



BEA WebLogic Commerce Server

Product Catalog Management

BEA WebLogic Commerce Server 3.1
Document Edition 1.0.1
March 2001

Copyright

Copyright © 2000 BEA Systems, Inc. All Rights Reserved.

Restricted Rights Legend

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

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

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

Trademarks or Service Marks

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

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

Product Catalog Management

Document Edition	Date	Software Version
1.0.1	March 2001	BEA WebLogic Commerce Server 3.1

Contents

What You Need to Know	xii
e-docs Web Site	xiii
How to Print the Document	xiii
Related Information	xiv
Contact Us!	xiv
Documentation Conventions	xv
1. Introduction to the Product Catalog	
What Does the Product Catalog Provide?	1-3
Catalog Hierarchy	1-6
Product Catalog Development Roles	1-8
How Are the Product Catalog Features and Other Commerce Features Linked? ... 1-9	
Using Commerce Server and Personalization Server Features in an Application .. 1-11	
Next Step	1-11
2. The Product Catalog Schema	
The Entity-Relation Diagram	2-1
The Catalog Schema Is Based on Dublin Core Standard	2-4
The WLCS_CATEGORY Database Table	2-5
The WLCS_PRODUCT Database Table	2-9
The WLCS_PRODUCT_CATEGORY Database Table	2-15
The WLCS_PRODUCT_KEYWORD Database Table	2-16
The WLCS_CAT_PROP_* Database Tables for Custom Attributes	2-17
The WLCS_CAT_ENTITY_ID Database Table	2-17
The WLCS_CAT_PROP_ID Database Table	2-18

The WLCS_CAT_PROP_BOOLEAN Database Table.....	2-19
The WLCS_CAT_PROP_INTEGER Database Table.....	2-19
The WLCS_CAT_PROP_FLOAT Database Table.....	2-19
The WLCS_CAT_PROP_TEXT Database Table.....	2-20
The WLCS_CAT_PROP_DATETIME Database Table.....	2-20
The WLCS_CAT_PROP_USER_DEFINED Database Table.....	2-21
The SQL Files and Defined Constraints.....	2-21

3. Using the Product Catalog Database Loader

The Input File for DBLoader.....	3-2
The dbloader.properties File.....	3-4
Running the DBLoader Program.....	3-7
To Run the Program.....	3-8
DBLoader Log Files.....	3-9
DBLoader Validations.....	3-9
Important Database Considerations.....	3-10
Using Database-specific Data Loaders.....	3-11
Using Third-party Data Loaders.....	3-13

4. Catalog Administration Tasks

Starting the Server.....	4-2
Starting the Administration Tool.....	4-4
Changing the Administrator Password.....	4-8
Loading Data into the Product Catalog.....	4-11
Adding Categories to the Catalog.....	4-12
Adding Items to the Catalog.....	4-18
Controlling the Visibility of Items in the Catalog.....	4-22
Assigning Items to Categories.....	4-23
What if I Have a Large Amount of Data?.....	4-23
Using the Administration Screens to Assign Items to Categories.....	4-24
Edit the Attributes for Categories and Items.....	4-27
Editing Category Attributes.....	4-27
Editing Product Item Attributes.....	4-30
Edit the Availability of an Item.....	4-34
How Are Categories and Items Displayed to the Web Site User?.....	4-36

Deleting Items or Removing Items from One or More Categories	4-37
Caching Considerations.....	4-38
Deleting an Item from the Catalog.....	4-38
Removing an Item from One or More Categories.....	4-41
Removing Categories	4-43
Moving Items from One Category to Another Category.....	4-46
Define Custom Attributes for Items	4-46
Improving Catalog Performance by Optimizing the Catalog Cache.....	4-48
Cache-Related Values in WeblogicCommerce.properties	4-49
Considering Hardware Costs Versus the Cost of Dissatisfied Web Site Users	
4-52	
What's in Each Cache Initially?.....	4-52
The Catalog Cache Administration Screen	4-53
Using the wlcs-catalog.properties File	4-55
Location.....	4-56
Some Property Values You Might Modify	4-56
Editing the Catalog Schema Definition.....	4-59

5. The Product Catalog JSP Templates and Tag Library

Introduction	5-3
JSP Templates Overview.....	5-3
On Which JavaServer Page Will My Users Start?.....	5-4
Web Applications and the weblogic.properties File	5-4
XML Deployment Descriptor Files	5-5
commercef Property Set and DestinationDeterminer	5-6
Sequence Review and the Browser View	5-8
JavaServer Pages (JSPs).....	5-12
main.jsp Template	5-14
Sample Browser View	5-14
Location in the WebLogic Commerce Server Directory Structure...	5-17
Tag Library Imports	5-17
Java Package Imports.....	5-17
Location in the Default Webflow.....	5-18
Included JSP Templates	5-18
Events.....	5-19

Dynamic Data Display	5-20
Form Field Specification	5-24
browse.jsp Template.....	5-25
Sample Browser View.....	5-28
Location in the WebLogic Commerce Server Directory Structure...	5-30
Tag Library Imports	5-31
Java Package Imports	5-31
Location in the Default Webflow	5-31
Included JSP Templates	5-32
Events	5-42
Dynamic Data Display	5-44
Form Field Specification	5-46
details.jsp Template	5-47
Sample Browser View.....	5-47
Location in the WebLogic Commerce Server Directory Structure...	5-49
Tag Library Imports	5-49
Java Package Imports	5-50
Location in the Default Webflow	5-50
Included JSP Templates	5-50
Events	5-51
Dynamic Data Display	5-52
Form Field Specification	5-53
search.jsp	5-54
Sample Browser View.....	5-54
Location in the WebLogic Commerce Server Directory Structure...	5-56
Tag Library Imports	5-57
Java Package Imports	5-57
Location in the Default Webflow	5-57
Included JSP Templates	5-58
Events	5-58
Dynamic Data Display	5-60
Form Field Specification	5-63
searchresults.jsp.....	5-64
Sample Browser View.....	5-64
Location in the WebLogic Commerce Server Directory Structure...	5-66

Tag Library Imports	5-66
Java Package Imports	5-67
Location in the Default Webflow	5-67
Included JSP Templates	5-67
Events	5-68
Dynamic Data Display	5-69
Form Field Specification	5-73
Query-based Search Syntax	5-74
Using Comparison Operators to Construct Queries	5-76
Searchable Catalog Attributes	5-77
Controlling the Number of Search Results	5-78
Input Processors	5-80
CatalogIP	5-80
GetProductItemIP	5-81
GetCategoryIP	5-82
KeywordSearchIP	5-83
ExpressionSearchIP	5-84
MoveAttributeIP	5-85
RemoveAttributeIP	5-86
Pipeline Components	5-87
CatalogPC	5-87
GetCategoryPC	5-88
GetProductItemPC	5-89
GetParentPC	5-90
GetAncestorsPC	5-91
GetProductItemsPC	5-92
GetSubcategoriesPC	5-93
MoveAttributePC	5-94
RemoveAttributePC	5-95
SearchPC	5-96
The Catalog JSP Tag Library: cat.tld	5-97
The <catalog:getProperty> Tag	5-98
Format	5-98
Attributes	5-98
Example	5-99

The <catalog:iterateViewIterator> Tag	5-100
Format	5-100
Attributes	5-100
Examples	5-101
The <catalog:iterateThroughView> Tag	5-102
Format	5-102
Attributes	5-102
Examples	5-103

6. Using the API to Extend the Product Catalog

Overview of the Product Catalog API	6-2
Catalog Architecture and Services	6-3
Catalog Architecture	6-3
Catalog Manager	6-5
Product Item Manager	6-8
Category Manager	6-9
Custom Data Manager	6-13
Catalog Query Manager	6-14
The Catalog Cache	6-15
Writing Your Own Catalog Service	6-17
Create New Services	6-19
Sample Source Code	6-20
Changes to ejb-jar.xml	6-33
Changes to weblogic-ejb-jar.xml	6-36

7. Product Catalog Internationalization Support

Support for Multiple Languages	7-2
Language and Country Codes	7-2
About the CatalogRequest Object	7-3
Persisting Language Information to the Catalog Database	7-4
Product Items and Categories	7-4
Image Support	7-5
Limiting Search Results by Language	7-5
Using the Catalog Architecture to Maintain Internationalized Product Catalogs ..	7-7

Method 1: Filtering Product Catalog Content	7-7
Method 2: Parsing Language-Specific Data.....	7-8
Two Languages	7-9
Multiple Languages.....	7-9
Method 3: Multiple Product Catalog Instances.....	7-10
Method 4: Language-Based Service Routing.....	7-12

Index



About This Document

This document explains how to use BEA WebLogic Commerce Server™ to build and customize a Web-based product catalog.

The WebLogic Commerce Server product catalog provides commonly used items and attributes that are found on e-commerce Web sites. JavaServer Page (JSP) templates are provided as a starting point, and you can easily customize the presentation of each page to match your business branding requirements and design preferences. The documented product catalog schema identifies the structure and relationships of the database tables. Behind the scenes, pre-built Enterprise Java Beans (EJBs) and other Commerce Server features provide the computing infrastructure and scalability that enables your site to support many concurrent users. Property files and administration screens allow you to manage the product catalog's behavior and content.

This document includes the following topics:

- Chapter 1, “Introduction to the Product Catalog,” sets the stage by summarizing the features provided by the WebLogic Commerce Server features and the job-based roles of the people who will build and customize the catalog.
- Chapter 2, “The Product Catalog Schema,” explains the structure of the product catalog. This understanding is essential to moving your existing data into the catalog database, or adding new data to the catalog.
- Chapter 3, “Using the Product Catalog Database Loader,” describes a bulk loader program that you can use to insert, update, to delete large numbers of category and items records in the product catalog database.
- Chapter 4, “Catalog Administration Tasks,” describes the administration screens to find, add, edit, or remove categories and items in the product catalog. In addition, the chapter explains how to tune the catalog for optimal performance by adjusting in-memory cache settings for categories and items.

-
- Chapter 5, “The Product Catalog JSP Templates and Tag Library,” describes the JSP templates provided by WebLogic Commerce Server. You can use the templates as a starting point for your e-commerce Web pages. You can then customize the pages to match your corporate branding requirements, design preferences, navigation options, and the content of your catalog.
 - Chapter 6, “Using the API to Extend the Product Catalog,” describes the various options available for extending, customizing, or writing third-party integrations for the WebLogic Commerce Server product catalog.
 - Chapter 7, “Product Catalog Internationalization Support,” describes how your product catalog can be localized for users in other countries, and provides instructions about how you might accomplish tasks related to internationalization.

What You Need to Know

This document is intended primarily for Web content developers, JSP developers, Java developers, and administrators who will work together to deliver an e-commerce product catalog. While those roles describe the types of tasks involved in building the site, it is recognized that people in your organization often have job assignments that span multiple roles.

- **Web content developers** use the JavaServer Page templates and add customized HTML tags to match your corporate branding requirements, design preferences, navigation options, and the content of your catalog.
- **JSP developers and or Java programmers** handle any customization of the JavaServer Pages and might use the JSP tags provided by WebLogic Commerce Server (or custom tags) to extend the functionality provided on the pages. Java programmers might also modify or add scriptlets on the pages. **Business analysts** bring their market research to the design table and help the developers understand the required product items, categories, and pricing.
- **System or Web administrators** maintain the data in the catalog, either adding, editing, or removing categories and items from the catalog. Administrators also optimize the performance of the catalog by adjusting settings for the in-memory cache of categories and items in the catalog.

e-docs Web Site

BEA product documentation is available on the BEA corporate Web site. From the BEA Home page, click on Product Documentation or go directly to the “e-docs” Product Documentation page at <http://e-docs.bea.com>. The WebLogic Commerce Server and WebLogic Personalization Server 3.1 documentation starts at <http://e-docs.bea.com/wlcs310/index.htm>.

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 by clicking the PDF Files link on the WebLogic Commerce Server and WebLogic Personalization Server documentation Home page, at <http://e-docs.bea.com/wlcs310/index.htm>. A PDF version of this document is also available on your local system if you installed the separate WebLogic Commerce Server documentation kit. In the installed WebLogic Commerce Server directory, the documentation’s default starting location is:

```
WL_COMMERCE_HOME\server\public_html\docs\index.htm
```

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

For more information about the Java 2 Enterprise Edition (J2EE) APIs, see the Sun Microsystems, Inc. Web site at <http://java.sun.com/j2ee/>.

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.
<i>italics</i>	Indicates emphasis or book titles.
monospace text	Indicates code samples, commands and their options, data structures and their members, data types, directories, and filenames and their extensions. Monospace text also indicates text that you must enter from the keyboard. <i>Example:</i> <pre>public interface Item extends ConfigurableEntity { public ItemValue getItemByValue() throws RemoteException; public void setItemByValue(ItemValue value) throws RemoteException; //... }</pre>
monospace boldface text	Identifies significant words in code. <i>Example:</i> <pre>void commit ()</pre>
<i>monospace italic text</i>	Identifies variables in code. <i>Example:</i> <pre>String <i>expr</i></pre>
UPPERCASE TEXT	Indicates device names, environment variables, and logical operators. <i>Examples:</i> <pre>LPT1 SIGNON OR</pre>

Convention	Item
{ }	Indicates a set of choices in a syntax line. The braces themselves should never be typed.
[]	Indicates optional items in a syntax line. The brackets themselves should never be typed. <i>Example:</i> buildobjclient [-v] [-o name] [-f file-list]... [-l file-list]...
	Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.
...	Indicates one of the following in a command line: <ul style="list-style-type: none"> ■ That an argument can be repeated several times in a command line ■ That the statement omits additional optional arguments ■ That you can enter additional parameters, values, or other information The ellipsis itself should never be typed. <i>Example:</i> buildobjclient [-v] [-o name] [-f file-list]... [-l file-list]...
.	Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed.

1 Introduction to the Product Catalog

Internet-savvy consumers today have high expectations about their online shopping experience. They love the convenience of finding just about any product item for sale on the Web. They expect that an e-commerce Web site's pages will load quickly in their browser, and do not care if there happen to be a thousand other concurrent users accessing the site's servers. They want to be able to pay for the items securely with their credit card, and have the items delivered directly to their home or business.

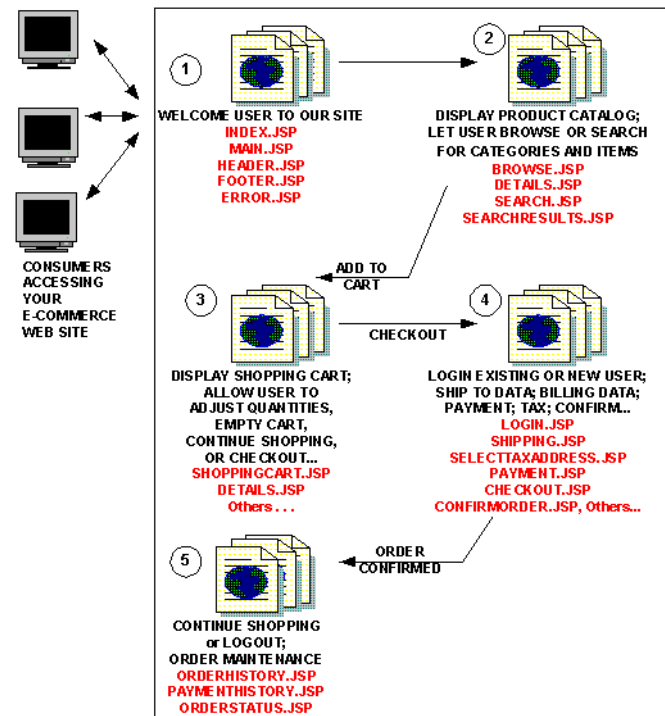
In the Web application development industry, we are all familiar with the model for an e-commerce Web site. One goal is to attract consumers who might have purchased items in a retail store, and instead get them to buy the items online, on **your** Web site. The companies that want to exploit the Web more effectively include:

- Already established companies with a solid market presence that pre-dates the Internet revolution. These companies are extending their marketing and sales reach by enabling existing and new customers to buy items on their e-commerce Web sites. They are moving from “bricks and mortar” to “bricks and clicks.”
- New online-only companies that have established their store front on the Internet.

Both types of companies are attempting to succeed in a highly competitive environment. Some already have the trained programming staff with the expertise in the J2EE APIs to create from scratch the Enterprise Java Beans and JavaServer Pages (JSPs) that will support the Web site's computing model. Many firms, however, need to get a jump on their competition as they work to go live with their e-commerce Web site as soon as possible. They cannot afford to wait six months or a year to develop the standardized database resources, EJB programming expertise, and JSP templates that together will provide the commonly expected Web site functions shown in Figure 1-1.

1 Introduction to the Product Catalog

Figure 1-1 Simple View of E-commerce Web Site Functions



Of course, writing the code to create, build, deploy, and maintain all the processing shown in Figure 1-1 would require a lot of time and effort. In addition to the front-end presentation layer that you see in the previous diagram, high-volume Web sites also require an underlying software infrastructure that makes it possible to support peak usage of their commerce site by eager consumers.

So how can a company get a jump on the competition and implement a scalable e-commerce Web site? The answer is: by using the pre-built features that come with BEA WebLogic Commerce Server!

What Does the Product Catalog Provide?

The WebLogic Commerce Server product catalog provides the following features:

- **A well-designed database schema** and build scripts that define the commonly used product items and attributes found on Web-based catalog sites. The metadata for the product catalog is based on the Dublin Core Open Standard. In the current release, schemas are provided for Oracle® and Cloudscape® databases.

The schema establishes a `CATEGORY_ID` field as the unique, primary key of the category metadata, which uses the `WLCS_CATEGORY` database table. The schema also establishes the `SKU` field (an acronym for “Stock Keeping Unit”) as the unique, primary key of the product item metadata, which uses the `WLCS_PRODUCT` database table.

The schema is described in Chapter 2, “The Product Catalog Schema,” of this document.

- **A bulk loader program called DBLoader** that takes a data input file and populates the product catalog database. With DBLoader, you can add large volumes of product item and category records in a single command. Adding the records is done by specifying the `-insert` option on the DBLoader command line, and is probably the option you will use most often. However, you can also use the `-update` or `-delete` option with DBLoader.

The DBLoader program and third-party data loader utilities are described in Chapter 3, “Using the Product Catalog Database Loader,” of this document.

- **Browser-based administration screens** that you can use to manage the product catalog’s content and behavior. The screens allow you to find, add, edit, or remove product categories or items. Figure 1-2 shows a portion of a sample administration screen, where the administrator is editing the values for an existing item.

1 Introduction to the Product Catalog

Figure 1-2 Sample Administration Screen

The screenshot shows a web browser window titled "Edit Item Core Attributes - Microsoft Internet Explorer". The address bar shows the URL: `http://qatest:7501/tools/application/admin?dest=%2Ftools%2Fcatalog%2Fitem_edit.jsp&wics_catalog_item_sku=9-WD1`. The page header features the BEA logo and "Administration Tools" with navigation links for "home", "help", and "finished". The main content area is titled "Edit Item Information" and contains the following fields:

SKU:	9-WD10116	The Stock Keeping Unit
Item Name*:	lubricant-9-WD10116	The name of the item.
Short Description*:	lubricant: special purpose; wd-40; 12 per pack	Short description associated with the item.
Visible:	<input checked="" type="checkbox"/>	Include or exclude the item from the catalog.
Long Description:	lubricant; special purpose; wd-40; 12 per pack; 16 oz; maintenance tools, loctite,	Long description of the item.
Price Amount:	49.5	The selling price of the item.
Price Currency:	USD	The currency for the selling price.
MSRP Amount:	50.5	The manufacturer suggested retail price.
MSRP Currency:	USD	The currency for the manufacturer suggested retail price.
Tax Code:	73212	The Tax Code for the item.
Shipping Code:		The Shipping Code for the item.
Summary JSP URL:	/commerce/catalog/includes/itemssummary.jsp	The JSP associated with the item summary.
Detail JSP URL:	/commerce/catalog/includes/itemdetails.jsp	The JSP associated with the item details.
Type:		The nature of the item.

In addition to finding, adding, editing, and deleting product items and categories, you can also use the administration screens to tune the performance of your product catalog. You can adjust the in-memory cache of items and categories, a feature that can significantly improve the satisfaction level of customers shopping on your Web site.

Web site administrators should read Chapter 4, "Catalog Administration Tasks," in this document.

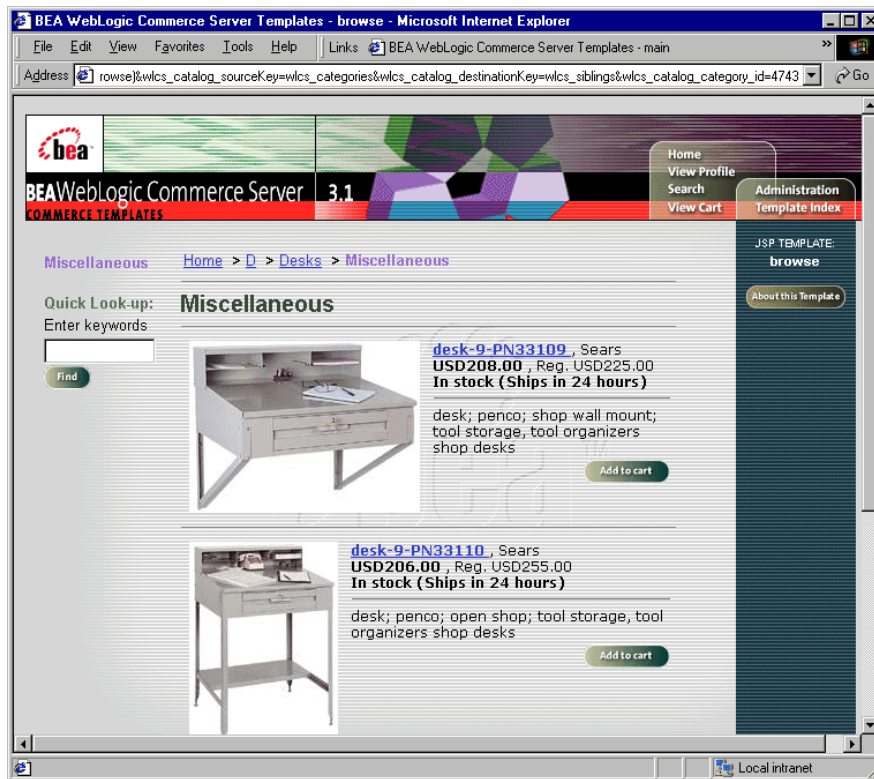
- **JavaServer Page (JSP) templates** that you can use as a starting point for your e-commerce Web development, and easily customize. You can change the presentation of each page to match your corporate branding requirements, design preferences, navigation options, and the content of your catalog. The JSP

What Does the Product Catalog Provide?

templates use a combination of HTML code, Java scriptlets, and JSP tags to provide the presentation layer of your Web-based product catalog.

For example, Figure 1-3 shows the running output of a sample `browse.jsp` file provided by WebLogic Commerce Server.

Figure 1-3 Running Output of a Sample JSP Template



For details about the JSPs used with the product catalog, see Chapter 5, “The Product Catalog JSP Templates and Tag Library,” in this document. For related details about the JSPs used with order processing (shopping cart, shipping, tax, checkout, and so on), see *BEA WebLogic Commerce Server Order Processing Package*. For related details about the JSPs used to register users, see *BEA WebLogic Commerce Server Registration and User Processing Package* in the online documentation.

1 Introduction to the Product Catalog

- Behind the scenes, an **Application Programming Interface (API)** of pre-built Enterprise Java Beans (EJBs) and other Commerce Server features that provide the computing infrastructure and scalability that enables your site to support many concurrent users.

Chapter 6, “Using the API to Extend the Product Catalog,” describes the various options available for extending, customizing, or writing third-party integrations for the WebLogic Commerce Server product catalog. The API is also described in the WebLogic Commerce Server *online Javadoc* for the `com.beasys.commerce.ebusiness.catalog.*` packages.

