BEA WebLogic Commerce Server

Order Processing Package

BEA WebLogic Commerce Server 3.1
Document Edition 1.2
August 2001

Order Processing Package

Document Edition	Date	Software Version
1.2	August 2001	WebLogic Commerce Server 3.1

Contents

About This Document

This document explains how to use the services available within the BEA WebLogic Commerce Server Order 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.

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 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 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 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 BEA WebLogic Commerce Server documents contain information that is relevant to using the Order Processing package and understanding how to customize or extend the provided services.

- *BEA WebLogic Commerce Server Webflow and Pipeline Management*
- *BEA WebLogic Commerce Server Registration and User Processing Package*
- *BEA WebLogic Commerce Server Product Catalog Management*

Contact Us!

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

In your e-mail message, please indicate that you are using the documentation for the BEA WebLogic Commerce Server 3.1 release.

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

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

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

Documentation Conventions

The following documentation conventions are used throughout this document.

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

Convention	Item
italics	Indicates emphasis or book titles.

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

`monospace boldface text`	Identifies significant words in code. *Example*: `void` **`commit`** `( )`
`monospace italic text`	Identifies variables in code. *Example*: `String` *`expr`*
UPPERCASE TEXT	Indicates device names, environment variables, and logical operators. *Examples*: LPT1 SIGNON OR
`{ }`	Indicates a set of choices in a syntax line. The braces themselves should never be typed.

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

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

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

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

CHAPTER

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:

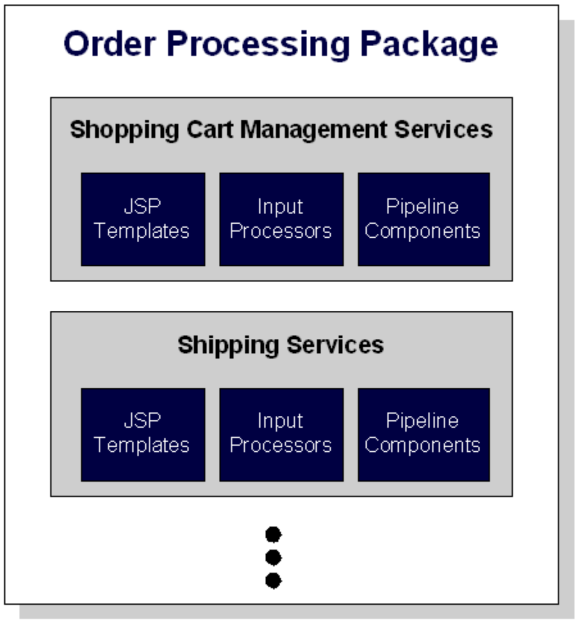
- What Is the Order Processing Package?
- High-level Architecture
- Development Roles
- Next Steps

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 *BEA WebLogic Commerce Server WebFlow and Pipeline Management.*

In addition to the services available for order processing, the BEA 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 *BEA WebLogic Commerce Server Product Catalog Management*. For information on services related to registration and user processing, see *BEA WebLogic Commerce Server Registration and User Processing Package*.

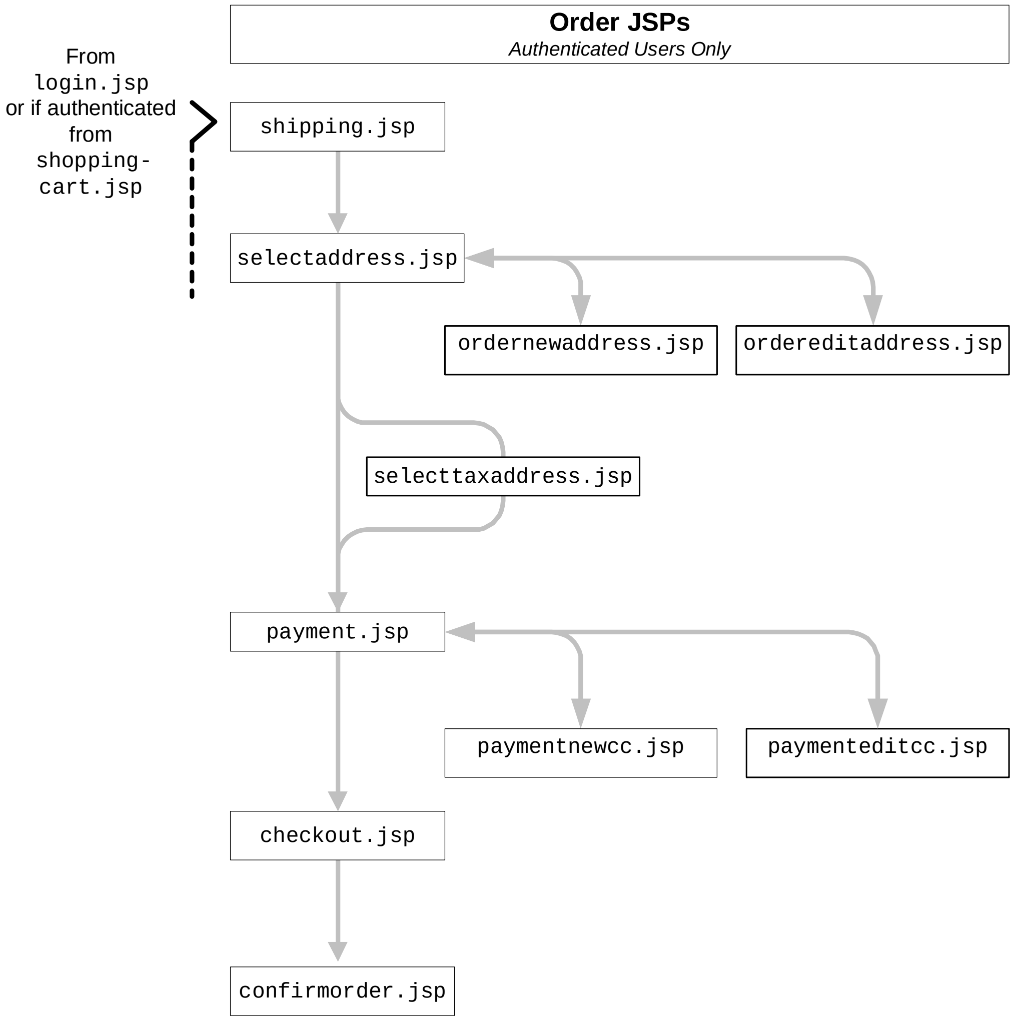
High-level Architecture

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 *BEA WebLogic Commerce Server Webflow and Pipeline Management*.
- For more information about the Registration and User Processing package, see *BEA 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).

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 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"

- “Shipping Services”
- “Taxation Services”
- “Payment Services”
- “Order Summary and Confirmation Services”

CHAPTER

2 The Order Processing Database Schema

This topic describes the database schema for the BEA 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.

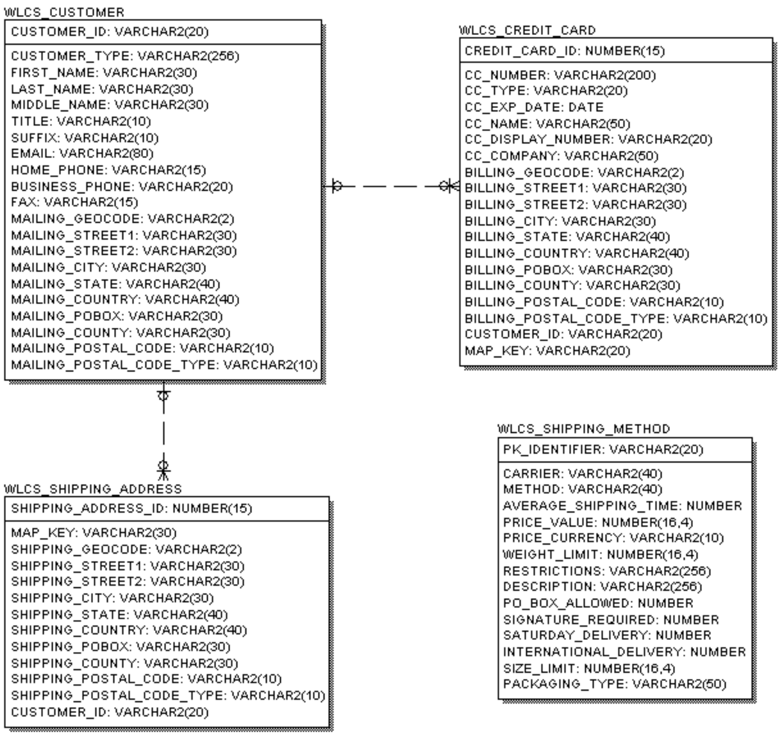
This topic includes the following sections:

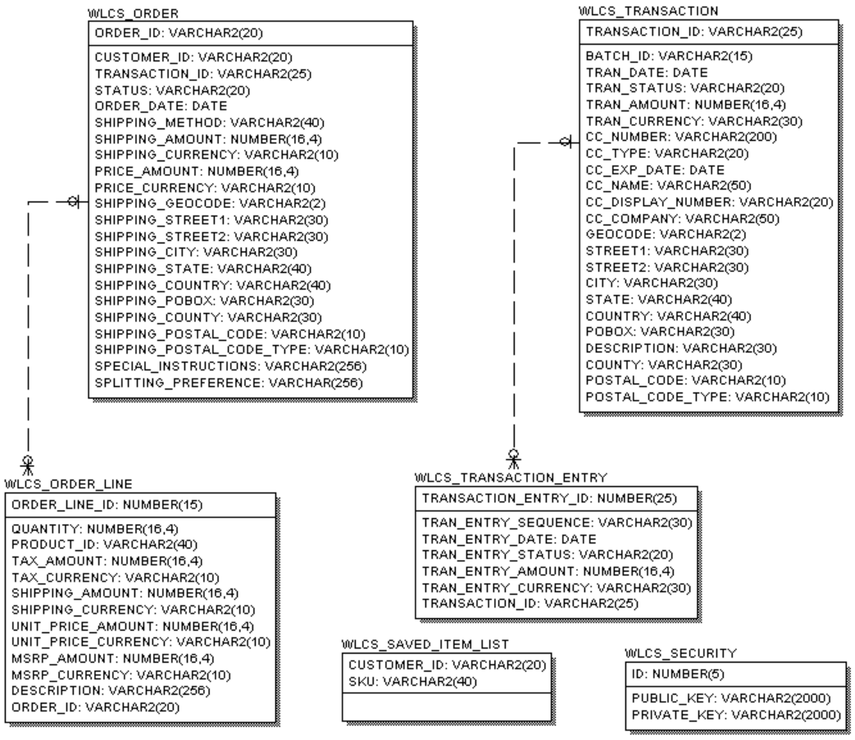
- The Entity-Relation Diagram
- The WLCS_CUSTOMER Database Table
- The WLCS_SHIPPING_ADDRESS Database Table
- The WLCS_TRANSACTION Database Table
- The WLCS_TRANSACTION_ENTRY Database Table
- The WLCS_SAVED_ITEM_LIST Database Table
- The WLCS_ORDER Database Table
- The WLCS_ORDER_LINE Database Table
- 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 BEA WebLogic Commerce Server order processing database.

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


WLCS_CUSTOMER
CUSTOMER_ID: VARCHAR2(20)
CUSTOMER_TYPE: VARCHAR2(256)
FIRST_NAME: VARCHAR2(30)
LAST_NAME: VARCHAR2(30)
MIDDLE_NAME: VARCHAR2(30)
TITLE: VARCHAR2(10)
SUFFIX: VARCHAR2(10)
EMAIL: VARCHAR2(80)
HOME_PHONE: VARCHAR2(15)
BUSINESS_PHONE: VARCHAR2(20)
FAX: VARCHAR2(15)
MAILING_GEOCODE: VARCHAR2(2)
MAILING_STREET1: VARCHAR2(30)
MAILING_STREET2: VARCHAR2(30)
MAILING_CITY: VARCHAR2(30)
MAILING_STATE: VARCHAR2(40)
MAILING_COUNTRY: VARCHAR2(40)
MAILING_POBOX: VARCHAR2(30)
MAILING_COUNTY: VARCHAR2(30)
MAILING_POSTAL_CODE: VARCHAR2(10)
MAILING_POSTAL_CODE_TYPE: VARCHAR2(10)
WLCS_CREDIT_CARD
CREDIT_CARD_ID: NUMBER(15)
CC_NUMBER: VARCHAR2(200)
CC_TYPE: VARCHAR2(20)
CC_EXP_DATE: DATE
CC_NAME: VARCHAR2(50)
CC_DISPLAY_NUMBER: VARCHAR2(20)
CC_COMPANY: VARCHAR2(50)
BILLING_GEOCODE: VARCHAR2(2)
BILLING_STREET1: VARCHAR2(30)
BILLING_STREET2: VARCHAR2(30)
BILLING_CITY: VARCHAR2(30)
BILLING_STATE: VARCHAR2(40)
BILLING_COUNTRY: VARCHAR2(40)
BILLING_POBOX: VARCHAR2(30)
BILLING_COUNTY: VARCHAR2(30)
BILLING_POSTAL_CODE: VARCHAR2(10)
BILLING_POSTAL_CODE_TYPE: VARCHAR2(10)
CUSTOMER_ID: VARCHAR2(20)
MAP_KEY: VARCHAR2(20)
WLCS_SHIPPING_METHOD
PK_IDENTIFIER: VARCHAR2(20)
CARRIER: VARCHAR2(40)
METHOD: VARCHAR2(40)
AVERAGE_SHIPPING_TIME: NUMBER
PRICE_VALUE: NUMBER(16,4)
PRICE_CURRENCY: VARCHAR2(10)
WEIGHT_LIMIT: NUMBER(16,4)
RESTRICTIONS: VARCHAR2(256)
DESCRIPTION: VARCHAR2(256)
PO_BOX_ALLOWED: NUMBER
SIGNATURE_REQUIRED: NUMBER
SATURDAY_DELIVERY: NUMBER
INTERNATIONAL_DELIVERY: NUMBER
SIZE_LIMIT: NUMBER(16,4)
PACKAGING_TYPE: VARCHAR2(50)
WLCS_SHIPPING_ADDRESS
SHIPPING_ADDRESS_ID: NUMBER(15)
MAP_KEY: VARCHAR2(30)
SHIPPING_GEOCODE: VARCHAR2(2)
SHIPPING_STREET1: VARCHAR2(30)
SHIPPING_STREET2: VARCHAR2(30)
SHIPPING_CITY: VARCHAR2(30)
SHIPPING_STATE: VARCHAR2(40)
SHIPPING_COUNTRY: VARCHAR2(40)
SHIPPING_POBOX: VARCHAR2(30)
SHIPPING_COUNTY: VARCHAR2(30)
SHIPPING_POSTAL_CODE: VARCHAR2(10)
SHIPPING_POSTAL_CODE_TYPE: VARCHAR2(10)
CUSTOMER_ID: VARCHAR2(20)


WLCS_ORDER
ORDER_ID: VARCHAR2(20)
CUSTOMER_ID: VARCHAR2(20)
TRANSACTION_ID: VARCHAR2(25)
STATUS: VARCHAR2(20)
ORDER_DATE: DATE
SHIPPING_METHOD: VARCHAR2(40)
SHIPPING_AMOUNT: NUMBER(16,4)
SHIPPING_CURRENCY: VARCHAR2(10)
PRICE_AMOUNT: NUMBER(16,4)
PRICE_CURRENCY: VARCHAR2(10)
SHIPPING_GEOCODE: VARCHAR2(2)
SHIPPING_STREET1: VARCHAR2(30)
SHIPPING_STREET2: VARCHAR2(30)
SHIPPING_CITY: VARCHAR2(30)
SHIPPING_STATE: VARCHAR2(40)
SHIPPING_COUNTRY: VARCHAR2(40)
SHIPPING_POBOX: VARCHAR2(30)
SHIPPING_COUNTY: VARCHAR2(30)
SHIPPING_POSTAL_CODE: VARCHAR2(10)
SHIPPING_POSTAL_CODE_TYPE: VARCHAR2(10)
SPECIAL_INSTRUCTIONS: VARCHAR2(256)
SPLITTING_PREFERENCE: VARCHAR(256)
WLCS_TRANSACTION
TRANSACTION_ID: VARCHAR2(25)
BATCH_ID: VARCHAR2(15)
TRAN_DATE: DATE
TRAN_STATUS: VARCHAR2(20)
TRAN_AMOUNT: NUMBER(16,4)
TRAN_CURRENCY: VARCHAR2(30)
CC_NUMBER: VARCHAR2(200)
CC_TYPE: VARCHAR2(20)
CC_EXP_DATE: DATE
CC_NAME: VARCHAR2(50)
CC_DISPLAY_NUMBER: VARCHAR2(20)
CC_COMPANY: VARCHAR2(50)
GEOCODE: VARCHAR2(2)
STREET1: VARCHAR2(30)
STREET2: VARCHAR2(30)
CITY: VARCHAR2(30)
STATE: VARCHAR2(40)
COUNTRY: VARCHAR2(40)
POBOX: VARCHAR2(30)
DESCRIPTION: VARCHAR2(30)
COUNTY: VARCHAR2(30)
POSTAL_CODE: VARCHAR2(10)
POSTAL_CODE_TYPE: VARCHAR2(10)
WLCS_ORDER_LINE
ORDER_LINE_ID: NUMBER(15)
QUANTITY: NUMBER(16,4)
PRODUCT_ID: VARCHAR2(40)
TAX_AMOUNT: NUMBER(16,4)
TAX_CURRENCY: VARCHAR2(10)
SHIPPING_AMOUNT: NUMBER(16,4)
SHIPPING_CURRENCY: VARCHAR2(10)
UNIT_PRICE_AMOUNT: NUMBER(16,4)
UNIT_PRICE_CURRENCY: VARCHAR2(10)
MSRP_AMOUNT: NUMBER(16,4)
MSRP_CURRENCY: VARCHAR2(10)
DESCRIPTION: VARCHAR2(256)
ORDER_ID: VARCHAR2(20)
WLCS_TRANSACTION_ENTRY
TRANSACTION_ENTRY_ID: NUMBER(25)
TRAN_ENTRY_SEQUENCE: VARCHAR2(30)
TRAN_ENTRY_DATE: DATE
TRAN_ENTRY_STATUS: VARCHAR2(20)
TRAN_ENTRY_AMOUNT: NUMBER(16,4)
TRAN_ENTRY_CURRENCY: VARCHAR2(30)
TRANSACTION_ID: VARCHAR2(25)
WLCS_SAVED_ITEM_LIST
CUSTOMER_ID: VARCHAR2(20)
SKU: VARCHAR2(40)
WLCS_SECURITY
ID: NUMBER(5)
PUBLIC_KEY: VARCHAR2(2000)
PRIVATE_KEY: VARCHAR2(2000)

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 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	Cloudscape Type	Oracle Type	Description and Recommendations
CUSTOMER_ID	VARCHAR(20)	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	VARCHAR(256)	VARCHAR2(256)	A label for the customer (such as preferred, standard, or business).
FIRST_NAME	VARCHAR(30)	VARCHAR2(30)	The customer's first name.
LAST_NAME	VARCHAR(30)	VARCHAR2(30)	The customer's last name.
MIDDLE_NAME	VARCHAR(30)	VARCHAR2(30)	The customer's middle name.
TITLE	VARCHAR(10)	VARCHAR2(10)	The customer's preferred title (Mr., Mrs., Ms.).
SUFFIX	VARCHAR(10)	VARCHAR2(10)	The customer's preferred suffix (Jr., Sr.).
EMAIL	VARCHAR(80)	VARCHAR2(80)	The customer's email address.
HOME_PHONE	VARCHAR(15)	VARCHAR2(15)	The customer's home phone number.

