown in Listing 6-10.

### Listing 6-10 Security Settings in `weblogiccommerce.properties`

---

```
#####
# Properties required for the Security Service
#####

# Security services are turned on by setting this property to true.
# Commenting out the property or setting it to false will disable
# security.

is.encryption.enabled=true
```

## 6 Payment Services

---

```
# The name of the security table and column names for the public
# and private keys can be specified using the properties below.

security.table.name=WLCS_SECURITY
security.backup.table=WLCS_SECURITY_BACKUP
public.key.column.name=PUBLIC_KEY
private.key.column.name=PRIVATE_KEY

# The key bit size desired
# Key bit length and length of data that can be encrypted are related
# as follows:

# KEY BIT LENGTH(bits)          DATA LENGTH (bytes)
#      512                      53
#      1024                     117
#      2048 (MAX LENGTH)245

key.bit.size=1024

# WARNING! Remember that setting this property will start up the
# server without prompting for a password. The password will be read
# from this property which makes the encryption vulnerable to an
# inside attack.

private.key.password=WLCS
```

---

First, the `is.encryption.enabled` property enables encryption mechanisms. Please note that a value of `false` (or no value at all) will disable encryption mechanisms. BEA has assigned this property a default value of `true`.

Next is a series of properties that allow you to specify the names of the security tables (primary and backup) and the columns in which the public and private keys will be stored. BEA has assigned default values to these properties, but you can modify them based on your database.

Following the properties related to the database, the `key.bit.size` property allows you to specify the encryption key length. BEA has assigned this property a default value of 1024, but you can adjust this value. Table 6-10 illustrates the possible key bit values.

**Table 6-10 Key Bit Values**

| <b>Length (Bits)</b> | <b>Data Length (Bytes)</b> |
|----------------------|----------------------------|
| 512                  | 53                         |
| 1024                 | 117                        |
| 2048                 | 245                        |

Lastly, the `private.key.password` property allows you to specify, in the `weblogiccommerce.properties` file, the password used to encrypt the private key. Please note that BEA does not recommend use of this property. Rather, the private key should be supplied by an administrator during server startup. For more information about supplying the private key, see “Methods for Supplying the Private Key Encryption Password” on page 6-44.

**Note:** If not used, the `private.key.password` property should be commented out with a # symbol. BEA has assigned this property a default value of `WLCS`, but this is for demonstration purposes only.

## Methods for Supplying the Private Key Encryption Password

As previously mentioned, the private key used to encrypt customer credit cards is itself encrypted with a password before being stored in the database. There are three methods by which you can supply this password:

- Specify the password in the `weblogiccommerce.properties` file, which will be read by a startup class (not recommended).
- Specify the password at server startup using the console (recommended).
- Specify the password after server startup using a secure Web form (recommended).

### Specifying the Password in `weblogiccommerce.properties` (Default)

The first method for specifying the private key encryption password is to specify the password as a value for a property in the `weblogiccommerce.properties` file.

**Note:** BEA does not recommend this method because by providing the password in a simple text file, you leave yourself vulnerable to security attacks. Anyone who gains access to this file can read the password you use to encrypt the private key, and thus gain access to it.

To use this method, follow these steps:

1. 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 `private.key.password` property to specify the password.
2. In the `weblogic.properties` file (located in the `WL_COMMERCE_HOME` directory, where `WL_COMMERCE_HOME` is the directory in which you installed WebLogic Commerce Server), ensure that the `weblogic.system.startupClass.KeyBootstrap` class is enabled (that is, not commented out), as shown in the last line of Listing 6-11.

**Listing 6-11 Encryption Section of weblogic.properties file**

---

```
#####
# ENCRYPTION SERVICES
# -----

#
# Specify a method for supplying the password for decrypting
# private keys. This may be one of two mechanisms:
#
# (1) Servlet-based password entry
# (2) Property specification by way of a startup class
#
# NOTE: Make sure that the property is.encryption.enabled
# in weblogiccommerce.properties is commented out or set to false
# if neither the servlet nor the startup class is being used.
#

#
# Startup class password entry
#
# Reads a private key encryption password from the
# weblogiccommerce.properties file if the private.key.password
# property has a non-empty value. If keys are already present in the
# database and the password used to generate them differs from the
# one specified by this property, the user must enter the password
# on the console.
#
# OR
#
# Prompts the user to enter a password on the console at server
# startup. If there are no public and private keys in the database
# the user is prompted to specify that new keys be created. If new
# key generation is not desired, encryption should be turned off
# by way of the is.encryption.enabled property in the
# weblogiccommerce.properties file. If the user knows that keys are
# already generated, he/she should stop the server and check the
# security database tables and properties in the
# weblogiccommerce.properties file.

weblogic.system.startupClass.KeyBootstrap=com.beasys.commerce.
ebusiness.security.KeyBootstrap
```

---

### Specifying the Password at Server Startup Using the Console

The second method for specifying the private key encryption password is for an administrator to specify the password at server startup using the server console.

To use this method, follow these steps:

1. 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), comment out the `private.key.password` property line with a `#` symbol.
2. In the `weblogic.properties` file (located in the `WL_COMMERCE_HOME` directory, where `WL_COMMERCE_HOME` is the directory in which you installed WebLogic Commerce Server), ensure that the `weblogic.system.startupClass.KeyBootstrap` class is enabled (that is, not commented out), as shown in the last line of Listing 6-11.

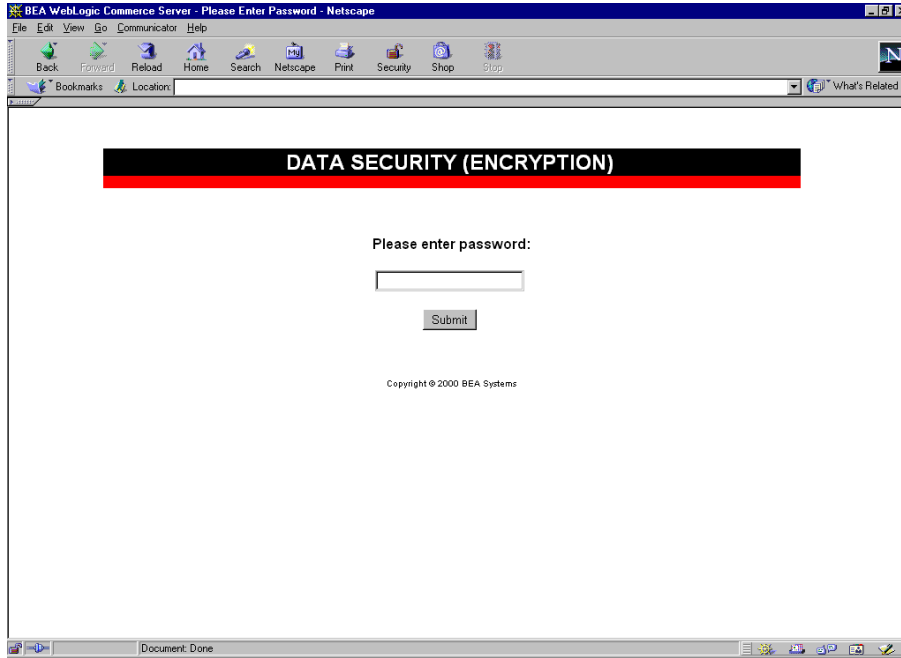
### Specifying the Password After Server Startup Using a Secure Web Form

The third method for specifying the private key encryption password allows an administrator to enter the password on a secure Web form, so the password is stored in memory on your system instead of in a text file.