- **Flexible internationalization support** that allows you to choose among many architectural options when developing multilingual product catalogs. These features will help you internationalize your system and render a localized version of each category or item on a Web page, including text descriptions, images, item cost, type of currency, and so on.

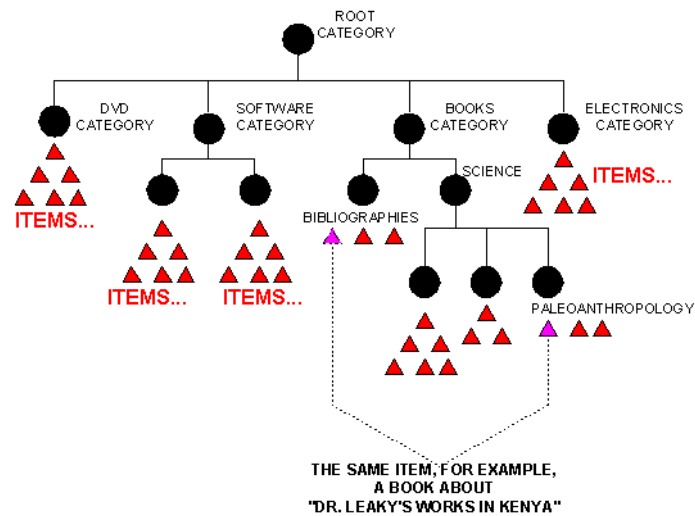
Chapter 7, “Product Catalog Internationalization Support,” describes how you can internationalize your product catalogs in more detail.

Catalog Hierarchy

Categories in the product catalog exist in a hierarchy, as illustrated in Figure 1-4.

The example shown in the figure deviates from the sample data that is seen when you run the WebLogic Commerce Server templates. However, the following example illustrates a point about how items can reside in more than one category.

Figure 1-4 Sample Product Catalog Hierarchy



Note that any given category needs to be aware of the following:

- Items in this category.
- The parent category. If the category is already a top-level category in the hierarchy, then the parent category is the catalog’s root category.
- Sibling categories that exist at the same level as the current category.

Also note that an individual item can reside in more than one category. For example, a hardcopy book about the works of paleoanthropologist Dr. Richard Leaky in Kenya might reside in a Books → Bibliographies category and also in a subcategory of Books → Science → Paleoanthropology.

If you delete the instance of an item in one category, it continues to reside in any other categories in which it might exist. Also, if you delete all the categories that an item belongs to, the item is moved to a separate, “uncategorized items” category in the catalog. Sometimes these uncategorized items are referred to as “orphaned items.”

Unlike items, categories cannot reside outside of their hierarchical path. In other words, a category that resides in one path of the hierarchy cannot also reside in another path of the hierarchy.

Product Catalog Development Roles

Given the goal of building a Web-based product catalog, what are the roles of the people on the development team? The product catalog development would most likely be done by a team of people in your organization who collaborate to deliver an e-commerce Web solution. While the roles that are summarized in the following list describe the types of tasks involved in building the site, it is recognized that people often have job assignments that span multiple roles.

- **Web content developers** use the JavaServer Page (JSP) templates and add customized HTML tags to match your corporate branding requirements, design preferences, navigation options, and the content of your catalog.
- **JSP developers and Java programmers** handle any customization of the JavaServer Pages and might use the JSP tags provided by WebLogic Commerce Server (or custom tags) to extend the functionality provided on the pages. Java programmers might also modify or add scriptlets on the pages. **Business analysts** bring their market research to the design table and help the developers understand the required product items, categories, and pricing. The business analysts usually are not involved in the page design or customization steps that others on the team manage.
- **System or Web administrators** maintain the data in the catalog, either adding, editing, or removing categories and items from the catalog. Administrators also optimize the performance of the catalog by adjusting settings for the in-memory cache of categories and items in the catalog.

How Are the Product Catalog Features and Other Commerce Features Linked?

The WebLogic Commerce Server product catalog is only a portion of the features provided in this release. Related features, of course, are implemented by the order processing and user registration packages. When a user of your Web site decides to click the Add to Cart button (or equivalent) on one of the catalog pages, by default the user is directed to the shopping cart portion of the order processing package.

Once the user clicks Add to Cart, a series of JavaServer Pages under the control of background processing are employed to step the user through the process of entering the information required to complete the order.

You can modify the actual sequence of pages, or Webflow, by editing a configuration file called `webflow.properties`. A default Webflow file is provided by WebLogic Commerce Server. The advantage here is that the flow of the website (that is, what page to go to next) is not hardwired into each page, but is managed by an external flow manager. Rather than having to edit HTML and JSP files to change the page sequence, the administrator simply modifies `webflow.properties`. Also, you do not have to restart the WebLogic Server instance in which your WebLogic Commerce Server applications is running, allowing for dynamic site management.

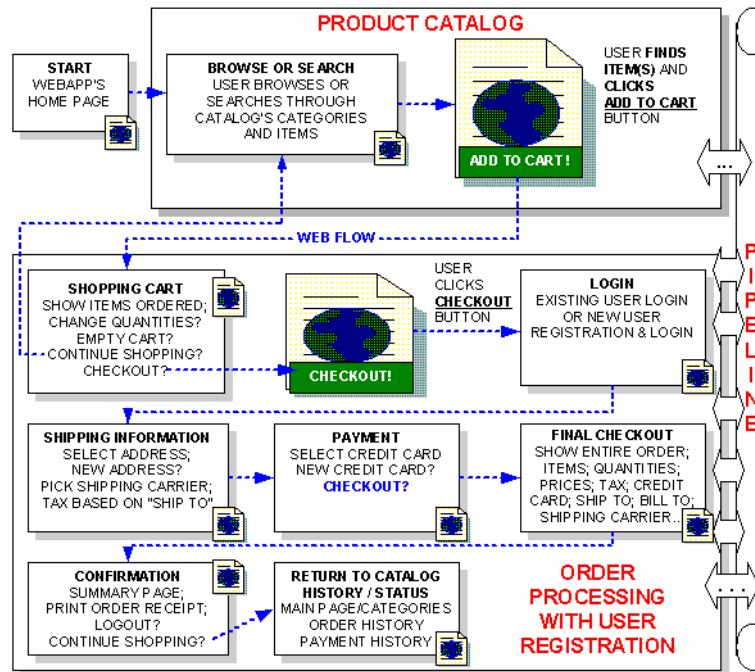
Underlying all this processing for both catalog management and order management is another important feature called the WebLogic Commerce Server pipeline. The pipeline is a mechanism for binding together a sequence of services into a single named service. While the JavaServer Pages and tags manage the presentation layer of the catalog and order fulfillment site, the pipeline manages the processing of the business data. A pipeline configuration file, `pipeline.properties`, contains properties that describe the execution of a series of business methods.

By modifying the `pipeline.properties` file, it is easy to change a business process (such as checking an order status) by adding or removing steps in the pipeline configuration, without any programming.

Figure 1-5 shows the link between the product catalog processing and the order processing package. The diagram illustrates conceptually the Webflow (arrows) and the pipeline that is processing the business data. For instance, it is the pipeline that passes the data about the item(s) the consumer has selected for purchase to the Order Fulfillment services that will process the order.

1 Introduction to the Product Catalog

Figure 1-5 Link Between Catalog and Order Fulfillment



For details about configuring the site's Webflow and pipelines, see [BEA WebLogic Commerce Server Webflow and Pipeline Management](#). For details about the order processing, see [BEA WebLogic Commerce Server Order Processing Package](#). For details about user registration, see [BEA WebLogic Commerce Server Registration and User Processing Package](#).

Using Commerce Server and Personalization Server Features in an Application

You can use WebLogic Commerce Server and WebLogic Personalization Server features in an application. For example, you may want to have a portal as the entry point for your users, but you wish to build e-commerce functionality into your application. For more information, please see the topic [“Using the Catalog Application in a Portal”](#) in the online documentation.

Next Step

We suggest you read Chapter 2, “The Product Catalog Schema,” which explains the structure of the product catalog’s tables in the Commerce database. Understanding the product catalog schema is essential to moving your existing data into the database, or adding new data to the catalog.

1 *Introduction to the Product Catalog*

2 The Product Catalog Schema

This chapter documents the database schema for the WebLogic Commerce Server product catalog. The following topics are covered:

- The Entity-Relation Diagram
- The Catalog Schema Is Based on Dublin Core Standard
- The WLCS_CATEGORY Database Table
- The WLCS_PRODUCT Database Table
- The WLCS_PRODUCT_CATEGORY Database Table
- The WLCS_PRODUCT_KEYWORD Database Table
- The WLCS_CAT_PROP_* Database Tables for Custom Attributes
- The SQL Files and Defined Constraints

The Entity-Relation Diagram

Figure 2-1 shows the Entity-Relation diagram for the WebLogic Commerce Server core product catalog tables in the Commerce database. The data types shown are for an Oracle database. The data types have different syntax for Cloudscape databases, but the type of storage is the same. See the subsequent sections in this chapter for information about the data type syntax for both Oracle and Cloudscape databases.

Figure 2-1 Entity-Relation Diagram for the Core Product Catalog Tables (Shown with Oracle Data Types)

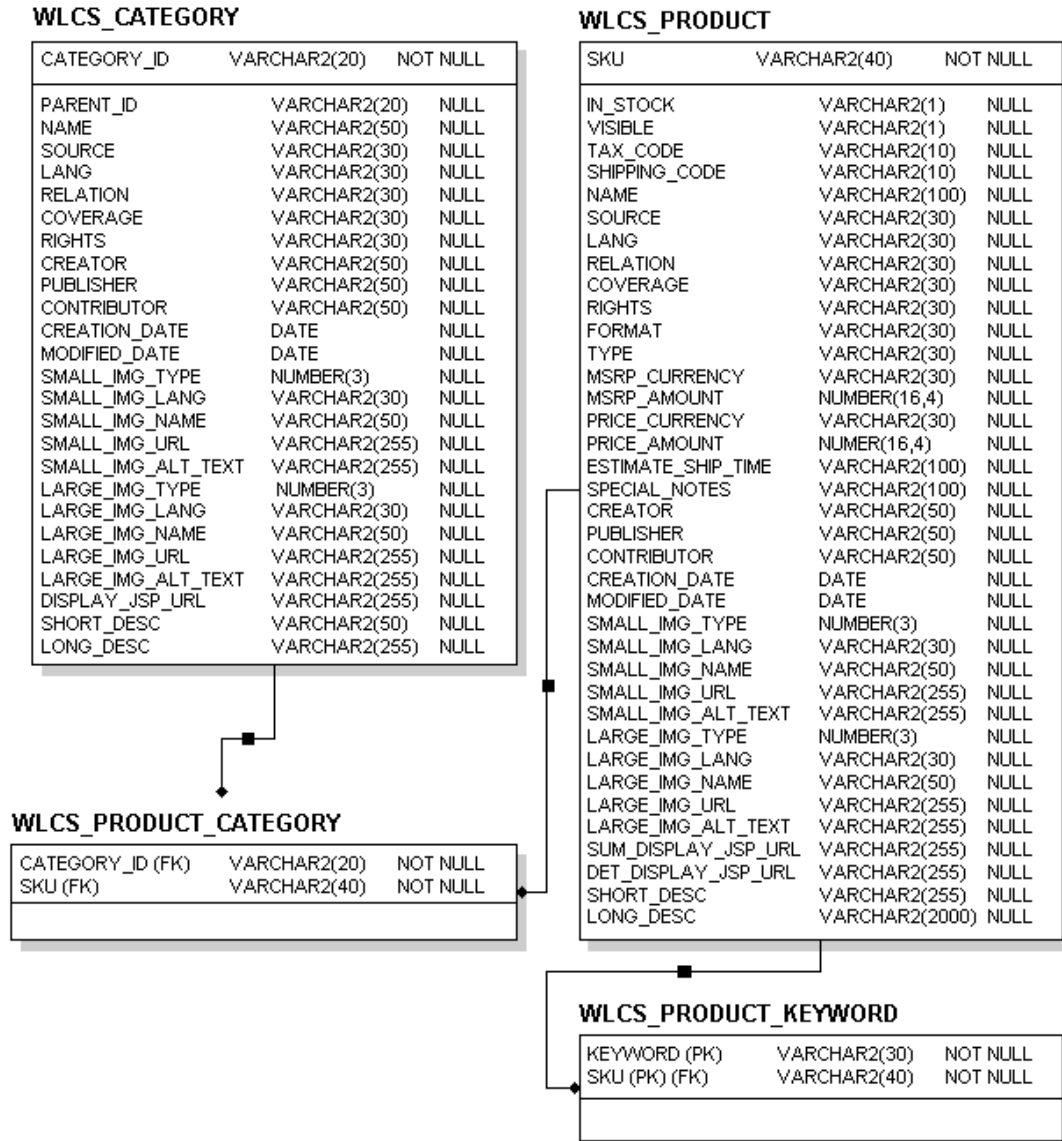


Figure 2-2 shows the database tables that are used to store different types of custom attributes for product items. These custom attributes are optionally defined by a Web site administrator and are implemented as Property Sets. For more information about defining custom attributes, see the section “Define Custom Attributes for Items” on page 4-46.

Figure 2-2 Custom Attribute Tables for Catalog

WLCS_CAT_ENTITY_ID			WLCS_CAT_PROP_ID		
JNDI_HOME_NAME	VARCHAR2(256)	NOT NULL	ENTITY_ID	NUMBER	NOT NULL
PK_STRING	VARCHAR2(480)	NOT NULL	SCOPE_NAME	VARCHAR2(100)	
ENTITY_ID	NUMBER	NOT NULL	PROPERTY_NAME	VARCHAR(100)	NOT NULL
			PROPERTY_TYPE	NUMBER	
			PROPERTY_META_DATA_ID	NUMBER	
			SCHEMA_HAS_CHANGED	NUMBER	
			PROPERTY_ID	NUMBER	NOT NULL

WLCS_CAT_PROP_BOOLEAN			WLCS_CAT_PROP_INTEGER		
PROPERTY_ID	NUMBER	NOT NULL	PROPERTY_ID	NUMBER	NOT NULL
VALUE	NUMBER		VALUE	NUMBER	

WLCS_CAT_PROP_FLOAT			WLCS_CAT_PROP_TEXT		
PROPERTY_ID	NUMBER	NOT NULL	PROPERTY_ID	NUMBER	NOT NULL
VALUE	NUMBER		VALUE	VARCHAR(256)	

WLCS_CAT_PROP_DATE			WLCS_CAT_PROP_USER_DEFINED		
PROPERTY_ID	NUMBER	NOT NULL	PROPERTY_ID	NUMBER	NOT NULL
VALUE	DATE		VALUE	LONG RAW	

Explanations for the columns in the tables are provided in the remainder of this chapter.

The Catalog Schema Is Based on Dublin Core Standard

The metadata for items in WebLogic Commerce Server product catalog are based on the Dublin Core Metadata Open Standard. This standard offers a number of advantages for a Web-based catalog:

- **Simplicity**

The Dublin Core is intended to be usable by non-catalogers as well as resource description specialists. Most of the elements have commonly understood semantics that is roughly the complexity of a library catalog card.

- **Semantic interoperability**

In an Internet environment, disparate description models interfere with the ability to search across discipline boundaries. Promoting a commonly understood set of descriptors that helps to unify other data content standards increases the possibility of semantic interoperability across disciplines.

- **International consensus**

Recognition of the international scope of resource discovery on the Web is critical to the development of effective discovery infrastructure. The Dublin Core benefits from active participation and promotion in some 20 countries in North America, Europe, Australia, and Asia.

- **Extensibility**

The Dublin Core provides an economical alternative to more elaborate description models such as the full MARC cataloging of the library world. Additionally, Dublin Core includes sufficient flexibility and extensibility to encode the structure and more elaborate semantics inherent in richer description standards

- **Metadata modularity on the Web**

The diversity of metadata needs on the Web requires an infrastructure that supports the coexistence of complementary, independently maintained metadata packages. The World Wide Web Consortium (W3C) has begun implementing an architecture for metadata for the Web. The Resource Description Framework, or

RDF, is designed to support the many different metadata needs of vendors and information providers. Representatives of the Dublin Core effort are actively involved in the development of this architecture, bringing the digital library perspective to bear on this important component of the Web infrastructure.

For more information about the Dublin Core Metadata Open Standard, please see <http://purl.org/dc>.

The WLCS_CATEGORY Database Table

Table 2-1 describes the metadata for the WebLogic Commerce Server WLCS_CATEGORY table. This table is used to store categories in the Commerce database. The descriptions shown in the table reflect the “recommended best practice” for the use of that field by the Dublin Core standard.

Table 2-1 WLCS_CATEGORY Table

Column Name	Cloudscape Type	Oracle Type	Description and Recommendations
CATEGORY_ID	VARCHAR (20)	VARCHAR2 (20)	A unique identifier for a category; the primary key for this table. This field cannot be NULL. All other fields in the WLCS_CATEGORY table can be NULL.
PARENT_ID	VARCHAR (20)	VARCHAR2 (20)	The value of the CATEGORY_ID of the parent category in the hierarchy of categories that comprise your product catalog. If this is a top-level user-defined category, the PARENT_ID will be com.beasys.ROOT.
NAME	VARCHAR (50)	VARCHAR2 (50)	The name of the category in the product catalog.

2 The Product Catalog Schema

SOURCE	VARCHAR (30)	VARCHAR2 (30)	A reference to a category from which the present category is derived.
LANG	VARCHAR (30)	VARCHAR2 (30)	A language of the intellectual content of the category. The recommended best practice for the values of the language element is defined by RFC 1766, which includes a two-letter Language Code (taken from the ISO 639 standard), such as: en for English; fr for French, or de for German. The language code can, optionally, be followed by a two-letter Country Code (taken from the ISO 3166 standard [ISO3166]). For example, en-uk for English used in the United Kingdom.
RELATION	VARCHAR (30)	VARCHAR2 (30)	A reference to a related category.
COVERAGE	VARCHAR (30)	VARCHAR2 (30)	The extent or scope of the content of the category.
RIGHTS	VARCHAR (30)	VARCHAR2 (30)	Information about rights held in and over the category.
CREATOR	VARCHAR (50)	VARCHAR2 (50)	An entity primarily responsible for making the content of the category.
PUBLISHER	VARCHAR (50)	VARCHAR2 (50)	An entity responsible for making the category available.
CONTRIBUTOR	VARCHAR (50)	VARCHAR2 (50)	An entity responsible for making contributions to the content of the category.

The WLCS_CATEGORY Database Table

CREATION_DATE	TIMESTAMP	DATE	A date associated with an event in the life cycle of the category. Recommended best practice for encoding the date value is defined in a profile of ISO 8601 and follows the YYYY-MM-DD format.
MODIFIED_DATE	TIMESTAMP	DATE	A date associated with an event in the life cycle of the category, such as an update or insert by the DBLoader program that is provided with WebLogic Commerce Server. The recommended best practice for encoding the date value is defined in a profile of ISO 8601 and follows the YYYY-MM-DD format.
SMALL_IMG_TYPE	INT	NUMBER (3)	A type field of your own design that relates to the graphic. For example, you can implement your own numbering scheme, such as: 0 = display a low resolution graphic for users with low bandwidth 1 = display a high resolution graphic for users with high bandwidth
SMALL_IMG_LANG	VARCHAR (30)	VARCHAR2 (30)	The language of the thumbnail image for the category. For related information, see the description of the LANG column.
SMALL_IMG_NAME	VARCHAR (50)	VARCHAR2 (50)	The name of the thumbnail image for the category.

2 The Product Catalog Schema

SMALL_IMG_URL	VARCHAR (256)	VARCHAR2 (256)	The URL of the thumbnail image for the category.
SMALL_IMG_ALT_TEXT	VARCHAR (256)	VARCHAR2 (256)	The alternate text to display when the user has their cursor over the thumbnail image for the category, or if they have disabled the display of graphics in their browser settings.
LARGE_IMG_TYPE	INT	NUMBER (3)	A type field of your own design that relates to the graphic. For example, you can implement your own numbering scheme, such as: 0 = display a low resolution graphic for users with low bandwidth 1 = display a high resolution graphic for users with high bandwidth
LARGE_IMG_LANG	VARCHAR (30)	VARCHAR2 (30)	The language of the full-size image for the category. For related information, see the description of the LANG column.
LARGE_IMG_NAME	VARCHAR (50)	VARCHAR2 (50)	The name of the full-size image for the category.
LARGE_IMG_URL	VARCHAR (256)	VARCHAR2 (256)	The URL of the full-size image for the category.

The WLCS_PRODUCT Database Table

LARGE_IMG_ALT_TEXT	VARCHAR (256)	VARCHAR2 (256)	The alternate text to display when the user has their cursor over the full-size image for the category, or if they have disabled the display of graphics in their browser settings.
DISPLAY_JSP_URL	VARCHAR (256)	VARCHAR2 (256)	The URL to the JSP used to display the category. For example: <code>/commerce/catalog/includes/category.jsp</code>
SHORT_DESC	VARCHAR (50)	VARCHAR2 (50)	A short description of the content of the category.
LONG_DESC	VARCHAR (2000)	VARCHAR2 (256)	A long description of the content of the category.

See the section “The SQL Files and Defined Constraints” on page 2-21 for information about the constraint defined for this table.

The WLCS_PRODUCT Database Table

Table 2-2 describes the metadata for the WebLogic Commerce Server WLCS_PRODUCT table. This table is used to store item records in the Commerce database. The descriptions shown in the table reflect the “recommended best practice” for the use of that field by the Dublin Core standard.

2 The Product Catalog Schema

Table 2-2 WLCS_PRODUCT Database Table

Column Name	Cloudscape Type	Oracle Type	Description and Recommendations
SKU	VARCHAR (40)	VARCHAR2 (40)	A unique identifier (the “Stock Keeping Unit,” or SKU) for a product item. This field is the table’s primary key and cannot be NULL. All other fields in the WLCS_PRODUCT table can be NULL.
IN_STOCK	CHAR (1)	VARCHAR2 (1)	Provide 1 if in stock, or 0 if out of stock.
VISIBLE	CHAR (1)	VARCHAR2 (1)	Indicates whether the item should be displayed to the user. Enter 1 if visible or 0. If not specified in the database, the default is 1. See the section “Controlling the Visibility of Items in the Catalog” on page 4-22 for important information about this field.
TAX_CODE	VARCHAR (10)	VARCHAR2 (10)	The code used by the Taxware® system to identify the specific tax category to which this item belongs.
SHIPPING_CODE	VARCHAR (10)	VARCHAR2 (10)	The code used by the shipping company for this item.
NAME	VARCHAR (100)	VARCHAR2 (100)	A name given to the product item.
SOURCE	VARCHAR (30)	VARCHAR2 (30)	A reference to another product item from which the present item is derived.

The WLCS_PRODUCT Database Table

LANG	VARCHAR (30)	VARCHAR2 (30)	A language of the intellectual content of the product item. The recommended best practice for the values of the language element is defined by RFC 1766, which includes a two-letter Language Code (taken from the ISO 639 standard), such as: en for English; fr for French, or de for German. The language code can, optionally, be followed by a two-letter Country Code (taken from the ISO 3166 standard). For example, en-uk for English used in the United Kingdom.
RELATION	VARCHAR (30)	VARCHAR2 (30)	A reference to a related product item.
COVERAGE	VARCHAR (30)	VARCHAR2 (30)	The extent or scope of the content of the product item.
RIGHTS	VARCHAR (30)	VARCHAR2 (30)	Information about rights held in and over the item.
FORMAT	VARCHAR (30)	VARCHAR2 (30)	The physical or digital manifestation of the item.
TYPE	VARCHAR (30)	VARCHAR2 (30)	The nature or genre of the content of the item.
MSRP_CURRENCY	VARCHAR (30)	VARCHAR2 (30)	The currency type of the manufacturer's recommended price.
MSRP_AMOUNT	DOUBLE PRECISION	NUMBER (16 , 4)	The manufacturer's recommended price.
PRICE_CURRENCY	VARCHAR (30)	VARCHAR2 (30)	The currency type of our catalog price for this item.