Table 2-1 WLCS_CUSTOMER Table Metadata

Column Name	Cloudscape Type	Oracle Type	Description and Recommendations
BUSINESS_PHONE	VARCHAR(20)	VARCHAR2(20)	The customer’s business phone number.
FAX	VARCHAR(15)	VARCHAR2(15)	The customer’s fax number.
MAILING_GEOCODE	VARCHAR(10)	VARCHAR2(2)	The code used by the TAXWARE system to identify taxes for the order based on jurisdiction.
MAILING_STREET1	VARCHAR(30)	VARCHAR2(30)	The first line in the customer’s street address.
MAILING_STREET2	VARCHAR(30)	VARCHAR2(30)	The second line in the customer’s street address.
MAILING_CITY	VARCHAR(30)	VARCHAR2(30)	The city in the customer’s address.
MAILING_STATE	VARCHAR(40)	VARCHAR2(40)	The state in the customer’s address.
MAILING_COUNTRY	VARCHAR(40)	VARCHAR2(40)	The country in the customer’s address.
MAILING_POBOX	VARCHAR(30)	VARCHAR2(30)	The post office box in the customer’s address.
MAILING_COUNTY	VARCHAR(30)	VARCHAR2(30)	The county in the customer’s address.
MAILING_POSTAL_CODE	VARCHAR(10)	VARCHAR2(10)	The postal (zip) code in the customer’s address.
MAILING_POSTAL_CODE _TYPE	VARCHAR(10)	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 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	Cloudscape Type	Oracle Type	Description and Recommendations
SHIPPING_ADDRESS_ID	INTEGER	NUMBER(15)	A unique identifier for the shipping address. This field is the table's primary key and cannot be `NULL`. All other fields in the `WLCS_SHIPPING_ADDRESS` table can be `NULL`.
CUSTOMER_ID	VARCHAR(20)	VARCHAR2(20)	A unique identifier for the customer.
MAP_KEY	VARCHAR(30)	VARCHAR2(30)	Key that maps multiple shipping addresses with a single customer.
SHIPPING_GEOCODE	VARCHAR(2)	VARCHAR2(2)	The code used by the TAXWARE system to identify taxes for the order based on jurisdiction.
SHIPPING_STREET1	VARCHAR(30)	VARCHAR2(30)	The first line in the customer's shipping address.
SHIPPING_STREET2	VARCHAR(30)	VARCHAR2(30)	The second line in the customer's shipping address.

Table 2-2 WLCS_SHIPPING_ADDRESS Table Metadata

Column Name	Cloudscape Type	Oracle Type	Description and Recommendations
SHIPPING_CITY	VARCHAR(30)	VARCHAR2(30)	The city in the customer's shipping address.
SHIPPING_STATE	VARCHAR(40)	VARCHAR2(40)	The state in the customer's shipping address.
SHIPPING_COUNTRY	VARCHAR(40)	VARCHAR2(40)	The country in the customer's shipping address.
SHIPPING_POBOX	VARCHAR(30)	VARCHAR2(30)	The post office box in the customer's shipping address.
SHIPPING_COUNTY	VARCHAR(30)	VARCHAR2(30)	The county in the customer's shipping address.
SHIPPING_POSTAL_CODE	VARCHAR(10)	VARCHAR2(10)	The postal (zip) code in the customer's shipping address.
SHIPPING_POSTAL_CODE _TYPE	VARCHAR(10)	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 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	Cloudscape Type	Oracle Type	Description and Recommendations
CREDIT_CARD_ID	INTEGER	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	VARCHAR(20)	VARCHAR2(20)	A unique identifier for the customer.
MAP_KEY	VARCHAR(20)	VARCHAR2(20)	Key that maps multiple credit cards with a single customer.
CC_NUMBER	VARCHAR(200)	VARCHAR2(200)	The customer's credit card number. This is encrypted if `is.encryption.enable` is set to `true` in the `weblogiccommerce.properties` file.
CC_TYPE	VARCHAR(20)	VARCHAR2(20)	The customer's credit card type (VISA, MasterCard, and so on).
CC_EXP_DATE	DATE	DATE	The expiration date on the customer's credit card.

Table 2-3 WLCS_CREDIT_CARD Table Metadata

Column Name	Cloudscape Type	Oracle Type	Description and Recommendations
CC_NAME	VARCHAR(50)	VARCHAR2(50)	The credit card holder's name.
CC_DISPLAY_NUMBER	VARCHAR(20)	VARCHAR2(20)	The version of the credit card number that is displayed (all Xs except last 4-digits).
CC_COMPANY	VARCHAR(50)	VARCHAR2(50)	The name of the credit card company.
BILLING_GEOCODE	VARCHAR(2)	VARCHAR2(2)	The code used by the TAXWARE system to identify taxes for the order based on jurisdiction.
BILLING_STREET1	VARCHAR(30)	VARCHAR2(30)	The first line in the customer's billing address.
BILLING_STREET2	VARCHAR(30)	VARCHAR2(30)	The second line in the customer's billing address.
BILLING_CITY	VARCHAR(30)	VARCHAR2(30)	The city in the customer's billing address.
BILLING_STATE	VARCHAR(40)	VARCHAR2(40)	The state in the customer's billing address.
BILLING_COUNTRY	VARCHAR(40)	VARCHAR2(40)	The country in the customer's billing address.
BILLING_POBOX	VARCHAR(30)	VARCHAR2(30)	The post office box in the customer's billing address.
BILLING_COUNTY	VARCHAR(30)	VARCHAR2(30)	The county in the customer's billing address.
BILLING_POSTAL_CODE	VARCHAR(10)	VARCHAR2(10)	The postal (zip) code in the customer's billing address.

Table 2-3 WLCS_CREDIT_CARD Table Metadata

Column Name	Cloudscape Type	Oracle Type	Description and Recommendations
BILLING_POSTAL_CODE _TYPE	VARCHAR(10)	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 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	Cloudscape Type	Oracle Type	Description and Recommendations
TRANSACTION_ID	VARCHAR(25)	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	VARCHAR(15)	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	DATE	The date of the transaction (date on which the transaction was first started).
TRAN_STATUS	VARCHAR(20)	VARCHAR2(20)	The current status of the transaction (Settled, Authorized, MarkedForSettle, PendingSettle, Retry, Settled).

Table 2-4 WLCS_TRANSACTION Table Metadata

Column Name	Cloudscape Type	Oracle Type	Description and Recommendations
TRAN_AMOUNT	DOUBLE PRECISION	NUMBER(16,4)	The most recent amount applied to the transaction (MarkForSettle amounts can be different from the authorization amount).
TRAN_CURRENCY	VARCHAR(30)	VARCHAR2(30)	The currency of the transaction.
CC_NUMBER	VARCHAR(200)	VARCHAR2(200)	The customer's credit card number. This is encrypted if `is.encryption.enable` is set to `true` in the `weblogiccommerce.properties` file.
CC_TYPE	VARCHAR(20)	VARCHAR2(20)	The customer's credit card type (VISA, MasterCard, and so on).
CC_EXP_DATE	DATE	DATE	The expiration date on the customer's credit card.
CC_NAME	VARCHAR(50)	VARCHAR2(50)	The credit card holder's name.
CC_DISPLAY_NUMBER	VARCHAR(20)	VARCHAR2(20)	The version of the credit card number that is displayed (all Xs except last 4-digits).
CC_COMPANY	VARCHAR(50)	VARCHAR2(50)	The name of the credit card company.
GEOCODE	VARCHAR(2)	VARCHAR2(2)	The code used by the TAXWARE system to identify taxes for the order based on jurisdiction.

Table 2-4 WLCS_TRANSACTION Table Metadata

Column Name	Cloudscape Type	Oracle Type	Description and Recommendations
STREET1	VARCHAR(30)	VARCHAR2(30)	The first line in the customer’s street address.
STREET2	VARCHAR(30)	VARCHAR2(30)	The second line in the customer’s street address.
CITY	VARCHAR(30)	VARCHAR2(30)	The city in the customer’s address.
STATE	VARCHAR(40)	VARCHAR2(40)	The state in the customer’s address.
COUNTRY	VARCHAR(40)	VARCHAR2(40)	The country in the customer’s address.
POBOX	VARCHAR(30)	VARCHAR2(30)	The post office box in the customer’s address.
COUNTY	VARCHAR(30)	VARCHAR2(30)	The county in the customer’s address.
POSTAL_CODE	VARCHAR(10)	VARCHAR2(10)	The postal (zip) code in the customer’s address.
POSTAL_CODE_TYPE	VARCHAR(10)	VARCHAR2(10)	Format or type of postal code, generally determined by country (such as zip code in the United States).
DESCRIPTION	VARCHAR(30)	VARCHAR2(30)	Any additional data. Can be `NULL`.

The WLCS_TRANSACTION_ENTRY Database Table

Table 2-5 describes the metadata for the 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	Cloudscape Type	Oracle Type	Description and Recommendations
TRANSACTION_ENTRY_ID	INTEGER	NUMBER(25)	A unique identifier for the transaction entry. This field is the table's primary key and cannot be `NULL`. All other fields in the `WLCS_TRANSACTION_ENTRY` table can be `NULL`.
TRAN_ENTRY_SEQUENCE	VARCHAR(30)	VARCHAR2(30)	Represents the running count per transaction.
TRAN_ENTRY_DATE	DATE	DATE	The date of the log entry.
TRAN_ENTRY_STATUS	VARCHAR(20)	VARCHAR2(20)	The status of the transaction when this entry was made.
TRAN_ENTRY_AMOUNT	DOUBLE PRECISION	NUMBER(16,4)	The amount of the transaction when the log entry was made.
TRAN_ENTRY_CURRENCY	VARCHAR(30)	VARCHAR2(30)	The currency of the transaction.
TRANSACTION_ID	VARCHAR(25)	VARCHAR2(25)	A unique identifier for the transaction.

The WLCS_SAVED_ITEM_LIST Database Table

Table 2-6 describes the metadata for the 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

Column Name	Cloudscape Type	Oracle Type	Description and Recommendations
CUSTOMER_ID	VARCHAR(20)	VARCHAR2(20)	A unique identifier for the customer.
SKU	VARCHAR(40)	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 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 BEA 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	Cloudscape Type	Oracle Type	Description and Recommendations
ORDER_ID	VARCHAR(20)	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	VARCHAR(20)	VARCHAR2(20)	A unique identifier for the customer.
TRANSACTION_ID	VARCHAR(25)	VARCHAR2(25)	A unique identifier for the transaction.
STATUS	VARCHAR(20)	VARCHAR2(20)	The status of the order.
ORDER_DATE	DATE	DATE	The date the order was placed.
SHIPPING_METHOD	VARCHAR(40)	VARCHAR2(40)	The method by which the order is to be shipped.
SHIPPING_AMOUNT	DOUBLE PRECISION	NUMBER(16,4)	The shipping amount for the order.
SHIPPING_CURRENCY	VARCHAR(10)	VARCHAR2(10)	The currency associated with the shipping amount.

Table 2-7 WLCS_ORDER Table Metadata

Column Name	Cloudscape Type	Oracle Type	Description and Recommendations
PRICE_AMOUNT	DOUBLE PRECISION	NUMBER(16,4)	The price of the order.
PRICE_CURRENCY	VARCHAR(10)	VARCHAR2(10)	The currency associated with the price.
SHIPPING_GEOGODE	VARCHAR(2)	VARCHAR2(2)	The code used by the TAXWARE system to identify taxes for the order based on jurisdiction.
SHIPPING_STREET1	VARCHAR(30)	VARCHAR2(30)	The first line in the customer’s shipping address.
SHIPPING_STREET2	VARCHAR(30)	VARCHAR2(30)	The second line in the customer’s shipping address.
SHIPPING_CITY	VARCHAR(30)	VARCHAR2(30)	The city in the customer’s shipping address.
SHIPPING_STATE	VARCHAR(40)	VARCHAR2(40)	The state in the customer’s shipping address.
SHIPPING_COUNTRY	VARCHAR(40)	VARCHAR2(40)	The country in the customer’s shipping address.
SHIPPING_POBOX	VARCHAR(30)	VARCHAR2(30)	The post office box in the customer’s shipping address.
SHIPPING_COUNTY	VARCHAR(40)	VARCHAR2(30)	The county in the customer’s shipping address.
SHIPPING_POSTAL_CODE	VARCHAR(10)	VARCHAR2(10)	The postal (zip) code in the customer’s shipping address.

Table 2-7 WLCS_ORDER Table Metadata

Column Name	Cloudscape Type	Oracle Type	Description and Recommendations
SHIPPING_POSTAL_CODE _TYPE	VARCHAR(10)	VARCHAR2(10)	Format or type of postal code, generally determined by country (such as zip code in the United States).
SPECIAL_INSTRUCTIONS	VARCHAR(256)	VARCHAR2(256)	Any special shipping instructions associated with the order.
SPLITTING_PREFERENCE	VARCHAR(256)	VARCHAR2(256)	The splitting preferences for the customer's order.

The WLCS_ORDER_LINE Database Table

Table 2-8 describes the metadata for the 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	Cloudscape Type	Oracle Type	Description and Recommendations
ORDER_LINE_ID	INTEGER	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	DOUBLE PRECISION	NUMBER(16,4)	The quantity of the item in the shopping cart.
PRODUCT_ID	VARCHAR(40)	VARCHAR2(40)	An identification number for the item in the shopping cart.
TAX_AMOUNT	DOUBLE PRECISION	NUMBER(16,4)	The tax amount for the order.
TAX_CURRENCY	VARCHAR(10)	VARCHAR2(10)	The currency associated with the tax amount.
SHIPPING_AMOUNT	DOUBLE PRECISION	NUMBER(16,4)	The shipping amount for the order.
SHIPPING_CURRENCY	VARCHAR(10)	VARCHAR2(10)	The currency associated with the shipping amount.
UNIT_PRICE_AMOUNT	DOUBLE PRECISION	NUMBER(16,4)	The unit price amount for the item.

Table 2-8 WLCS_ORDER_LINE Table Metadata

Column Name	Cloudscape Type	Oracle Type	Description and Recommendations
UNIT_PRICE_CURRENCY	VARCHAR(10)	VARCHAR2(10)	The currency associated with the unit price.
MSRP_AMOUNT	DOUBLE PRECISION	NUMBER(16,4)	The MSRP amount for the item.
MSRP_CURRENCY	VARCHAR(10)	VARCHAR2(10)	The currency associated with the MSRP amount.
DESCRIPTION	VARCHAR(256)	VARCHAR2(256)	The name of the item that is part of the order.
ORDER_ID	VARCHAR(20)	VARCHAR2(20)	A unique identifier for the order.

The WLCS_SHIPPING_METHOD Database Table

Table 2-9 describes the metadata for the Commerce Server WLCS_SHIPPING_METHOD table. This table is used to store information about the shiping method in the order processing database.

Table 2-9 WLCS_SHIPPING_METHOD Table Metadata

Column Name	Cloudscape Type	Oracle Type	Description and Recommendations
PK_IDENTIFIER	VARCHAR(20)	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	VARCHAR(40)	VARCHAR2(40)	The carrier being used to ship the order (such as UPS, FedEx, and so on).
METHOD	VARCHAR(40)	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	INTEGER	NUMBER	The average number of days it will take the order to arrive.
PRICE_VALUE	DOUBLE PRECISION	NUMBER(16,4)	The amount it will cost to ship the order.