To use this method, follow these steps:

1. In the `weblogic.properties` file (located in the `WL_COMMERCE_HOME` directory, where `WL_COMMERCE_HOME` is the directory in which you installed WebLogic Commerce Server), disable the `weblogic.system.startupClass.KeyBootstrap` class by commenting out this line with a `#` symbol (see the last line of Listing 6-11).
2. Point your Web browser to `<hostname>:port/tools/security/security_getPassword.html`, to load the secure Web form shown in Figure 6-5.

Figure 6-5 security\_getPassword.html



3. Specify the private key encryption password in the form field and click the Submit button.

On submission, this page will invoke the `EncryptionServlet` and `KeyGeneratorServlet` registered in the `web.xml` file (located in the `WL_COMMERCE_HOME/server/webapps/admin/Web-inf` directory, where `WL_COMMERCE_HOME` is the directory in which you installed WebLogic Commerce Server).

### Important Notes About Supplying Your Password

You must supply the password for all nodes in a cluster. Should one node in the cluster fail, other machines that know the private key encryption password can be used for failover.

The first time you enter the password, you will be asked to confirm whether or not you want to generate new keys. If this is indeed the first time you are entering the password, you do want to generate new keys. However, be sure to select a password that is memorable. All credit cards accepted by your site will be encrypted using this password, and cannot be decrypted if you forget your password.

If you are asked to confirm whether or not you want to generate new keys and you are using the same password, then the keys cannot be found in the database. If no data was encrypted using the old keys, you can regenerate the keys. However, if data has already been encrypted using the old keys, this data will be lost because it cannot be read using the new keys. If you have data encrypted with the old keys, you should stop the server, check the database, and verify the properties in the `weblogiccommerce.properties` file to ensure that the system is set up properly.

During server startup, any orders placed before the password is entered will be persisted with a payment transaction in the `RETRY` state. After supplying the password, administrators should use the Payment Management Administration Tool to reauthorize the transaction. For more information about using the Payment Management Administration Tool, see Chapter 9, “Using the Order and Payment Management Pages.”

### What if I Want to Change My Password?

Because all the credit cards that have been encrypted use the private key encryption password, it is not recommended that you change this password. However, there may be the rare occasion (for example, if the password has been compromised) when you need to change the password. Changing the password means changing the public and private key pair. Therefore, you must follow this process when changing the password:

- Use the old password (and thus the old key pair) to decrypt old credit card numbers. The credit card numbers will now be in plain text. Store the credit card numbers in a data structure that preserves the original organization.
- Create a new key pair using a new password.



- Using the new key pair, re-encrypt the plain text credit card numbers from the data structure.

**Note:** Changing the password is especially difficult if you have a lot of encrypted data. Again, this process is not recommended and should not be done unless absolutely required.

## **6** *Payment Services*

---

**6-50** Order Processing Package

# 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

## 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 [Summary of JSP Templates](#) documentation.

### checkout.jsp Template

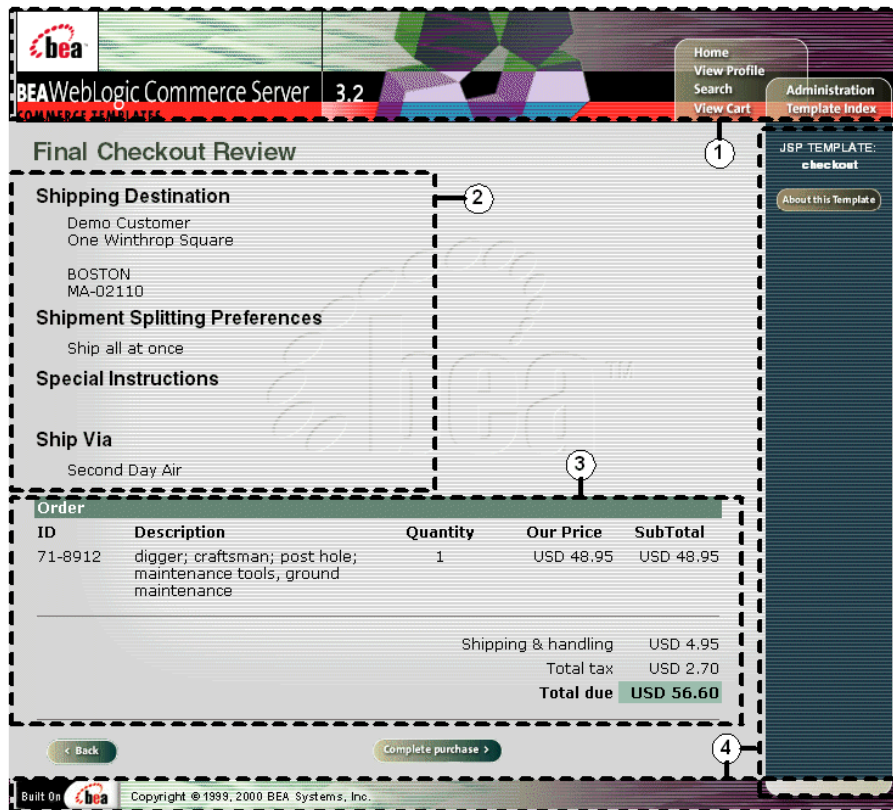
The `checkout.jsp` template (shown in Figure 7-1) provides a customer with a final look at all the details of their order, before the customer commits or cancels the order. Information displayed includes the shipping address, shipping details, a list of the items ordered (including the item name, short 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. The dashed lines and numbers in the diagram are not part of the template; they are referenced in the explanation that follows the screen shot.

Figure 7-1 Annotated checkout.jsp Template



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

1. The page header (top banner) is created from an import of the header2.jsp template. This is standard across many of the JSP templates provided by WebLogic Commerce Server. The import call is:  

```
<%@ include file="/commerce/includes/header2.jsp" %>
```
2. Region 2 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.

## 7 Order Summary and Confirmation Services

---

3. Region 3 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 shipping, tax, and total amounts for the entire order.

4. The `checkout.jsp` template's content in region 4 contains the included `footer2.jsp` template. The include call in `checkout.jsp` is:

```
<%@ include file="/commerce/includes/footer2.jsp" %>
```

`footer2.jsp` consists of the horizontal footer at the bottom of the page, plus the right-side vertical column that describes (for the benefit of you and your development team) the name of the current template and links to its *About* information. In the `footer2.jsp` file, the right-side vertical column is an include file:

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

### 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%\server\webapps\wlcs\commerce\order\  
checkout.jsp (Windows)  
$WL_COMMERCE_HOME/server/webapps/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" %>
```

**Note:** For more information on the WebLogic Server JSP tags or the WebLogic Personalization Server JSP tags, see [“JSP Tag Reference”](#) in the *WebLogic Personalization Server* documentation. For more information about the Pipeline JSP tags, see [WebLogic Commerce Server Webflow and Pipeline Management](#).

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 `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.*" %>
```

## 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 the Order Processing Package”](#) on page 1-1.

### Included JSP Templates

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

- `header2.jsp`, which creates the top banner.
- `footer2.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 `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 7-24.