2 The Product Catalog Schema

PRICE_AMOUNT	DOUBLE PRECISION	NUMBER (16 , 4)	Our current price for this item in the catalog.
ESTIMATE_SHIP _TIME	VARCHAR (100)	VARCHAR2 (100)	Inventory: number of days/weeks before the item can be shipped.
SPECIAL_NOTES	VARCHAR (100)	VARCHAR2 (100)	Inventory related message to display with the item.
CREATOR	VARCHAR (50)	VARCHAR2 (50)	An entity primarily responsible for making the content of the product item.
PUBLISHER	VARCHAR (50)	VARCHAR2 (50)	An entity responsible for making the product item available.
CONTRIBUTOR	VARCHAR (50)	VARCHAR2 (50)	An entity responsible for making contributions to the content of the product item.
CREATION_DATE	TIMESTAMP	DATE	A date associated with an event in the life cycle of the product item. The recommended best practice for encoding the date value is defined in a profile of ISO 8601 and follows the YYYY-MM-DD format.
MODIFIED_DATE	TIMESTAMP	DATE	A date associated with an event in the life cycle of the item, such as an update or insert by the DBLoader program that is provided with WebLogic Commerce Server. The recommended best practice for encoding the date value is defined in a profile of ISO 8601 and follows the YYYY-MM-DD format.

The WLCS_PRODUCT Database Table

SMALL_IMG_TYPE	INT	NUMBER (3)	A type field of your own design that relates to the graphic. For example, you can implement your own numbering scheme, such as: 0 = display a low resolution graphic for users with low bandwidth 1 = display a high resolution graphic for users with high bandwidth
SMALL_IMG_LANG	VARCHAR (30)	VARCHAR2 (30)	The language of the thumbnail image for the item. For related information, see the description of the LANG column.
SMALL_IMG_NAME	VARCHAR (50)	VARCHAR2 (50)	The name of the thumbnail image for the item.
SMALL_IMG_URL	VARCHAR (256)	VARCHAR2 (256)	The URL of the thumbnail image for the item.
SMALL_IMG_ALT_TEXT	VARCHAR (256)	VARCHAR2 (256)	The alternate text to display when the user has their cursor over the thumbnail image for the item, or if they have disabled the display of graphics in their browser settings.

2 The Product Catalog Schema

LARGE_IMG_TYPE	INT	NUMBER (3)	A type field of your own design that relates to the graphic. For example, you can implement your own numbering scheme, such as: 0 = display a low resolution graphic for users with low bandwidth 1 = display a high resolution graphic for users with high bandwidth
LARGE_IMG_LANG	VARCHAR (30)	VARCHAR2 (30)	The language of the full-size image for the item. For related information, see the description of the LANG column.
LARGE_IMG_NAME	VARCHAR (50)	VARCHAR2 (50)	The name of the full-size image for the item.
LARGE_IMG_URL	VARCHAR (256)	VARCHAR2 (256)	The URL of the full-size image for the item.
LARGE_IMG_ALT_TEXT	VARCHAR (256)	VARCHAR2 (256)	The alternate text to display when the user has their cursor over the full-size image of the item, or if they have disabled the display of graphics in their browser settings.
SUM_DISPLAY_JSP_URL	VARCHAR (256)	VARCHAR2 (256)	The URL to the JSP used to display the item in summary form. For example: <code>/commerce/catalog/includes/itemssummary.jsp</code>

The WLCS_PRODUCT_CATEGORY Database Table

DET_DISPLAY_JSP_URL	VARCHAR (256)	VARCHAR2 (256)	The URL to the JSP used to display the item in detailed form. For example: /commerce/catalog/includes/itemdetails.jsp
SHORT_DESC	VARCHAR (256)	VARCHAR2 (256)	A short description of the content of the product item.
LONG_DESC	VARCHAR (2000)	VARCHAR2 (2000)	A long description of the content of the product item.

See the section “The SQL Files and Defined Constraints” on page 2-21 for information about the constraint defined for this table.

The WLCS_PRODUCT_CATEGORY Database Table

Table 2-3 describes the metadata for the WebLogic Commerce Server WLCS_PRODUCT_CATEGORY table in the Commerce database. This table is used to join categories and items.

Table 2-3 WLCS_PRODUCT_CATEGORY Database Table

Column Name	Cloudscape Type	Oracle Type	Description
SKU	VARCHAR (40)	VARCHAR2 (40)	A unique identifier (the “Stock Keeping Unit,” or SKU) for an item. NOT NULL.
CATEGORY_ID	VARCHAR (20)	VARCHAR2 (20)	A unique identifier for a category. NOT NULL.

See the section “The SQL Files and Defined Constraints” on page 2-21 for information about the constraint defined for this table.

The WLCS_PRODUCT_KEYWORD Database Table

Table 2-4 describes the metadata for the WebLogic Commerce Server WLCS_PRODUCT_KEYWORD table in the Commerce database. This table stores the keywords that you associate with each product item. The keywords enable rapid retrieval of item records via the Search functions on the Web site's pages or Administration pages.

Table 2-4 WLCS_PRODUCT_KEYWORD Database Table

Column Name	Cloudscape Type	Oracle Type	Description
KEYWORD	VARCHAR (30)	VARCHAR2 (30)	Contains a keyword that you associate with the product item assigned to the unique SKU. NOT NULL. Recommendation: for a given item, select a value from a controlled vocabulary or formal classification scheme implemented in your company.
SKU	VARCHAR (40)	VARCHAR2 (40)	A unique identifier (the "Stock Keeping Unit," or SKU) for an item. NOT NULL.

See the section "The SQL Files and Defined Constraints" on page 2-21 for information about the two constraints defined for this table.

The WLCS_CAT_PROP_* Database Tables for Custom Attributes

This section describes several database tables that are used to store different types of custom attributes for product items. These custom attributes are optionally defined by a Web site administrator and are implemented as Property Sets. For more information about defining custom attributes, see the section “Define Custom Attributes for Items” on page 4-46.

- The WLCS_CAT_ENTITY_ID Database Table
- The WLCS_CAT_PROP_ID Database Table
- The WLCS_CAT_PROP_BOOLEAN Database Table
- The WLCS_CAT_PROP_INTEGER Database Table
- The WLCS_CAT_PROP_FLOAT Database Table
- The WLCS_CAT_PROP_TEXT Database Table
- The WLCS_CAT_PROP_DATETIME Database Table
- The WLCS_CAT_PROP_USER_DEFINED Database Table

The WLCS_CAT_ENTITY_ID Database Table

Table 2-5 describes the metadata for the WebLogic Commerce Server WLCS_CAT_ENTITY_ID table in the Commerce database. This table stores unique identification numbers for configurable entities.

2 The Product Catalog Schema

Table 2-5 WLCS_CAT_ENTITY_ID Database Table

Column Name	Cloudscape Type	Oracle Type	Description
JNDI_HOME_NAME	VARCHAR (256)	VARCHAR2 (256)	The class name for the configurable entity. Either: com.beasys.ebusiness.catalog.ProductCategory or com.beasys.ebusiness.catalog.ProductItem
PK_STRING	VARCHAR (480)	VARCHAR2 (480)	The primary key string for the category or item.
ENTITY_ID	LONGINT	NUMBER	NOT NULL

The WLCS_CAT_PROP_ID Database Table

Table 2-6 describes the metadata for the WebLogic Commerce Server WLCS_CAT_PROP_ID table in the Commerce database. This table stores unique identification numbers for scoped property names that are associated with configurable entities.

Table 2-6 WLCS_CAT_PROP_ID Database Table

Column Name	Cloudscape Type	Oracle Type
ENTITY_ID	LONGINT	NUMBER
SCOPE_NAME	VARCHAR (100)	VARCHAR2 (100)
PROPERTY_NAME	VARCHAR (100)	VARCHAR2 (100)
PROPERTY_TYPE	INT	NUMBER
PROPERTY_META_DATA_ID	LONGINT	NUMBER
SCHEMA_HAS_CHANGED	INT	NUMBER
PROPERTY_ID	LONGINT	NUMBER

The WLCS_CAT_PROP_BOOLEAN Database Table

Table 2-7 describes the metadata for the WebLogic Commerce Server WLCS_CAT_PROP_BOOLEAN table in the Commerce database. This table stores Boolean property values that are associated with configurable entities.

Table 2-7 WLCS_CAT_PROP_BOOLEAN Database Table

Column Name	Cloudscape Type	Oracle Type
PROPERTY_ID	LONGINT	NUMBER
VALUE	INT	NUMBER

The WLCS_CAT_PROP_INTEGER Database Table

Table 2-8 describes the metadata for the WebLogic Commerce Server WLCS_CAT_PROP_INTEGER table in the Commerce database. This table stores integer property values that are associated with configurable entities.

Table 2-8 WLCS_CAT_PROP_INTEGER Database Table

Column Name	Cloudscape Type	Oracle Type
PROPERTY_ID	LONGINT	NUMBER
VALUE	INT	NUMBER

The WLCS_CAT_PROP_FLOAT Database Table

Table 2-9 describes the metadata for the WebLogic Commerce Server WLCS_CAT_PROP_FLOAT table in the Commerce database. This table stores integer property values that are associated with configurable entities.

Table 2-9 WLCS_CAT_PROP_FLOAT Database Table

Column Name	Cloudscape Type	Oracle Type
PROPERTY_ID	LONGINT	NUMBER
VALUE	INT	NUMBER

The WLCS_CAT_PROP_TEXT Database Table

Table 2-10 describes the metadata for the WebLogic Commerce Server WLCS_CAT_PROP_TEXT table in the Commerce database. This table stores text property values that are associated with configurable entities.

Table 2-10 WLCS_CAT_PROP_TEXT Database Table

Column Name	Cloudscape Type	Oracle Type
PROPERTY_ID	LONGINT	NUMBER
VALUE	VARCHAR (256)	VARCHAR (256)

The WLCS_CAT_PROP_DATETIME Database Table

Table 2-11 describes the metadata for the WebLogic Commerce Server WLCS_CAT_PROP_DATETIME table in the Commerce database. This table stores timestamp property values that are associated with configurable entities.

Table 2-11 WLCS_CAT_PROP_DATETIME Database Table

Column Name	Cloudscape Type	Oracle Type
PROPERTY_ID	LONGINT	NUMBER
VALUE	TIMESTAMP	DATE

The WLCS_CAT_PROP_USER_DEFINED Database Table

Table 2-12 describes the metadata for the WebLogic Commerce Server WLCS_CAT_PROP_USER_DEFINED table in the Commerce database. This table stores user-defined (object) property values that are associated with configurable entities.

Table 2-12 WLCS_CAT_PROP_USER_DEFINED Database Table

Column Name	Cloudscape Type	Oracle Type
PROPERTY_ID	LONGINT	NUMBER
VALUE	LONG BIT VARYING	LONG RAW

The SQL Files and Defined Constraints

WebLogic Commerce Server provides four SQL files to create and populate the Cloudscape and Oracle versions of the Commerce 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-catalog-cloudscape.sql
- create-catalog-oracle.sql
- insert-catalog-data-cloudscape.sql
- insert-catalog-data-oracle.sql

You can run the create-* and insert-* procedures 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)
- create-all-cloudscape.sh (UNIX)

2 The Product Catalog Schema

- `create-all-oracle.sql`

In each `create-catalog-*.sql` file, the database tables described earlier in this chapter are created. In addition, the SQL files define constraints. Table 2-13 shows the table name and describes the constraint(s) defined for it.

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

Table 2-13 Constraints Defined on Product Catalog Database Tables

Table Name	Constraints as Defined in <code>create-catalog-oracle.sql</code>
WLCS_CATEGORY	The WLCS_CATEGORY_PARENT_FK constraint enforces the deletion of subcategories when the current category is deleted (Cascading deletes are not supported in Cloudscape, however.) The constraint for the schema in Oracle is: <pre>ALTER TABLE WLCS_CATEGORY ADD CONSTRAINT WLCS_CATEGORY_PARENT_FK1 FOREIGN KEY (PARENT_ID) REFERENCES WLCS_CATEGORY (CATEGORY_ID) ON DELETE CASCADE;</pre>
WLCS_PRODUCT	The WLCS_PROD_ITEM_PK constraint sets the SKU as the primary key, which must be unique and cannot be null. The constraint is: <pre>CONSTRAINT WLCS_PROD_ITEM_PK PRIMARY KEY (SKU)</pre>
WLCS_PRODUCT_CATEGORY	The WLCS_PRODUCT_CATEGORY_PK constraint sets the CATEGORY_ID and SKU fields as the primary keys in the mapping table. For example: <pre>CONSTRAINT WLCS_PRODUCT_CATEGORY_PK PRIMARY KEY (CATEGORY_ID, SKU)</pre> <p>This table is used to join categories with items.</p>

Table 2-13 Constraints Defined on Product Catalog Database Tables

Table Name	Constraints as Defined in create-catalog-oracle.sql
WLCS_PRODUCT_KEYWORD	<p>Two constraints are defined for this table.</p> <p>The first constraint, WLCS_PRODUCT_KEYWORD_PK, sets the KEYWORD and SKU fields as the primary keys. For example:</p> <pre>CONSTRAINT WLCS_PRODUCT_KEYWORD_PK PRIMARY KEY (KEYWORD, SKU)</pre> <p>The second constraint, WLCS_PRODUCT_KEYWORD_FK2, sets the SKU field as a foreign key that references the SKU in the WLCS_PRODUCT table. For example:</p> <pre>CONSTRAINT WLCS_PRODUCT_KEYWORD_FK2 FOREIGN KEY (SKU) REFERENCES WLCS_PRODUCT(SKU)</pre> <p>This table stores the keywords that you associate with each product item. The keywords enable rapid retrieval of item records from the database (or cache) via the Search functions on the Web site's pages or Administration pages.</p>

2 *The Product Catalog Schema*

3 Using the Product Catalog Database Loader

WebLogic Commerce Server provides a DBLoader program that you can use to bulk load data into the product catalog database. While you could use a WebLogic Commerce Server administration screen to add new item or category data, one record at a time, this is impractical when you need to load hundreds or thousands of records. The DBLoader program is also useful if you want to load legacy data from an existing database into the WebLogic Commerce Server database.

You can also use a database vendor's specific loader program such as Oracle SQL*Loader, or a data loader by a third-party company, to populate the product catalog database.

The topic includes the following sections:

- The Input File for DBLoader
- The dbloader.properties File
- Running the DBLoader Program
- DBLoader Log Files
- DBLoader Validations
- Important Database Considerations
- Using Database-specific Data Loaders

- Using Third-party Data Loaders

The Input File for DBLoader

The WebLogic Commerce Server DBLoader program loads data that you provide in a text file into the product catalog database. Data is loaded one table at a time; create a separate input file for each table that you want to update.

The input data file is, by default, a comma-separated value (CSV) text file. The input file has the following structure:

- First row: header containing the table name
- Second row: field names for that table
- Third row: data types for the fields listed on the second row
- Fourth through N row: input data

First Row

The header of the file must identify:

- The number of records to be loaded. DBLoader will use this number as a reference point only. It will process all the records in the file regardless of this indicator.
- The name of the table to be loaded with data in the Catalog database.

For example, the header line might contain:

```
130 , WLCS_PRODUCT
```

Second Row

The second row identifies the table field (column) names into which you are loading data. You must include the primary key field or fields in the input file. Preface each primary key field name with a plus sign (+). Apart from primary keys in tables, all other fields are defaulted as null. Thus, you may omit field names where null is an acceptable value, and specify only those with non-null values.

For example, the second line of the input data file might contain:

```
+SKU,NAME,IN_STOCK,EST_SHIP_TIME,SPECIAL_NOTES,CREATION_DATE
```

Third Row

The third row specifies the data type of each field being loaded. See Chapter 2, “The Product Catalog Schema,” for information about the product catalog schema and the datatypes used.

For example, the third line of the input data file might contain:

```
VARCHAR, VARCHAR, CHAR, VARCHAR, VARCHAR, DATE
```

Notes: On the data type line of the input file, it is not necessary to include the length of the data type, such as `VARCHAR(20)` or `VARCHAR2(20)`. Simply use `VARCHAR` for strings. Use `NUMBER` instead of (for example) `NUMBER(16,4)`. Use `DOUBLE` instead of `DOUBLE PRECISION`.

Fourth Through N Rows

All subsequent lines in the input data file contain the data values. The following is an example of a simple input file:

```
3,WLCS_PRODUCT
+SKU,NAME,IN_STOCK,EST_SHIP_TIME,SPECIAL_NOTES,CREATION_DATE
VARCHAR, VARCHAR, CHAR, VARCHAR, VARCHAR, DATE
P123,CoolKid,N,Out of stock until further notice,Special order
only,02-Oct-2000
P124,FastKid,Y,One week,No special order,02-Oct-2000
P125,RadSneakers,Y,,regular stock,02-Oct-2000
```

Note: `DATE` field values should always be entered in the format `DD-MMM-YYYY`. It cannot be an empty string. Its values are either null or a valid date.

Empty input strings from the data file are inserted into database as empty strings. You must account for each unspecified field in the input record by including the delimiter character (by default, a comma) in the correct position (matching the position of the fields you listed in line 2, the field names). For example:

```
P125,RadSneakers,Y,,regular stock,02-Oct-2000
```

3 Using the Product Catalog Database Loader

In the previous example a value for the fourth identified field (`EST_SHIP_TIME`) was not specified. This condition is fine because this field is not a primary key for the database record. The field's value is stored as an empty string.

Note: If your intention is to store a null value in the database for a non-primary-key field, you should enter `NULL` in the correct position for the field in that record. Do not enclose `NULL` in quotes; enclosing the word `'NULL'` in quotes will cause the field to be stored as a string.

The `dbloader.properties` File

The WebLogic Commerce Server DBLoader program uses a properties file named `dbloader.properties` to decide what driver, database, or login to use.

This file resides in the `WL_COMMERCE_HOME` directory. `WL_COMMERCE_HOME` is the directory where you installed WebLogic Commerce Server.

The `dbloader.properties` file is unrelated to the `weblogic.properties` file, although you can use `weblogic.properties` as an example for specifying JDBC drivers.

Comment lines are prefixed with the `#` character. Both comment lines and blank lines are allowed.

The following table describes the values you can set in this property file.

Property Name	Default Value	Description
<code>jdbcdriver</code>	<code>COM.cloudscape.core.JDBCdriver</code>	Specify which JDBC driver to use to connect to your database. Default driver is Cloudscape JDBC driver that ships with WebLogic. Supported drivers for DBLoader program: <code>COM.cloudscape.core.JDBCdriver</code> <code>oracle.jdbc.driver.OracleDriver</code> <code>weblogic.jdbc.oci.Driver</code> <code>weblogic.jdbc20.oci.Driver</code>

The *dbloader.properties* File

Property Name	Default Value	Description
connection	<code>jdbc:cloudscape:Commerce</code>	Database name where loaded data should go. Commerce is the name of the default database that ships with WebLogic. Location of this database is specified by the system property: <code>cloudscape.system.home</code> .
dblogin	None	The database user name. The default Cloudscape database does not require a user name to be specified. The login name must have read/write privileges on the affected tables.
dbpassword	None	The database user password. The default Cloudscape database does not require a user password to be specified.
delimiter	,	You can change the recognized delimiter character that is used to separate values in the input data file. For example, this might be necessary if you use commas as punctuation in an item's Long Description (LONG_DESC). Choose another character, such as the circumflex (^) as a delimiter.
timestampable	<code>WLCS_CATEGORY,</code> <code>WLCS_PRODUCT</code>	Identifies the product catalog database tables to which DBLoader will track updates (for these two tables). The field name is fixed in the schema provided by WebLogic Commerce Server. However, if you are using DBLoader for other tables (not WLCS tables), you can specify other field names of your own.

3 Using the Product Catalog Database Loader

Property Name	Default Value	Description
timestampfield	MODIFIED_DATE	Specifies the field in the WLCS_CATEGORY and WLCS_PRODUCT tables that identifies the last time this record in the table was modified. The value of the field specified is used by DBLoader to learn when the most recent update was made in each record in the product catalog tables identified in the timestamptable property. The field name is fixed in the schema provided by WebLogic Commerce Server. However, if you are using DBLoader for other tables (not WLCS tables), you can specify other field names of your own
commitTxn	50	Sets how many records are loaded before committing the updates in the database. If the value is less than or equal to one, DBLoader will commit after loading each record.
encoding	Not specified in the dbloader.properties file; therefore, the default is the Java 2 SDK's platform default.	Sets the multibyte character encoding type. The property value supplied can be UCS2 or UTF8. When writing data into and reading data out of the product catalog, Java will transparently convert from the native character encoding used by your systems and Unicode 2.0. There is nothing special that you must do. However, if you need to write/read data to/from the catalog that is encoded differently than your system's native encoding, you will have to explicitly perform the translation. For more information, see the section "Important Database Considerations" on page 3-10.

Listing 3-1 shows a sample dbloader.properties file.

Listing 3-1 Sample dbloader.properties File

```
jdbcdriver=COM.cloudscape.core.JDBCdriver
connection=jdbc:cloudscape:Commerce
```

```
dblogin=none
dbpassword=none
delimiter=,
timestampable=wlcs_category
timestampfield=modified_date
encoding=UTF8
commitTxn=50
```

Running the DBLoader Program

You use the `loaddata` script to run the DBLoader program.

Depending on the platform you are using, the script is in one of the following directories:

- `WL_COMMERCE_HOME\bin\win32`
- `WL_COMMERCE_HOME/bin/unix`

The `loaddata` script performs the following:

- Configures your environment for the duration of execution of this program
- Specifies where to find the data input file
- Launches the DBLoader program

Before you can run the `loaddata` script, make sure that the `set-environment` script specifies the same database as the `dbloader.properties` file. The `set-environment` script resides in the same directory as the `loaddata` script.

For example, if the `dbloader.properties` file uses `'jdbc:cloudscape:Commerce'` connections, then `set-environment` script should have `SET DATABASE=CLOUDSCAPE`.

3 Using the Product Catalog Database Loader

As we mentioned earlier, DBLoader runs independently of WebLogic Commerce Server. Therefore you do not need to stop the server if you are planning to run the loader. However, if you are running Commerce Server with a Cloudscape database, the database itself does not allow more than one connections at a time. In that case, you would need to stop the server.

If you are running Commerce Server with Oracle, then the draw back might be a slower performance for the time the data is being loaded into the database.

Note: You might want to backup the particular tables that you are about to update before running DBLoader. The DBLoader program does not keep history records in the database.

To Run the Program

The command to run the program has the following format:

```
prompt> loaddata { -insert | -update | -delete } input-file.csv
```

On UNIX systems, the loaddata.sh file needs to have its default protections set to include execute privilege. A typical way to do this is with the command:

```
$ chmod +x loaddata.sh
```

You must select one of the three possible operations: `-insert`, `-update`, or `-delete`.

For example:

```
prompt> loaddata -update category.csv
```

In the previous example, the DBLoader program will update rows in the product catalog database that match the primary keys specified in the `category.csv` input file.

To insert, update or delete data in several tables, run the `loaddata` script separately for each table, providing corresponding input file name as a parameter. The order of tables being updated should use the same data integrity rules as all other SQL statements. For example, insert rows into the parent table with the primary key constraint before inserting rows into the child table with the foreign key constraint.

DBLoader Log Files

The WebLogic Commerce Server DBLoader creates two audit trail logs:

- `dbloader.log`
- `dbloader.err`

If these files do not already exist, they are created. Otherwise, the existing audit trails are overwritten by each DBLoader operation. Both files reside in the same directory where you run the `loaddata` script.