Table 2-9 WLCS_SHIPPING_METHOD Table Metadata

Column Name	Cloudscape Type	Oracle Type	Description and Recommendations
PRICE_CURRENCY	VARCHAR(10)	VARCHAR2(10)	The currency associated with the `PRICE_VALUE` column (such as dollars, pounds, lira, and so on).
WEIGHT_LIMIT	DOUBLE PRECISION	NUMBER(16,4)	The weight limit for the shipment.
RESTRICTIONS	VARCHAR(256)	VARCHAR2(256)	Any restrictions associated with the shipment.
DESCRIPTION	VARCHAR(256)	VARCHAR2(256)	A description of the shipping method (such as FedEx Overnight or Standard).
PO_BOX_ALLOWED	INTEGER	NUMBER	Specifies whether or not the shipment can be left at a post office box.
SIGNATURE_REQUIRED	INTEGER	NUMBER	Specifies whether or not a signature is required upon receipt of the shipment.
SATURDAY_DELIVERY	INTEGER	NUMBER	Specifies whether or not the shipment can be delivered on Saturday.
INTERNATIONAL_ DELIVERY	INTEGER	NUMBER	Specifies whether or not international delivery is an option.
SIZE_LIMIT	DOUBLE PRECISION	NUMBER(16,4)	The size limit for the shipment.
PACKAGING_TYPE	VARCHAR(50)	VARCHAR2(50)	The packaging type for the shipment.

The WLCS_SECURITY Database Table

Table 2-10 describes the metadata for the 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 BEA WebLogic Commerce Server product.

Table 2-10 WLCS_SECURITY Table Metadata

Column Name	Cloudscape Type	Oracle Type	Description and Recommendations
ID	INTEGER	NUMBER(2)	A unique identifier for the key pair. This field is the table's primary key and cannot be `NULL`.
PUBLIC_KEY	VARCHAR(2000)	VARCHAR2(2000)	The public key to be used for encryption/decryption of credit cards.
PRIVATE_KEY	VARCHAR(2000)	VARCHAR2(2000)	The private key to be used for encryption/decryption of credit cards.

The SQL Files and Defined Constraints

The BEA 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

Table Name	Constraints as Defined in create-order-oracle.sql
WLCS_SHIPPING_ADDRESS	If a customer is deleted from the database, the CUSTOMER_FK constraint causes all their associated shipping addresses to be deleted. The constraint for the schema in Oracle is: CONSTRAINT CUSTOMER_FK REFERENCES WLCS_CUSTOMER(CUSTOMER_ID) ON DELETE CASCADE
WLCS_CREDIT_CARD	If a customer is deleted from the database, the CUSTOMER_CREDIT_CARD_FK constraint causes all their associated credit cards to be deleted. The constraint for the schema in Oracle is: CONSTRAINT CUSTOMER_CREDIT_CARD_FK REFERENCES WLCS_CUSTOMER(CUSTOMER_ID) ON DELETE CASCADE
WLCS_TRANSACTION_ENTRY	If a transaction is deleted from the database, the WLCS_TRANSACTION_FK constraint causes all the associated transaction entries be deleted. The constraint for the schema in Oracle is: CONSTRAINT WLCS_TRANSACTION_FK REFERENCES WLCS_TRANSACTION(TRANSACTION_ID) ON DELETE CASCADE
WLCS_ORDER_LINE	If an order is deleted from the database, the WLCS_ORDER_FK constraint causes all the associated order line items to be deleted. The constraint for the schema in Oracle is: CONSTRAINT ORDER_FK REFERENCES WLCS_ORDER(ORDER_ID) ON DELETE CASCADE

CHAPTER

3 Shopping Cart Management Services

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

This topic includes the following sections:

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

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.

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.

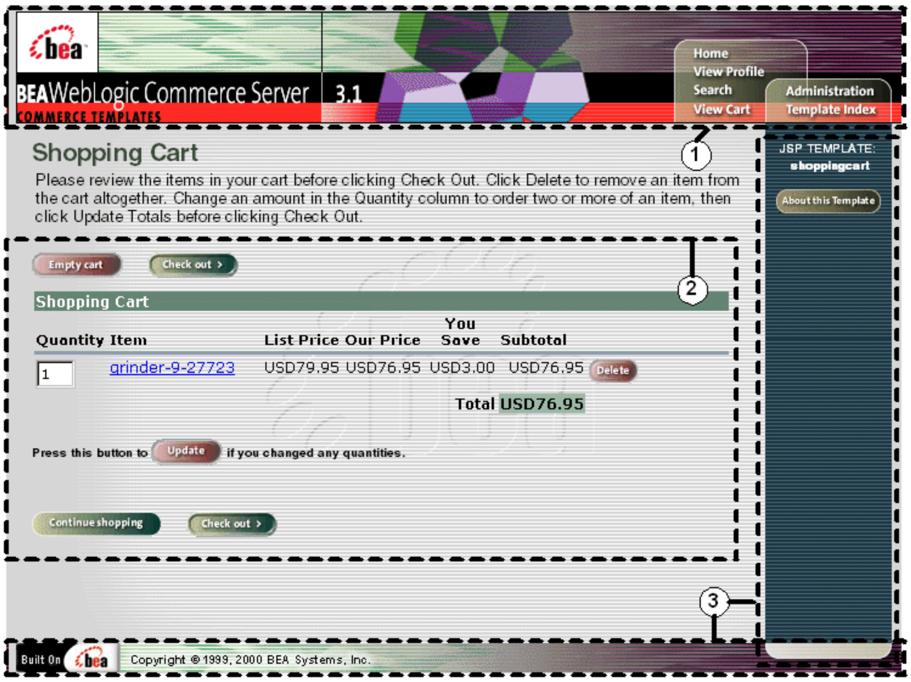
Note: If the customer is not logged into your e-commerce site, they will be prompted to do so before continuing to the next part of the checkout process.

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.

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



Shopping Cart

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

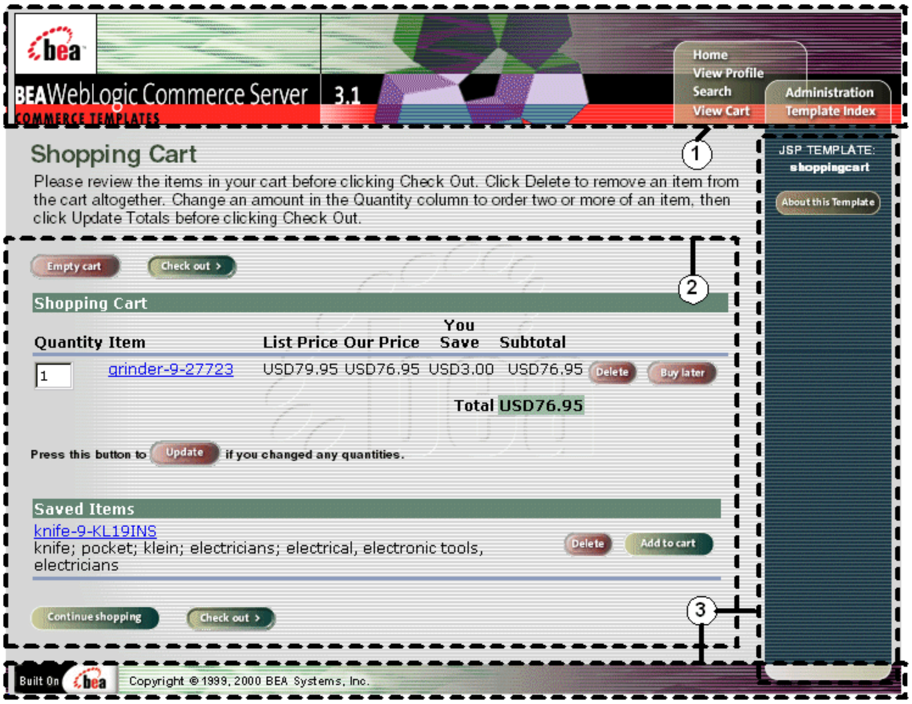
Shopping Cart

Quantity	Item	List Price	Our Price	You Save	Subtotal	
1	grinder-9-27723	USD79.95	USD76.95	USD3.00	USD76.95	Delete
					Total USD76.95	

Press this button to Update if you changed any quantities.

Copyright © 1999, 2000 BEA Systems, Inc.

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



Shopping Cart

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

Shopping Cart

Quantity	Item	List Price	Our Price	You Save	Subtotal
1	grinder-9-27723	USD79.95	USD76.95	USD3.00	USD76.95

Total USD76.95

Press this button to Update if you changed any quantities.

Saved Items

knife-9-KL19INS
knife; pocket; klein; electricians; electrical, electronic tools, electricians

Copyright © 1999, 2000 BEA Systems, Inc.

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 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. 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 *BEA WebLogic Personalization Server* documentation. For more information about the Pipeline JSP tags, see *BEA 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 login at the `login.jsp` template (prior to loading the `shipping.jsp` template). For more information about the `login.jsp` template, see *BEA 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.

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`
`button(checkout)`	`InitShippingMethodListIP`
`button(deleteItemFromShoppingCart)`	`DeleteProductItemFromShoppingCartIP`
`button(deleteItemFromSavedList)`	`UpdateSkuIP` `DeleteProductItemFromSavedList`
`button(emptyShoppingCart)`	`EmptyShoppingCartIP`
`button(moveItemToSavedList)`	`UpdateSkuIP` `MoveProductItemToSavedList`
`button(moveItemToShoppingCart)`	`UpdateSkuIP` `MoveProductItemToShoppingCart`
`button(updateShoppingCartQuantities)`	`UpdateShoppingCartQuantitiesIP`

Table 3-2 briefly describes each of the Pipelines from Table 3-1, as they are defined in the `pipeline.properties` file. For more information about individual Pipeline components, see "Pipeline Components" on page 3-18.

Table 3-2 Shopping Cart Pipelines

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

Table 3-3 shoppingcart.jsp Pipeline Session Attributes

Attribute	Type	Description
`PipelineSessionConstants .SAVED_SHOPPING_CART`	`com.beasys.commerce.ebusiness .shoppingcart.ShoppingCart`	The saved shopping cart (source of the saved items).
`PipelineSessionConstants .SHOPPING_CART`	`com.beasys.commerce.ebusiness .shoppingcart.ShoppingCart`	The currently active 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 *BEA 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
`getShoppingCartLineCollection()`	A collection of the individual lines in the shopping cart (that is, `ShoppingCartLine`).

`getTotal(int totalType)`	The total amount specified by the `totalType` parameter. Valid parameters include:
	
	`ShoppingCartConstants.LINE_UNIT_PRICE_TIMES_QUANTITY`
	`ShoppingCartConstants.LINE_SHIPPING`
	`ShoppingCartConstants.LINE_TAX`
	
	Note: The `getTotal()` method also allows you to combine different total types. For more information, see the *Javadoc*.

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

Table 3-5 ShoppingCartLine Accessor Methods/Attributes

Method/Attribute	Description
`getQuantity()`	The quantity of the item.
`getProductItem()`	The product item in the shopping cart line.
`getUnitPrice()`	The current price for the item at the time it was added to the shopping cart. May be different from MSRP.

Table 3-5 ShoppingCartLine Accessor Methods/Attributes

Method/Attribute	Description

`getLineTotal(int totalType)`	The total amount specified by the `totalType` parameter. Valid parameters include:
	
	`ShoppingCartConstants.LINE_UNIT_PRICE_TIMES_QUANTITY`
	`ShoppingCartConstants.LINE_SHIPPING`
	`ShoppingCartConstants.LINE_TAX`
	
	Note: The `getLineTotal()` method also allows you to combine different total types. For more information, see the *Javadoc*.

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

Note: The `ProductItem` object is described in the *BEA 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 *BEA 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 (`shoppingcart.jsp`), used by the Webflow.
`HttpRequestConstants.CATALOG_ITEM_SKU`	Hidden	SKU of the item that the event is to operate on.

Table 3-6 shoppingcart.jsp Form Fields

Parameter Name	Type	Description
`NewQuantity_<SKU>` where <*SKU*> is replaced with the SKU of the item on the shopping cart line.	Textbox	The new quantity for the item in the shopping cart. It is the only form field on this page that requires input from the customer.

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

Input Processors

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

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

DeleteProductItemFromShoppingCartIP

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

EmptyShoppingCartIP

Class Name	`com.beasys.commerce.ebusiness.shoppingcart.webflow.EmptyShoppingCartIP`
Description	Creates a new shopping cart and stores it in the Pipeline session. The old shopping cart is discarded.
Required `HTTPServletRequest` Parameters	None
Required Pipeline Session Attributes	None
Updated Pipeline Session Attributes	`PipelineSessionConstants.SHOPPING_CART`
Removed Pipeline Session Attributes	None
Validation	None
Exceptions	None

InitShoppingCartIP

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

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

UpdateShoppingCartQuantitiesIP

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

UpdateSkuIP

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

Pipeline Components

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

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

Class Name	`com.beasys.commerce.ebusiness.shoppingcart.pipeline.DeleteProductItemFromSavedListPC`
Description	Removes the item from the saved list and updates the `WLCS_SAVED_ITEM_LIST` table in the database.

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

Updated Pipeline Session Attributes	`PipelineSessionConstants.SAVED_SHOPPING_CART`
Removed Pipeline Session Attributes	None
Type	Session bean
JNDI Name	`com.beasys.commerce.ebusiness.shoppingcart.pipeline.DeleteProductItemFromSavedListPC`

Exceptions	`PipelineFatalException`, thrown if the required Pipeline session attributes are not available.

MoveProductItemToSavedListPC

Class Name	`com.beasys.commerce.ebusiness.shoppingcart.pipeline.MoveProductItemToSavedListPC`
Description	Removes the item from the shopping cart, adds it to the saved list, and then updates the `WLCS_SAVED_ITEM_LIST` table in the database.

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

Updated Pipeline Session Attributes	`PipelineSessionConstants.SAVED_SHOPPING_CART` `PipelineSessionConstants.SHOPPING_CART`
Removed Pipeline Session Attributes	None
Type	Session bean
JNDI Name	`com.beasys.commerce.ebusiness.shoppingcart.pipeline.MoveProductItemToSavedListPC`
Exceptions	`PipelineFatalException`, thrown if the required Pipeline session attributes are not available.

MoveProductItemToShoppingCartPC

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

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

Updated Pipeline Session Attributes	`PipelineSessionConstants.SAVED_SHOPPING_CART` `PipelineSessionConstants.SHOPPING_CART`
Removed Pipeline Session Attributes	None
Type	Session bean
JNDI Name	`com.beasys.commerce.ebusiness.shoppingcart.` `pipeline.MoveProductItemToShoppingCartPC`
Exceptions	`PipelineFatalException`, thrown if the required Pipeline session attributes are not available.

RefreshSavedListPC

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

CHAPTER

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.

shipping.jsp Template

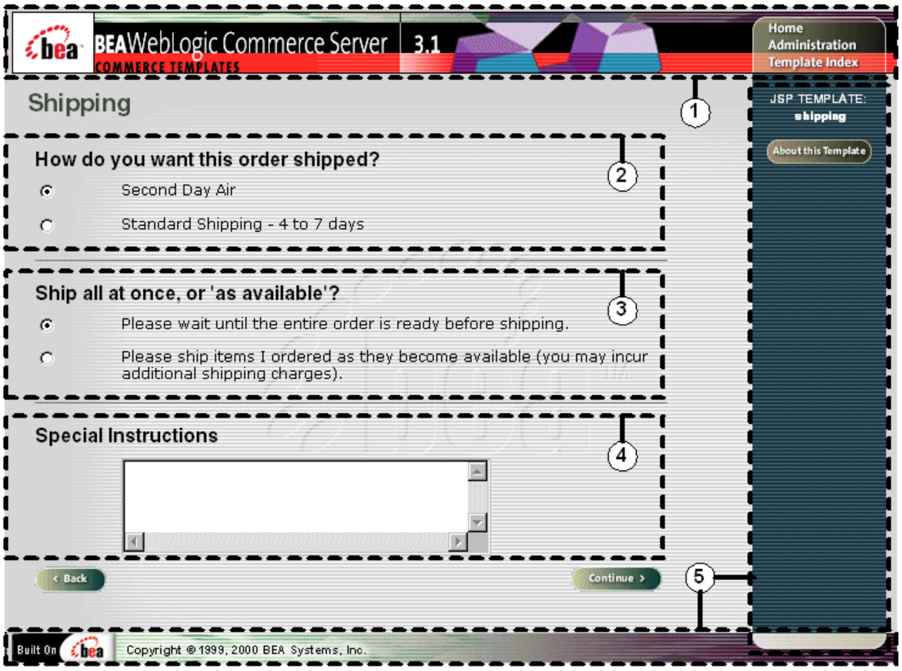
The `shipping.jsp` template (shown in Figure 4-1) allows the customer to select and input shipping details for the order. Shipping details include the shipping method (such as standard, second day air, and so on), shipping preference (all at once or as items become available) and any special shipping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



Shipping

How do you want this order shipped?

[✓ Second Day Air]

Standard Shipping - 4 to 7 days

Ship all at once, or 'as available'?

[✓ Please wait until the entire order is ready before shipping.]

Please ship items I ordered as they become available (you may incur additional shipping charges).

Special Instructions

Copyright © 1999, 2000 BEA Systems, Inc.