**Table 7-2 Checkout Review Pipelines**

| Pipeline                 | Description   |
|--------------------------|---|
| <code>CommitOrder</code> | Contains <code>CommitOrderPC</code> , <code>AuthorizePaymentPC</code> , <code>CalculateTaxLineLevelCommitPC</code> , <code>ResetCheckoutPC</code> , and is 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 *WebLogic Personalization Server* documentation.

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.<br>SHIPPING_ADDRESS     | com.beasys.commerce.axiom<br>.contact.Address                  | The address the order is being shipped to.            |
| PipelineSessionConstants.<br>SHIPPING_METHOD      | com.beasys.commerce.ebusiness<br>.shipping.shippingMethodValue | Identifies the shipping method the customer selected. |
| PipelineSessionConstants.<br>SHOPPING_CART        | com.beasys.commerce.ebusiness<br>.shoppingcart.ShoppingCart    | The shopping cart that was ordered.                   |
| PipelineSessionConstants.<br>SPLITTING_PREFERENCE | java.lang.String   | The splitting preference the customer selected.       |

## 7 Order Summary and Confirmation Services

---

**Table 7-3 checkout.jsp Pipeline Session Attributes (Continued)**

| Attributes  | Type             | Description                                      |
|---|------------------|--|
| PipelineSessionConstants.<br>SPECIAL_INSTRUCTIONS | java.lang.String | Any special instructions the customer specifies. |

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

### 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 [WebLogic Commerce Server Webflow and Pipeline Management](#).

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 *WebLogic Personalization Server* documentation.

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>getCounty()</code>     | The county 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 (i.e. <code>ShoppingCartLine</code> ). |

## 7 Order Summary and Confirmation Services

---

**Table 7-5 ShoppingCart Accessor Methods/Attributes (Continued)**

| Method/Attribute                     | Description  |
|--------------------------------------|--|
| <code>getTotal(int totalType)</code> | <p>The total amount specified by the <code>totalType</code> parameter. Valid parameters include:</p> <p><code>ShoppingCartConstants.LINE_UNIT_PRICE_TIMES_QUANTITY</code><br/><code>ShoppingCartConstants.LINE_SHIPPING</code><br/><code>ShoppingCartConstants.LINE_TAX</code></p> <p><b>Note:</b> The <code>getTotal()</code> method also allows you to combine different total types. For more information, see the <a href="#">Javadoc</a>.</p> |

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

**Table 7-6 ShoppingCartLine Accessor Methods/Attributes**

| Method/Attribute                         | Description  |
|--|--|
| <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>getLineTotal(int totalType)</code> | <p>The total amount specified by the <code>totalType</code> parameter. Valid parameters include:</p> <p><code>ShoppingCartConstants.LINE_UNIT_PRICE_TIMES_QUANTITY</code><br/><code>ShoppingCartConstants.LINE_SHIPPING</code><br/><code>ShoppingCartConstants.LINE_TAX</code></p> <p><b>Note:</b> The <code>getLineTotal()</code> method also allows you to combine different total types. For more information, see the <a href="#">Javadoc</a>.</p> |

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">
      <%=shoppingCartLine.getProductItem().getKey().getIdentifier()%>
    </div>
  </td>

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

  <td align="center" valign="top">
    <div class="tabletext">
      <%=WebflowJSPHelper.quantityFormat(shoppingCartLine.getQuantity() %>
    </div>
  </td>

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

  <td align="right" nowrap valign="top">
    <% Money lineTotal=shoppingCartLine.getLineTotal
      (ShoppingCartConstants.LINE_UNIT_PRICE_TIMES_QUANTITY); %>
    <div class="tabletext">
      <%=lineTotal.getCurrency()%>
      <%=WebflowJSPHelper.priceFormat(lineTotal.getValue())%>
    </div>
  </td>
</tr><tr>

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

## 7 Order Summary and Confirmation Services

---

```
</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>

  <td align="right" bgcolor="#99BBAA" nowrap>
    <% Money total=shoppingCart.getTotal(ShoppingCartConstants.
    LINE_UNIT_PRICE_TIMES_QUANTITY + ShoppingCartConstants.LINE_SHIPPING +
    ShoppingCartConstants.LINE_TAX); %>
    <div class="tabletext"><b>
      <%=total.getCurrency()%>
      <%=WebflowJSPHelper.priceFormat(total.getValue())%>
    </b></div>
  </td>
</tr>
```

---

## **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. The dashed lines and numbers in the diagram are not part of the template; they are referenced in the explanation that follows the screen shot.

Figure 7-2 Annotated `confirmorder.jsp` Template

**Confirm Order**

Thank you for shopping here with BEA WebLogic Commerce Server 3.1. Don't forget to stop back for more great deals, contests, new store openings and specials on jsp templates.

**Order Confirmation #2004**

**Will be shipped to:**  
 Demo Customer  
 One Winthrop Square  
 BOSTON  
 MA-02110

**Preferences:**  
 Ship all at once

| ID      | Description    | Quantity | Unit Price | Total                      |
|---------|----------------|----------|------------|----------------------------|
| 71-8912 | digger-71-8912 | 1        | USD 48.95  | USD 48.95                  |
|         |                |          |            | <b>Shipping</b> USD 4.95   |
|         |                |          |            | <b>Total Tax</b> USD 2.70  |
|         |                |          |            | <b>Total Due</b> USD 56.60 |

Print this page for your records.

[Continue shopping](#)

Built On Copyright © 1999, 2000 BEA Systems, Inc.



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

1. The page header (top banner) is created from an import of the `header2.jsp` template. This is standard across many of the JSP templates provided by WebLogic Commerce Server. The import call is:

```
<%@ include file="/commerce/includes/header2.jsp" %>
```

2. Region 2 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.
3. Region 3 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 #2 and region #4, this provides the customer with a record of the shipping information as it was entered on previous JSP templates.
4. Region 4 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 2 and region 3, 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.
5. The `confirmorder.jsp` template's content in region 5 contains the included `footer2.jsp` template. The include call in `checkout.jsp` is:

```
<%@ include file="/commerce/includes/footer2.jsp" %>
```

`footer2.jsp` consists of the horizontal footer at the bottom of the page, plus the right-side vertical column that describes (for the benefit of you and your development team) the name of the current template and links to its *About* information. In the `footer2.jsp` file, the right-side vertical column is an include file:

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

### 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" %>
```

**Note:** For more information on the WebLogic Server JSP tags or the WebLogic Personalization Server JSP tags, see [“JSP Tag Reference”](#) in the *WebLogic Personalization Server* documentation. For more information about the Pipeline JSP tags, see [WebLogic Commerce Server Webflow and Pipeline Management](#).

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 `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 the Order Processing Package” on page 1-1.

## Included JSP Templates

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

- `header2.jsp`, which creates the top banner.
- `footer2.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

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.

## 7 Order Summary and Confirmation Services

---

### 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 *WebLogic Personalization Server* documentation.

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.       |

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 [WebLogic Commerce Server Webflow and Pipeline Management](#).

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 *WebLogic Personalization Server* documentation.

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.

**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>getCounty()</code>  | The county in the customer’s address.             |
| <code>getState()</code>   | The state in the customer’s address.              |

## 7 Order Summary and Confirmation Services

---

**Table 7-8 Address Accessor Methods/Attributes (Continued)**

| Method/Attribute             | Description                                    |
|------------------------------|--|
| <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> | <p>The total amount specified by the <code>totalType</code> parameter. Valid parameters include:</p> <p><code>OrderConstants.LINE_UNIT_PRICE_TIMES_QUANTITY</code><br/><code>OrderConstants.LINE_SHIPPING</code><br/><code>OrderConstants.LINE_TAX</code></p> <p><b>Note:</b> The <code>getTotal()</code> method also allows you to combine different total types. For more information, see the <a href="#">Javadoc</a>.</p> |
| <code>orderLines</code>              | A collection of the lines in the shopping cart that make up the customer's order.   |

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

**Table 7-11 OrderLine Accessor Methods/Attributes**

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

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

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