The `dbloader.log` file contains the following information:

- The input file name, and the action taken: insert, update, or delete.
- The number of records processed during the load operation.
- The start and end time of the database load processing.

If any errors occurred during the attempted database load operation, the `dbloader.err` file captures the following information:

- The input file name, and the action taken: insert, update, or delete.
- The timestamp when the failure or exception occurred on the record.
- The index of the failed data record in the input file.
- The reason for the failure or exception and actual the input record's values.

DBLoader Validations

The DBLoader program checks the number of fields affected by the load (as specified in the second line of the input data file) against the number of input fields in each record. Because the field delimiter is a comma (by default), this character is not allowed in a string input field. If extra commas are supplied inadvertently, such as punctuation in a `LONG_DESC` (Long Description) field, an error will result and is noted in the `dbloader.err` file. To avoid this type of error, carefully check the number of

commas you are using to separate the input data field values. Or select a different delimiter character and specify it in the `dbloader.properties` file. For more information, see the section “The `dbloader.properties` File.”

All errors and exceptions are displayed in the console where the DBLoader program is running. Records with errors in them will be skipped, and the processing continues until the end of the file. (The program does not roll back a transaction if an error has occurred.)

Important Database Considerations

This section describes some important database considerations that you should keep in mind while using the DBLoader program.

- The schema for the product catalog enforces referential integrity between tables with the use of table constraints; for example, the primary key constraint on `WLCS_PRODUCT` and `WLCS_CATEGORY`, or the foreign key constraint on `WLCS_PRODUCT_CATEGORY`. These constraints determine the order in which data can be inserted, or updated, or deleted from these tables. If you are using Oracle database, you might be less concerned about the order of delete operation because of Oracle’s cascading delete functionality, an option that causes the deletion of a record from the primary table to automatically delete all related records in the related child table or tables.

Because Cloudscape does not provide this functionality, run the `loaddata` script to delete records from child tables before deleting records from primary tables.

For related information, please see Chapter 2, “The Product Catalog Schema.”

- All Strings in Java are represented as a series of Unicode 2.0 characters. Unicode 2.0 is a 16-bit character encoding that supports the world’s major languages. Therefore, when reading text into and writing text out of the JVM, an encoding scheme must be used to convert the “native” encoding used by the operating system to or from Unicode 2.0. Data in text files is automatically converted to Unicode 2.0 when its encoding matches the default file encoding of the Java Virtual Machine (and that of the operating system).

You can identify the default file encoding by checking the System property named `file.encoding`, as follows:

```
System.out.println(System.getProperty("file.encoding"));
```

If the `file.encoding` property differs from the encoding of the text data you want to process, then you must perform the conversion yourself.

Currently, the Java 2 SDK 1.2.2 can convert several files encoding into Unicode 2.0. For details, please see

<http://java.sun.com/products/jdk/1.2/docs/guide/internet/encoding.doc.html>.

What this means to your development group:

- a. When writing data into and reading data out of the Catalog, Java will transparently convert from the native character encoding used by your systems and Unicode 2.0. There is nothing special that you must do.
- b. A conversion can be done only if the encoding is one supported by Java. For details, please see <http://java.sun.com/products/jdk/1.2/docs/guide/internet/encoding.doc.html>.
- c. If you need to write/read data to/from the catalog that is encoded differently than your system's native encoding (for example, if you would like to enter text into a Catalog item description that resides in a file on a machine that has been encoded in UTF8), you will have to explicitly do the translation. This capability is supported by the input process (by allowing you to specify an encoding type in the DBLoader property file), but you must programmatically use the appropriate Java API during the output process.

Normally, you enter or extract data using the default encoding used by your operating system. Therefore, the case shown in item "a" in the previous list is the usual behavior.

Using Database-specific Data Loaders

Most database management systems provide a data loader utility. In the case of Oracle, the data load utility is known as SQL*Loader. This section summarizes the capabilities of SQL*Loader. For details about SQL*Loader, see the Oracle 8i Utilities Guide. An online copy is available at <http://technet.oracle.com/>.

3 Using the Product Catalog Database Loader

The examples used in this section are based on a simple ASCII file containing a few comma separated values (SKU, IN_STOCK, VISIBLE, NAME) taken from a sample WLCS_PRODUCT table, which is described in Chapter 2, “The Product Catalog Schema.”

Note: The example shown in this section does not use all of the columns from WLCS_PRODUCT. Your actual comma-separated values (CSV) file may contain more columns than we show here. We are merely attempting to show you how to conduct the import operation once you have the CSV file ready.

The name we will use for our sample data file is `sample.csv`. The contents of that file might look like the following (based on the columns we mentioned earlier – SKU, IN_STOCK, VISIBLE, NAME).

```
"0,3,4",0,0,"Growing Herbs from Seed, Cutting, and Root"  
"0,2,1",0,0,"The Perfect Storm: A True Story of Men Against the Sea"  
"0,2,2",0,0,"The Worst-Case Scenario Survival Handbook"  
"0,4,0",0,0,"Acute Asthma: Assessment and Management"  
"0,4,1",0,0,"Communications Technology Explained"  
"0,4,2",0,0,"Modern Plastics Handbook"
```

Once you have your data file ready to populate the WLCS_PRODUCT table, you must create a control file which will be used by SQL*Loader. The control file identifies the data file to be read in, how the pieces of information are delimited in the file, and, of course, the actual column locations of the destination table. We will name our control file `sample.ctl`.

```
LOAD DATA  
INFILE 'sample.csv'  
APPEND INTO TABLE WLCS_PRODUCT  
FIELDS TERMINATED BY "," OPTIONALLY ENCLOSED BY '' '  
(SKU, IN_STOCK, VISIBLE, NAME)
```

At a system prompt, invoke SQL*Loader with a command such as:

```
sqlldr userid=bea_systems/bea_systems control=sample.ctl log=sample.log
```

To review the results of your data load, see `sample.log`.

Using Third-party Data Loaders

There are a variety of data loaders available on the market today to assist in the extraction and loading of information. Please be sure to research the use of these tools to ensure success within your environment.

- DataStage by Informix

The Data Movement module of DataStage provides a comprehensive data extraction, transformation, and loading toolset designed for building Operational Data Stores (ODS), data marts and enterprise data warehouses. For more information, please see <http://www.informix.com>.

- PowerConnect by Informatica

PowerConnect is Informatica's family of packaged software products that helps customers easily extract data and metadata from hard-to-access ERP and other legacy applications. This data is then delivered to PowerCenter, Informatica's data integration software hub, which provides robust capabilities for transforming the data and delivering it to downstream data warehouses, data marts and analytic applications. For more information, please see <http://www.informatic.com>.

- Data Junction

Data Junction is a visual design tool for rapidly integrating and transforming data between hundreds of applications and structured data formats. For more information, please see <http://www.datajunction.com>.

- ETI-EXTRACT by ETI

ETI-EXTRACT moves and integrates data across the value chain and multiple business processes. The product automates the writing of programs that retrieve the data needed from any system, transform it and load it into any other system while capturing a complete history of that process. For more information, please see <http://www.eti.com>.

3 *Using the Product Catalog Database Loader*

4 Catalog Administration Tasks

WebLogic Commerce Server provides administrators with command-line scripts, property files, and JSP-based administration screens that you can open in your browser. A common administration screen is provided for WebLogic Commerce Server and WebLogic Personalization Server. The administration functions allow you to manage the behavior and content of your product catalog.

For catalog administration, the tasks include:

- Starting the Server
- Starting the Administration Tool
- Changing the Administrator Password
- Loading Data into the Product Catalog
- Adding Categories to the Catalog
- Adding Items to the Catalog
- Controlling the Visibility of Items in the Catalog
- Assigning Items to Categories
- Edit the Attributes for Categories and Items
- Edit the Availability of an Item
- How Are Categories and Items Displayed to the Web Site User?
- Deleting Items or Removing Items from One or More Categories

4 *Catalog Administration Tasks*

- Removing Categories
- Moving Items from One Category to Another Category
- Define Custom Attributes for Items
- Improving Catalog Performance by Optimizing the Catalog Cache
- Using the `wlcs-catalog.properties` File

Note: In this chapter, the environment variable `WL_COMMERCE_HOME` is used to represent the directory in which you installed the WebLogic Commerce Server software.

Starting the Server

The WebLogic Commerce Server administration tools run as a Web application in the WebLogic Server environment. The server must be running before you can use the administration tools.

When you start the server for a WebLogic Commerce Server application, you are in fact passing WLCS-specific property values (and other required values, such as the location of the Java 2 SDK) to the WebLogic Server. This information gives the server instance the context it needs to provide services to the application.

For administrators, WebLogic Commerce Server provides a `StartCommerce.bat` (Windows) or `StartCommerce.sh` (Unix) script. The files reside in the directory where you installed WebLogic Commerce Server. This directory is pointed to by the `WL_COMMERCE_HOME` environment variable. You can run the `StartCommerce` script from a DOS command prompt (Windows) or a system prompt (Unix).

Another option on Windows systems is to start the server by using the following Start menu path:

Start → Programs → WebLogic Commerce Server 3.1 → Start Commerce Server

This Start menu option invokes the `StartCommerce` script.

The StartCommerce script includes the following command to start the WebLogic Server. (The formatting of the command's parameters has been modified for space reasons.)

```
REM ----- Start WebLogic with the above parameters -----
%JDK_HOME%\bin\java -classic -ms64m -mx128m -classpath %JAVA_CLASSPATH%
-Dweblogic.class.path=%WEBLOGIC_CLASSPATH% -Dweblogic.system.name=%SYSTEM_NAME%
-Dweblogic.system.home=%SYSTEM_HOME% -Dweblogic.home=%WEBLOGIC_HOME%
-Djava.security.manager -Djava.security.policy=%WEBLOGIC_HOME%\weblogic.policy
-Dcommerce.properties=%WL_COMMERCE_HOME%\weblogiccommerce.properties
-Dweblogic.properties=%WL_COMMERCE_HOME%\weblogic.properties
-Dpipeline.properties=%WL_COMMERCE_HOME%\pipeline.properties
-Dwebflow.properties=%WL_COMMERCE_HOME%\webflow.properties weblogic.Server
```

When the script runs, the values that are assigned to the referenced environment variables (read from your PATH) are used. You can see the output in the console listing as the server is starting. For example:

```
C:\jdk1.2.2\bin\java -classic -ms64m -mx128m -classpath
C:\jdk1.2.2\lib\tools.jar;C:\weblogic\lib\weblogic510sp5boot.jar;
C:\weblogic\classes\boot
-Dweblogic.class.path=C:\weblogic\lib\weblogic510sp5.jar;
C:\weblogic\lib\WebLogic_RDBMS.jar;C:\weblogic\license;C:\weblogic\classes;
C:\weblogic\lib\weblogicaux.jar;
C:\weblogicCommerce\lib\weblogic-tags-server.jar;C:\weblogicCommerce\license;
C:\weblogicCommerce\classes;C:\weblogicCommerce\lib\rules.jar;
C:\weblogicCommerce\lib\jruleserviceprovider.jar;
C:\weblogicCommerce\deploy\bmp\classes;
C:\weblogicCommerce\eval\win32\Taxware\classes;
C:\weblogic\eval\cloudscape\lib\cloudscape.jar;
C:\weblogic\eval\cloudscape\lib\tools.jar;
C:\weblogic\eval\cloudscape\lib\client.jar;\xp.jar;\xt.jar;\sax.jar;
-Dweblogic.system.name=server -Dweblogic.system.home=C:\weblogicCommerce
-Dweblogic.home=C:\weblogic -Djava.security.manager
-Djava.security.policy=C:\weblogic\weblogic.policy
-Dcommerce.properties=C:\weblogicCommerce\weblogiccommerce.properties
-Dweblogic.properties=C:\weblogicCommerce\weblogic.properties
-Dpipeline.properties=C:\weblogicCommerce\pipeline.properties
-Dwebflow.properties=C:\weblogicCommerce\webflow.properties weblogic.Server
```

The command passes property values from the following files to the server instance being started:

- Your local copy of `weblogic.properties`, which resides in the `WL_COMMERCE_HOME` directory.

4 Catalog Administration Tasks

- The `weblogiccommerce.properties` file, also in `WL_COMMERCE_HOME`; this file sets properties that are specific to WebLogic Commerce Server, such as the size of the in-memory cache for the catalog's `WLCS_CATEGORY` records.
- The `pipeline.properties` and `webflow.properties` files, in `WL_COMMERCE_HOME`. Each pipeline component defines a set of business logic that execute step-by-step until it is complete, or an exception is thrown. The webflow specifies the sequence of steps to take when the current webflow step (which may be a pipeline invocation) succeeds; if an exception is thrown in that step, the webflow indicates the next step to take based on the exception type, providing additional selectivity in handling the errors. For details, see the document [Webflow and Pipeline Management](#) in the WebLogic Commerce Server online documentation.

Subsequent sections in this chapter include discussions about property values from those files that relate to the administration of the product catalog.

When the `StartCommerce` script completes successfully and the WebLogic Server is running, the command window remains open and contains the following line:

```
<timestamp> <WebLogic Server> WebLogic Server started
```

Starting the Administration Tool

With the server running, as outlined in the previous section, you can start the main administration screen by opening the following URL in your browser:

```
http://localhost:7501/tools
```

The previous URL assumes that you are running the server on your local machine and you want to run administration tasks for the local machine. If the WebLogic Commerce Server application is running on a remote node, you can specify the remote node name to invoke its administration screen. For example, if the remote node is named `blues`, you can use the following URL:

```
http://blues:7501/tools
```

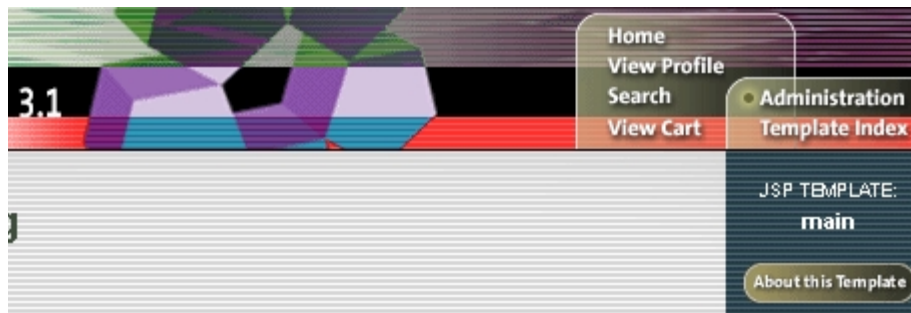
Another option on Windows systems is to start the main administration screen by using the following Start menu path:

Starting the Administration Tool

Start → Programs → WebLogic Commerce Server 3.1 → Administration Tool (Server must be running)

A third option is to click the Administration link that appears in the top banner of most JSP templates. Figure 4-1 shows the link in an extract of the `main.jsp` template.

Figure 4-1 Link to Administration Screens from JSP Template



Before the browser can open the page, you must log into the Administrator account, as shown in Figure 4-2:

Figure 4-2 Administration Login Screen



4 Catalog Administration Tasks

WebLogic Commerce Server defines an Administrator account name for you. The initial password is `password` (lowercase characters). (See the section Changing the Administrator Password for a related discussion.)

Following valid login credentials, the main administration screen is displayed, as shown in Figure 4-3.

Figure 4-3 Main Administration Screen

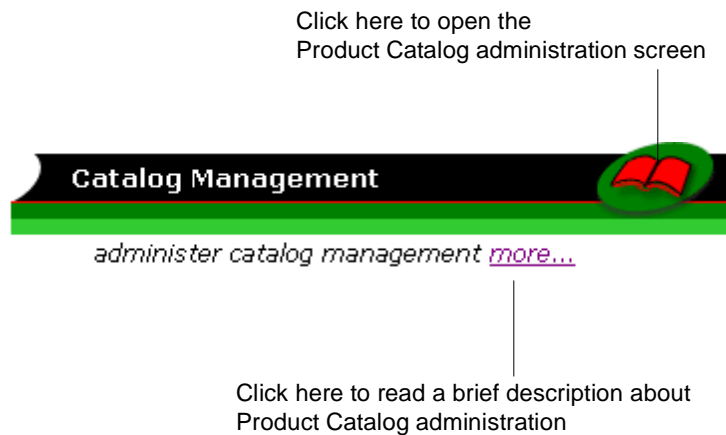


Most of the remaining sections in this chapter describe the tasks you can perform when you click the Catalog Management item on the main administration screen.

Figure 4-4 identifies the two types of links on the Catalog Management graphic.

4-6 Product Catalog Management

Figure 4-4 Links on the Catalog Management Graphic



When you click the icon that looks like a catalog book, the main Catalog Administration screen is displayed, as shown in Figure 4-5.

Figure 4-5 Main Catalog Management Screen

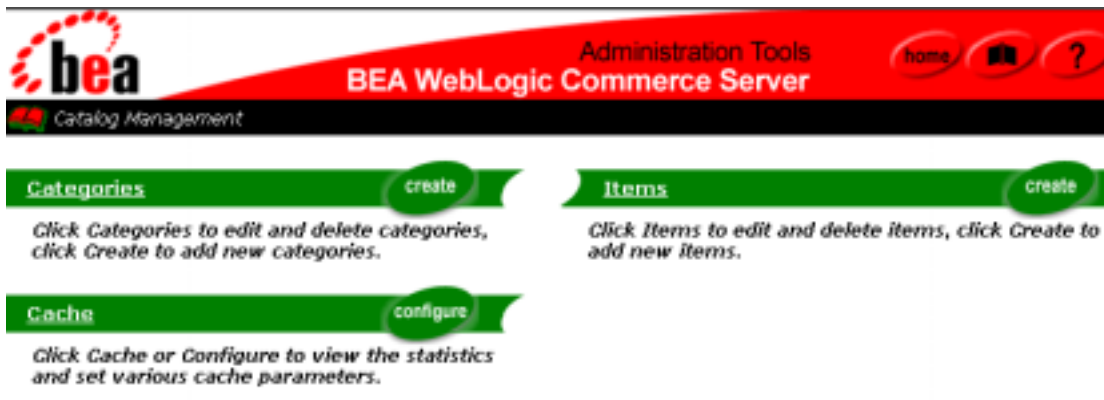


Table 4-1 summarizes the catalog management functional areas.

4 Catalog Administration Tasks

Table 4-1 Catalog Management Functional Areas

Catalog Management Functional Areas	Description
Categories	You can add categories, edit the attributes for categories, move items from one category to another category, and remove categories from the product catalog.
Items	You can add items, edit the attributes for items, add custom attributes for items, , and remove items from the product catalog.
Cache	You can tune the performance of your catalog by adjusting the size and Time To Live (TTL) properties for the separate, in-memory caches defined for categories and items. Balancing the cache to match your peak usage can significantly improve the satisfaction level of customers using your e-commerce Web site.

Changing the Administrator Password

As noted earlier, the initial account information for the administrator account supplied with WebLogic Commerce Server is as follows:

```
Username: administrator  
Password: password
```

To change the password of the administrator account, follow these steps:

1. Open and log into the main Administration tool screen, as described in “Starting the Administration Tool” on page 4-4. The server must be running. One way to start the main administration screen is to enter the URL in your browser:

```
http://localhost:7501/tools
```

2. When you are prompted to login, enter the current password.
3. On the main administration screen, click the User Management graphic, as shown in Figure 4-6:

Figure 4-6 Link to User Management on Main Administration Screen



4. On the User Management screen, click the underlined Users link, as shown in Figure 4-7.

Figure 4-7 Users Link on the User Management Screen



5. On the Users screen, search for the administrator account name, as shown in Figure 4-8.

Figure 4-8 Searching for the Administrator Account



After you enter administrator in the Username input box, click the Search button.

4 Catalog Administration Tasks

6. In the Search Results screen, click the underlined administrator link, as shown in Figure 4-9.

Figure 4-9 Administrator Link in the User Account Search Results

Search Results

Click a title link to edit a user. You can delete a user by clicking its associated Delete icon.

Result for "administrator"

administrator 

Warning: Do not click the red X to the right of the account name. Clicking the red X will delete the account.

7. On the Users: administrator screen, click the Edit button, as shown in Figure 4-10.

Figure 4-10 Edit Button on Users: administrator Screen



8. On the Users: username screen, enter the new password twice, as shown in Figure 4-11.

Figure 4-11 Entering the New Password

The screenshot shows the BEA WebLogic Commerce Server Administration Tools interface. At the top, there is a red header with the BEA logo on the left, the text "Administration Tools" and "BEA WebLogic Commerce Server" in the center, and navigation icons for "home", a book icon, and a question mark on the right. Below the header is a teal bar with the text "Users: Username". The main content area has a grey header with a pencil icon and the text "Edit User information". Below this is a grey box with the instruction "Enter changes to the user information then click Save." The form contains three fields: "Username:" with the value "administrator", "Password:" with a masked input field and a red asterisk, and "Verify Password:" with a masked input field and a red asterisk. At the bottom right of the form are two green buttons labeled "back" and "save".

Warning: Make sure you remember the new password for the administrator account! That sounds obvious, but there is a potential “Catch-22” situation. To change the account password again, you must log into the administration screens, a step that requires the current administrator account. If you forget the account’s password, you would have to recreate the Commerce database.

After you enter the new password, click the Save button. Then click the Home button to return to the main administration screen.

Loading Data into the Product Catalog

WebLogic Commerce Server provides a DBLoader program that you can use to bulk load data into the product catalog database. While you could use a WebLogic Commerce Server administration screen to add new item or category data, one record at a time, this is impractical when you need to load hundreds or thousands of records. The DBLoader program is also useful if you want to load legacy data from an existing database into the WebLogic Commerce Server database.

4 Catalog Administration Tasks

You can also use a database vendor's specific loader program such as Oracle SQL*Loader, or a data loader by a 3rd-party company, to populate the product catalog database.

For details, see Chapter 3, "Using the Product Catalog Database Loader."

Adding Categories to the Catalog

You can add a new category by using either the DBLoader program, as described in Chapter 3, "Using the Product Catalog Database Loader," or by using the administration screens. This section explains how to use the administration screens.

The steps are as follows:

1. Make sure the server is running. See the section "Starting the Server" on page 4-2.
2. Start the Administration tool. See the section "Starting the Administration Tool" on page 4-4.
3. On the main administration screen, click the Catalog Management graphic.
4. On the main Catalog Management screen, click the Create button on the Categories graphic, as shown in Figure 4-12.

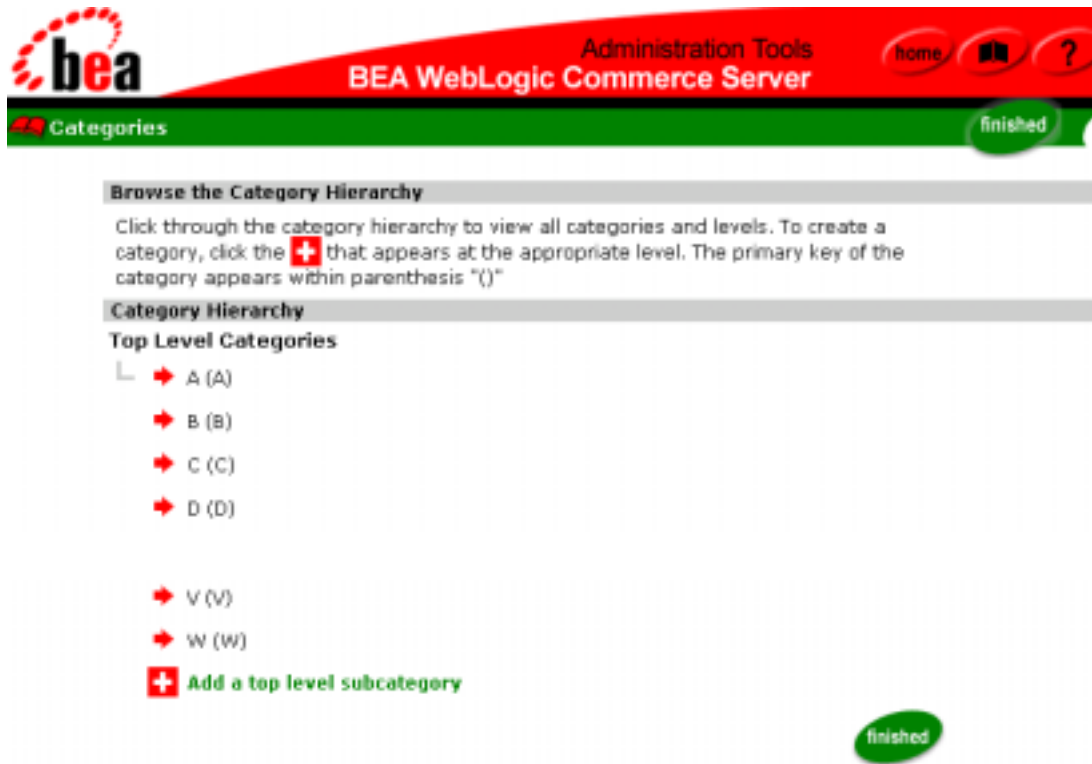
Figure 4-12 Create Button on Categories Graphic



Click Categories to edit and delete categories, click Create to add new categories.