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.

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 *BEA WebLogic Personalization Server* documentation. For more information about the Pipeline JSP tags, see *BEA 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 `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 login at the `login.jsp` template prior to loading the `shipping.jsp`. For more information about the `login.jsp` template, see *BEA 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.

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)
`button(back)`	No business logic required. Loads `shoppingcart.jsp`.
`button(continue)`	`ValidateShippingInfoIP`

Dynamic Data Display

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

First, the `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 .SHIPPING_METHOD_LIST`	List of `com.beasys.commerce.ebusiness .shipping.ShippingMethodValue`	The list of available shipping methods.

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

Listing 4-1 Retrieving the Shipping Method Attribute

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

Note: For more information on the `getPipelineProperty` JSP tag, see *BEA 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
`description`	A description of the shipping method.
`identifier`	Key in the database for the shipping method.

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

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

```
<table>

<tr>
  <td colspan=2>
    <b>Select Shipping Method</b>
  </td>
</tr>

<wl:repeat set=”<%=shippingMethodList%>” id=”shippingMethodValue”
type=”ShippingMethodValue” count=”100”

<tr>
  <td>
    <input type=”radio” name=”<%HttpRequestConstants.SHIPPING_METHOD%>”
     value=”<%=shippingMethodValue.identifier%>”>
  </td>
  <td>
    <%=shippingMethodValue.description%>
  </td>
</tr>

</wl:repeat>

</table>
```

Note: For more information on the WebLogic Server JSP tags, see “JSP Tag Reference” in the *BEA 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 (`shipping.jsp`), used by the Webflow.
`HttpRequestConstants.SHIPPING_METHOD`	Radio button	Identifies the shipping method the customer selects.
`HttpRequestConstants.SPECIAL_INSTRUCTIONS`	Textbox	Any special instructions the customer specifies.
`HttpRequestConstants.SPLITTING_PREFERENCE`	Radio button	String representing the splitting preference the customer selects.

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

selectaddress.jsp Template

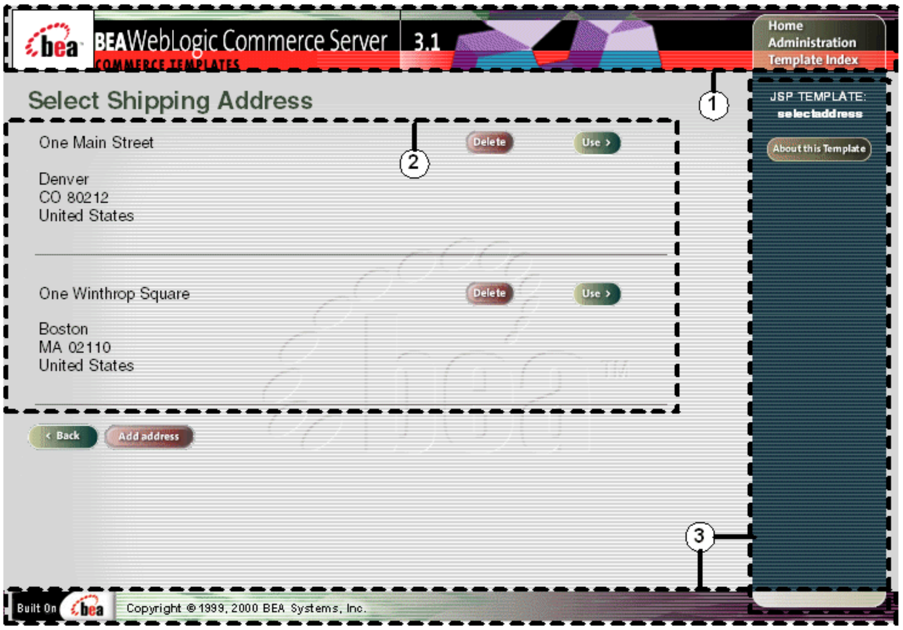
The `selectaddress.jsp` template (shown in Figure 4-2) displays a list of shipping addresses that have previously been associated with the customer. If the customer clicks the Use button associated with a particular address, that address will be used as the shipping address and the customer will continue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

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

Sample Browser View

Figure 4-2 shows an annotated version of the `selectaddress.jsp` template. The 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



One Main Street

Denver
CO 80212
United States

One Winthrop Square

Boston
MA 02110
United States

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.

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 *BEA WebLogic Personalization Server* documentation. For more information about the Pipeline JSP tags, see *BEA 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.

Events

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

Table 4-5 selectaddress.jsp Events

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

`button(useShippingAddress)`	`UpdateShippingAddressIP`
	`TaxVerifyShippingAddress`
	`CalculateShippingCost`
	`TaxCalculateLineLevel`

Table 4-6 briefly describes each of the Pipelines from Table 4-5, 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

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

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

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

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

```
<% Iterator iterator =((Map)shippingAddressMap).keySet().iterator();
while(iterator.hasNext())

{

String addressKey = (String)iterator.next();
Address shippingAddress = (Address)((Map)shippingAddressMap).get(addressKey);

%>

<table width="90%" border="0" cellpadding="6" cellspacing="0">
  <tr>
    <td align="left" valign="top" width="40%" nowrap>
      <p><%=shippingAddress.getStreet1()%><br>
      <% if(shippingAddress.getStreet2().length() != 0) {%>
      <%=shippingAddress.getStreet2()%><br>
      <% } %>
      <%=shippingAddress.getCity()%><br>
      <%=shippingAddress.getState()%> <%=shippingAddress.getPostalCode()%><br>
      <%= shippingAddress.getCountry() %>
    </td>

    <td align="left" valign="top" width="5%" >
      <div class="commentary">
      <a href="<%=WebflowJSPHelper.createWebflowURL(pageContext,
       "selectaddress.jsp", "button(deleteShippingAddress)","&" +
       HttpRequestConstants.ADDRESS_KEY + "=" + addressKey, true)%>">
       <img src="<%=com.beasys.commerce.webflow.WebflowJSPHelper.createGIFURL
       (request, response,"/commerce/images/btn_delete.gif")%>" 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)%>">
       <img src="<%=com.beasys.commerce.webflow.WebflowJSPHelper.createGIFURL
       (request, response,"/commerce/images/btn_use.gif")%>" border="0">
      </a>
      </div>
    </td>
  </tr>

  <tr>
    <td colspan="3">
```

```
      <hr size="1">
    </td>
  </tr>
</table>

<%

}

%>
```

Form Field Specification

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

addaddress.jsp Template

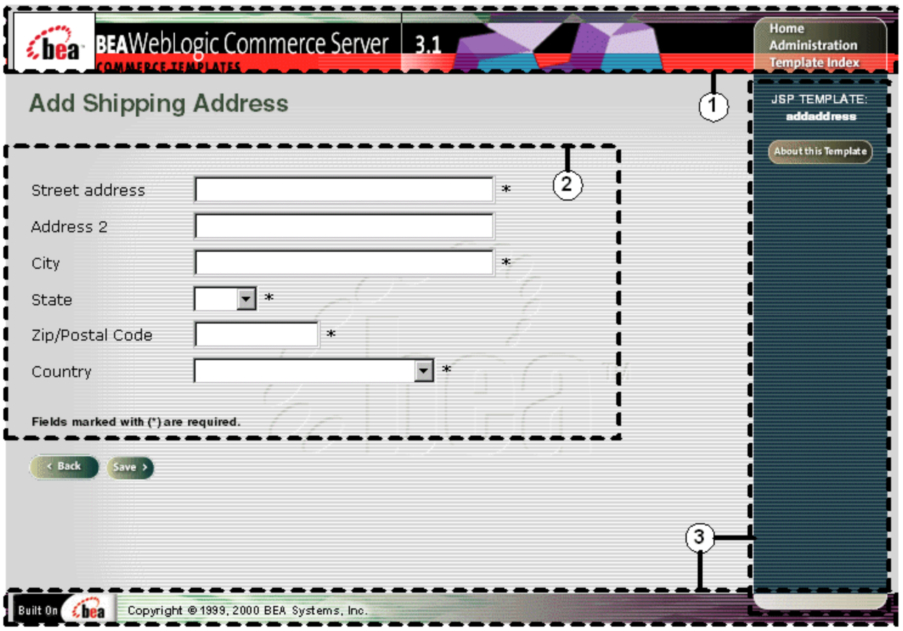
The `addaddress.jsp` template (shown in Figure 4-3) collects information about a new shipping address from the customer. This information includes two lines of a street address (one required), a city, a state, a zip code, and a country (all required).

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

Sample Browser View

Figure 4-3 shows an annotated version of the `addaddress.jsp` template. The 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



Add Shipping Address

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

Fields marked with (*) are required.

Copyright © 1999, 2000 BEA Systems, Inc.

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 *BEA 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)
`button(back)`	No business logic required. Loads `selectaddress.jsp`.
`button(addNewShippingAddress)`	`ValidateAddressIP` `AddShippingAddress`

Table 4-9 briefly describes each of the Pipelines from Table 4-8, 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 *BEA 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 (`addaddress.jsp`), used by the Webflow.
`HttpRequestConstants.CUSTOMER_SHIPPING_ADDRESS1`	Textbox	The first line of the shipping street address.

Table 4-10 addaddress.jsp Form Fields

Parameter Name	Type	Description
`HttpRequestConstants. CUSTOMER_SHIPPING_ADDRESS2`	Textbox	The second line of the shipping street address.
`HttpRequestConstants. CUSTOMER_SHIPPING_CITY`	Textbox	The city in the shipping address.
`HttpRequestConstants. CUSTOMER_SHIPPING_STATE`	Textbox	The state in the shipping address.
`HttpRequestConstants. CUSTOMER_SHIPPING_ZIPCODE`	Textbox	The zip/postal code in the shipping address.
`HttpRequestConstants. CUSTOMER_SHIPPING_COUNTRY`	Textbox	The country in the shipping address.

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

Input Processors

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

InitShippingMethodListIP

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

UpdateShippingAddressIP

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

ValidateAddressIP

Class Name	`com.beasys.commerce.ebusiness.shipping.webflow.ValidateAddressIP`
Description	Validates the address and places it in the Pipeline session.

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

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

ValidateShippingInfoIP

Class Name	`com.beasys.commerce.ebusiness.shipping.webflow.ValidateShippingInfoIP`
Description	Places the shipping method, splitting preference, and special instructions into the Pipeline session.

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

Required Pipeline Session Attributes	None

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

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

Pipeline Components

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

Notes: For information about the `TaxVerifyShippingAddressPC` and `TaxCalculateLineLevelPC` Pipeline components, see "Taxation Services" on page 5-1.

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

AddShippingAddressPC

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

CalculateShippingPC

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

DeleteShippingAddressPC

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

CHAPTER

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.

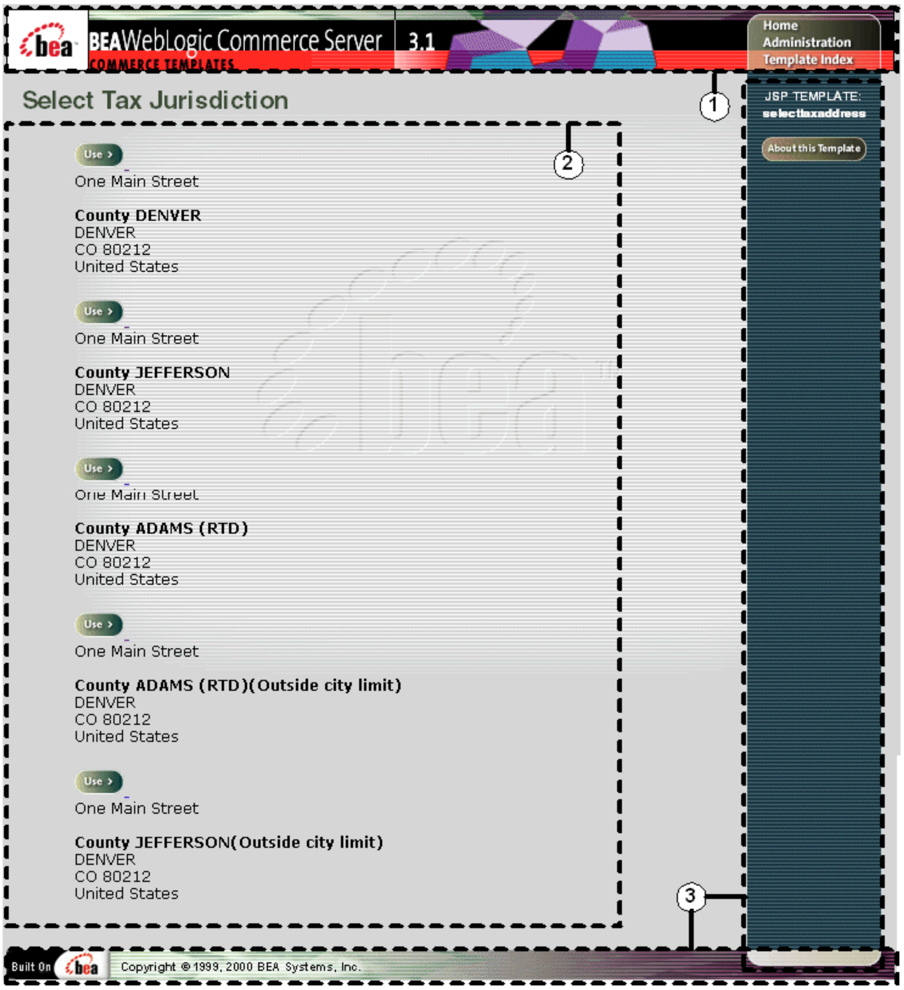
selecttaxaddress.jsp Template

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



Select Tax Jurisdiction

Use ›
One Main Street

County DENVER
DENVER
CO 80212
United States

Use ›
One Main Street

County JEFFERSON
DENVER
CO 80212
United States

Use ›
One Main Street

County ADAMS (RTD)
DENVER
CO 80212
United States

Use ›
One Main Street

County ADAMS (RTD)(Outside city limit)
DENVER
CO 80212
United States

Use ›
One Main Street

County JEFFERSON(Outside city limit)
DENVER
CO 80212
United States

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 `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 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 *BEA WebLogic Personalization Server* documentation. For more information about the Pipeline JSP tags, see *BEA 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

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)
`button(use)`	`UpdateTaxShippingAddressIP`

Dynamic Data Display

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

First, the `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
`PipelineSessionConstants.` `VERIZIP_SHIPPING_ADDRESSES`	List of `com.beasys.commerce.axiom` `.contact.Address`	List of the possibilities for the more detailed shipping address.

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

Listing 5-1 Retrieving the Address Selection Attribute

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

Note: For more information on the `getPipelineProperty` JSP tag, see *BEA 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
`getStreet1()`	The first line of the street in the shipping address.
`getStreet2()`	The second line of the street in the shipping address.
`getCity()`	The city in the shipping address.
`getCounty()`	The county in the shipping address.
`getState()`	The state in the shipping address.
`getPostalCode()`	The zip/postal code in the shipping address.
`getCountry()`	The country in the shipping address.

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

Listing 5-2 Using <wl> 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 *BEA 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 selectataxddress.jsp Form Fields

Parameter Name	Type	Description
`"event"`	Hidden	Indicates which event has been triggered. It is used by the Webflow to determine what happens next.
`"origin"`	Hidden	The name of the current page (`selecttaxaddress.jsp`), used by the Webflow.

Table 5-4 selectataxddress.jsp Form Fields

Parameter Name	Type	Description
`PipelineSessionConstants.` `TAX_SHIPPING_ADDRESS`	Hidden	Identifies the more specific address selected by the customer.

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

Input Processors

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

DecideShippingAddressPageIP

Class Name	`com.beasys.commerce.ebusiness.tax.webflow.` `DecideShippingAddressPageIP`
Description	Makes the decision about whether to display `selecttaxaddress.jsp` 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 `payment.jsp`. Otherwise, this input processor redirects the Webflow to `selecttaxaddress.jsp`.
Required `HTTPServletRequest` Parameters	None
Required Pipeline Session Attributes	`PipelineSessionConstants.SHIPPING_ADDRESS` `PipelineSessionConstants.VERIZIP_SHIPPING_ADDRESSES`
Updated Pipeline Session Attributes	`PipelineSessionConstants.SHIPPING_ADDRESS` (in the case of a single address)
Removed Pipeline Session Attributes	None
Validation	None
Exceptions	`MultipleAddressFoundException`, thrown if the VERAZIP service returns more than one address.

UpdateShippingAddressIP

Class Name	`com.beasys.commerce.ebusiness.shipping.webflow.UpdateShippingAddressIP`
Description	Updates the shipping address attribute in the Pipeline session based on the tax address the customer selects.
Required `HTTPServletRequest` Parameters	`HTTPRequestConstants.TAX_SHIPPING_ADDRESS`
Required Pipeline Session Attributes	`PipelineSessionConstants.SHIPPING_ADDRESS` `PipelineSessionConstants.VERIZIP_SHIPPING_ADDRESSES`
Updated Pipeline Session Attributes	`PipelineSessionConstants.SHIPPING_ADDRESS`
Removed Pipeline Session Attributes	None
Validation	None
Exceptions	None

Pipeline Components

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

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

TaxCalculateLineLevelPC

Class Name	`com.beasys.commerce.ebusiness.tax.pipeline.` `TaxCalculateLineLevelPC`
Description	Calculates the tax and provides line-level information about the taxability of an item. This Pipeline component is used to display the tax information to the customer.
Required Pipeline Session Attributes	`PipelineSessionConstants.SHOPPING_CART` `PipelineSessionConstants.SHIPPING_ADDRESS`
Updated Pipeline Session Attributes	`PipelineSessionConstants.SHOPPING_CART`
Removed Pipeline Session Attributes	None
Type	Java class
JNDI Name	None
Exceptions	None

TaxCalculateAndCommitLineLevelPC

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

TaxVerifyShippingAddressPC

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

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

Integration with TAXWARE

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

- The SALES/USE Tax System is a TAXWARE product that 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 that 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).