```

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

```

## 7 Order Summary and Confirmation Services

---

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

<td align="right" valign="top" nowrap>
    <% Money lineTotal=orderLine.getLineTotal(OrderConstants.
        LINE_UNIT_PRICE_TIMES_QUANTITY); %>
    <div class="tabletext">
        <%=lineTotal.getCurrency()%>
        <%=WebflowJSPHelper.priceFormat(lineTotal.getValue())%>
    </div>
</td>
</tr>
</wl:repeat>

<tr>

    <td colspan="2" rowspan="3" valign="middle" align="center" bgcolor="#99BBAA">
        <div class="commentary">Print this page for your records.</div>
    </td>

    <td colspan="2" align="right">
        <div class="tabletext"><b>Shipping</b><br>
            <font size="1"><%= shippingMethodDescription %></font>
        </div>
    </td>

    <td align="right" nowrap valign="top">
        <% Money shipping=orderValue.getTotal(OrderConstants.LINE_SHIPPING); %>
        <div class="tabletext">
            <%=shipping.getCurrency()%>
            <%=WebflowJSPHelper.priceFormat(shipping.getValue())%>
        </div>
    </td>
</tr><tr>

    <td align="right" colspan="2">
        <div class="tabletext"><b>Total Tax</b></div>
    </td>

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



```
<td align="right" colspan="2">
  <div class="tabletext"><b>Total Due</b></div>
</td>