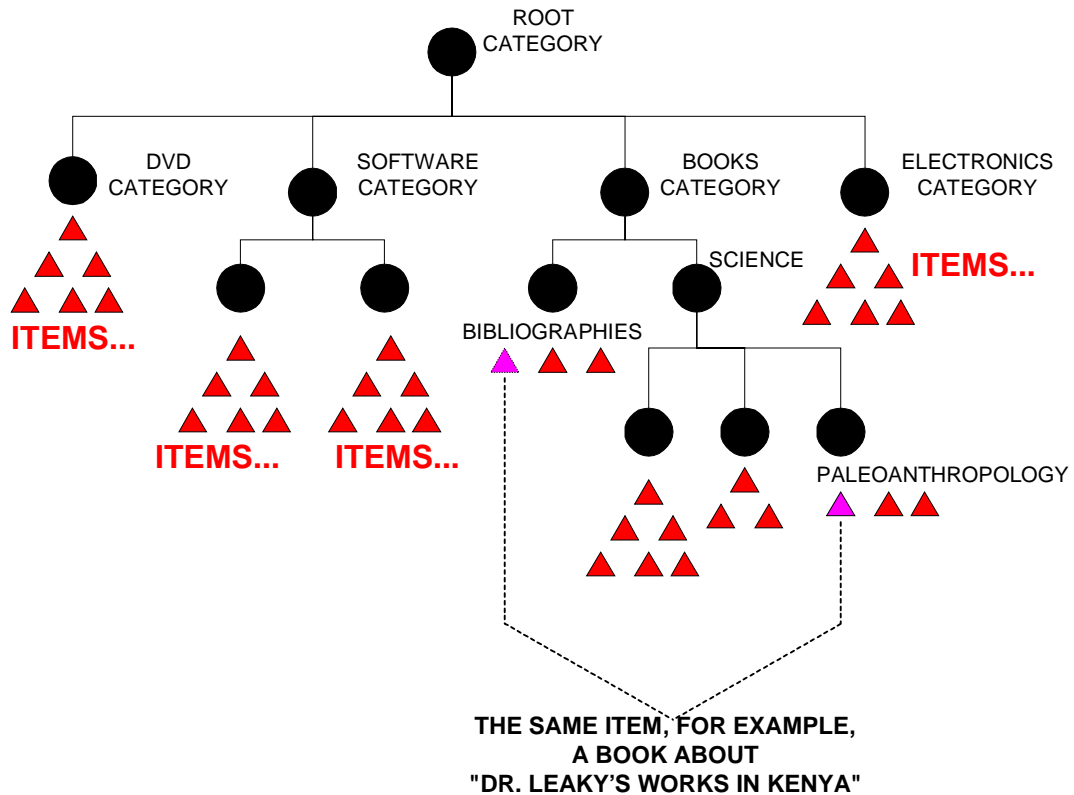
On the Categories screen, you can either add a new category at the current level in the catalog hierarchy, or you can click an existing category name to create a subcategory. Figure 4-13 shows a portion of the Categories screen. To save space in this document, only a subset of the categories defined in the sample data are shown in the figure.

Figure 4-13 Portion of Categories Screen When You Clicked the Create Button



As you will recall from the section “Catalog Hierarchy” on page 1-6, the categories in the product catalog are organized as a hierarchy. It is useful to repeat the diagram that illustrates this concept:

4 Catalog Administration Tasks



An individual item can reside in more than one category. However, categories cannot reside outside of their hierarchical path. In other words, a category that resides in one path of the hierarchy cannot also reside in another path of the hierarchy.

5. To add a new category at the current level in the hierarchy, click the red and white plus sign icon on the Categories screen. If you are on the highest level of the categories hierarchy, the link text next to the plus-sign icon is: "Add a subcategory to top level". For example, assume that you want to add a "Z" category because the electronic store will now offer different types of zithers, which are a type of stringed musical instruments. After you create the category Z, you will then create a "Zithers" subcategory.

- Figure 4-14 and Figure 4-15 show the top portion and bottom portion of the Create New SubCategory screen, after you click the plus-sign icon. (The screen is split into two figures for space reasons only.)

Figure 4-14 Top Portion of the Create New SubCategory Screen

Figure 4-15 Bottom Portion of the Create New SubCategory Screen

Image Properties Associated with the Category

	Image Name	Image Type (Number)	Image URL	Alternate Text	Language
Small Image	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
Large Image	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>

- Enter the required fields, which are indicated by the asterisk (*). The required field are:

4 Catalog Administration Tasks

- Category Identifier (also the primary key of the WLCS_CATEGORY table in the Commerce database). In this simple example, enter Z.
- Category Name. In this simple example, enter Z.
- Short Description. Enter a short description such as “Contains subcategories that start with Z.”

You have the option of leaving the other fields blank for now because they are not required fields or primary keys in the WLCS_CATEGORY table. For more information about the category fields, see the section “The WLCS_CATEGORY Database Table” on page 2-5.

8. After you enter the field values, click the Create button to create the new category. (Or click the Back button on the screen to cancel the current create category operation.)
9. When the new category has been created, a confirmation message is displayed on the refreshed Categories screen. Figure 4-16 shows a portion of a sample screen after we added the Z category under the hierarchical catalog’s top-level root:

Figure 4-16 Confirmation Screen After New Category ‘Z’ Created

The screenshot shows the BEA WebLogic Commerce Server Administration Tools interface. At the top, there is a red header with the BEA logo on the left, "Administration Tools" and "BEA WebLogic Commerce Server" in the center, and "home", "back", and "help" buttons on the right. Below the header, a green bar contains the word "Categories". A red message box displays "Category : Z created successfully." Below this, a grey bar contains the title "Create New Subcategory of Category : 'ROOT'" and the instruction "Enter the appropriate information then click Create. Fields marked * are required." The form contains three fields: "Category Identifier*" with the value "Z", "Category Name*" with the value "Z", and "Short Description*" with the value "Contains subcategories that start with Z". Each field has a small help icon to its right.

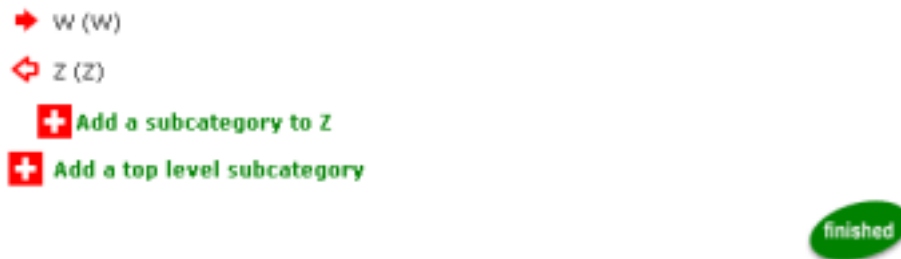
10. When you click the Back button near the bottom of the create category screen (not the browser’s Back button), the Category screen is refreshed to include the category you just added, as shown in Figure 4-17.

Figure 4-17 Newly Added Category Near Bottom of Category Screen



11. To create subcategories in an existing category, first click the red arrow to the left of the category name. For example, assume that you want to add the Zithers category under the Z category. To do this, click the red right-arrow next to the Z category. Figure 4-18 shows how the updated screen looks at this point.

Figure 4-18 Updated Screen While Adding a Subcategory



The icon next to the Z category changes to a red, hollow, left-facing arrow. To proceed, click the plus-sign icon that appears indented below the category. In this example, you can click the plus-sign icon next to the text, Add to Subcategory to Z.

12. Figure 4-19 shows the top portion of the next screen: Create New Subcategory below Category : 'Z'. On this screen, we have just entered data into the first three fields.

Figure 4-19 Creating a New Subcategory

The screenshot shows the BEA WebLogic Commerce Server Administration Tools interface. The top navigation bar includes the BEA logo, the text 'Administration Tools BEA WebLogic Commerce Server', and buttons for 'home', a search icon, and a help icon. Below this is a green 'Categories' header. The main content area is titled 'Create New Subcategory of Category : 'Z'' and includes a note: 'Enter the appropriate information then click Create. Fields marked * are required.' The form contains the following fields:

Category Identifier*:	<input type="text" value="Z30-40"/>	Unique category identifier.
Category Name*:	<input type="text" value="Zithers"/>	Name of the category.
Short Description*:	<input type="text" value="Musical instruments, flat box, 30-40 strings"/>	Short description associated with
Long Description:	<input type="text" value="Zithers are musical instruments constructed of a flat sounding box with about 30 to 40 strings"/>	Long description of the category.

Click the Create button to create the new category Zithers under the category Z.

The process for creating other categories in the catalog hierarchy is the same as outlined in previous steps.

Adding Items to the Catalog

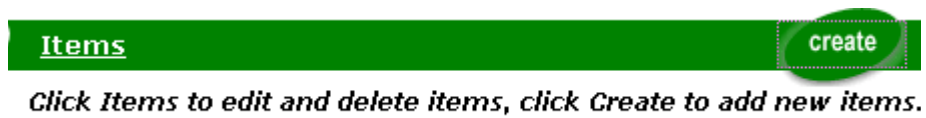
You can add one or more product items to an existing category by using either the DBLoader program, as described in Chapter 3, “Using the Product Catalog Database Loader,” or by using the administration screens. This section explains how to use the administration screens. After the catalog’s categories and items are stored in the database, you will perform a separate set of steps to assign items to categories, as described in the section “Assigning Items to Categories” on page 4-23.

The steps are as follows:

1. Make sure the server is running. See the section “Starting the Server” on page 4-2.
2. Start the Administration tool. See the section “Starting the Administration Tool” on page 4-4.

3. On the main administration screen, click the Catalog Management graphic.
4. On the main Catalog Management screen, click the Create button on the Items graphic, as shown in Figure 4-20.

Figure 4-20 Create Button on Items Graphic



5. The Create New Item screen is displayed. Figure 4-21 and Figure 4-22 shows the top portion and bottom portion of the screen.

Figure 4-21 Top Portion of the Create New Item Screen

SKU*:	<input type="text"/>	The Stock Keeping Unit for the item.
Item Name*:	<input type="text"/>	The name of the item.
Short Description*:	<input type="text"/>	Short description associated with the item.
Visible:	<input type="checkbox"/>	Is the item displayed during catalog browsing
Long Description:	<input type="text"/>	Long description of the item.
Price Amount:	<input type="text"/>	The selling price of the item.
Price Currency:	<input type="text"/>	The currency for the selling price of the item.
MSRP Amount:	<input type="text"/>	The manufacturer suggested retail price of this item.
MSRP Currency:	<input type="text"/>	The currency for the msrp of this item.
Tax Code:	<input type="text"/>	The Tax Code for the item.
Shipping Code:	<input type="text"/>	The Shipping Code for the item.
Summary JSP URL:	<input type="text"/>	The jsp associated with the summary display of this item.
Detail JSP URL:	<input type="text"/>	The jsp associated with the detailed view of this item.
Type:	<input type="text"/>	The nature of the content of the item.
Format:	<input type="text"/>	The physical or digital manifestation of the item.
Creator:	<input type="text"/>	The person who created this item.
Publisher:	<input type="text"/>	The publisher of the item.
Contributor:	<input type="text"/>	The entity responsible for contributing to the item
Creation Date:	<input type="text"/>	Dates are generally associated with the creation of the resource.

Figure 4-22 Bottom Portion of Create New Item Screen

Modified Date:

Language:

Relation:

Rights:

Coverage:

Source:

Keywords for the Item

Keywords:

Images for the Item

	Image Name	Image Type (Number)	Image URL	Alternate Text	Language
Small Image	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
Large Image	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>

back create

6. Edit the product item fields. The required fields, indicated with an asterisk (*) next to the field name, are:

- SKU, an acronym for Stock Keeping Unit. This is the unique identifier for the item in the catalog. It is the primary key of the WLCS_PRODUCT table in the Commerce database.
- Item Name
- Short Description

You have the option of leaving the other fields blank for now because they are not required fields or primary keys in the WLCS_PRODUCT table. For more information about the product item fields, see the section “The WLCS_PRODUCT Database Table” on page 2-9.

Note: See the section “Controlling the Visibility of Items in the Catalog” on page 4-22 for important information about the Visible checkbox.

7. For example, assume that you need to add a new product item, a “Yakaha 30-String Zither.” Figure 4-23 shows the top portion of the screen as it appears when information is being added about this item.

Figure 4-23 Adding a New Product Item

Create a New Item		
Enter the appropriate information then click Create.		
SKU*:	<input type="text" value="YAK-Z30S"/>	The Stock Keeping Unit for
Item Name*:	<input type="text" value="Yakaha 30-String Zither"/>	The name of the item.
Short Description*:	<input type="text" value="30-string Zither from Yakaha!"/>	Short description associate
Visible:	<input checked="" type="checkbox"/>	Include or exclude the item
Long Description:	<input type="text" value="For aficionados of zithers, this 30-string flat-box beauty is unparalleled in"/>	Long description of the item
Price Amount:	<input type="text" value="299.95"/>	The selling price of the item
Price Currency:	<input type="text" value="USD"/>	The currency for the selling
MSRP Amount:	<input type="text" value="329.99"/>	The manufacturer suggests
MSRP Currency:	<input type="text" value="USD"/>	The currency for the msrp o

When you add an item, make sure you add a full set of keywords that describe the item. Doing so will make it easier for the Web site users to find items in the catalog via keyword-based searches. Adding lots of descriptive keywords will also make it easier for you to find items in the administration screens when you want to assign them to one or more categories. You can always remove keywords later if too many keywords are returning spurious results in keyword searches.

For example, Figure 4-24 shows the portion of the Create a New Item screen where you can enter keywords. Type the each keyword into the left-side text box, and click the right arrow to add it to the list. If necessary, highlight an unwanted keyword in the right-side list and then click the left arrow to remove it from the list.

Figure 4-24 Entering Keywords on the Create a New Item Screen



Note: By default, all keywords that are entered on this administration screen are converted to lowercase characters. Because the search engine is case sensitive, this consistent conversion means that your subsequent searches for items via keyword searches should use lowercase characters. You can enter keywords in mixed case or ALL CAPS, but the keywords are stored in lowercase, by default. This behavior is set in the `wlcs-catalog.properties` file. For more information, see the section “Using the `wlcs-catalog.properties` File” on page 4-55.

8. After you enter the field values, click the Create button to create the new item. (Or click the Back button on the screen to cancel the create item operation.)
9. When the new item has been created, a confirmation message is displayed on the refreshed Item screen: `The item was created.`

Controlling the Visibility of Items in the Catalog

The visibility attribute of items in the product catalog is currently only applied to the results of keyword searches and query-based searches. Thus if users of your deployed Web site are **browsing** for an item (by clicking through the catalog’s hierarchy of categories and subcategories), by default they will see the item even if you left the Visible checkbox unchecked.

To make an item invisible to the end-users of the catalog, in the current release you must:

- Mark the item as invisible (set or leave the Visible checkbox unchecked)
- Remove the item from all categories (orphan the item)

An **uncategorized item**, also known as an **orphaned item**, is one that has been removed from all categories, but still resides in the `WLCS_PRODUCT` table in the Commerce database.

For searches by Web site users of the catalog, the default end-user `CatalogRequest` object has the `showAll` attribute set to `false`; thus invisible items are not displayed. In contrast the default administrator's `CatalogRequest` object has the `showAll` attribute set to `true`, ensuring that administrators can see invisible items when they perform search operations via the Administration screens.

Assigning Items to Categories

Assuming that you have already entered categories and items in the Commerce database, either through a bulk loader program like `DBLoader` or through the administration screens, you can assign each item to one or more categories. This section shows how to use a WebLogic Commerce Server administration screen to make the assignments.

What if I Have a Large Amount of Data?

If you have a catalog with hundreds of categories and thousands or tens of thousands of items, making the assignments via the administration screen is obviously not practical. An alternative is to use the `DBLoader`. Create a text data input file to populate the `WLCS_PRODUCT_CATEGORY` table in the database. As illustrated in Figure 2-1 (the Entity-Relation diagram for the Catalog tables), the `WLCS_PRODUCT_CATEGORY` table maps the `CATEGORY_ID` primary key from `WLCS_CATEGORY` table to the `SKU` primary key for the `WLCS_PRODUCT` table.

For related information, also see:

- “The `WLCS_PRODUCT_CATEGORY` Database Table” on page 2-15.
- Chapter 3, “Using the Product Catalog Database Loader.”

Using the Administration Screens to Assign Items to Categories

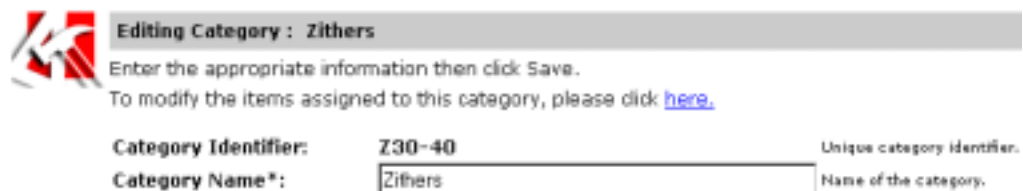
The steps to assign items to categories in the administration screens are as follows:

1. Make sure the server is running. See the section “Starting the Server” on page 4-2.
2. Start the Administration tool. See the section “Starting the Administration Tool” on page 4-4.
3. On the main administration screen, click the Catalog Management graphic.
4. On the main Catalog Management screen, click the underlined Categories link.
5. In the Catalog hierarchy display, click the category or subcategory into which you want to add the item.

Note: To expand a current category and reveal its subcategories (if any), make sure you click the red arrow to the left of a category name. If you instead click the current category’s underlined name, you are taken to its Edit Category screen.

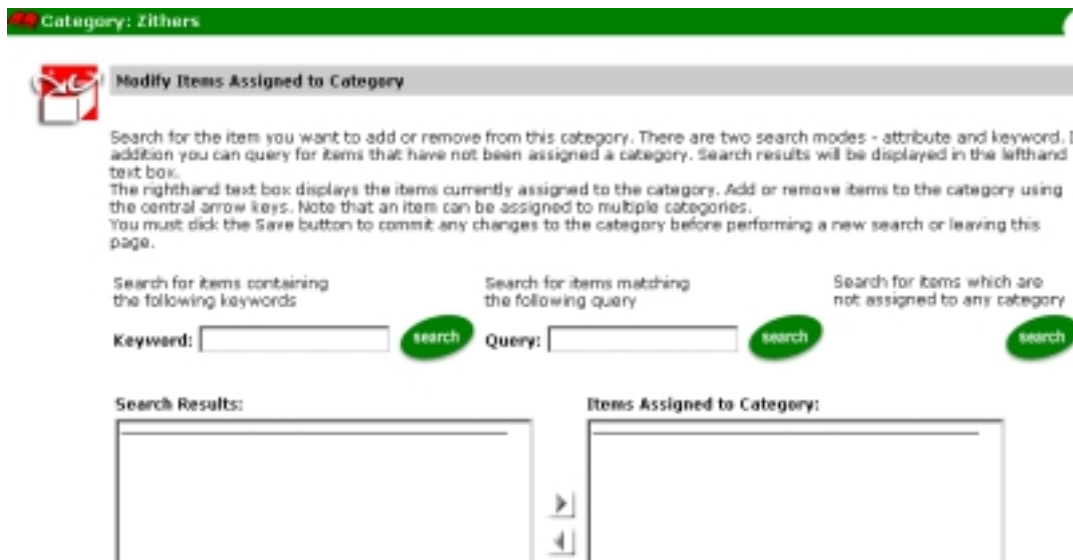
6. When the category or subcategory that you want to add items to is shown in the hierarchy, click its underlined link. For example, assume that you clicked the red arrow next to the Z category, then clicked the underlined Zithers (Z30-40) link. (These sample categories were added to the catalog in the previous section, “Adding Categories to the Catalog” on page 4-12.)
7. Figure 4-25 shows the top portion of a sample Editing Category: Zithers screen. Notice the link near the top of the display: To modify the items assigned to this category, click [here](#).

Figure 4-25 Sample Editing Category Screen with Modify Link



- Click the link titled, To modify the items assigned to this category, click [here](#). When you do, a screen is displayed that is similar to the one shown in Figure 4-26. In this particular screen, there are currently no items assigned to the Zithers category.

Figure 4-26 Modify Items Assigned to Category Screen



- The Items Assigned to Category textbox shows the items that are already in this category. (In this example so far, there are no items in the Zithers category.) You can search for the item you want to add or remove. There are three modes of search: keyword-based search, query-based searches, or orphaned-items search (“Uncategorized Items”) search. The search results are displayed on the left side text box. To add an item to the category, you can move the item to the right-side textbox by clicking on the right arrow.

Warning: You must click the Save button to commit any changes to the category before performing a new search or leaving this page.

- For example, assume that you need to assign the Yakaha 30-String Zither to the Zithers category. (This item was added to the sample data in the catalog in the section “Adding Items to the Catalog” on page 4-18.) You can search for the keywords that we entered for the item earlier in this chapter. Figure 4-27 shows a

4 Catalog Administration Tasks

sample Modify Items Assigned to Category screen where the instrument keyword was entered, the search results found the item, and the results were displayed in the left-side Search Results box.

Figure 4-27 Sample Keyword Search Results on Modify Items Assigned to Category Screen

The screenshot shows three search options at the top: "Search for items containing the following keywords" with a "Keyword:" field containing "instrument" and a "search" button; "Search for items matching the following query" with a "Query:" field and a "search" button; and "Search for items which are not assigned to any category" with a "search" button. Below these are two textboxes: "Search Results:" containing "(YAK-Z385) Yakaha 38-String Zither" and "Items Assigned to Category:" which is empty. A right arrow button is positioned between the two textboxes, and a left arrow button is below it.

11. To add one or more items to the current category, select them in the left-side textbox and then click the right-arrow button. This step will move the items to the right-wide textbox. If you moved items, remember to click the Save button near the bottom of the page before you perform another search or leave this page. If no errors occur after you click the Save button, WebLogic Commerce Server displays a confirmation message. In this example, the message was: Category 'Zithers' has been updated.
12. To find items, via the query-based search, that you want to add to the category, you can use an expression such as the following examples:

```
price > 100 && price <= 300
name like 'Y*'
!(price > 100) || (msrpAmount >= 300)
modifiedDate < now
```

When the search results are displayed in the left-side textbox, use the right-arrow button to move the appropriate items into the right-side Category Items textbox. If you move items from the left textbox to the right textbox, or from the right textbox to the left textbox, remember to click the Save button near the bottom of the page before you perform another search or leave this page. For details about the query-based search syntax, see the section "Query-based Search Syntax" on page 5-74.