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 BEA 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 BEA 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 does become 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 vs. International Taxes*: The TAXWARE products included in the BEA WebLogic Commerce Server product handle tax calculations for the United States and Canada only.
- *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, and Universal Tax Link (UTL) systems are shipped within the BEA Weblogic Commerce Server product to provide out-of-the-box TAXWARE functionality. The Commerce Server's installation program will install these TAXWARE products along with the Commerce Server, and will also uninstall them upon uninstallation of the Commerce Server.

The versions of the TAXWARE products installed with the BEA 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

Installation Directory Structure

The TAXWARE product files installed with the BEA 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, Java classes, DLLs, and preloaded data files needed for Win32 installation reside in subdirectories beneath `WL_COMMERCE_HOME\eval\win32\Taxware`, where `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

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.`
`\classes`	Contains Java classes for UTL, including `taxmain.class` and `taxcommon.class.`
`\data`	Contains preloaded data files for SALES/USE and VERAZIP 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 two `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.

Notes: The BEA WebLogic Commerce Server product's installation program automatically copies 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 5-26.

UNIX

All TAXWARE audit files, Java classes, shared objects, and preloaded data files needed for UNIX installation reside in subdirectories beneath `WL_COMMERCE_HOME\eval\solaris2\Taxware`, where `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

Subdirectory	Description
`\audit`	Contains audit files for all tax transactions.
`\classes`	Contains UTL Java classes, including `taxmain.class` and `taxcommon.class.`
`\data`	Contains preloaded data files for SALES/USE and VERAZIP 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 `libsalesusetax.so`, `libstep.so`, `libtaxcommon.so`, `libtaxcommono.so`, and `libverazip.so.`
`\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\bin` directory, where `WL_COMMERCE_HOME` is the directory in which you installed WebLogic Commerce Server.
3. To test the SALES/USE component of TAXWARE, type `runsample.bat commonsu.in`.
4. To test the VERAZIP component of TAXWARE, type `runsample.bat vzip.in`. The result should be a long line that begins with: `0000010000`.
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\bin` 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 BEA 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 Commerce Server ships with a working version of TAXWARE, complete with the correct DLLs and sample data files. If your organization already makes use of TAXWARE products and has installed these files in a different location, you may want to point the BEA WebLogic Commerce Server product's Taxation Services to a different directory structure. For more information about changing the TAXWARE directory structure, see "Run-Time Configuration" on page 5-26.

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 BEA WebLogic Commerce Server product via our programmatic interface (API). Finally, 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 will need to take into consideration when using TAXWARE products. The information described focuses on the configuration properties in the `weblogiccommerce.properties` file that enable tax calculations.

Addresses and Taxation

In many cases, the proper calculation of taxes requires that you specify a number of addresses, including the location from which 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.

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

For each of the relevant address fields (street, city, state, and so on), there is a separate line in the properties file (see Listing 5-3). The minimum information that you are required to specify is the city, state, zip, and country. If the TAXWARE products determine that this information is enough to identify a unique tax jurisdiction, then it is possible to default the county code and GeoCode by commenting out these properties in the `weblogiccommerce.properties` file. In some cases, however, it may be necessary to provide a specific county and GeoCode. This is something that you will need to confirm when installing the additional TAXWARE components.

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
#

shipfrom.countycode=000
shipfrom.state=MA
shipfrom.city=SALEM
shipfrom.zip=01970
shipfrom.geocode=00
shipfrom.country=USA

################################################################
```

```
# Order Acceptance Address
#-----------------------------------------------------------------

# OrderAcceptance is the address where orders are accepted
# Please review Taxware documentation when setting these properties
#

orderacceptance.countycode=000
orderacceptance.state=MA
orderacceptance.city=SALEM
orderacceptance.zip=01970
orderacceptance.geocode=00
orderacceptance.country=USA

#################################################################

# Order Origin Address
#----------------------------------------------------------------

# Order Origin is the address where orders are Originated
# Please review Taxware documentation when setting these properties
#

orderorigin.countycode=000
orderorigin.state=MA
orderorigin.city=SALEM
orderorigin.zip=01970
orderorigin.geocode=00
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
```

```
#titlepassage=shipto
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.

Specifying a Currency

It is important that the ISO 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 a single 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 only supports USD)
# ------------------------------------------------------------------

tax.currency = USD
```

Specifying Your Company's ID

When you configure TAXWARE, you will also need to provide some indentification information for your company to calculate taxes. 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 Specifying Company ID in the weblogiccommerce.properties File

```
#################################################################

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

companyId=CompanyId
```

Specifying Your Tax Type

Depending on the nature of your business, you will need to select the type of taxes you want to calculate. The BEA WebLogic Commerce Server product defaults to calculating sales tax for hard and soft goods. TAXWARE 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 5-7.

Listing 5-7 Specifying TaxType in the weblogiccommerce.properties File

```
#################################################################

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

#taxtype=use
#taxtype=rental
#taxtype=consumeruse
```

```
#taxtype=services
taxtype=sales
```

Note: The tax calculation Pipeline components that ship with 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 Tax Type parameters.

Specifying Calculation of Jurisdiction

Setting the `TaxSelParm` property (shown in Listing 5-8) 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`, `orderacceptance`, and `orderorigin` address properties from the `weblogiccommerce.properties` file, as they will not be required (see Listing 5-3).

Listing 5-8 Specifying Jurisdiction Calculations in the weblogiccommerce.properties File

```
################################################################

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

#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 5-7.

Table 5-7 Differences in WebLogic Commerce Server and TAXWARE Directory Structures

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

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

- In the `set-environment.bat` file, change the `WLCS_CLASSPATH` environmental variable to the directory where the TAXWARE Java Class files reside.
- In the `StartCommerce.bat` file, change the `PATH` environment variable in `StartCommerce.bat` to the directory where the TAXWARE DLL files reside.
- In the `avptax.ini`, `avpzip.ini`, and `taxware.ini` files, change the location of the address verification data and tax tables. These files are located in the `winnt` directory. For an example, see Listing 5-11.

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 5-9.

Listing 5-9 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 5-10.
2. Set the TAXWARE-specific environment variables to the correct data directories. For an example, see Listing 5-11.
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 theses changes to take effect, you need to restart your server.

Listing 5-10 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 5-11 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 STEPOUT

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

BT_SHARE=N
export BT_SHARE
```

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

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

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 BEA WebLogic Commerce Server product.
- Determine the `ShipFrom` address.

- Determine the `OrderAcceptance` 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 5-26) are set properly.
- Establish a process by which tax data is periodically updated.
- Establish a process by which tax audit files are archived.

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 5-12.

Listing 5-12 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 5-13.

Listing 5-13 CommitOrder Pipeline Without Tax Pipeline Component

```
# CommitOrder

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

4. 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.

2. Locate and remove (or comment out) all lines in the `webflow.properties` file that reference the following Pipeline components:
   - `TaxVerifyShippingAddressPC`
   - `TaxCalculateLineLevelPC`
   - `TaxCalculateAndCommitLineLevelPC`

3. Locate the Select Shipping Address Page section of the `webflow.properties` file, as shown in Listing 5-14. Notice that in the default configuration, the `TaxVerifyShippingAddress` Pipeline is invoked upon successful execution of the `UpdateShippingAddressIP` input processor.

Listing 5-14 Default Shipping Address Page in the webflow.properties File

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

SelectShippingAddress_UpdateShippingAddress.inputprocessor.
success=TaxVerifyShippingAddress.pipeline

...
```

4. Replace the `TaxVerifyShippingAddress.pipeline` with `CalculateShippingCost.pipeline`, so the Select Shipping Address Page section in the `webflow.properties` file is as shown in Listing 5-15.

Listing 5-15 Shipping Address Page Without Tax Pipeline

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

SelectShippingAddress_UpdateShippingAddress.inputprocessor.
success=CalculateShippingCost.pipeline

...
```

5. Locate the success path for the `CalculateShippingCost` Pipeline in the `webflow.properties` file, as shown in Listing 5-16.

Listing 5-16 Default Success Path for CalculateShippingCost Pipeline

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

...
...

CalculateShippingCost.pipeline.success=TaxCalculateLineLevel.
pipeline

...
```

6. Replace the `TaxCalculateLineLevel` Pipeline with `commerce/order/payment.jsp`, so the success path for the `CalculateShippingCost` Pipeline is as shown in Listing 5-17.

Listing 5-17 Success Path for CalculateShippingCost Pipeline without Tax Pipeline

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

...
...

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

...
```

7. 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`.
   - The Pipeline components `TaxVerifyShippingAddressPC`, `TaxCalculateLineLevelPC`, and `TaxCalculateAndCommitLineLevelPC`.

8. 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 BEA 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 5-30), 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 calcuations 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*.

CHAPTER

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 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.

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.

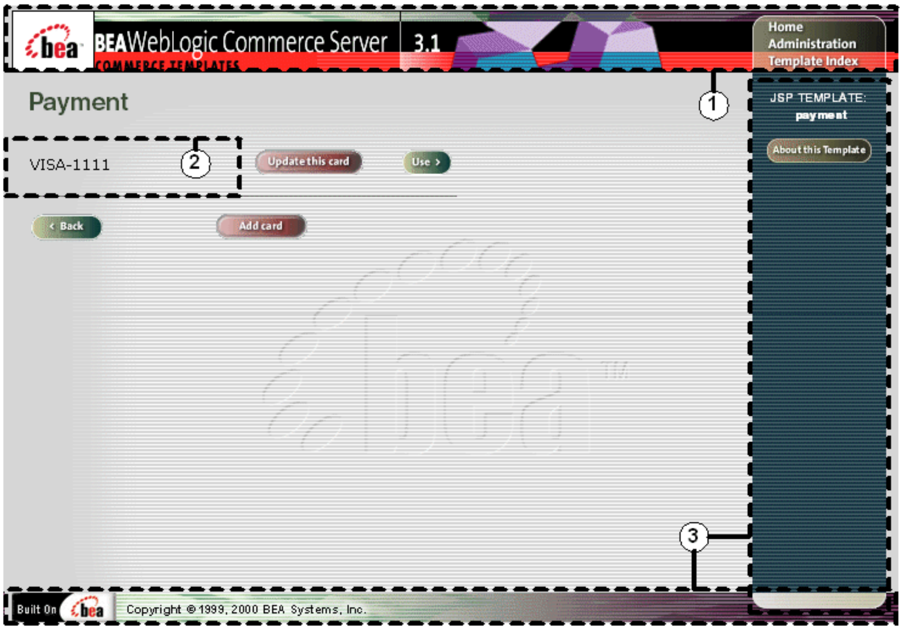
Note: For more information about user profiles, see "Customer Profile Services" in the *BEA WebLogic Commerce Server Registration and User Processing Package* documentation.

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 *BEA 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)
`button(addNewCreditCard)`	No business logic required. Loads `paymentnewcc.jsp`.
`button(continue)`	`AuthorizePaymentIP`
`button(updatePaymentInfo)`	No business logic required. Loads `paymenteditcc.jsp`.

Dynamic Data Display

The purpose of the `payment.jsp` template is to display a list of the customer’s previously saved credit cards. This is accomplished on the `payment.jsp` template using a combination of WebLogic Server and WebLogic 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)credtCardsMapObject).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 *BEA WebLogic Personalization Server* documentation.

Form Field Specification

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

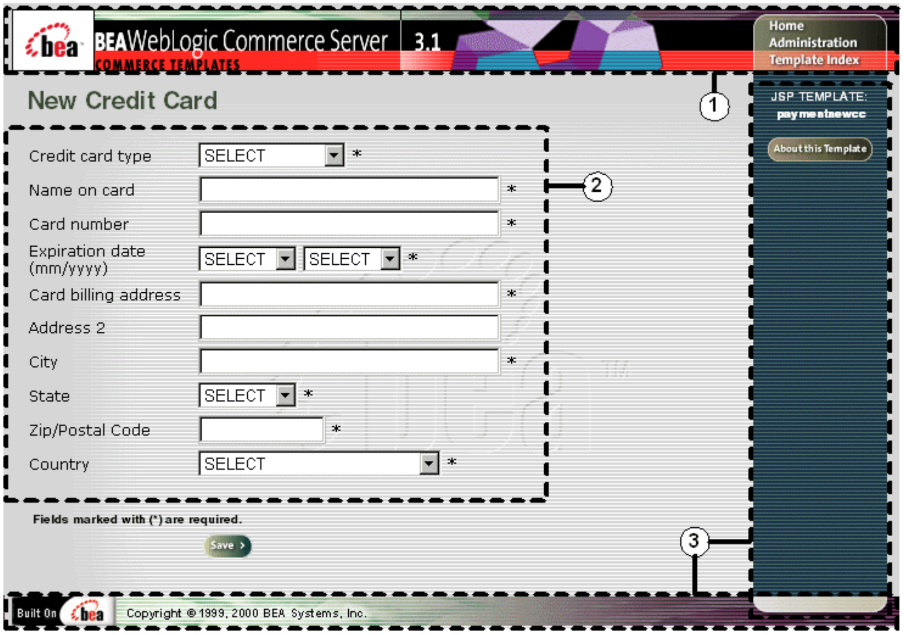
paymentnewcc.jsp Template

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



New Credit Card

Credit card type	SELECT *
Name on card	*
Card number	*
Expiration date (mm/yyyy)	SELECT SELECT *
Card billing address	*
Address 2	
City	*
State	SELECT *
Zip/Postal Code	*
Country	SELECT *

Fields marked with (*) are required.

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 `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 *BEA 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 *BEA WebLogic Commerce Server Registation 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)
`button(save)`	`UpdatePaymentInfoIP`

Dynamic Data Display

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

Form Field Specification

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

Table 6-3 paymentnewcc.jsp Form Fields

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

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

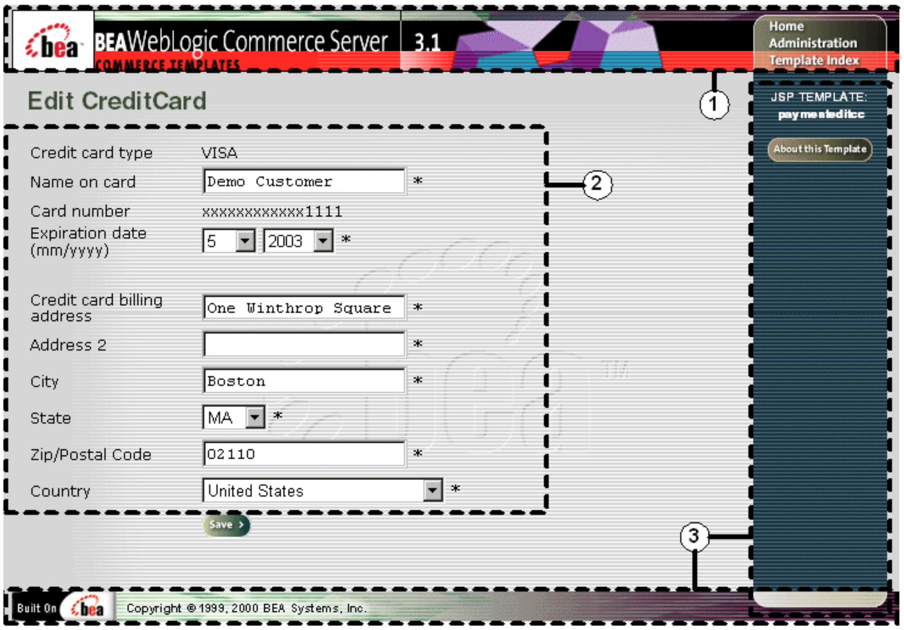
paymenteditcc.jsp Template

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



Edit CreditCard

Credit card type	VISA
Name on card	Demo Customer *
Card number	xxxxxxxxxxxx1111
Expiration date (mm/yyyy)	5 2003 *
Credit card billing address	One Winthrop Square *
Address 2	*
City	Boston *
State	MA *
Zip/Postal Code	02110 *
Country	United States *