<td align="right" nowrap>
  <% Money total=orderValue.getTotal(OrderConstants.LINE_UNIT_PRICE_TIMES_
    QUANTITY + OrderConstants.LINE_SHIPPING + OrderConstants.LINE_TAX); %>
  <div class="tabletext">
    <%=total.getCurrency()%>
    <%=WebflowJSPHelper.priceFormat(total.getValue())%>
  </div>
</td>
</tr>
</table>
```

---

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

|   |  |
|---|--|
| <b>Class Name</b>                           | <code>com.beasys.commerce.ebusiness.order.pipeline.CommitOrderPC</code>  |
| <b>Description</b>                          | 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. |
| <b>Required Pipeline Session Attributes</b> | <code>PipelineSessionConstants.USER_NAME</code><br><code>PipelineSessionConstants.SHOPPING_CART</code><br><code>PipelineSessionConstants.SPLITTING_PREFERENCE</code><br><code>PipelineSessionConstants.SPECIAL_INSTRUCTIONS</code><br><code>PipelineSessionConstants.ORDER_CONFIRMATION_NUMBER</code>                          |
| <b>Updated Pipeline Session Attributes</b>  | <code>PipelineSessionConstants.ORDER_HANDLE</code> (Request scope)<br><code>PipelineSessionConstants.ORDER_VALUE</code> (Request scope)<br><code>PipelineSessionConstants.ORDER_SHIPPING_METHOD</code> (Request scope)   |

|  |  |
|--|--|
| <b>Removed Pipeline Session Attributes</b> | PipelineSessionConstants.SHIPPING_METHOD   |
| <b>Type</b>                                | Java object  |
| <b>JNDI Name</b>                           | None   |
| <b>Exceptions</b>                          | PipelineFatalException, thrown when the required Pipeline session attributes are not available or if the shopping cart is empty. |

## ResetCheckoutPC

|   |   |
|---|---|
| <b>Class Name</b>                           | com.beasys.commerce.ebusiness.order.pipeline.<br>ResetCheckoutPC  |
| <b>Description</b>                          | Removes all Pipeline session attributes relating to the customer's checkout process.  |
| <b>Required Pipeline Session Attributes</b> | None  |
| <b>Updated Pipeline Session Attributes</b>  | None  |
| <b>Removed Pipeline Session Attributes</b>  | PipelineSessionConstants.SHOPPING_CART<br>PipelineSessionConstants.SHIPPING_ADDRESS<br>PipelineSessionConstants.SPLITTING_PREFERENCE<br>PipelineSessionConstants.SHIPPING_METHOD<br>PipelineSessionConstants.SPECIAL_INSTRUCTIONS<br>PipelineSessionConstants.PAYMENT_AUTHORIZATION_AMOUNT<br>PipelineSessionConstants.VERAZIP_SHIPPING_ADDRESS<br>PipelineSessionConstants.PAYMENT_CREDIT_CARD |
| <b>Type</b>                                 | Java object   |
| <b>JNDI Name</b>                            | None  |
| <b>Exceptions</b>                           | None  |

## **7** *Order Summary and Confirmation Services*

---

# 8 Extending the Data Model

This chapter explains how to extend the Order Package data model. 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 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

The User Registration and Order Processing packages are two core components of the WebLogic Commerce Server 3.2. These packages implement use-cases that deal with customer self-registration, customer management, shopping cart experience, and order processing (including shipping, payment and taxation).

These packages implement 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 WebLogic Commerce Server *Webflow and Pipeline Management* documentation.

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 *User Registration and Order Processing Package* 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.

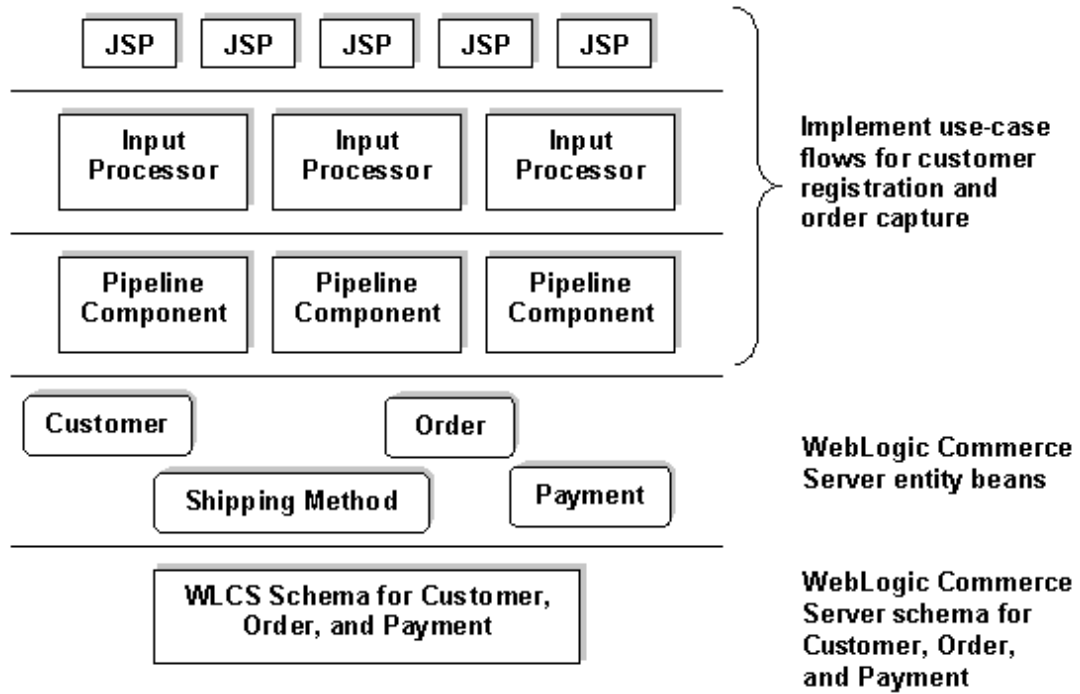
This chapter addresses the following:

- 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 WLCS\_CUSTOMER, WLCS\_ORDER, WLCS\_TRANSACTION and WLCS\_SHIPPING\_METHOD tables.
- Extending WLCS\_ORDER\_LINE table (the case of one-to-many associations).
- How to persist and query additional attributes on entities such as customer, order, and payment transaction.
- How to demarcate transactions with such extensions.

**Note:** This chapter does not cover extensions to other WebLogic Commerce Server packages (such as extensions for catalog). You can periodically check the WebLogic Commerce Server documentation for future updates on how to extend other packages.

## 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 [Registration and Order Processing Package](#). 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 Customer Registration and Order Processing Packages**

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

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

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

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



| Table  | Description                       |
|--|-----------------------------------|
| <b>Entity:</b> <code>com.beasys.commerce.ebusiness.customer.Customer</code>          |                                   |
| <code>WLCS_CUSTOMER</code>   | Customer description              |
| <code>WLCS_CREDIT_CARD</code>  | Credit cards                      |
| <code>WLCS_SHIPPING_ADDRESS</code>   | Shipping address                  |
| <b>Entity:</b> <code>com.beasys.commerce.ebusiness.order.Order</code>                |                                   |
| <code>WLCS_ORDER</code>  | Order description                 |
| <code>WLCS_ORDER_LINES</code>  | Order lines                       |
| <b>Entity:</b> <code>com.beasys.commerce.ebusiness.payment.PaymentTransaction</code> |                                   |
| <code>WLCS_TRANSACTION</code>  | CyberCash transaction description |
| <code>WLCS_TRANSACTION_ENTRY</code>  | CyberCash transaction entries     |
| <b>Entity:</b> <code>com.beasys.commerce.ebusiness.shipping.ShippingMethod</code>    |                                   |
| <code>WLCS_SHIPPING_METHOD</code>  | 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 WebLogic Personalization Server documentation.

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

You can find examples of adding attributes for customer data in the Pipeline components under the `com.beasys.commerce.ebusiness.customer.pipeline` package. To add attributes to the customer data, the WebLogic Commerce Server 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 *Personalization Server Developer's Guide*.

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 the order-processing package 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 order-processing package extend the `com.beasys.commerce.foundation.ConfigurableEntity` interface, which provides the following methods for adding and manipulating run-time attributes.

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

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

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

This method returns a previously added property.

For more information, including the API, 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 collection additional data from the user interface.
- Modifying the WebFlow to change the flow of user interaction.
- Extending the WebLogic Commerce Server schema.

**Note:** Almost all the data in the Order-Processing package is meaningful across your business, 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 backend applications may depend on this data. Any extension to the schema of the order-processing package 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 backend order fulfillment application. You may want to display this tracking number on WebLogic Commerce Server order history pages for customers to view the tracking information. This is a domain-specific attribute that best be persisted in the WLCS\_ORDER table (or another table that you created for this purpose).

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

1. Adding attributes against WLCS\_CUSTOMER, WLCS\_ORDER, WLCS\_TRANSACTION, and WLCS\_SHIPPING\_METHOD
2. Adding attributes against WLCS\_ORDER\_LINE, WLCS\_SHIPPING\_ADDRESS, and WLCS\_CREDIT\_CARD tables

**Note:** These two cases are discussed separately because the tables in case 2 participate in a one-to-many association with WLCS\_ORDER and WLCS\_CUSTOMER tables in case 1.

## Overview of Approach to Extending the WebLogic Commerce Server Schema

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

**Figure 8-2 Extending the Data Model**

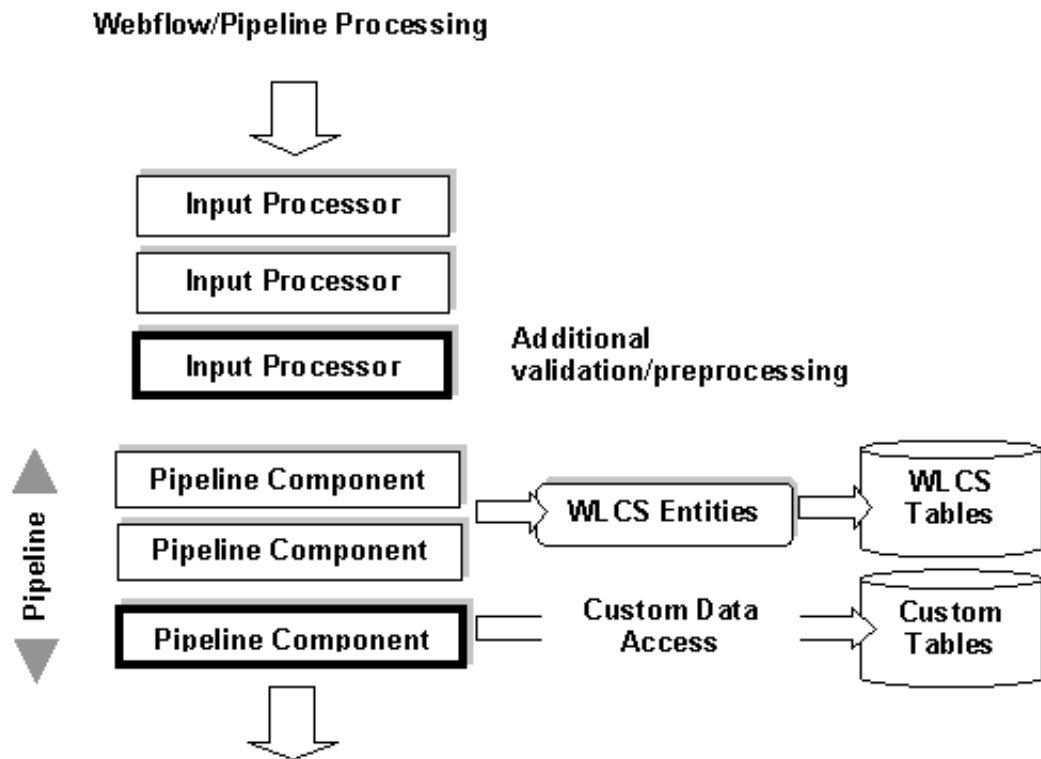


Figure 8-2 demonstrates how a given WebFlow/Pipeline processing can be modified to process additional data, without modifying existing input processors and Pipeline components. In Figure 8-2, the blocks with heavy borders are new input processors and Pipeline components inserted to process the additional 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 WLCS\_CUSTOMER, WLCS\_ORDER, WLCS\_TRANSACTION and WLCS\_SHIPPING\_METHOD Tables**

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

### **Step 1: Design new tables.**

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

### **Step 2: Modify corresponding JSP templates.**

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

### **Step 3: Implement new input processor.**

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

### **Step 4: Include the new input processor.**

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

### **Step 5: Implement a new Pipeline component.**

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

### **Step 6: Obtain a database connection.**

To obtain a database connection, use the `getConnection()` method in the abstract base class