13. To find a product item that is no longer associated with any categories in the catalog, but the item is still in the catalog, click the underlined Uncategorized Items link.
14. Again, to add one or more items (that you located via one of the three search options) to the current category, select them in the left-side textbox and then click the right-arrow button. This step will move the items to the right-wide textbox. If you moved items, remember to click the Save button near the bottom of the page before you perform another search or leave this page.

Edit the Attributes for Categories and Items

You can use the catalog administration screens to edit the attributes for existing categories and items.

Editing Category Attributes

To edit the attributes for a category, follow these steps:

1. Make sure the server is running. See the section “Starting the Server” on page 4-2.
2. Start the Administration tool. See the section “Starting the Administration Tool” on page 4-4.
3. On the main administration screen, click the Catalog Management graphic.
4. On the main Catalog Management screen, click the underlined Categories button on the Categories graphic.
5. On the main Categories screen, you can find the category you want to modify by either:
 - Entering its category identifier in the search input box. In the sample data that comes with the catalog tables in the Commerce database, the category identifier assigned to each alphabetized main category is the same letter as the category. However, a number is assigned to each of the subcategories. Figure 4-28 shows a portion of the resulting Categories screen after we

4 Catalog Administration Tasks

entered the category identifier 149 in the search field and clicked the Search button. In the figure, notice how the found category name is shown in red type.

Figure 4-28 Sample Category Search Result

Search for Categories

Click through the hierarchy to find a category, or search for the category by its primary key shown in parenthesis. Click the category title to edit it, or click **X** to delete the category.

Warning: deleting a category will also delete its subcategories and cannot be undone.

Category Identifier:

search

Category Hierarchy

Top Level Categories

- ↳ [A\(A\)](#) X
- ↳ [B\(B\)](#) X
- ↳ [C\(C\)](#) X
- ↳ [Caddies \(144\)](#) X
- ↳ [Cans \(145\)](#) X
- ↳ [Chargers \(149\)](#) X
- ↳ [Chests \(150\)](#) X

- Or you can find the category you want to modify by simply browsing through the hierarchy of categories and subcategories. You can click the solid, red, right-facing arrows to expand a category and view its subcategories.
6. When the category you want to modify is displayed, click the name of the category. For example, Figure 4-29 shows a portion of the resulting screen when we clicked the Chargers (149) category.

Figure 4-29 Editing Category: Chargers Sample Screen

Editing Category : Chargers

Enter the appropriate information then click Save.
To modify the items assigned to this category, please click [here](#).

Category Identifier: 149 Unique category identifier.

Category Name*: Chargers Name of the category.

Short Description*: Short description associated with

Long Description: Long description of the category.

Display JSP URL: /commerce/catalog/includes/category.jsp JSP associated with the display c

7. On this screen, you can add, change, or remove the category attributes. See the screen itself online for the full set of attributes that can be modified. Also see “The WLCS_CATEGORY Database Table” on page 2-5 for a description of the fields.

Warning: By design, you cannot modify the category’s unique identifier. If you had to change a category identifier, you would have to delete the category itself and then create a new category with a new, unique identifier. But be careful! As noted on the administration screen, removing a category removes it and all of its subcategories. In Oracle databases, this feature is known as a cascading delete operation. This feature is currently not supported in Cloudscape.

Also, product items that are associated with removed categories may be orphaned (unless they belong to another category in the hierarchy). Orphaned items are allowed to remain in the catalog, and can be reassigned later to one or more categories. The caveat here is that by deleting a category, you may be inadvertently removing many subcategories from the catalog. Check the hierarchy carefully before clicking the red X next to a category name.

8. After you edit the attributes for the category, click the Save button to commit your changes to the database. Or, to exit the screen without committing your changes, click the Back button.

Editing Product Item Attributes

The steps to edit the attributes for a product item are as follows:


1. Make sure the server is running. See the section “Starting the Server” on page 4-2.
2. Start the Administration tool. See the section “Starting the Administration Tool” on page 4-4.
3. On the main administration screen, click the Catalog Management graphic.
4. On the main Catalog Management screen, click the underlined Items button on the Items graphic.
5. On the Item Search screen, you can find the product item by either entering one of its keywords, or by entering an query-based search expression, or by searching for orphaned items. For example, Figure 4-30 shows a portion of the resulting screen after we entered the keyword `hammer`. The list of matching items show the value for each item’s Name field.

Figure 4-30 Sample Keyword Search Results for Product Items

Item Search

Search for the items you would like to edit. There are query for items that have not been assigned a category delete the item.
Warning: deleting an item is a permanent operation

Search for items containing the following keywords

Keyword: 

Search Results [1-10](#) [Next](#)

1. [drill-9-27870](#) ✖
2. [drill-9-27499](#) ✖
3. [drill-9-27205](#) ✖
4. [drill-9-27808](#) ✖
5. [drill-9-27718](#) ✖
6. [hammer-9-18864](#) ✖
7. [hammer-9-18915](#) ✖
8. [hammer-71-UF30588](#) ✖
9. [hammer-9-18894](#) ✖
10. [wrench-9-MIRN7125](#) ✖

6. Another search option on the Item Search screen is entering an query-based search expression such as one of the following:

```
price > 10 && price <= 50  
name like 'L*'  
!(price > 20) || (msrpAmount >= 30)  
modifiedDate < now
```

If more than 10 search results have been returned, you can click the underlined Next button or Previous button (if any) to read through the list. For details

4 Catalog Administration Tasks

about the query-based search syntax, see the section “Query-based Search Syntax” on page 5-74.

Figure 4-31 shows the results of a search for `price > 10 && price <= 50`.

Figure 4-31 Sample Query-based Search to Find a Product Item

Search for items containing the following keywords

Keyword:

Search for items matching the following query

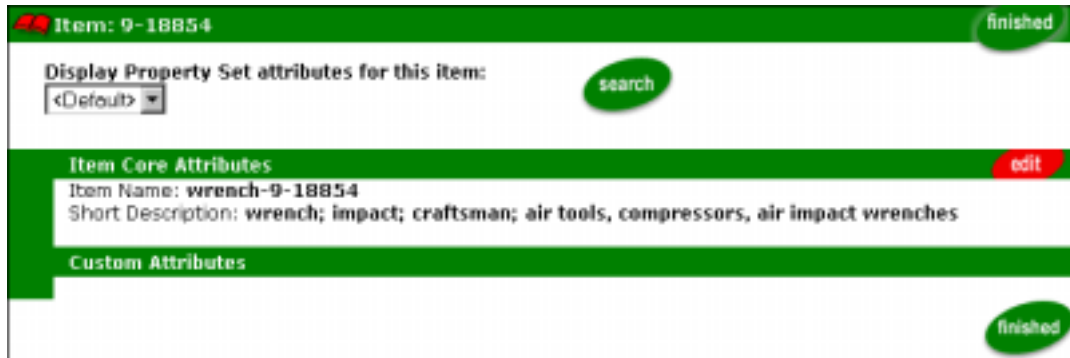
Query:

Search Results [1-10](#) [Next](#)

1. [wrench-9-18854](#) ✖
2. [hammer-9-18915](#) ✖
3. [converter-9-WTTC253A](#) ✖
4. [wipe-9-KC03046](#) ✖
5. [extension-9-WF8417](#) ✖
6. [cutter-9-RG31642](#) ✖
7. [abrasive-9-NT27984](#) ✖
8. [wrench-9-MI8812](#) ✖
9. [manifold-9-FB40133](#) ✖
10. [wrench-9-MI8811](#) ✖

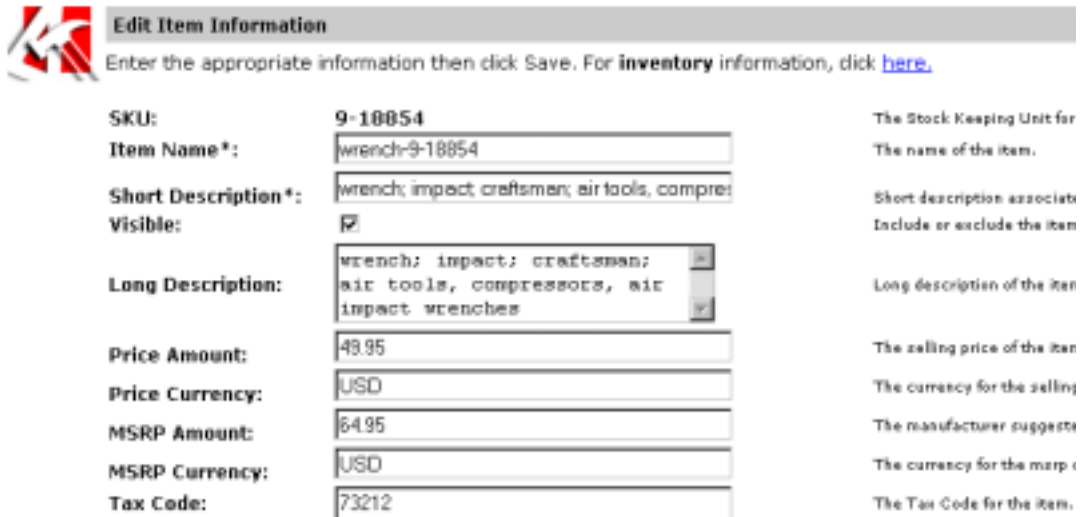
7. After you find the item you want to modify, click the underlined item name to edit its attributes. Figure 4-32 shows the resulting screen when we click the search result for `wrench-9-18854`.

Figure 4-32 Sample Initial Edit Item Attributes Screen



8. Click the red and white Edit button. Figure 4-33 shows a portion of a resulting Edit Item Information screen. (Notice how the price is indeed greater than 10 and less than or equal to 50, from the prior search that returned this as a matching item.)

Figure 4-33 Sample Edit Item Information Screen



9. On this screen, you can add, change, or remove the item's attributes. For example, you can change the item's price, short and long descriptions, its basic inventory setting, and its visibility in the catalog. See an Edit Item Information screen online for the full set of attributes that can be modified. See the section "The WLCS_PRODUCT Database Table" on page 2-9 for details about the item's fields. Also see the section "Controlling the Visibility of Items in the Catalog" on page 4-22 for important related information.
10. Before you save any changes on this Edit Item Information screen, you can view or change the item's inventory setting on a related screen. For information about this setting, see the next section, "Edit the Availability of an Item."
11. After you edit the attributes for the item, click the Save button to commit your changes to the database. Or, to exit the screen without committing your changes, click the Back button. If no errors occur when you save your changes, WebLogic Commerce Server displays the message "The item was updated successfully".

Edit the Availability of an Item

In the current release, WebLogic Commerce Server provides a simple inventory setting in its administration pages. The Edit Item Information screen includes a link to the inventory function. (Please see the previous section, "Editing Product Item Attributes," for information about how to find an item and get to the Edit Item Information screen.)

Figure 4-33 in the previous section showed the top portion of an Edit Item Information screen, which included the link, "For **inventory** information, click [here](#)." Figure 4-34 shows the Item Inventory screen that is displayed when you click this link.

Figure 4-34 Sample Item Inventory Screen

The screenshot shows a web interface for editing item inventory. At the top, there is a green header bar with a red book icon and the text "Items". Below this is a grey header bar with a red and white icon and the text "Item Inventory". Underneath the grey bar is a grey instruction box that says "Enter the appropriate information then click Save." The main form area contains four fields: "Item Name:" with the value "wrench-9-18854"; "In Stock:" with radio buttons for "Yes" (selected) and "No"; "Shipping Time:" with a text input field containing "Ships in 24 hours"; and "Comments:" with an empty text area. Each of the three input fields has a vertical scrollbar on its right side.

On this screen, you can click the Yes or No option to indicate whether the specific product item is in stock. Enter a text string (which can be displayed to a user of your Web site on the item details screen) to indicate the shipping time. You can optionally enter additional comments about the item's inventory.

Note: In the current release, WebLogic Commerce Server does not implement an automated inventory-count calculation.

If you made any changes on the Item Inventory screen, click the Save button. Otherwise, click the Back button.

How Are Categories and Items Displayed to the Web Site User?

You can control the format and content of category and item data that is displayed to the Web site's users by setting the Display URLs field for each category or item. The WebLogic Commerce Server catalog includes the URL of three JSPs for items and categories:

- Display JSP for a category
- Summary display JSP for an item
- Detail display JSP for an item

This approach allows, for example, the colors and layout of individual items to be controlled on as fine a grained level as necessary. Categories can be assigned different JSPs to define a "look and feel" for the category. Items can be given different layout logic that is appropriate to the item.

By simply editing the name of the display JSPs in the Catalog Management administration tools, you can introduce promotional text, seasonal greetings, or special offers in the catalog. The default values that appear in the Display JSP URL field on the create/edit category or item screen are read from the `wlcs-catalog.properties` file in `WL_COMMERCE_HOME/classes`. For example:

```
# Default JSP for Categories
catalog.category.jsp.default =/commerce/catalog/includes/category.jsp

# Default JSPs for Product Items
catalog.item.jsp.default.summary =/commerce/catalog/includes/itemssummary.jsp
catalog.item.jsp.default.details =/commerce/catalog/includes/itemdetails.jsp
```

The URL values defined in the `wlcs-catalog.properties` file are relative to the WLCS Web application directory. However, you can prepend any URL or URI string that you need. Although you could use fully qualified local URLs, that could introduce portability problems if you need to move the application to another server.

Deleting Items or Removing Items from One or More Categories

You can change these Display JSP defaults in `wlcs-catalog.properties`. You also can overwrite the defaults URLs on the Administration screen by providing a pointer to another JSP template of your own design. And you can modify the format and content of the default or customized included JSP files. The values for the Display JSP URLs are stored in the following fields in the Commerce database:

- `DISPLAY_JSP_URL` in the `WLCS_CATEGORY` table
- `DET_DISPLAY_JSP_URL` and `SUM_DISPLAY_JSP_URL` fields in the `WLCS_PRODUCT` table

In the catalog's master JSP files (the JSPs that contain the `<jsp:include...>` tags), a related design decision is made by the Web designer or programmer about the type of summary or detail pages to be presented. For example, the `searchdetails.jsp` template is coded to return the summary JSP for items:

```
<%-- Get the summary JSP from the current product item --%>
<catalog:getProperty object="<%= item %>"
  propertyName="Jsp"
  getterArgument="<%= new Integer(ProductItem.SUMMARY_DISPLAY_JSP_INDEX) %>"
  id="summaryJsp"
  returnType="com.beasys.commerce.ebusiness.catalog.JspInfo" />

<%-- Included the summary JSP --%>
<jsp:include page="<%= summaryJsp.getUrl() %>" flush="true"/>
</catalog:iterateThroughView>
.
.
.
```

Deleting Items or Removing Items from One or More Categories

As described earlier in this chapter, you can assign an item to one or more categories. If necessary, you could subsequently use administration screens to:

- Delete the item entirely from the catalog.
- Remove the item from a particular category. The item could continue to be assigned to another category.

- Remove the item, one category at a time, from all categories, creating an “orphaned item” (also called an “uncategorized item” that still resides in the database).

Caching Considerations

Whenever possible, you should perform any Catalog Management operations during non-peak Web site usage. When you delete an item or remove an item from a category, WebLogic Commerce Server will automatically remove the item record from the ProductItemCache or CategoryCache. This step ensures that subsequent Web site users get a valid view of the available, categorized items.

However, if you use an SQL tool to directly delete an item from the database (WLCS_PRODUCT database table), or remove an association between an item and a category (WLCS_PRODUCT_CATEGORY mapping database table), you should flush the caches for items and categories. Flushing these in-memory caches is an administration function. For more information about caching, see the section “Improving Catalog Performance by Optimizing the Catalog Cache” on page 4-48.

Deleting an Item from the Catalog

To permanently delete an item from the catalog database via administration screens, start by finding the item via one of the provided search options:

- Keyword-based search
- Query-based search
- Uncategorized items search

The steps to delete an item are as follows:


1. Make sure the server is running. See the section “Starting the Server” on page 4-2.
2. Start the Administration tool. See the section “Starting the Administration Tool” on page 4-4.

Deleting Items or Removing Items from One or More Categories











3. On the main administration screen, click the Catalog Management graphic.
4. On the main Catalog Management screen, click the underlined [Items](#) link.
5. On the Item Search screen, perform one of the following steps:
 - a. In the Keyword input box, enter a keyword that you know is associated with the item you need to delete, and then click its Search button. For example, Figure 4-35 shows the Search results on the refreshed Item Search screen after you search for the keyword `cabinet`.

Figure 4-35 Sample Search for an Item Using Cabinet Keyword

Search for items containing
the following keywords

Keyword: 

Search Results [1-10](#) [Next](#)

1. [cabinet-9-65459](#) 
2. [cabinet-9-65439](#) 
3. [cabinet-9-JR25300](#) 
4. [cabinet-9-JR25302](#) 
5. [cabinet-9-JR25450](#) 
6. [cabinet-9-65243](#) 
7. [cabinet-9-65242](#) 
8. [cabinet-9-JR25600](#) 
9. [cabinet-9-JR25602](#) 
10. [cabinet-9-DBC4630](#) 

In the Search Results list, when the item that you need to delete is displayed, click the red X icon to the right of the item name.

- Warning:** When you click the red X icon, the delete operation is immediate and permanent; a confirmation screen is not displayed.
- b. Or in the Query input box, enter an query-based search, such as one of the following examples to find the item you need to delete; then click its Search button.


4 Catalog Administration Tasks

```
price > 100 && price <= 150  
name like 'cabinet*'  
modifiedDate < now
```

Note: The query expression `name like` is case sensitive. For example, figure shows the results of a query search that used the expression: `name like 'cabinet*'`. Had the search string been `name like 'Cabinet*'` the search would have yielded no string matches given the sample data provided by the product. For more information about the query syntax, see the section “Query-based Search Syntax” on page 5-74.

Figure 4-36 Sample Query-based Search Results Screen

Search for items containing the following keywords	Search for items matching the following query
Keyword: <input type="text"/>	Query: <input type="text" value="name like 'cabinet*'"/>



Search Results 1-10 Next	
1.	cabinet-9-65459 
2.	cabinet-9-65439 
3.	cabinet-9-JR25300 
4.	cabinet-9-JR25302 
5.	cabinet-9-JR25450 
6.	cabinet-9-65243 
7.	cabinet-9-65242 
8.	cabinet-9-JR25600 
9.	cabinet-9-JR25602 
10.	cabinet-9-DBC4630 

In the Search Results list, when the item that you need to delete is displayed, click the red X icon to the right of the item name.

Warning: When you click the red X icon, the delete operation is immediate and permanent; a confirmation screen is not displayed.

- c. Or click the Search button next to the text: Search for items which are not assigned to any category. Figure 4-37 shows the search results when you click this option. The layout of the text and results has been modified in the figure to fit into this document.

Figure 4-37 Sample Search Results for Uncategorized Items

Search for items which are
not assigned to any category



search

Search Results [1-6](#)

1. [wrench-9-IR1077XP](#) **X**
2. [wrench-9-CP828](#) **X**
3. [grinder-9-19911](#) **X**
4. [wrench-9-18821](#) **X**
5. [wrench-9-18820](#) **X**
6. [grinder-9-18814](#) **X**

In the Search Results list, when the item that you need to delete is displayed, click the red X to the right of the item name.

Warning: When you click the red X icon, the delete operation is immediate and permanent; a confirmation screen is not displayed.

Removing an Item from One or More Categories

The process to remove an item from one or more categories is similar to the process of assigning items to categories, which is explained in the section “Using the Administration Screens to Assign Items to Categories” on page 4-24.

For a given item, you can remove it (unassign it), one category at a time. If you remove the item from all categories, it remains in the Commerce database and is flagged as an uncategorized item, also known as an orphaned item.

Removing an item from one or more categories is different than deleting the item entirely from the database. If you need to delete an item entirely, see the section “Deleting an Item from the Catalog” on page 4-38.

4 Catalog Administration Tasks

The steps to remove an item from a category are covered in the following list. Due to their similarity, the sample screens that would duplicate the ones shown in the section “Using the Administration Screens to Assign Items to Categories” on page 4-24 are not repeated here.

Note: To remove an item from a particular category, you must know in advance the name of the category to which the item is currently assigned. As the administrator, if you do not have this information, run the catalog’s Web application and browse through the hierarchy of categories to find the item. If necessary, check with the development team to confirm that you are about to remove the item from the correct category (because the item can be associated with more than one category).

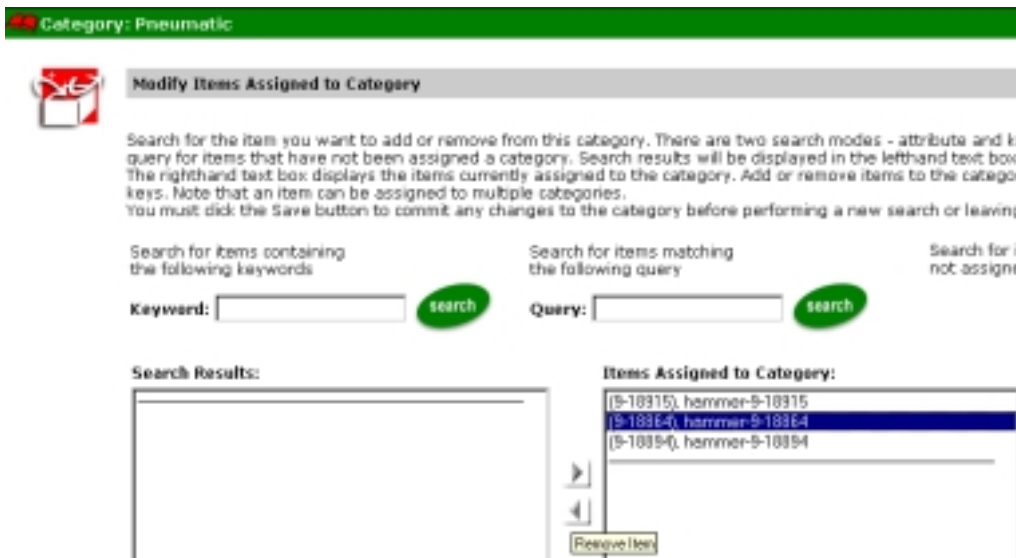
1. Make sure the server is running. See the section “Starting the Server” on page 4-2.
2. Start the Administration tool. See the section “Starting the Administration Tool” on page 4-4.
3. On the main administration screen, click the Catalog Management graphic.
4. On the main Catalog Management screen, click the underlined Categories link.
5. In the Catalog hierarchy display, click the category or subcategory into which you want to remove the item.

Note: To expand a current category and reveal its subcategories (if any), make sure you click the red arrow to the left of a category name. If you instead click the current category’s underlined name, you are taken to its Edit Category screen.

6. When the category or subcategory name is shown in the hierarchy, click its underlined link.
7. On the Editing Category: <category name> screen, click the following link: To modify the items assigned to this category, click here.
8. Figure 4-38 shows a sample screen In this example, assume that we need to remove one of the pneumatic hammer items from the following category:

Root → H → Hammers → Pneumatic

Figure 4-38 Removing an Item from a Category



On the screen, the second item in the Items Assigned to Category text box has been highlighted, and the cursor (not shown) is hovering over the left arrow.

9. Click the left arrow to remove the item from the category.
10. Click the Save button near the bottom of the screen to make the change. Or click the Back button near the bottom of the screen (before clicking the Save button) to cancel any updates you made on the screen.

If you need to remove an item from more than one category, you must do so one category at a time. After removing the item from the first category, follow the steps (starting at Step 4) to remove the category from the next category.

Removing Categories

To remove a category, find the category or subcategory via administration screens and click the red X icon to the right of the target category or subcategory name. When you click the red X icon, you are prompted for a confirmation before the deletion starts.