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 `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:

```
<%@ 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 *BEA WebLogic Commerce Server Webflow and Pipeline Management*. For more information on the WebLogic Personalization Server JSP tags, see "JSP Tag Reference" in the *BEA 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 *BEA WebLogic Commerce Server Registation 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)
`button(save)`	`UpdatePaymentInfoIP`

Dynamic Data Display

One purpose of the `paymenteditcc.jsp` template is to prepare the credit card information a customer had previously entered, so the `editcctemplate.jsp` template can display this information in the payment information form fields. This is accomplished on the `paymenteditcc.jsp` template using a combination 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 *BEA 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

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

Table 6-6 billingAddress Accessor Methods/Attributes

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

Form Field Specification

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

Table 6-7 paymenteditcc.jsp Form Fields

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

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

Input Processors

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

PaymentAuthorizationIP

Class Name	`com.beasys.commerce.ebusiness.payment.webflow.PaymentAuthorizationIP`
Description	Retrieves the shopping cart from the Pipeline session, the `CreditCardMapKey` from the request, and determines the total price of the order associated with the shopping cart. Adds the amount and credit card associated with the key to the Pipeline session.
Required `HTTPServletRequest` Parameters	`HttpRequestConstants.CREDITCARD_KEY`
Required Pipeline Session Attributes	`PipelineSessionConstants.SHOPPING_CART`
Updated Pipeline Session Attributes	`PipelineSessionConstants.PAYMENT_CREDIT_CARD` `PipelineSessionConstants.PAYMENT_AUTHORIZATION_AMOUNT`
Removed Pipeline Session Attributes	None
Validation	Verifies that the credit card key is valid and that it references an existing credit card.
Exceptions	`ProcessingException`, thrown for invalid types of `CREDITCARD_KEY`, `PAYMENT_CREDIT_CARD`, or `SHOPPING_CART`. Also thrown if these attributes are not available.

UpdatePaymentInfoIP

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

Pipeline Components

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

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

PaymentAuthorizationHostPC

Class Name	`com.beasys.commerce.ebusiness.payment.pipeline.PaymentAuthorizationHostPC`

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

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

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

Exceptions	`AuthorizationFailureException`, thrown when the credit card being used for authorization is invalid (that is, the number or other associated information is incorrect). `AuthorizationRejectedException`, thrown when the credit card used for authorization is valid but cannot be authorized (overdrawn, expired, and so on). `PipelineNonFatalException`, thrown when the external payment service is unavailable. The transaction is recorded for retry. `PipelineFatalException`, thrown when there is a configuration error, a general service error, or a system-level exception from a back-end component.

PaymentAuthorizationTerminalPC

Class Name	`com.beasys.commerce.ebusiness.payment.pipeline.` `PaymentAuthorizationTerminalPC`

Description	Authorizes a given credit card for a specified amount. Used for terminal-based payment models, shown in the `weblogiccommerce.properties` file as:
	`AUTO_MARK_AUTO_SETTLE`
	`AUTO_MARK_AUTO_SETTLE_AVS`
	`AUTO_MARK_MANUAL_SETTLE`
	`AUTO_MARK_MANUAL_SETTLE_AVS`
	`MANUAL_MARK_AUTO_SETTLE`
	`MANUAL_MARK_AUTO_SETTLE_AVS`
	`MANUAL_MARK_MANUAL_SETTLE`
	`MANUAL_MARK_MANUAL_SETTLE_AVS`

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

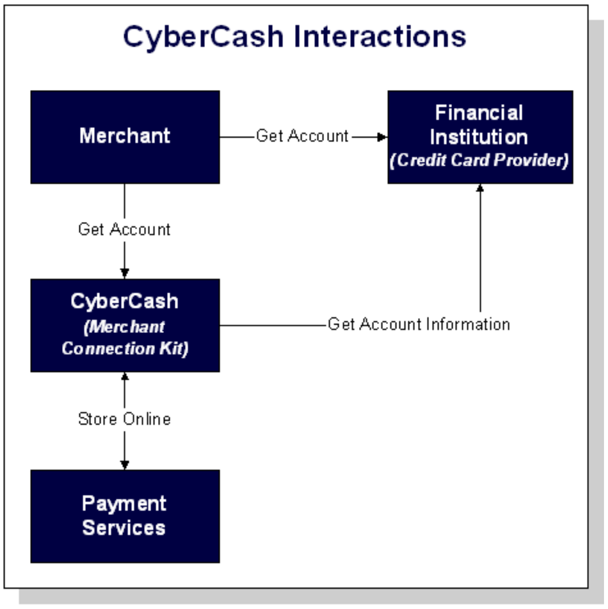
Updated Pipeline Session Attributes	None
Removed Pipeline Session Attributes	None
Type	Java object
JNDI Name	None

Exceptions	`AuthorizationFailureException`, thrown when the credit card being used for authorization is invalid (that is, the number or other associated information is incorrect).
	`AuthorizationRejectedException`, thrown when the credit card used for authorization is valid but cannot be authorized (overdrawn, expired, and so on).
	`PipelineNonFatalException`, thrown when the external payment service is unavailable. The transaction is recorded for retry.
	`PipelineFatalException`, thrown when there is a configuration error, a general service error, or a system-level exception from a back-end component.

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
```

```
#
# 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. MERCHANGS MUST ACQUIRE A
# CYBERCASH CONFIGURATION FILE FROM CYBERCASH. These will be furnished by
# CyberCash as part of the merchant agreement. Once a merchang 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 8, "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 transfered to the merchant's bank account.

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

Table 6-8 Terminal-based Payment Models

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

Note: Each of the terminal-based payment models may be suffixed by `_AVS`. This suffix indicates that merchants are also required to send an address. The BEA 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 BEA 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.
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 BEA 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, perform 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.*;

/**
```

```
* 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
// 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 8, “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 BEA 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 BEA 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
```

```
# 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

Length (Bits)	Data Length (Bytes)
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.

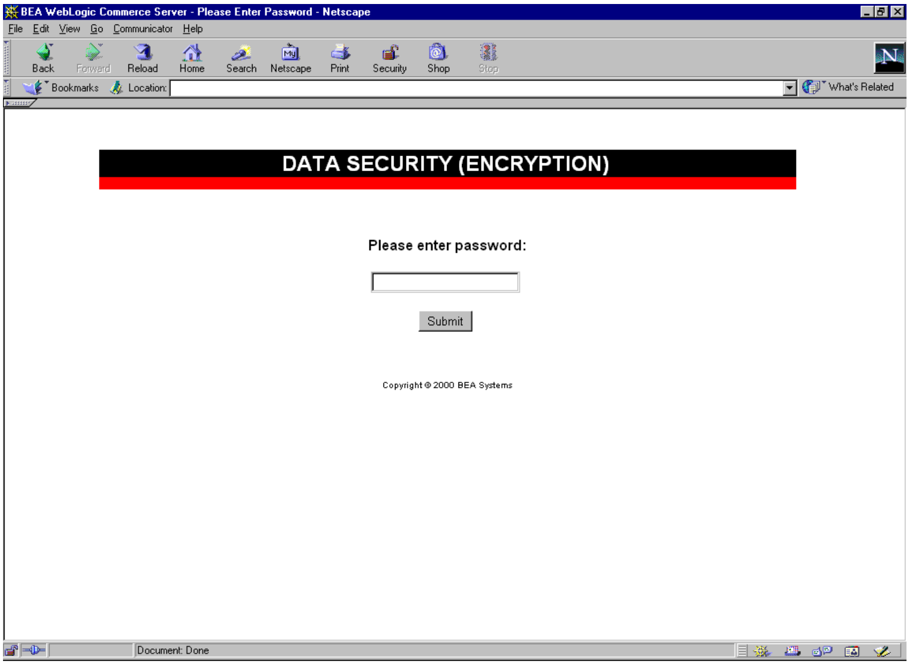
Specifying the Password After Server Startup Using a Secure Web Form

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


BEA WebLogic Commerce Server - Please Enter Password - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location: What's Related
Submit
Document: Done

DATA SECURITY (ENCRYPTION)

Please enter password:

Copyright © 2000 BEA Systems

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 8, “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.

CHAPTER

7 Order Summary and Confirmation Services

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

This topic includes the following sections:

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

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.

checkout.jsp Template

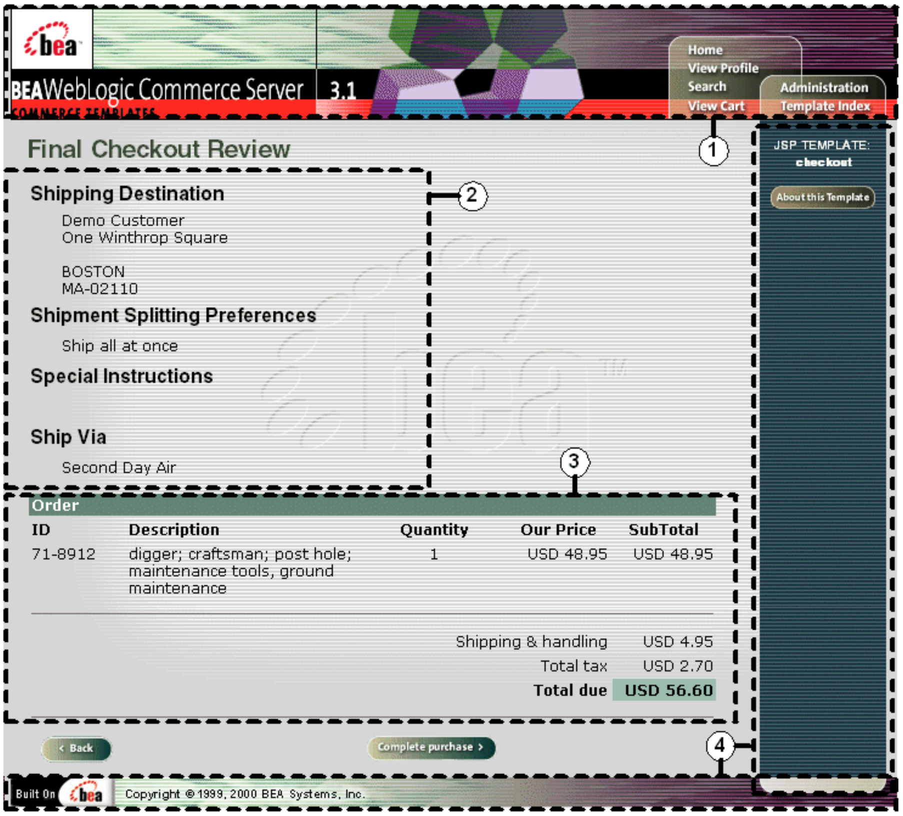
The `checkout.jsp` template (shown in Figure 7-1) provides a customer with a final look at all the details of their order, before the customer commits or cancels the order. Information displayed includes the shipping address, shipping details, a list of the items ordered (including the item name, short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



Final Checkout Review

Shipping Destination

Demo Customer
One Winthrop Square

BOSTON
MA-02110

Shipment Splitting Preferences

Ship all at once

Special Instructions

Ship Via

Second Day Air

Order

ID	Description	Quantity	Our Price	SubTotal
71-8912	digger; craftsman; post hole; maintenance tools, ground maintenance	1	USD 48.95	USD 48.95
			Shipping & handling	USD 4.95
			Total tax	USD 2.70
			Total due	**USD 56.60**

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 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.