```
com.beasys.commerce.foundation.pipeline.CommercePipelineComponent.
```

You may recall that all Pipeline components extend this abstract class. This method returns a connection from the `commercePool` setup in the `weblogic.properties` file. However, if you want to use a different connection pool, modify the property `commerce.jdbc.pool.url` 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 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_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-ehb-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.

# 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 Payment Management pages allow a site administrator to review and modify the status of 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
- 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 3.2 → 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.

**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 3.2 → 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 title bar to load the Order Management Search Page; to look up a customer's payment transactions, click the icon shown on the Payment Management section title bar 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 title bar. 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 Administration Tools interface. At the top, there is a navigation bar with the BEA logo, 'Administration Tools', and 'BEA WebLogic Commerce Server'. Below this, a breadcrumb trail shows 'Order Management'. A status message reads 'Obtained '2' orders for customer democustomer'. The main section is titled 'Order Search' and includes a brief instruction: 'Use this page to search for Orders. Click on order id to edit it.' There are 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 a table titled 'Order List' with the following data:

| Identifier        | Create Date | Price | Customer Id                  |
|-------------------|-------------|-------|------------------------------|
| <a href="#">1</a> | 2000-09-15  | 54.18 | <a href="#">democustomer</a> |
| <a href="#">2</a> | 2000-09-18  | 93.83 | <a href="#">democustomer</a> |

At the bottom of the table area, 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.

Figure 9-4 Sample Order Status Page

The screenshot shows the BEA WebLogic Commerce Server interface. At the top, there is a red navigation bar with the BEA logo, "Administration Tools", and icons for "home", a folder, and a question mark. Below this, a grey bar displays "Order Status: 2".

Order details are presented in two columns:

|                             |  |                    |                      |
|-----------------------------|--|--------------------|----------------------|
| <b>Confirmation Number</b>  | 2  | <b>Customer Id</b> | democustomer         |
| <b>Order Status</b>         | Submitted  | <b>Name</b>        | Customer Demo        |
| <b>Date Ordered</b>         | 2000-09-18   | <b>Work Phone</b>  | 708-555-5555         |
| <b>Splitting Preference</b> | Ship all at once                                       | <b>Home Phone</b>  | 617-555-5555         |
| <b>Special Instructions</b> |  | <b>Email</b>       | democustomer@bea.com |
| <b>Shipping Address</b>     | One Main Street<br>DENVER<br>CO-80212<br>United States |                    |                      |

Below the details is a table of items:

| ID       | Description        | Quantity | Unit Price |
|----------|--------------------|----------|------------|
| 9-WR8913 | extension-9-WR8913 | 1        | USD 82.50  |

At the bottom right, a summary of charges is shown:

|                                |           |
|--------------------------------|-----------|
| <b>Shipping &amp; Handling</b> | USD 4.95  |
| <b>Total Tax</b>               | USD 6.38  |
| <b>Total Due</b>               | USD 93.83 |

A "back" button is located at the bottom left of the 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 Administration Tools interface. At the top, there is a navigation bar with the BEA logo, 'Administration Tools', and 'BEA WebLogic Commerce Server'. Below this, a 'Order Management' section indicates 'Obtained '1' orders for order 2'. The main area 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' field and a 'search' button; 'Search By Order Identifier' with an 'Order ID' field and a 'search' button; and 'Search By Date Range' with 'From' and 'To' date pickers and a 'search' button. Below the filters is an 'Order List' table with the following data:

| Identifier        | Create Date | Price | Customer Id                  |
|-------------------|-------------|-------|------------------------------|
| <a href="#">2</a> | 2000-09-18  | 93.83 | <a href="#">democustomer</a> |

A 'back' button is located below the table.

The Order List shows the Order Identifier number, the date the customer placed the order, and the price of the order. To see details for a particular order (including the product items ordered, shipping information, tax, and so on), click the hyperlinked Order Identifier number to load the Order Status page (shown in Figure 9-6). To return to the main Administration Tools page instead, click the Back button

Figure 9-6 Sample Order Status Page

Administration Tools home ?

**BEA WebLogic Commerce Server**

Order Status: 2

|                             |  |                    |                      |
|-----------------------------|--|--------------------|----------------------|
| <b>Confirmation Number</b>  | 2  | <b>Customer Id</b> | democustomer         |
| <b>Order Status</b>         | Submitted  | <b>Name</b>        | Customer Demo        |
| <b>Date Ordered</b>         | 2000-09-18   | <b>Work Phone</b>  | 708-555-5555         |
| <b>Splitting Preference</b> | Ship all at once                                       | <b>Home Phone</b>  | 617-555-5555         |
| <b>Special Instructions</b> |  | <b>Email</b>       | democustomer@bea.com |
| <b>Shipping Address</b>     | One Main Street<br>DENVER<br>CO-80212<br>United States |                    |                      |

| ID       | Description        | Quantity | Unit Price |
|----------|--------------------|----------|------------|
| 9-WR8913 | extension-9-WR8913 | 1        | USD 82.50  |

|                                |           |
|--------------------------------|-----------|
| <b>Shipping &amp; Handling</b> | USD 4.95  |
| <b>Total Tax</b>               | USD 6.38  |
| <b>Total Due</b>               | USD 93.83 |