4 *Catalog Administration Tasks*

Warning: Removing a category removes it and all of its subcategories (if any). In Oracle databases, this feature is known as a cascading delete operation. Cascading deletes are not supported currently in Cloudscape.

Also, product items that are associated with removed categories may be orphaned (unless they belong to another category in the hierarchy). Orphaned items are allowed to remain in the catalog, and can be reassigned later to one or more categories. The caveat here is that by deleting a category, you may be inadvertently removing many subcategories from the catalog. Check the hierarchy carefully before clicking the red X next to a category name

With this caveat in mind, to remove a category from the catalog's hierarchy, follow these steps:


1. Make sure the server is running. See the section "Starting the Server" on page 4-2.
2. Start the Administration tool. See the section "Starting the Administration Tool" on page 4-4.
3. On the main administration screen, click the Catalog Management graphic.
4. On the main Catalog Management screen, click the underlined Categories button on the Categories graphic.
5. On the main Categories screen, you can find the category you want to modify by either:
 - Entering its category identifier in the search input box. In the sample data that comes with the catalog tables in the Commerce database, the category identifier assigned to each alphabetized main category is the same letter as the category. But a number is assigned to the subcategories. Figure 4-39 shows a portion of the resulting Categories screen after you enter the category identifier 149 in the search field and clicked the Search button. In the figure, notice how the found category name is shown in red type.

Figure 4-39 Sample Category Search Result

Search for Categories

Click through the hierarchy to find a category, or search for the category by its primary key shown in parenthesis. Click the category title to edit it, or click **X** to delete the category.

Warning: deleting a category will also delete its subcategories and cannot be undone.

Category Identifier: 

Category Hierarchy

Top Level Categories

- ↳ [A \(A\)](#) **X**
- ↳ [B \(B\)](#) **X**
- ↳ [C \(C\)](#) **X**
- ↳ [Caddies \(144\)](#) **X**
- ↳ [Cans \(145\)](#) **X**
- ↳ [Chargers \(149\)](#) **X**
- ↳ [Chests \(150\)](#) **X**

6. When the category you want to remove is displayed, click the red X icon to the right of the category name. Just to emphasize the caveat again, be aware that:
 - Deleting a category removes it and all of its subcategories (if any). In Oracle databases, this feature is known as a cascading delete operation. This operation is not currently supported in Cloudscape.
 - Product items that are associated with removed categories may be orphaned (unless they belong to another category in the hierarchy). Orphaned items are allowed to remain in the catalog, and can be reassigned later to one or more categories. The caveat here is that by deleting a category, you may be

inadvertently removing many subcategories from the catalog. Check the hierarchy carefully before clicking the red X next to a category name.

Moving Items from One Category to Another Category

To move an item from one category to another category:

- Assign the item to the other category, as described in the section “Using the Administration Screens to Assign Items to Categories” on page 4-24.
- Remove the item from an existing category, as described in the section “Removing an Item from One or More Categories” on page 4-41.

Define Custom Attributes for Items

You can define a property set that establishes custom attributes for an item. For a given item, a custom attribute that you define can be used in addition to the default attribute that was set by WebLogic Commerce Server in the product catalog database tables.

Warning: The default product catalog attributes that are provided by WebLogic Commerce Server (based on Dublin Core) are retrieved in a single SQL statement (from a single database table) and are cached. However, custom attributes typically require a single SQL statement (which involves multiple database tables) and are not cached. For optimal performance, BEA recommends that the use of custom attributes be minimized and that the catalog developer maps their attributes onto the default Dublin Core attributes. For related information about the category and item caches, see the section “Improving Catalog Performance by Optimizing the Catalog Cache” on page 4-48.

WebLogic Commerce Server uses the existing product architecture for defining property sets and assigning values to custom attributes. Both items and categories implement the `ConfigurableEntity` interface, allowing property sets to be used to define custom attributes for items and categories. At runtime, these properties can be accessed or modified using the standard `getProperty` or `setProperty` interfaces of `ConfigurableEntity`.

The property set editor was introduced with an earlier release of WebLogic Personalization Server. Now the property set editor extends support to a new class of Property Set for the WebLogic Commerce Server catalog, and the product provides administrative tools that allow the custom attributes of categories and items to be edited. These tools are based on the familiar user interface for editing the custom attributes of groups and user (respectively).

You therefore have a number of options for customizing the WebLogic Commerce Server product catalog:

■ Property Set Definition

You can define Property Sets for custom attributes. This is a runtime operation that you can perform without having to restart the server. Domain-specific catalog attributes that cannot be mapped onto the metadata provided in the WebLogic Commerce Server schema (which is based on the Dublin Core standard) can be added to a Property Set for the catalog.

■ Service Provider Interfaces

The functional areas of the WebLogic Commerce Server product catalog can be unplugged (either individually or as a whole) to customize the behavior or data sources of the catalog. The customizable services are:

- Product Item Manager – responsible for managing the explicit attributes for an item
- Category Manager – responsible for managing the mapping of items into categories
- Custom Data Manager – responsible for managing the custom attributes for an item
- Catalog Query Manager – responsible for executing keyword and attribute queries and returning collections of items.

These steps are described in Chapter 6, “Using the API to Extend the Product Catalog.”

- Entity Property Manager Customization

Custom attribute management has been delegated from the Custom Data Manager to the Entity Property Manager. A new implementation of the Entity Property Manager could be plugged for custom data management.

Improving Catalog Performance by Optimizing the Catalog Cache

WebLogic Commerce Server provides tools that allow you to tune the response-time efficiency of your product catalog by adjusting the size and behavior of in-memory cache settings. Two separate caches have been implemented:

- CategoryCache
- ProductItemCache

There is only one instance of each cache per server. In a WebLogic Server clustered environment, the cache is specific to each server. Thus if you make adjustments to a CategoryCache or ProductItemCache on a particular server in the cluster, and you want that adjustment to take effect on other servers in the cluster, you must invoke the administration tools on each server machine and make the updates separately.

Values for each cache are defined in the `weblogiccommerce.properties` file and on a catalog administration screen. As the site administrator, your primary tasks related to the setup and maintenance of the caches are as follows:

- Enable or disable each cache. Keeping the cache enabled is highly recommended. Each cache is enabled by default.
- Set the size of each cache. That is, the maximum number of category and item records in each cache.
- Set the Time to Live (TTL) for each cache.
- Increase or decrease the size of each cache.
- Flush the cache.

- View percentage statistics about the hit or miss rate of each cache.
- Reset the statistics.

By experimenting with the size and Time to Live (TTL) values for each cache, you can achieve optimal performance of the catalog. Most importantly, you can significantly improve the satisfaction level of customers who are shopping on your Web site. The goal of the cache is to avoid, as much as possible, excessive accesses to the Commerce database as the application software performs catalog data lookup requests.

Cache-Related Values in `WeblogicCommerce.properties`

The `WL_COMMERCE_HOME\weblogiccommerce.properties` file includes the following values related to the category and item caches:

```
# Cache entries

ProductItemCache.ttl=21600000
ProductItemCache.capacity=10000
ProductItemCache.enabled=true

CategoryCache.ttl=86400000
CategoryCache.capacity=1000
CategoryCache.enabled=true
```

Table 4-2 describes the values defined for each cache property:

4 Catalog Administration Tasks

Table 4-2 Cache Property Values in WeblogicCommerce.properties

Property Value	Default Value	Description
<code>ProductItemCache.ttl</code>	2160000 milliseconds (360 minutes, or 6 hours)	Sets the Time to Live (TTL) value that gets assigned to each item record that is added to the ProductItemCache. When the value has expired, the item is removed from the cache. Then as new item records are accessed by users (items retrieved from the database), a copy of each record is added to the server-wide item cache.
<code>ProductItemCache.capacity</code>	10000 items	Sets the maximum number of product items that can be loaded into the ProductItemCache.
<code>ProductItemCache.enabled</code>	true	Can be set to true or false. BEA recommends that you always leave the cache enabled; if your catalog has highly volatile product item data, use a lower TTL value to ensure that the data is refreshed at appropriate intervals

Table 4-2 Cache Property Values in `WeblogicCommerce.properties`

Property Value	Default Value	Description
<code>CategoryCache.ttl</code>	86400000 milliseconds (1440 minutes, or 24 hours)	Sets the Time to Live (TTL) value that is assigned to each category record in the <code>CategoryCache</code> . When the value has expired, the category record is removed from the cache. Then as new category records are accessed by users (categories retrieved from the database), a copy of each record is added to the server-wide category cache.
<code>CategoryCache.capacity</code>	1000 categories	Sets the maximum number of category records that can be loaded into the <code>CategoryCache</code> .
<code>CategoryCache.enabled</code>	true	Can be set to true or false. BEA recommends that you always leave this cache enabled; if your catalog has highly volatile category data, use a lower TTL value to ensure that the data is refreshed at appropriate intervals.

Use the caching statistics to understand whether the values that you select initially are too low or too high.

Note: Remember that in a WebLogic Server clustered environment, the cache is specific to each server. Thus if you make adjustments to a `CategoryCache` or `ProductItemCache` on a particular server in the cluster, and you want that adjustment to take effect on other servers in the cluster, you must perform the administration tasks on each server machine and make the updates separately.

Considering Hardware Costs Versus the Cost of Dissatisfied Web Site Users

BEA recommends that you provide sufficient hardware memory resources so that you can cache your entire product catalog. As you consider the options for implementing a caching strategy to support your e-commerce Web site's product catalog, ask the following question. In the long-term, what is more expensive:

- The costs associated with increasing RAM capacity on your Web servers...
- Or the cost of losing potential Web shoppers who grew impatient with the performance of your catalog's pages?

BEA suggests that, almost certainly, losing customers due to poor site performance will be far more costly than purchasing the necessary RAM hardware to support your site.

What's in Each Cache Initially?

When the server starts, it reads the property values for the `CategoryCache` and `ProductItemCache` from the `weblogiccommerce.properties` file. But those values simply set the behavior and potential size of the separate caches. Each cache is initially empty.

As users start accessing the site's pages and requests for category or product item data are received, the WebLogic Commerce Server software first checks to see if the requested category or item record is in the cache. If the record resides in cache, it is returned to the requesting user and displayed. If the record does not reside in cache, the record is retrieved from the Commerce database, put in cache, returned to the requesting user and displayed.

If you are interested in the architectural view of how the catalog service managers use cache, see the section "Catalog Architecture and Services" on page 6-3.

The Catalog Cache Administration Screen

To start the catalog cache administration screen, follow these steps:

1. Make sure the server is running. See the section “Starting the Server” on page 4-2.
2. Start the Administration tool. See the section “Starting the Administration Tool” on page 4-4.

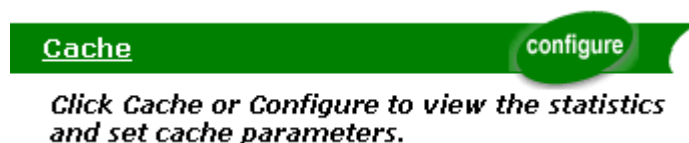
Note: As previously mentioned, in a WebLogic Server clustered environment the cache is specific to each server. Thus if you make adjustments to a CategoryCache or ProductItemCache on a particular server in the cluster, and you want that adjustment to take effect on other servers in the cluster, you must perform the administration tasks on each server and make the updates separately.

From your browser, you can administer another server in the cluster by specifying the other server’s name in the URL (assuming that the WebLogic server software is already running on the remote machine). For example, if the machine `Blues` is a remote server in the cluster, you can use the following URL from a browser sessions on your local machine to invoke the main administration screen for the remote server:

```
http://blues:7501/tools
```

3. On the main administration screen, click the Catalog Management graphic.
4. On the main Catalog Management screen, click the Configure button on the Cache graphic, as shown in Figure 4-40.

Figure 4-40 Configure Button on Cache Graphic



5. Figure 4-41 shows the resulting initial Cache Statistics and Configuration screen.

4 Catalog Administration Tasks

Figure 4-41 Initial Cache Statistics and Configuration Screen

Catalog's Cache Statistics and Configuration								
Enable	Cache Name	Capacity	Time to Live (min)	Items	Requests	Hits	Misses	Rate(%)
<input checked="" type="checkbox"/>	ProductItemCache	<input type="text" value="10000"/>	<input type="text" value="360"/>	0	0	0	0	0
<input checked="" type="checkbox"/>	CategoryCache	<input type="text" value="1000"/>	<input type="text" value="1440"/>	0	0	0	0	0

back save reset all flush all

- On the Cache Statistics and Configuration screen, you can:
 - Enable or disable each cache. The general recommendation is that you leave both caches enabled.
 - Click the underlined [ProductItemCache](#) link to view a separate screen that only shows item cache statistics. By going to the cache-specific screen, you can flush just that cache.
 - Click the underlined [CategoryCache](#) link to view a separate screen that only shows category cache statistics. By going to the cache-specific screen, you can flush just that cache.
 - Change the value for the ProductItemCache or CategoryCache capacity. This is the maximum number of items or categories allowed in each cache.
 - Change the value for the ProductItemCache or CategoryCache Time to Live (TTL), shown in minutes. This is the maximum time that an item will not be reread from the database.
 - View statistics about the following:
 - The number of product items or categories currently in the cache (Items column).
 - The number of requests for product items or categories (Requests column).
 - The number of times a requested product item or category was found in the cache (Hits).

The number of times a requested product item or category was not found in the cache (Misses).

The hit percentage; that is, the total requests divided by the number of hits (Rate% column).

Note: The Administration screens always bypass the cache. Thus any access you perform to category and item data in the Administration tools will not be seen in the statistics.

7. The buttons on the screen perform the following functions:
 - To return to the main catalog administration screen, click the Back button. If you do not click the Save button before clicking the Back button, any changes you made are not performed.
 - To save any changes you make, click the Save button.
 - To reset the statistics to zero, click the Reset All button.
 - To flush the contents of both caches, click the Flush All button.

A basic approach to working with the statistics is as follows:

1. Reset the caching statistics in the Administration screen
2. Let the Web site run for awhile
3. Examine the statistics and look at the Hit Rate (the % of times the requested record was found in the cache, instead of having to get the record from the database)
4. Adjust the caching values in the Administration screen

Repeat this process until the Hit Rate is high.

Using the wlbs-catalog.properties File

In the WebLogic Commerce Server's default product catalog configuration, the `wlbs-catalog.properties` file serves two purposes. The first part of the file defines values for internationalization string constants and constant configuration

4 Catalog Administration Tasks

parameters, while the second part of the file defines the names of the tables, columns, and the SQL statements used by the JDBC catalog implementation to perform persistence activities.

Note: To learn more about string constants and constant configuration parameters, see “Some Property Values You Might Modify” on page 4-56. To learn how to modify the names of tables, columns, and so on, see “Editing the Catalog Schema Definition” on page 4-59.

In most cases, you will not have to modify the content of the `wlcs-catalog.properties` file. However, some of the properties are relevant to developers who need to extend the catalog to suit specific business requirements, or to internationalize the catalog for non-English readers.

Location

The default `wlcs-catalog.properties` file is in `WL_COMMERCE_HOME\classes`, where `WL_COMMERCE_HOME` is the directory in which you installed the WebLogic Commerce Server software.

Some Property Values You Might Modify

The `wlcs-catalog.properties` file contains over 1000 lines of name/value properties. Table 4-3 lists a subset of the properties to introduce you to the type of information in the file. After reading this summary, you should be able to decide if your catalog’s environment would benefit by adjusting the values in `wlcs-catalog.properties`.

Table 4-3 Summary of Values in the `wlcs-catalog.properties` File

Property	Value and Description
<code>catalog.jsp.date.format</code>	<code>MM/dd/yyyy hh:mm:ss z</code> Sets the default format of <code>DATE</code> or <code>TIMESTAMP</code> fields in the database.

Table 4-3 Summary of Values in the *wlcs-catalog.properties* File

Property	Value and Description
<code>catalog.jsp.default.iterator.size</code>	10 Sets the default size of the <code>ViewIterators</code> created by Pipeline Components. You should set this parameter to a large enough value so that a typical set of items will not require an excessive number of trips to the database.
<code>catalog.searchresults.size</code>	-1 Sets the maximum search results returned by the catalog. -1 means unlimited search results size. You can dynamically change the searchresults by using the <code>get/set MaxSearchResults</code> methods on the <code>CatalogQuery</code> object.
<code>catalog.jsp.keyword.convert</code>	lower Used by the Administration pages to determine whether keywords should be case converted. Possible values are: <ul style="list-style-type: none">■ lower■ upper■ none
<code>catalog.category.jsp.default</code>	<code>/commerce/catalog/includes/category.jsp</code> Sets the default JSP file used to display a category.
<code>catalog.item.jsp.default.summary</code>	<code>/commerce/catalog/includes/itemsummary.jsp</code> Sets the default JSP file used to display an item's summary page.
<code>catalog.item.jsp.default.details</code>	<code>/commerce/catalog/includes/itemdetails.jsp</code> Sets the default JSP file used to display an item's details page.

4 Catalog Administration Tasks

Table 4-3 Summary of Values in the `wlcs-catalog.properties` File

Property	Value and Description
<code>catalog.custom.data.catalog.manager</code>	<code>com.beasys.commerce.ebusiness.catalog.CatalogManager</code> The JNDI name of the <code>CatalogManager</code> used to access Custom properties.

Messages:

The `wlcs-catalog.properties` file includes numerous properties for catalog-related messages. A few examples:

`user.message.error.format.date.log` =The date format is invalid. Please enter a valid date.

`user.message.error.duplicate.user` =A category with the specified identifier already exists. Category identifiers must be unique within the catalog.

`user.message.search.invalid.expression.user` =The search expression you have entered is invalid. Please try again.

If desired, you can change the value on the right side of the equal sign. For example, internationalization (I18N) developers can translate the values to a non-English language.

Types of messages:

- General user messages
- Category messages
- Item messages
- Search messages
- Tag messages
- Error messages

See the `wlcs-catalog.properties` files for the full set of messages.

Editing the Catalog Schema Definition

In addition to modifying some name/value pairs in the *wlcs-catalog.properties* file, you may also want to customize the names of the tables and/or columns used by the BEA WebLogic Commerce Server product catalog system.

The schema and persistence definition section of the *wlcs-catalog.properties* file are referenced by the Tier 2 JDBC catalog Service Providers. As such, the configuration parameters listed in this section are subject to change, and extreme care should be taken when editing these persistence parameters.

The name of the persistence definition (or schema) file used by the Tier 2 Service Providers is loaded from the deployment environment of the stateless session beans. Therefore, it is possible to deploy multiple instances of the catalog services that are bound to different schema files. This allows multiple instances of the WebLogic Commerce Server product catalog to be deployed, bound to distinct tables or columns.

By editing the schema file you can modify the names of tables and columns. The SQL scripts should also be modified to reflect the new names. An example from the default schema file is presented in Listing 4-1.

Listing 4-1 Default Schema File

```
#####  
###  
# CATALOG TABLE NAMES  
#####  
###  
  
CATALOG_TABLE_CATEGORY=WLCS_CATEGORY  
CATALOG_TABLE_PROD_ITEM=WLCS_PRODUCT  
CATALOG_TABLE_PROD_KEYWORD=WLCS_PRODUCT_KEYWORD
```

By editing the right-hand side of the equal sign, you can modify the names of the tables used by the WebLogic Commerce Server product catalog Tier 2 persistence mechanism.

4 *Catalog Administration Tasks*

Notes: It is not currently possible to add or remove attributes by editing the schema file. New attributes (schema additions) are easiest handled by writing a new `CustomDataManager` stateless session bean that reads and writes the values of the new properties, or by using the provided `CustomDataManager`, which will store the values in the `WLCS_CAT_` set of tables.

Please refer to the comments within the `wlcs-catalog.properties` file for additional details.

5 The Product Catalog JSP Templates and Tag Library

The BEA WebLogic Commerce Server provides JavaServer Page (JSP) templates and JSP tags that implement commonly used Web-based product catalog features. The product catalog JSP templates allow your customers to search for product items or browse through categories to locate items; the JSP tags are used to implement this functionality.

When you click the Add to Cart button on the JSPs that provide item details, the default Webflow for the WebLogic Commerce Server Web application passes data about the selected product items to the order management JSPs. For information about the JSPs that comprise the Order Processing package, see [BEA WebLogic Commerce Server Order Processing Package](#).

This topic includes the following sections:

- Introduction
- JSP Templates Overview
 - On Which JavaServer Page Will My Users Start?
 - Sequence Review and the Browser View
- JavaServer Pages (JSPs)
 - main.jsp Template
 - browse.jsp Template

- details.jsp Template
- search.jsp
- searchresults.jsp
- Input Processors
 - CatalogIP
 - GetProductItemIP
 - GetCategoryIP
 - KeywordSearchIP
 - ExpressionSearchIP
 - MoveAttributeIP
 - RemoveAttributeIP
- Pipeline Components
 - CatalogPC
 - GetCategoryPC
 - GetProductItemPC
 - GetParentPC
 - GetAncestorsPC
 - GetProductItemsPC
 - GetSubcategoriesPC
 - MoveAttributePC
 - RemoveAttributePC
 - SearchPC
- The Catalog JSP Tag Library: cat.tld
 - The <catalog:getProperty> Tag
 - The <catalog:iterateViewIterator> Tag
 - The <catalog:iterateThroughView> Tag

Note: In this chapter, the environment variable `WL_COMMERCE_HOME` is used to represent the directory in which you installed the WebLogic Commerce Server software.

Introduction

The JSP templates and JSP tags included in the BEA WebLogic Commerce Server allow you to easily customize the presentation of the product catalog. The names of the JSPs for categories and product items are stored in the database as attributes of the categories and items. (See Chapter 2, “The Product Catalog Schema,” for information about the `DISPLAY_JSP_URL` column in the `WLCS_CATEGORY` database table, and the `SUM_DISPLAY_JSP_URL` column [a pointer to the item’s summary page] and the `DET_DISPLAY_JSP_URL` column [a pointer to the item’s detail page] in the `WLCS_PROD_ITEM` database table.)

The Commerce Server product catalog integrates with the Webflow engine, which automatically selects the appropriate JSP for displaying a particular category or product item. The Webflow is set by entries in the `webflow.properties` file, as explained in the *BEA WebLogic Commerce Server Webflow and Pipeline Management* documentation.

JSP tag libraries allow you to easily retrieve the attributes of items and categories in the product catalog. You can then format these attributes using HTML tags. Any HTML editor can be used to create custom layouts. You can also include custom Java code within the JSPs to display categories and items.

JSP Templates Overview

The BEA WebLogic Commerce Server provides a number of JSP templates that span the start-to-finish experience for a Web site user who is shopping on your site. There are several sequence paths through the system; the following list outlines one possible path.

1. If the customer has previously registered and logged in, display a home page that welcomes the customer, and include links to the customer's Order History and Payment History. Otherwise, just display a generic home page.
2. Search for or browse product items in the catalog.
3. Add one or more items to a shopping cart.
4. Enter or update the customer's profile information, such as their default shipping address, default credit card billing address, and credit card number (encrypted).
5. Checkout.
6. Review an order.
7. View a confirmation of an order.
8. Optionally check the status of an order.

On Which JavaServer Page Will My Users Start?

Before you can understand which JavaServer Page your users will open first, it is important to understand how your catalog and order fulfillment system will eventually be deployed. This section introduces that topic, and you can follow along by opening the referenced files in the installed WebLogic Commerce Server directories.

Web Applications and the `weblogic.properties` File

The JSP templates and all the supporting Java packages are configured to run as a **Web application** on the WebLogic Server. In the `weblogic.properties` file that resides in the `WL_COMMERCE_HOME` directory, the following property is used to identify each Web application:

```
weblogic.httpd.webApp.myapp=location
```

The `myapp` parameter is the context name given to the Web application, and is included in the initial part of any URL request to the Web application. The `location` parameter is the root directory of the Web application, or the location of the Web Archive (`.war`) file that contains the Web application archive.