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 *BEA WebLogic Personalization Server* documentation. For more information about the Pipeline JSP tags, see *BEA 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)
`button(back)`	No business logic required. Loads `payment.jsp`.
`button(purchase)`	`CommitOrder`

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
`CommitOrder`	Contains `CommitOrderPC`, `AuthorizePaymentPC`, `CalculateTaxLineLevelCommitPC`, `ResetCheckoutPC`, 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 *BEA 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.SHIPPING_ADDRESS`	`com.beasys.commerce.axiom.contact.Address`	The address the order is being shipped to.
`PipelineSessionConstants.SHIPPING_METHOD`	`com.beasys.commerce.ebusiness.shipping.shippingMethodValue`	Identifies the shipping method the customer selected.
`PipelineSessionConstants.SHOPPING_CART`	`com.beasys.commerce.ebusiness.shoppingcart.ShoppingCart`	The shopping cart that was ordered.
`PipelineSessionConstants.SPLITTING_PREFERENCE`	`java.lang.String`	The splitting preference the customer selected.

Table 7-3 checkout.jsp Pipeline Session Attributes

Attributes	Type	Description
`PipelineSessionConstants.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 *BEA 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 *BEA 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
`getStreet1()`	The first line in the customer's street address.
`getStreet2()`	The second line in the customer's street address.
`getCity()`	The city in the customer's address.
`getCounty()`	The county in the customer's address.
`getState()`	The state in the customer's address.
`getPostalCode()`	The zip/postal code in the customer's address.
`getCountry()`	The country in the customer's address.

Table 7-5 ShoppingCart Accessor Methods/Attributes

Method/Attribute	Description
`getShoppingCartLineCollection()`	The individual lines in the shopping cart (i.e. `ShoppingCartLine`).

Table 7-5 ShoppingCart Accessor Methods/Attributes

Method/Attribute	Description

`getTotal(int totalType)`	The total amount specified by the `totalType` parameter. Valid parameters include:
	
	`ShoppingCartConstants.LINE_UNIT_PRICE_TIMES_QUANTITY`
	`ShoppingCartConstants.LINE_SHIPPING`
	`ShoppingCartConstants.LINE_TAX`
	
	Note: The `getTotal()` method also allows you to combine different total types. For more information, see the *Javadoc*.

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

Table 7-6 ShoppingCartLine Accessor Methods/Attributes

Method/Attribute	Description
`getQuantity()`	The quantity of the item.
`getProductItem()`	The product item in the shopping cart line.
`getUnitPrice()`	The current price for the item at the time it was added to the shopping cart. May be different from MSRP.

`getLineTotal(int totalType)`	The total amount specified by the `totalType` parameter. Valid parameters include:
	
	`ShoppingCartConstants.LINE_UNIT_PRICE_TIMES_QUANTITY`
	`ShoppingCartConstants.LINE_SHIPPING`
	`ShoppingCartConstants.LINE_TAX`
	
	Note: The `getLineTotal()` method also allows you to combine different total types. For more information, see the *Javadoc*.

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>
```

```
</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.

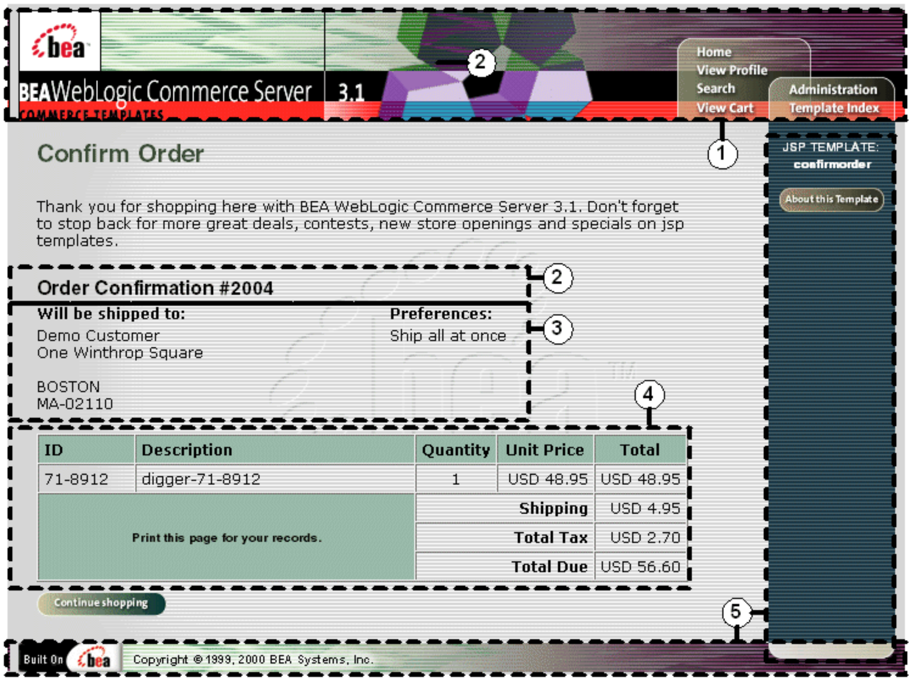
confirmorder.jsp Template

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:	Preferences:
Demo Customer One Winthrop Square BOSTON MA-02110	Ship all at once

ID	Description	Quantity	Unit Price	Total
71-8912	digger-71-8912	1	USD 48.95	USD 48.95
Print this page for your records.			Shipping	USD 4.95
			Total Tax	USD 2.70
			Total Due	USD 56.60

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 *BEA WebLogic Personalization Server* documentation. For more information about the Pipeline JSP tags, see *BEA 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.

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 *BEA 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
`PipelineSessionConstants.ORDER_VALUE`	List of `com.beasys.commerce.ebusiness.order.OrderValue`	List of the orders available for the customer.
`PipelineSessionConstants.SHIPPING_METHOD`	`com.beasys.commerce.ebusiness.shipping.ShippingMethodValue`	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 *BEA 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 *BEA 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
`getStreet1()`	The first line in the customer's street address.
`getStreet2()`	The second line in the customer's street address.
`getCity()`	The city in the customer's address.
`getCounty()`	The county in the customer's address.
`getState()`	The state in the customer's address.

Table 7-8 Address Accessor Methods/Attributes

Method/Attribute	Description
`getPostalCode()`	The zip/postal code in the customer's address.
`getCountry()`	The country in the customer's address.

Table 7-9 ShippingMethodValue Accessor Methods/Attributes

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

Table 7-10 OrderValue Accessor Methods/Attributes

Method/Attribute	Description
`createdDate`	The date the customer's order was created.
`identifier`	Key in the database for the order.

`getTotal(int totalType)`	The total amount specified by the `totalType` parameter. Valid parameters include:
	
	`OrderConstants.LINE_UNIT_PRICE_TIMES_QUANTITY`
	`OrderConstants.LINE_SHIPPING`
	`OrderConstants.LINE_TAX`
	
	Note: The `getTotal()` method also allows you to combine different total types. For more information, see the *Javadoc*.

`orderLines`	A collection of the lines in the shopping cart that make up the customer's order.

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()%>
      <%= 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

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

Required Pipeline Session Attributes	`PipelineSessionConstants.USER_NAME`
	`PipelineSessionConstants.SHOPPING_CART`
	`PipelineSessionConstants.SPLITTING_PREFERENCE`
	`PipelineSessionConstants.SPECIAL_INSTRUCTIONS`
	`PipelineSessionConstants.ORDER_CONFIRMATION_NUMBER`

Updated Pipeline Session Attributes	`PipelineSessionConstants.ORDER_HANDLE` (Request scope) `PipelineSessionConstants.ORDER_VALUE` (Request scope) `PipelineSessionConstants.ORDER_SHIPPING_METHOD` (Request scope)

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

ResetCheckoutPC

Class Name	`com.beasys.commerce.ebusiness.order.pipeline.` `ResetCheckoutPC`
Description	Removes all Pipeline session attributes relating to the customer's checkout process.
Required Pipeline Session Attributes	None
Updated Pipeline Session Attributes	None

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

Type	Java object
JNDI Name	None
Exceptions	None

8 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:

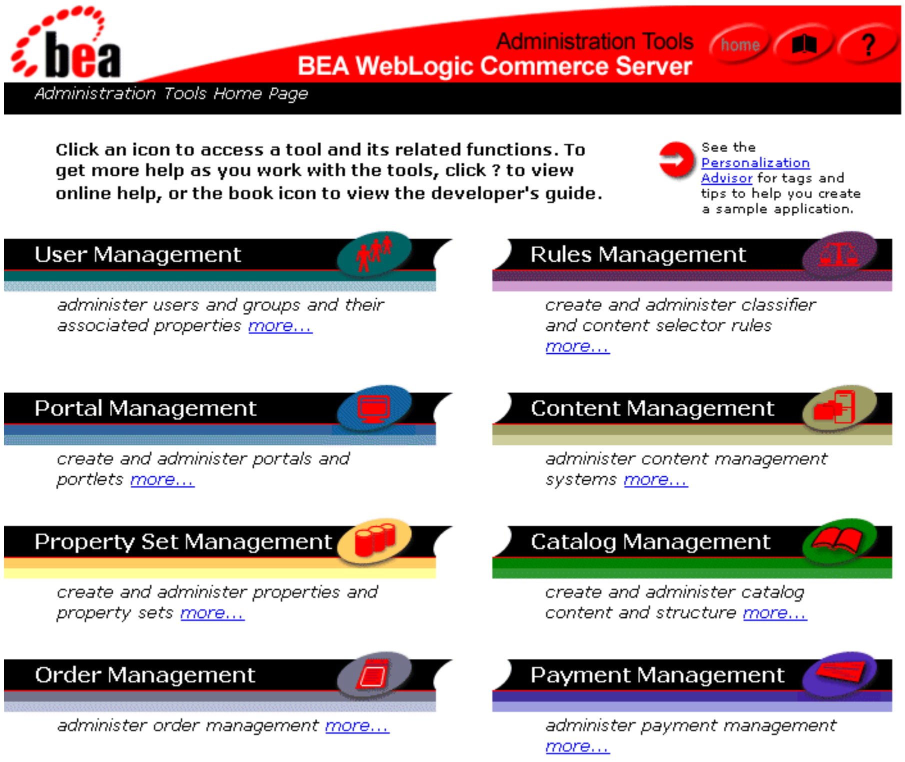
- Run `StartCommerce.bat` from the command line in the `WL_COMMERCE_HOME` directory, where `WL_COMMERCE_HOME` is the directory where you installed the WebLogic Commerce Server.
- From the Start menu, select Programs → WebLogic Commerce Server 3.1 → Start WebLogic Commerce Server.

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 8-1) 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.
- From the Start menu on a Windows system, select Programs → WebLogic Commerce Server 3.1 → Administration Tool.
- 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.

Figure 8-1 WebLogic Commerce Server Administration Tools Page



Click an icon to access a tool and its related functions. To get more help as you work with the tools, click ? to view online help, or the book icon to view the developer's guide.

See the Personalization Advisor for tags and tips to help you create a sample application.

User Management

administer users and groups and their associated properties more...

Rules Management

create and administer classifier and content selector rules more...

Portal Management

create and administer portals and portlets more...

Content Management

administer content management systems more...

Property Set Management

create and administer properties and property sets more...