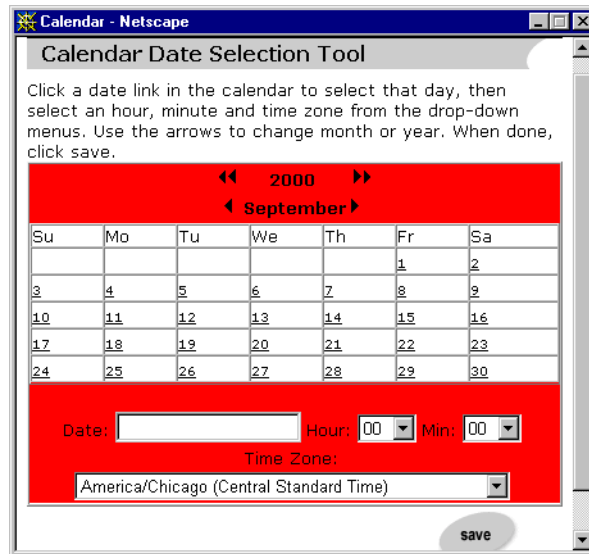
back

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

## Searching for an Order by Date Range

Another way in which an administrator of the site can search for a customer's order is by using a date range. Date ranges must be specified using the Calendar Date Selection Tool, shown in Figure 9-7.

Figure 9-7 The Calendar Date Selection Tool



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 shows the BEA WebLogic Commerce Server Administration Tools interface. At the top, there is a red navigation bar with the BEA logo, "Administration Tools", and "BEA WebLogic Commerce Server". Below this is a dark blue bar with "Order Management" and a status message: "Obtained '1' orders between 2000-09-18 and 2000-09-21". The main content area is titled "Order Search" and includes a search icon and the instruction: "Use this page to search for Orders. Click on order id to edit it." There are three search options: "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, calendar icons, and a "search" button. Below the search options is a table titled "Order List" with the following data:

| Identifier        | Create Date | Price | Customer Id                  |
|-------------------|-------------|-------|------------------------------|
| <a href="#">2</a> | 2000-09-18  | 93.83 | <a href="#">democustomer</a> |

At the bottom of the table 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

The screenshot shows the BEA WebLogic Commerce Server interface. At the top, there is a red navigation bar with the BEA logo on the left, "Administration Tools" and "BEA WebLogic Commerce Server" in the center, and "home", a book icon, and a question mark icon on the right. Below the navigation bar, a dark grey bar displays "Order Status: 2".

The main content area is divided into two columns of key-value pairs:

|                             |  |                    |                      |
|-----------------------------|--|--------------------|----------------------|
| <b>Confirmation Number</b>  | 2  | <b>Customer Id</b> | democustomer         |
| <b>Order Status</b>         | Submitted  | <b>Name</b>        | Customer Demo        |
| <b>Date Ordered</b>         | 2000-09-18   | <b>Work Phone</b>  | 708-555-5555         |
| <b>Splitting Preference</b> | Ship all at once                                       | <b>Home Phone</b>  | 617-555-5555         |
| <b>Special Instructions</b> |  | <b>Email</b>       | democustomer@bea.com |
| <b>Shipping Address</b>     | One Main Street<br>DENVER<br>CO-80212<br>United States |                    |                      |

Below this information is a table with the following columns: ID, Description, Quantity, and Unit Price.

| ID       | Description        | Quantity | Unit Price |
|----------|--------------------|----------|------------|
| 9-WR8913 | extension-9-WR8913 | 1        | USD 82.50  |

At the bottom right of the table, there is a summary of charges:

|                                |           |
|--------------------------------|-----------|
| <b>Shipping &amp; Handling</b> | USD 4.95  |
| <b>Total Tax</b>               | USD 6.38  |
| <b>Total Due</b>               | USD 93.83 |

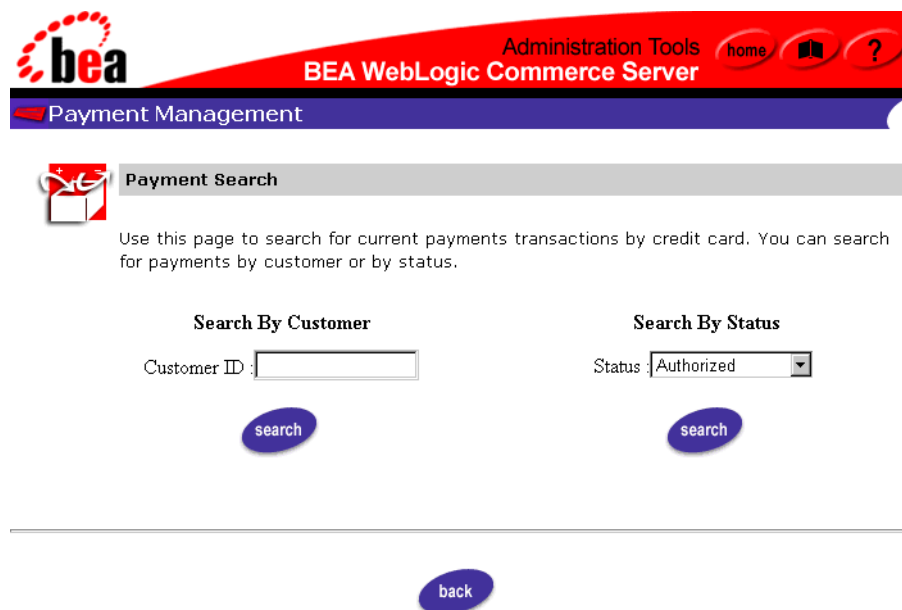
At the bottom left, there is a circular "back" button.

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

## Using the Payment Management Search Page

The Payment Management search page (shown in Figure 9-10) appears when you click the icon on the Payment Management section title bar. This section explains the three different searches and transaction modification activities that are available to an administrator for payment management.

**Figure 9-10** 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 9-3.

Figure 9-11 Sample Results for Payment Search by Customer ID

The screenshot shows the BEA WebLogic Commerce Server Administration Tools interface. The page title is "Payment Management" and it indicates "Obtained '2' transactions for customer democustomer". Below this is a "Payment Search" section with instructions: "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 text input for "Customer ID" and a "search" button, and "Search By Status" with a dropdown menu set to "Authorized" and a "search" button. Below the search options is a "Payment Transaction History" table with two rows of data. The first row shows a transaction from 2000-09-15 with a status of "MarkedForSettle" and a "Settle" button. The second row shows a transaction from 2000-09-18 with a status of "Retry" and an "Authorize" button. A "back" button is located below the table.

| Date       | Transaction Id | Amount | Status          | Credit Card    | New Amount           |  |
|------------|----------------|--------|-----------------|----------------|----------------------|--|
| 2000-09-15 | 1969038797504  | 54.18  | MarkedForSettle | xxxxxxxxxx1111 | <input type="text"/> | <input type="button" value="Settle"/>    |
| 2000-09-18 | 2969291417253  | 93.83  | Retry           | xxxxxxxxxx1111 |                      | <input type="button" value="Authorize"/> |

## **9** *Using the Order and Payment Management Pages*

---

For a detailed explanation of the Payment Transaction History fields and further payment management activities, refer to “Authorizing, Capturing, and Settling Payments” on page 9-16.

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 9-12.

Figure 9-12 Sample Results for Payment Search by Status

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 is a "Payment Management" section with a sub-header "Obtained '1' transactions for status Retry". A "Payment Search" section follows, with instructions: "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 text input for "Customer ID" and a "search" button, and "Search By Status" with a 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  | xxxxxxxxxxxx1111 | <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 9-16.

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 9-13).

Figure 9-13 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**

Customer ID:

**Search By Status**

Status:

search search

---

**Payment Transaction History**

| Date       | Transaction Id | Amount | Status          | Credit Card    | New Amount   |
|------------|----------------|--------|-----------------|----------------|--|
| 2000-09-15 | 1969038797504  | 54.18  | MarkedForSettle | xxxxxxxxxx1111 | <input type="text"/> <input type="button" value="Settle"/> |