Catalog Management

create and administer catalog content and structure more...

Order Management

administer order management more...

Payment Management

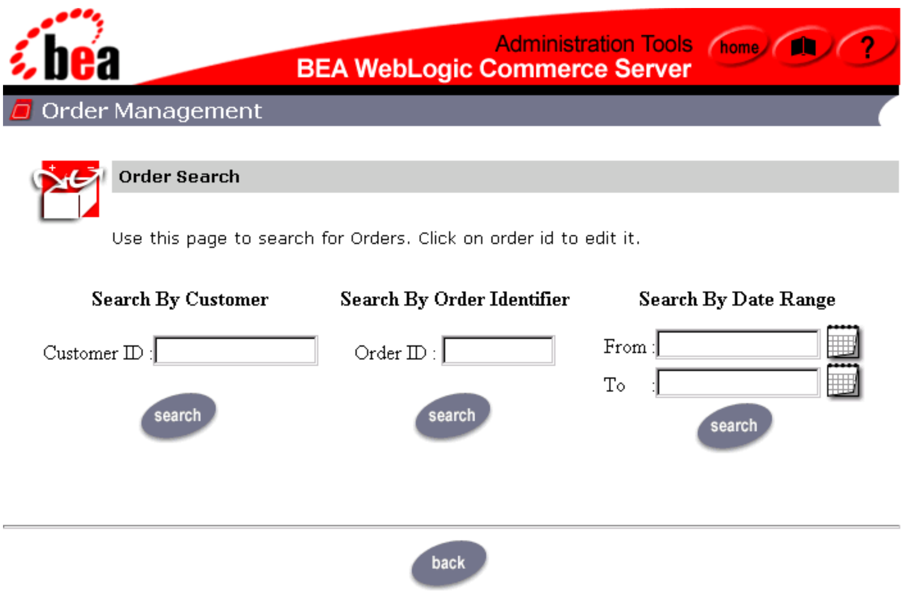
administer payment management more...

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 8-2) 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 8-2 The Order Management Search Page



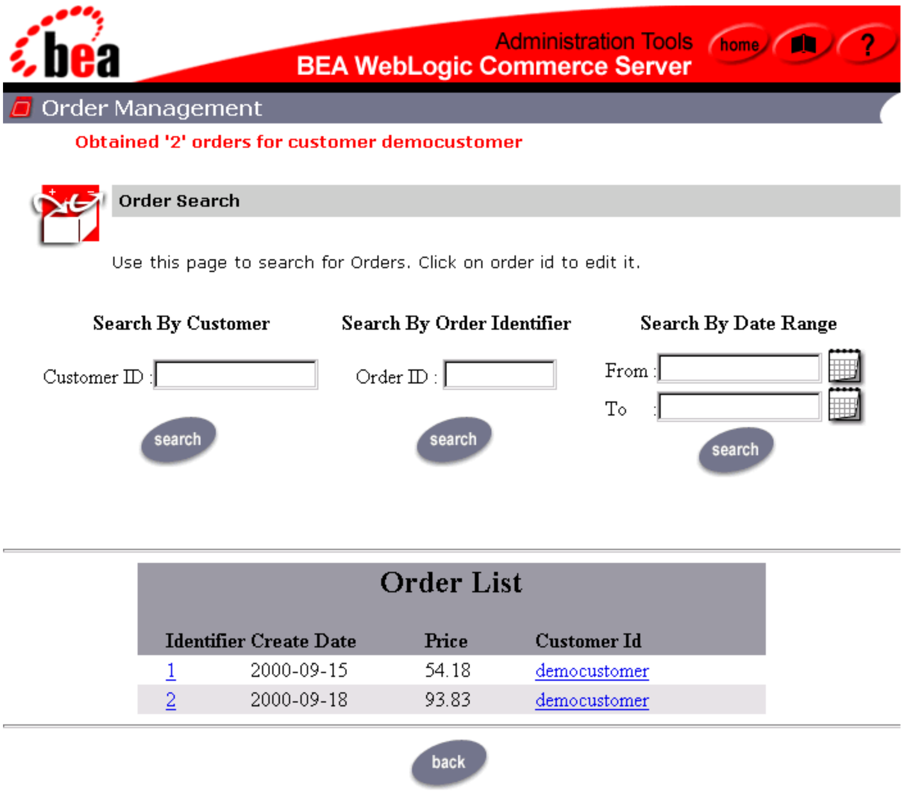
Use this page to search for Orders. Click on order id to edit it.

Search By Customer	Search By Order Identifier	Search By Date Range
Customer ID :	Order ID :	From :
		To :

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 8-3.

Figure 8-3 Sample Results for Order Search by Customer ID



Obtained '2' orders for customer democustomer

Order Search

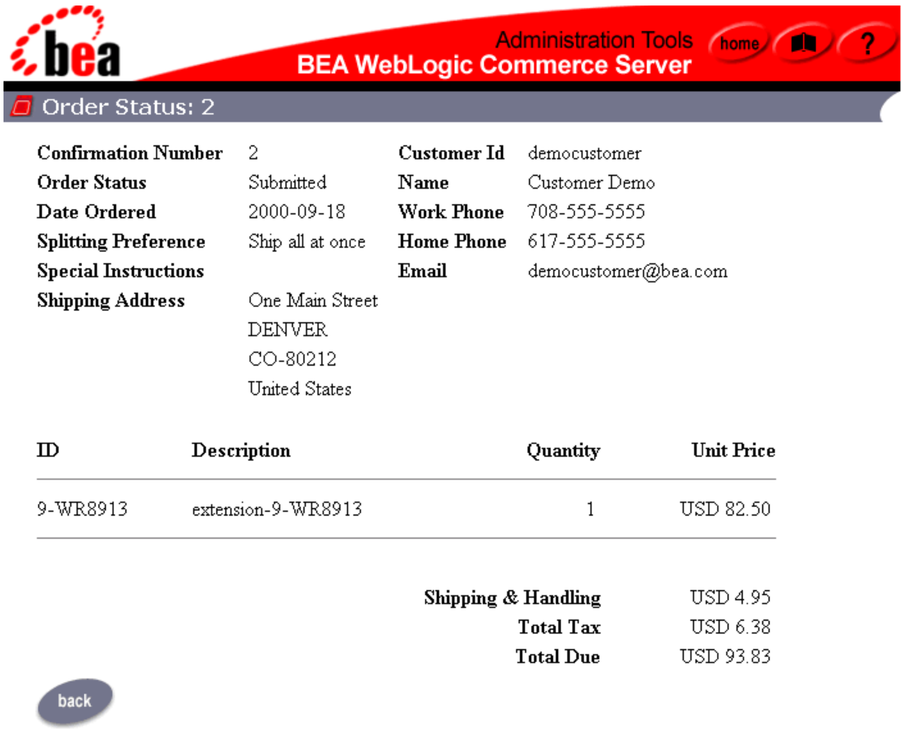
Use this page to search for Orders. Click on order id to edit it.

Order List

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

The Order List shows the Order Identifier number, the date the customer placed the order, and the price of the order. To see details for a particular order (including the product items ordered, shipping information, tax, and so on), click the hyperlinked Order Identifier number to load the Order Status page (shown in Figure 8-4). To return to the main Administration Tools page instead, click the Back button.

Figure 8-4 Sample Order Status Page



Order Status: 2

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

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

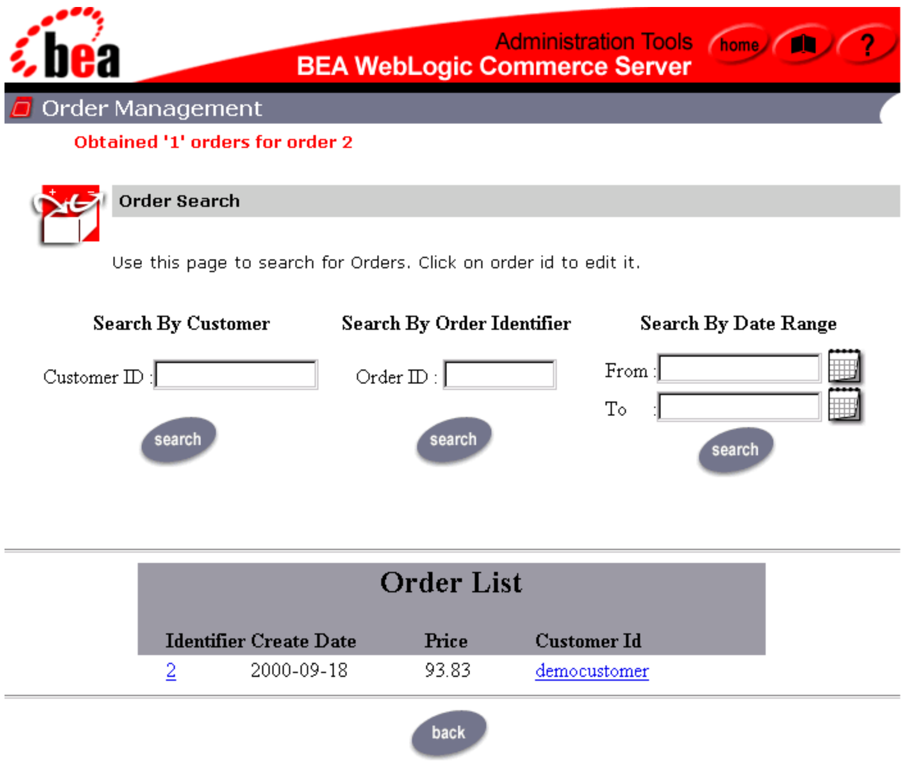
Shipping & Handling	USD 4.95
Total Tax	USD 6.38
Total Due	USD 93.83

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 8-5.

Figure 8-5 Sample Results for Order Search by Order Identifier Number



Obtained '1' orders for order 2

Order Search

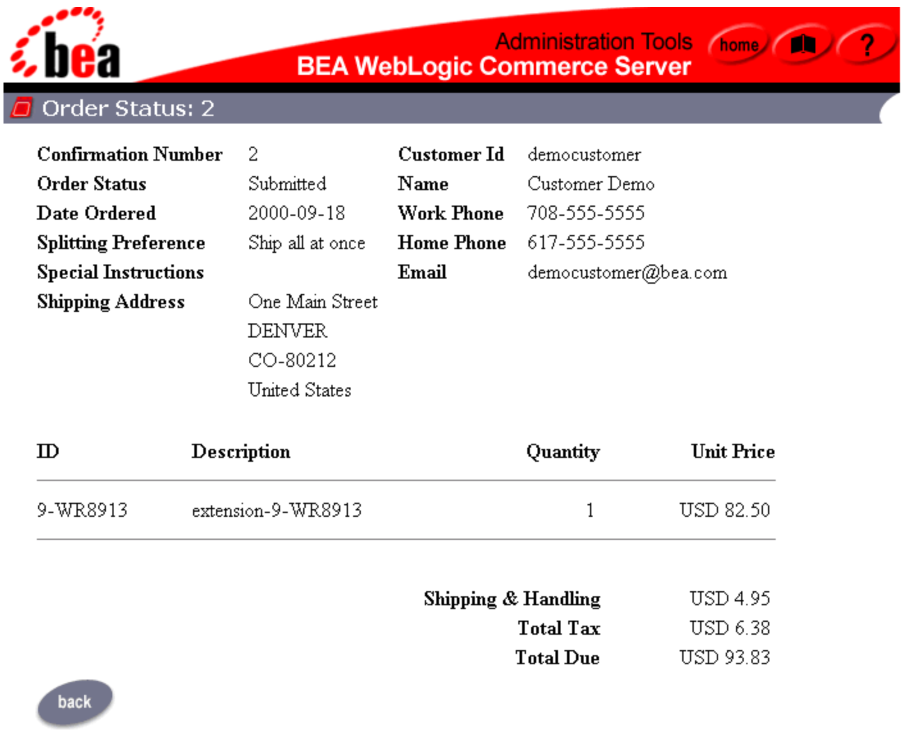
Use this page to search for Orders. Click on order id to edit it.

Order List

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

The Order List shows the Order Identifier number, the date the customer placed the order, and the price of the order. To see details for a particular order (including the product items ordered, shipping information, tax, and so on), click the hyperlinked Order Identifier number to load the Order Status page (shown in Figure 8-6). To return to the main Administration Tools page instead, click the Back button

Figure 8-6 Sample Order Status Page



Order Status: 2

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

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

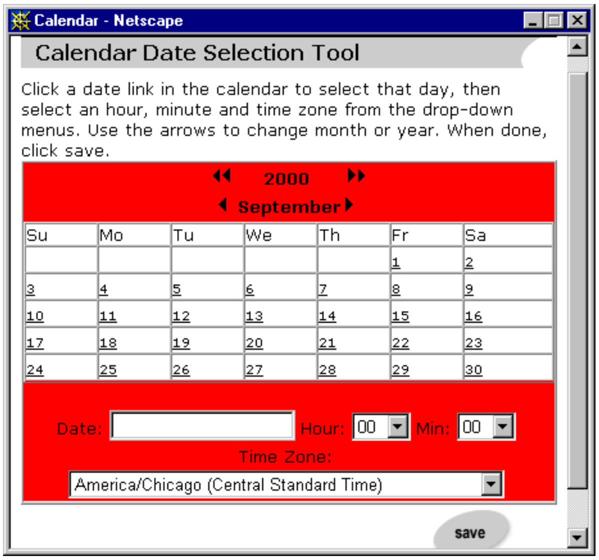
Shipping & Handling	USD 4.95
Total Tax	USD 6.38
Total Due	USD 93.83

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 8-7.

Figure 8-7 The Calendar Date Selection Tool



Click a date link in the calendar to select that day, then select an hour, minute and time zone from the drop-down menus. Use the arrows to change month or year. When done, click save.

Su	Mo	Tu	We	Th	Fr	Sa
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

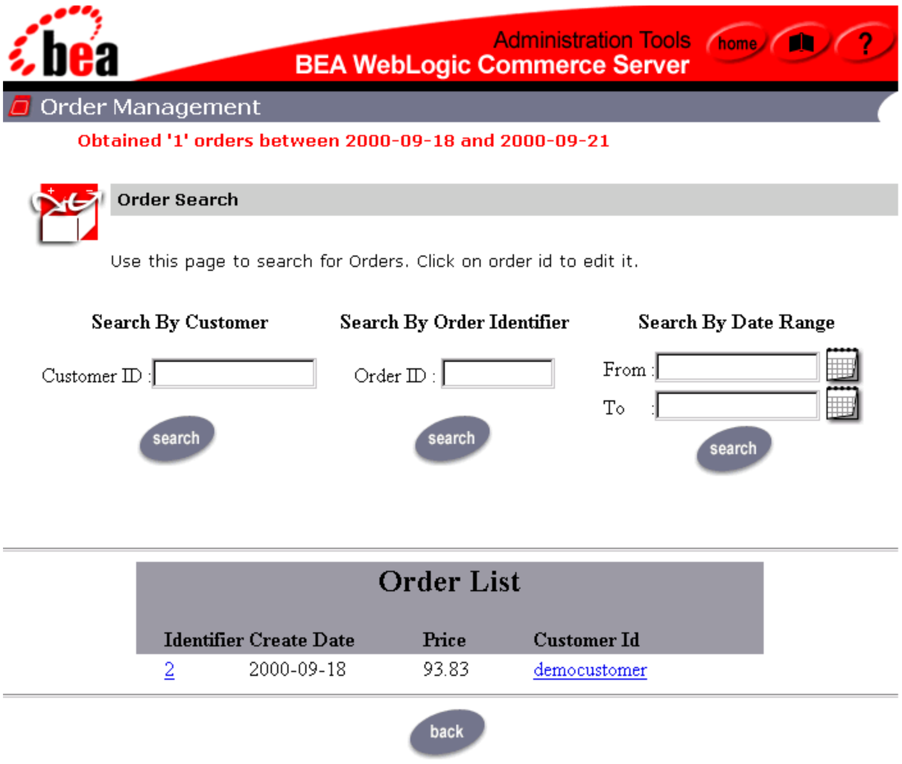
Date: Hour: 00 Min: 00
Time Zone: America/Chicago (Central Standard Time)

After clicking the Save button, the date, hour, minute and time zone you select with the Calendar Date Selection Tool appears in the From and To form fields, and you can now just click the Search button.

Note: The results for searches by date range are inclusive. That is, if you search for orders placed between July 22, 2000 and August 24, 2000, results will include orders placed on July 22 and orders placed on August 24.

A text message appears at the top of the page, indicating how many orders were found for the search. The actual results appear below the search fields in an Order List, as shown in Figure 8-8.

Figure 8-8 Sample Results for Order Search by Date Range



Obtained '1' orders between 2000-09-18 and 2000-09-21

Order Search

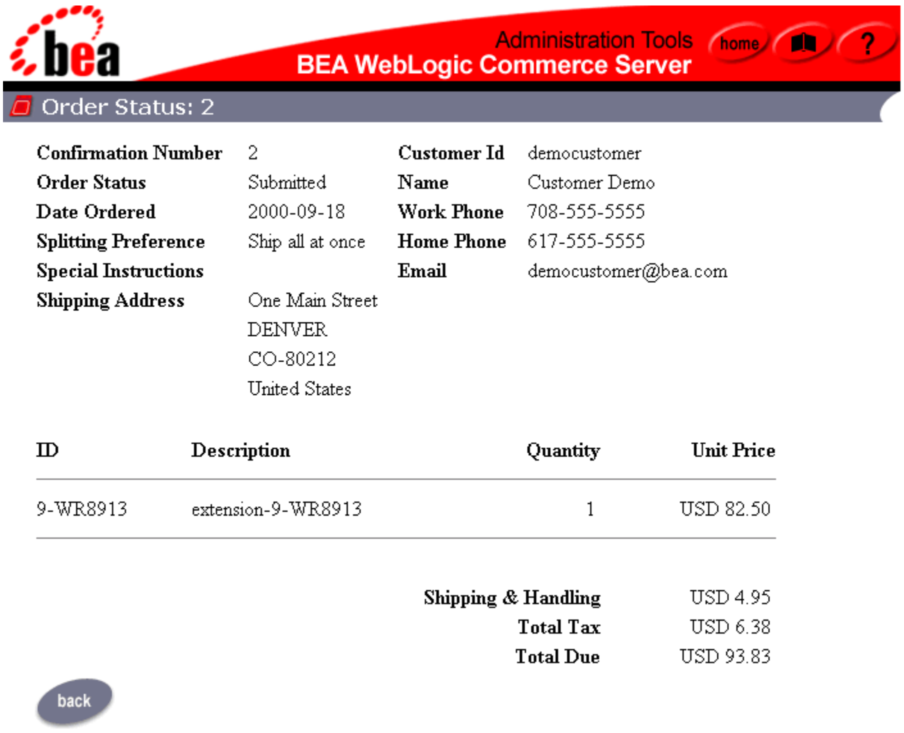
Use this page to search for Orders. Click on order id to edit it.

Order List

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

The Order List shows the Order Identifier number, the date the customer placed the order, and the price of the order. To see details for a particular order (including the product items ordered, shipping information, tax, and so on), click the hyperlinked Order Identifier number to load the Order Status page (shown in Figure 8-9). To return to the main Administration Tools page instead, click the Back button.

Figure 8-9 Sample Order Status Page



Order Status: 2

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

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

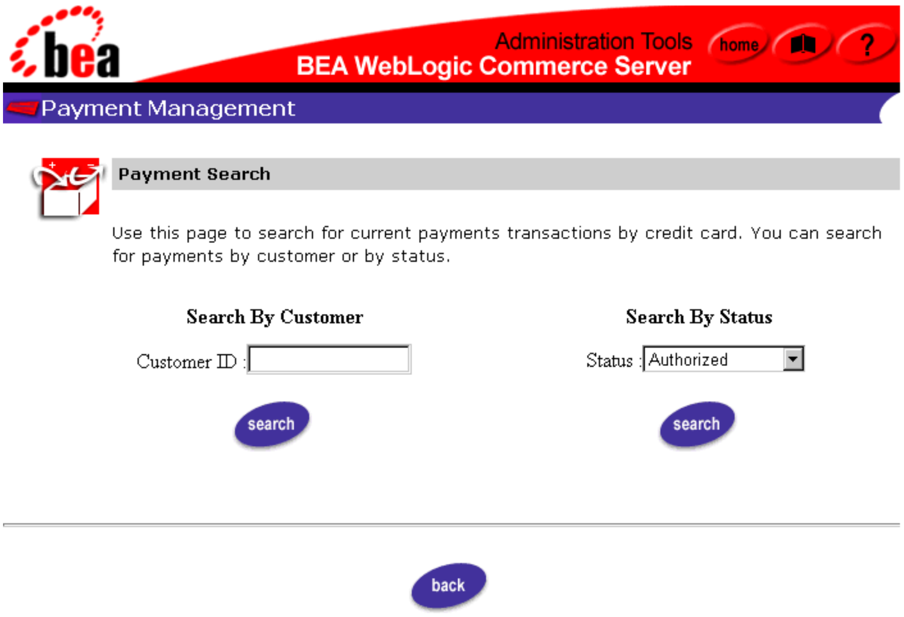
Shipping & Handling	USD 4.95
Total Tax	USD 6.38
Total Due	USD 93.83

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 8-10) 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 8-10 The Payment Management Search Page



Payment Search

Use this page to search for current payments transactions by credit card. You can search for payments by customer or by status.

Search By Customer

Customer ID :

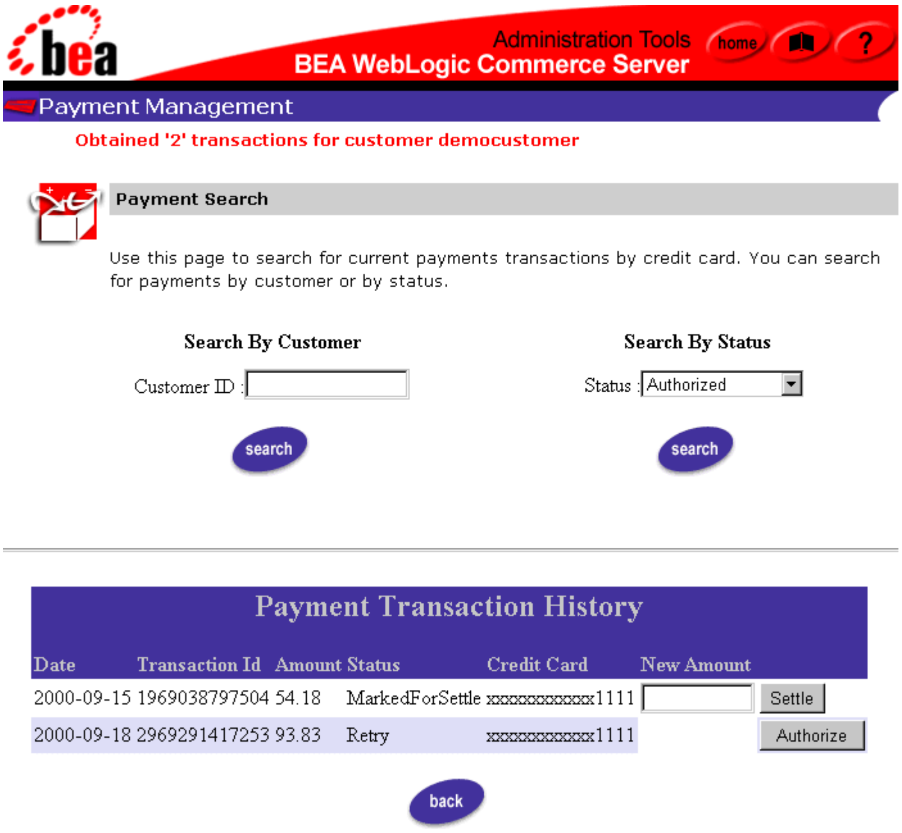
Search By Status

Status : Authorized

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 8-3.

Figure 8-11 Sample Results for Payment Search by Customer ID



Obtained '2' transactions for customer democustomer

Payment Search

Use this page to search for current payments transactions by credit card. You can search for payments by customer or by status.

Payment Transaction History

Date	Transaction Id	Amount	Status	Credit Card	New Amount	
2000-09-15	1969038797504	54.18	MarkedForSettle	xxxxxxxxxxxx1111		Settle
2000-09-18	2969291417253	93.83	Retry	xxxxxxxxxxxx1111		Authorize

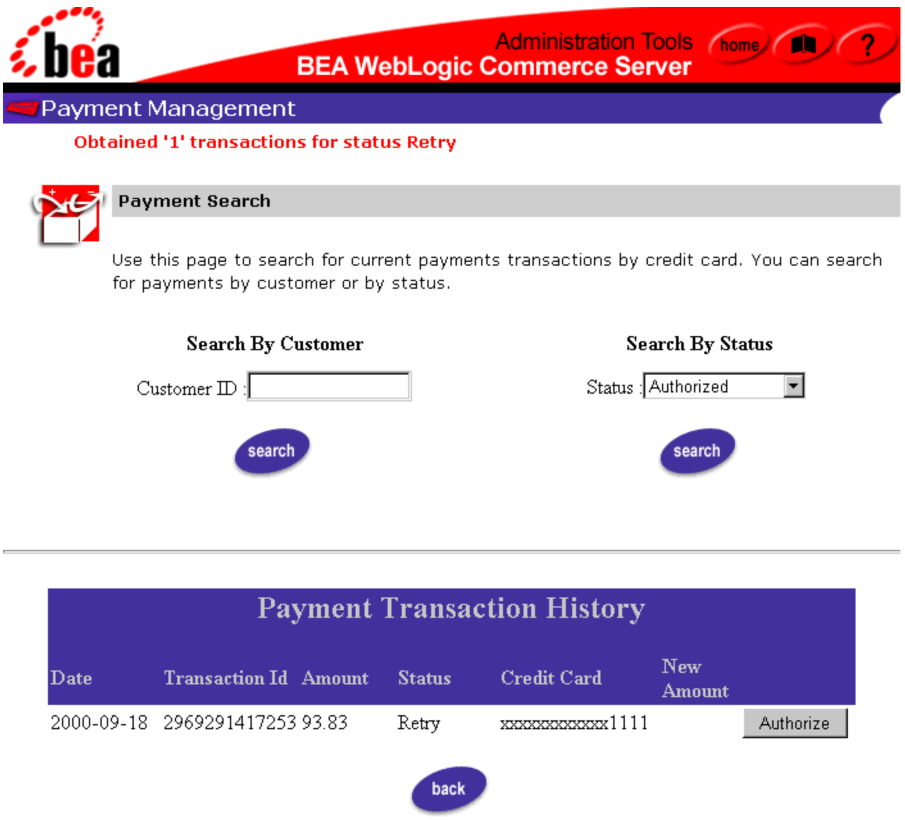
For a detailed explanation of the Payment Transaction History fields and further payment management activities, refer to “Authorizing, Capturing, and Settling Payments” on page 8-16 .

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 8-12.

Figure 8-12 Sample Results for Payment Search by Status



Obtained '1' transactions for status Retry

Payment Search

Use this page to search for current payments transactions by credit card. You can search for payments by customer or by status.

Search By Customer	Search By Status
Customer ID :	Status : Authorized

Payment Transaction History

Date	Transaction Id	Amount	Status	Credit Card	New Amount	
2000-09-18	2969291417253	93.83	Retry	xxxxxxxxxxxx1111		Authorize

For a detailed explanation of the Payment Transaction History fields and further payment management activities, refer to “Authorizing, Capturing, and Settling Payments” on page 8-16 .

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 8-1 provides a description for each of the possible payment status values.

Table 8-1 Payment Status Values

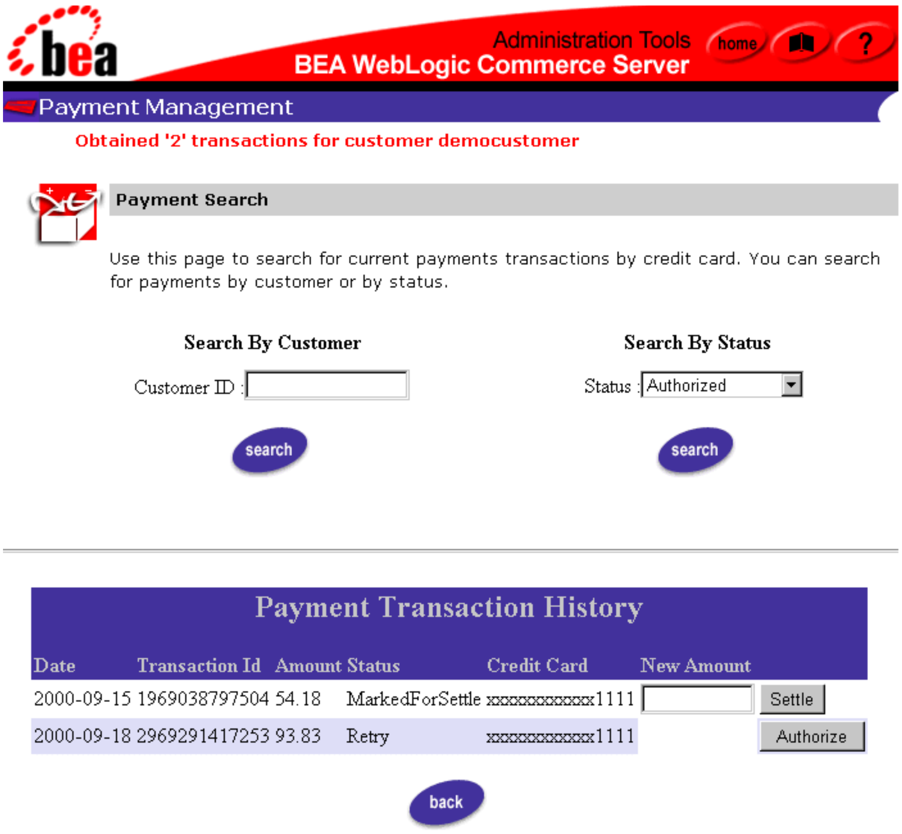
Status	Description
Authorized	The transaction has been successfully authorized, and is awaiting capture and settlement.
MarkedForSettle	The transaction has been batched for settlement (captured).
PendingSettle	The transaction settlement process has been initiated.
Settled	The transaction has been settled.
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 8-13).

Figure 8-13 Payment Transaction History With Authorize Button



Obtained '2' transactions for customer democustomer

Payment Search

Use this page to search for current payments transactions by credit card. You can search for payments by customer or by status.

Search By Customer

Customer ID :

Search By Status

Status : Authorized

Payment Transaction History

Date	Transaction Id	Amount	Status	Credit Card	New Amount	
2000-09-15	1969038797504	54.18	MarkedForSettle	xxxxxxxxxxxx1111		Settle
2000-09-18	2969291417253	93.83	Retry	xxxxxxxxxxxx1111		Authorize

Pressing this button will cause the BEA 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.

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 BEA 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 BEA 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