| 2000-09-18 | 2969291417253  | 93.83  | Retry           | xxxxxxxxxx1111 | <input type="button" value="Authorize"/>                   |

back

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 6-29.

## 9 Using the Order and Payment Management Pages

---

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 6-31.

### 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.



---

# Index

## A

- accessor method(s)
  - attributes of 3-10
  - ShoppingCart 3-10
  - ShoppingCartLine 3-11
- addaddress.jsp 4-19
- adding run-time attributes 8-6, 8-8
- Administration Tools page
  - loading 9-2
  - sample page 9-3
- attributes
  - accessor method 3-10
- authorizing transactions 9-16

## B

- browsing the product catalog 1-3
- business logic 1-2, 3-8

## C

- Calendar Date Selection Tool
  - about 9-8
- capturing transactions 9-18
- checkout process 3-3
- checkout.jsp 7-2
- confirmorder.jsp 7-14
- create-\* procedure 2-20
- credit card
  - customizing security settings 6-41
  - encryption/decryption 6-41

- private key encryption password 6-44
- processing using CyberCash 6-28
- security 6-40

## Customer ID

- searching for a payment by 9-13
- searching for an order by 9-1, 9-4
- customer support contact information xiv
- CyberCash
  - about 6-28
  - configuring 6-29
  - integration with 6-28

## D

- database schema 1-5, 2-1
- database table
  - WLCS\_CREDIT\_CARD 2-8
  - WLCS\_CUSTOMER 2-4
  - WLCS\_ORDER 2-14
  - WLCS\_ORDER\_LINE 2-16
  - WLCS\_SAVED\_ITEM\_LIST 2-13
  - WLCS\_SECURITY 2-19
  - WLCS\_SHIPPING\_ADDRESS 2-6
  - WLCS\_SHIPPING\_METHOD 2-17
  - WLCS\_TRANSACTION 2-10
  - WLCS\_TRANSACTION\_ENTRY 2-12
- date
  - range
    - searching for an order by 9-1, 9-8
- default Webflow 1-3
- documentation, where to find it xii

---

dynamic data display  
  checkout.jsp 7-7  
  confirmorder.jsp 7-17  
  payment.jsp 6-6  
  paymenteditcc.jsp 6-17  
  selectaddress.jsp 4-15  
  selecttaxaddress.jsp 5-6  
  shipping.jsp 4-6  
  shoppingcart.jsp 3-9

## E

entity beans 8-4  
Entity-Relation diagram 2-2  
event(s)  
  addaddress.jsp 4-22  
  checkout.jsp 7-6  
  payment.jsp 6-6  
  paymenteditcc.jsp 6-17  
  paymentnewcc.jsp 6-11  
  selecttaxaddress.jsp 5-6  
  shipping.jsp 4-6  
  shoppingcart.jsp 3-8

## G

getShoppingCartLineCollection() 7-10

## H

high-level architecture 1-3

## I

input processors  
  DecideShippingAddressPageIP 5-10  
  DeleteProductItemFromShoppingCartIP  
    3-14  
  EmptyShoppingCartIP 3-15  
  InitShippingMethodListIP 4-25  
  InitShoppingCartIP 3-15  
  PaymentAuthorizationIP 6-22

**I-2**    Order Processing Package

UpdatePaymentInfoIP 6-23  
UpdateShippingAddressIP 4-26, 5-11  
UpdateShoppingCartQuantitiesIP 3-16  
UpdateSkuIP 3-17  
ValidateAddressIP 4-27  
ValidateShippingInfoIP 4-28

## J

Java scriptlets 3-10  
JavaServer Page (JSP) templates  
  addaddress.jsp 4-19  
  checkout.jsp 7-2  
  confirmorder.jsp 7-14  
  Order Processing package 1-2  
  payment.jsp 6-2  
  paymenteditcc.jsp 6-14  
  paymentnewcc.jsp 6-8  
  selecttaxaddress.jsp 5-2  
  shoppingcart.jsp 3-2  
JSP tags  
  getPipelineProperty 3-9, 5-6  
  getProfile 4-15, 6-6, 6-17  
  getProperty 4-15, 6-6, 6-18

## O

Order Identifier number  
  searching for an order by 9-1, 9-6  
Order List  
  about 9-5  
Order Management page  
  loading Administration Tools page 9-2  
  search page 9-4  
order processing database 2-2  
  Cloudscape and Oracle versions 2-20  
Order Processing package  
  about 1-2  
  structure of 1-2  
Order Status page 9-5  
order(s)

---

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

## **P**

payment models  
host-based 6-31  
switching between two 6-33  
terminal-based 6-31  
Payment Transaction History  
about 9-13  
payment(s)  
search  
by Customer ID 9-13  
by status 9-14  
status 9-16  
payment.jsp 6-2  
paymenteditcc.jsp 6-14  
paymentnewcc.jsp 6-8  
persistence architecture 8-3  
Pipeline components  
AddShippingAddressPC 4-29  
CalculateShippingPC 4-30  
CommitOrderPC 7-24  
DeleteProductItemFromSavedListPC 3-18  
DeleteShippingAddressPC 4-31  
MoveProductItemToSavedListPC 3-19  
MoveProductItemToShippingCartPC 3-20  
PaymentAuthorizationHostPC 6-24  
PaymentAuthorizationTerminalPC 6-26  
RefreshSavedListPC 3-21  
ResetOrderCheckoutPC 7-25  
TaxCalculateAndCommitLineLevelPC 5-13  
TaxCalculateLineLevelPC 5-12  
TaxVerifyShippingAddressPC 5-13  
printing product documentation xiii

## **R**

registration and user processing 1-3  
related information xiii  
retrieving Pipeline session attributes  
shopping cart 3-10

## **S**

schema extension 8-9  
scriptlets, Java 3-10  
search  
order  
by Customer ID 9-1, 9-4  
by date range 9-1, 9-8  
by Order Identifier number 9-1, 9-6  
payment  
by Customer ID 9-13  
by status 9-14  
security  
credit card 6-40  
selectaddress.jsp 4-10  
selecttaxaddress.jsp 5-2  
services, order-related 1-1  
settling transactions 9-18  
Shipping Services  
about 4-1  
addaddress.jsp template 4-19  
selectaddress.jsp template 4-10  
shipping.jsp template 4-2  
shipping.jsp 4-2  
shopping cart  
managing 3-1  
Shopping Cart Management Services 3-1  
shoppingcart.jsp 3-2  
SQL files 2-20  
StartCommerce.bat 9-2  
StartCommerce.sh 9-2  
starting the WebLogic Commerce Server  
Administration Tools 9-1, 9-2  
status  
of payments

---

searching by 9-14  
values for 9-16  
support  
technical xiv

## T

Taxation Services  
removing 5-34

### TAXWARE

checklist 5-33  
configuration and deployment 5-21  
considerations 5-16  
installing on UNIX 5-18  
installing on Windows 5-17  
integrating with 5-15  
run-time configuration 5-29  
SALES/USE Tax System 5-15  
specific properties 5-24  
tax codes and product catalog 5-32  
to calculate taxes 5-15  
Universal Tax Link (UTL) System 5-15  
VERAZIP System 5-15

transactions

authorizing 9-16  
capturing 9-18  
managing 8-18  
settling 9-18

## U

Unified User Profile (UUP) technology 8-6  
use-cases 8-2

## W

Webflow

modifying properties file 5-35  
property file 6-34

Webflow/Pipeline infrastructure 1-3

WebLogic Commerce Server

Administration Tools

starting 9-1, 9-2

WebLogic Commerce Server schema  
extension 8-9

WLCS\_CREDIT\_CARD 2-8

WLCS\_CUSTOMER 2-4

WLCS\_ORDER 2-14

WLCS\_ORDER\_LINE 2-16

WLCS\_SAVED\_ITEM\_LIST 2-13

WLCS\_SECURITY 2-19

WLCS\_SHIPPING\_ADDRESS 2-6

WLCS\_SHIPPING\_METHOD 2-17

WLCS\_TRANSACTION 2-10

WLCS\_TRANSACTION\_ENTRY 2-12

**I-4** Order Processing Package