The two Web applications for the WebLogic Commerce Server are:

```
weblogic.httpd.webApp.tools=location
weblogic.httpd.webApp.wlcs=location
```

When you installed the WebLogic Commerce Server, the installation procedure substituted the directory path in the location parameter. For example, if you installed the BEA WebLogic Commerce Server product on a Windows system to `d:\WebLogicCommerce`, the Web application properties would be defined in the `WL_COMMERCE_HOME\weblogic.properties` file as shown in Listing 5-1.

Listing 5-1 WLCS/Tools Application Definitions in the weblogic.properties File

```
#####
# BEA WEBLOGIC COMMERCE SERVER WEB APPLICATIONS
# -----
# Defines the admin tools Web Application.
weblogic.httpd.webApp.tools=d:/weblogiccommerce/server/webapps/admin/
# Defines the wlcs Web Application.
weblogic.httpd.webApp.wlcs=d:/weblogiccommerce/server/webapps/wlcs/
```

The Web application named `tools` comprises all the administration tool JSPs, Java packages, and related files. The Web application named `wlcs` comprises all the catalog and order fulfillment JSPs, Java packages, and related files.

XML Deployment Descriptor Files

All other configuration information for the Web applications is described in each one's XML deployment descriptor files. The WebLogic Commerce Server Web application's `web.xml` deployment descriptor file resides in the following directory:

```
%WL_COMMERCE_HOME%\server\webapps\wlcs\Web-inf\web.xml (Windows)
$WL_COMMERCE_HOME/server/webapps/wlcs/Web-inf/web.xml (UNIX)
```

Included in the `web.xml` file for the WebLogic Commerce Server Web application is a setting for the initial page, or welcome page of the application, as shown in Listing 5-2.

Listing 5-2 Setting for Welcome Page in the web.xml File

```
<!-- Welcome file for the WLCS -->
<welcome-file-list>
  <!-- This is the entry point to a WLCS site.
  Change this appropriately -->
  <welcome-file>/index.jsp</welcome-file>
</welcome-file-list>
```

For the WebLogic Commerce Server Web application, the `index.jsp` file resides in the root directory for the application, or in:

```
%WL_COMMERCE_HOME%\server\webapps\wlcs\index.jsp (Windows)
$WL_COMMERCE_HOME/server/webapps/wlcs/index.jsp (UNIX)
```

commercef Property Set and DestinationDeterminer

If you open `index.jsp` with a text editor, you will see that it has a single line of JSP code, as follows:

```
<jsp:forward page="/application/commercef"/>
```

The WebLogic Commerce Server Web application uses an application initialization property set named `commercef`. This property set specifies a `DestinationDeterminer` property that has the following value:

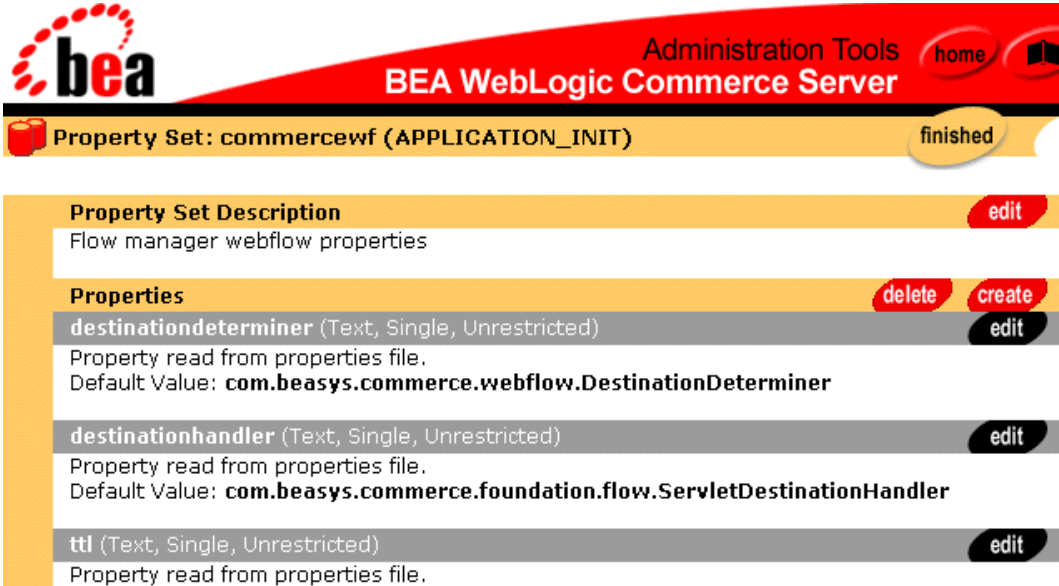
```
com.beasys.commere.webflow.DestinationDeterminer
```

To see the `commercef` property set definition yourself, start by opening the Administration Tools home page at:

```
http://localhost:7501/tools/application/admin
```

On this page, select the Property Set Management link. On the Property Sets page, under the Application Initialization Property Sets heading, select the `commercef` property set's link. The Property Set page for `commercef` is displayed, as shown in Figure 5-1.

Figure 5-1 Property Set Page for commercewf



The `com.beasys.commerce.webflow.DestinationDeterminer` reads values from the `webflow.properties` file. This file sets the order in which the JSPs, input processors, and Pipelines are executed, based on the activity initiated by a user of the Web site. The JSPs control the presentation of the information, while the input processors and Pipelines manage the business logic.

Notes: The `webflow.properties` file resides in `WL_COMMERCE_HOME`.

For more information about the `webflow.properties` file, see the [BEA WebLogic Commerce Server Webflow and Pipeline Management](#) documentation.

In the first line of the `webflow.properties` file that is not a comment, the initial step involves obtaining the parameters for the catalog's top-level categories. This is accomplished by the `GetTopCategories` input processor, which if successful, initiates execution of the `GetTopCategories` Pipeline. If both of these mechanisms have executed successfully, the `main.jsp` template is sent to the browser, as shown in Listing 5-3.

Listing 5-3 Webflow Processing Leading to the Display main.jsp

```
begin=GetTopCategories.inputprocessor
.
.
.
GetTopCategories.inputprocessor.success=GetTopCategories.pipeline
# Now display the main page
GetTopCategories.pipeline.success=commerce/main.jsp
```

Thus, when the initial page is opened in a browser and the URL references the context name for the WebLogic Commerce Server Web application (`wlcs`), the first JSP that is displayed (already populated with values from the Pipeline) is the `main.jsp` template.

Note: Although the `begin` state is only set the first time the page is accessed, the process of obtaining the product catalog's top categories is initiated by the Webflow each time a customer attempts to access the home page, as shown in the following `webflow.properties` statement:

```
*.jsp.link(home)=GetTopCategories.inputprocessor
```

Sequence Review and the Browser View

Let's review how these files and parameters are working in sequence to open the initial page of the catalog/order site in a customer's browser:

1. The server is started by running the `StartCommerce.bat` (Windows) or `StartCommerce.sh` (UNIX) procedure from a system prompt. This procedure resides in `WL_COMMERCE_HOME`, the top-level directory where you installed WebLogic Commerce Server. The procedure sets up the run-time environment for WebLogic Commerce Server and starts the WebLogic Server.
2. As the WebLogic Server starts, it finds the properties defined in many `*.properties` files, including the `weblogic.properties` file that resides in `WL_COMMERCE_HOME`. In this file, the `weblogic.httpd.webApp.wlcs` property deploys the `wlcs` Web application on the WebLogic Server and specifies the root

directory of the application. Consequently, the Web application server knows that the `wlcs` context name is included in the initial part of any URL request of the Web application.

3. You can see this in action by opening `http://localhost:7501/wlcs/` in a browser. (Prerequisite: start the server on your machine as described in step 1).
4. When the page loads, notice how the server switches the URL in your browser's Address/Location bar to:

```
http://localhost:7501/wlcs/index.jsp
```

This is because the `web.xml` file for the `wlcs` Web application specified `index.jsp` as the initial page in the `<welcome-file>` XML tag.

5. The `index.jsp` page simply references the `/application/commercewf/` application initialization property set.
6. The `commercewf` property set defines a `DestinationDeterminer` property. The default value is `com.beasys.commere.webflow.DestinationDeterminer`.
7. The `Webflow DestinationDeterminer` Java class reads values from the `webflow.properties` file.
8. The `webflow.properties` file includes a few initial steps (carried out by the `GetTopCategories` input processor and corresponding Pipeline) to get category data. Then, on successful completion, the Webflow calls for display of the `main.jsp` template, as shown below:

```
GetTopCategories.pipeline.success=commerce/main.jsp
```

Note: For more information about the `main.jsp` template, see “`main.jsp` Template” on page 5-14.

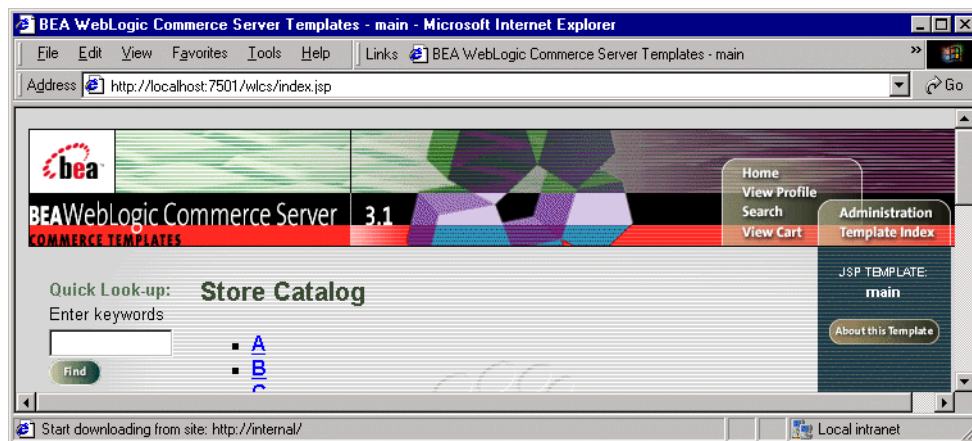
Figure 5-2 shows a portion of the Web application's home page. Notice that before we click on any links on this page, the URL is

```
http://localhost:7501/wlcs/index.jsp
```

but the displayed template is `main.jsp`.

5 The Product Catalog JSP Templates and Tag Library

Figure 5-2 Home Page Display for the WLCS Application

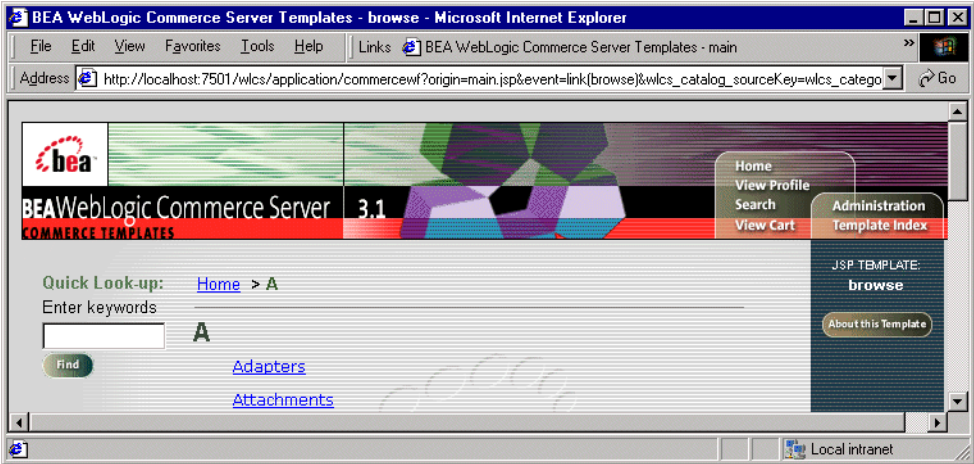


When you take any action on the page, the convention for the URI is to combine the `/wlcs/` context name for the Web application, the `/application/commercewf/` property set name, and the result of the operation with the Webflow and Pipeline processing. For example, if you click the [A](#) category in the Store Catalog, the URI changes to:

```
http://localhost:7501/wlcs/application/commercewf?origin=main.jsp
&event=link(browse)&wlcs_catalog_sourceKey=wlcs_categories&wlcs_c
atalog_destinationKey=wlcs_siblings&wlcs_catalog_category_id=1
```

Figure 5-3 shows a portion of this URI on the resulting page, which displays the `browse.jsp` template.

Figure 5-3 Resulting URI, From main.jsp to browse.jsp in the A Category



In this way, the flow of presentation-level JavaServer Pages and the processing of the business logic by input processors and Pipelines are all operating in the context of the running WebLogic Commerce Server Web application. The JSPs are not simply linked from one JSP file to another JSP file.

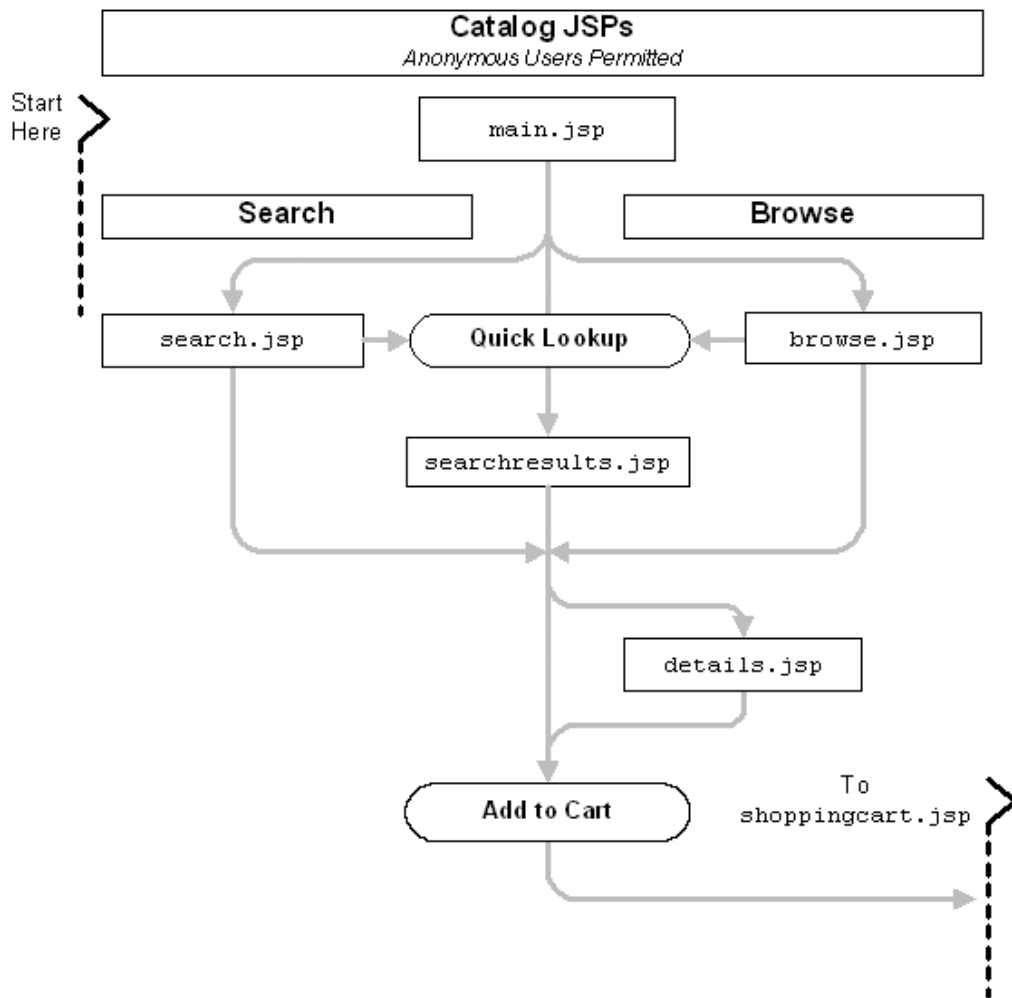
JavaServer Pages (JSPs)

The WebLogic Commerce Server Web application contains a number of JavaServer Pages (JSPs) that allow your customers to display a catalog's categories and product items. You can choose to utilize these pages in their current form, or adapt them to meet your specific needs. This section describes each page in detail.

Note: For a description of the complete set of JSPs used in the WebLogic Commerce Server Web application and a listing of their locations in the directory structure, see the [Summary of JSP Templates](#) documentation.

Figure 5-4 illustrates the JSP templates that participate in the product catalog portion of the Webflow.

Figure 5-4 Flow of Catalog JSP Templates



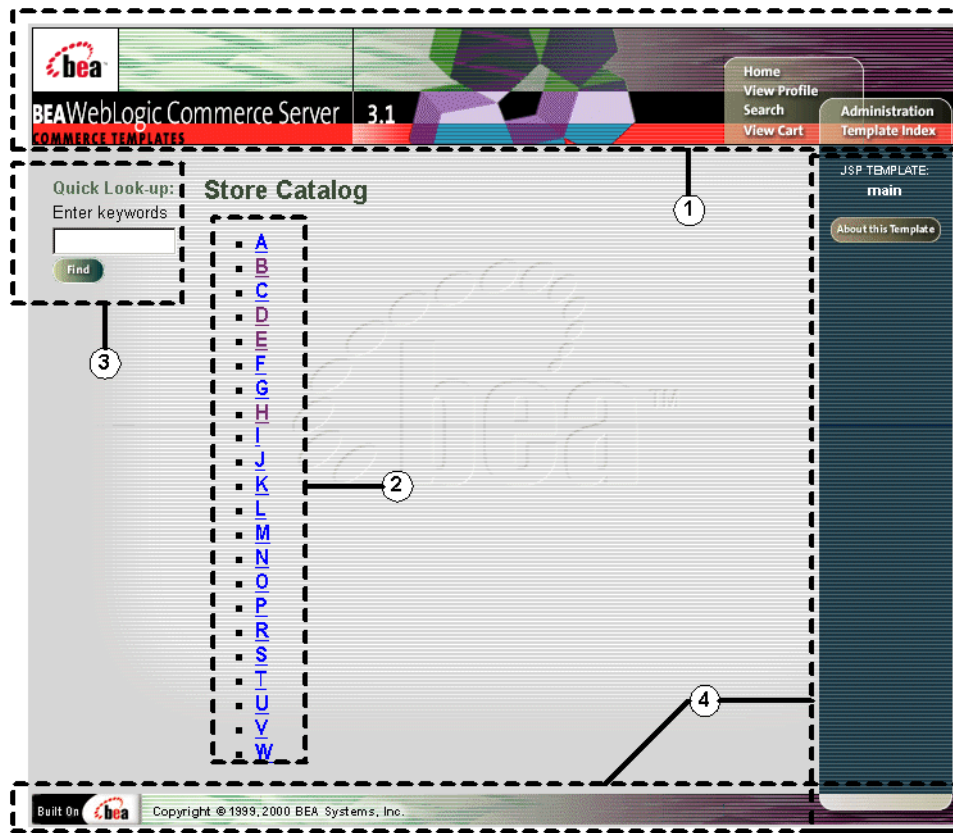
main.jsp Template

The `main.jsp` template is the default home page for the product catalog. As noted in “On Which JavaServer Page Will My Users Start?” on page 5-4, the WebLogic Commerce Server Web application’s home page is actually `index.jsp`. However, `index.jsp` refers processing to an `application/commercewf/` property set, which defines a Java class that reads values from the application’s `webflow.properties` file. In the `webflow.properties` file, the initial steps involve looking up the catalog’s top-level categories (from in-memory cache or, if necessary, from the Commerce database). If these categories can be successfully located, the `main.jsp` template will be loaded to display them.

Sample Browser View

Figure 5-5 shows an annotated version of the `main.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 5-5 Annotated main.jsp Template Before User Login



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

1. The page header is created from an import of the `header.jsp` template. This is standard across most of the JSP templates provided by WebLogic Commerce Server. The import call is:

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

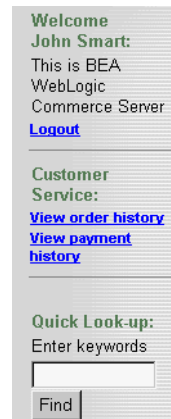
The `header.jsp` file creates the top banner and reserves space for the left-side column. The contents of the column are determined by other processing. In this example, only the `quicksearch.jsp` resides in the left column (see item 3 in this list).

2. The content in region 2 on the `main.jsp` template is generated by a series of Pipeline JSP tags that obtain the top category (in this case, the root category for the entire catalog) and use it to obtain all of its subcategories. Then, the Pipeline JSP tags display each category name in a hyperlinked list.
3. The content in region 3 depends on whether the user is logged in. (In the figure, the user had not logged in.) For that reason, only the `quicksearch.jsp` template is shown. The `quicksearch.jsp` template is always included into the `main.jsp` template. The include statement is:

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

However, if the user is logged in, the user is considered a registered customer, welcomed with a greeting, and presented with links to View Order History and View Payment History. In addition, the customer can choose to Logout of their account. Figure 5-6 shows only the left-side column when a user is logged in:

Figure 5-6 Left-column of main.jsp When the User is Logged In



4. The `main.jsp` template's content in region 4 contains the included `footer.jsp` template. The include statement is:

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

The `footer.jsp` file 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 `footer.jsp` file, the right-side vertical column is also an include file:

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

Location in the WebLogic Commerce Server Directory Structure

You can find the `main.jsp` 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\main.jsp (Windows)
$WL_COMMERCE_HOME/server/webapps/wlcs/commerce/main.jsp (UNIX)
```

Tag Library Imports

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

```
<%@ taglib uri="pipeline.tld" prefix="pipeline" %>
<%@ taglib uri="cat.tld" prefix="catalog" %>
<%@ taglib uri="um.tld" prefix="um" %>
```

Note: For more information on the Pipeline JSP tags, see [BEA WebLogic Commerce Server Webflow and Pipeline Management](#). For more information about the Catalog JSP tags, see “The Catalog JSP Tag Library: `cat.tld`” on page 5-97. For more information on the WebLogic Personalization Server's User Management 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 `main.jsp` template uses Java classes in the following packages and therefore includes these import statements:

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

5 The Product Catalog JSP Templates and Tag Library

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

Location in the Default Webflow

The `main.jsp` template is the first page you or your customers will see upon starting the WebLogic Commerce Server Web application. From this page, customers can browse the store catalog by clicking on a link to a particular category (which loads the `browse.jsp` template). Customers can also enter keywords and click the Find button to perform a Quick Look-up of a particular product item (which loads the `searchresults.jsp` template). If the customer is logged into the site, the customer can also choose to logout (which loads a different version of the `main.jsp` template), view their order history (which loads the `orderhistory.jsp` template), or view their payment history (which loads the `paymenthistory.jsp` template).

Note: For more information about the default Webflow, see Figure 5-4.

Included JSP Templates

The following JSP templates are included into the `main.jsp` template:

- `header.jsp`, which creates the top banner, and also includes the `leftside.jsp` template; the `leftside.jsp` template reserves column space for generated content that is displayed in the `main.jsp` template.
- `columnbreak.jsp`, which creates space (a vertical column) between the left-side column and the main display area.
- `quicksearch.jsp`, which provides a keyword-based search tool for finding product items via keywords that have already been assigned.
- `footer.jsp`, which creates a horizontal footer at the bottom of the page, and also includes the `rightside.jsp` template. The `rightside.jsp` file 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 link or a button on the `main.jsp` template, 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 5-1 provides information about these events and the business logic they invoke.

Table 5-1 main.jsp Events

Event	Webflow Response(s)
<code>link/logout</code>	<code>LogoutCustomerIP</code>
<code>link/viewOrderHistory</code>	<code>RefreshOrderHistory</code>
<code>link/viewPaymentHistory</code>	<code>RefreshPaymentHistory</code>
<code>link/browse</code>	<code>BrowseCategory (IP)</code> <code>MoveSiblingResults (IP)</code> <code>GetBrowseDetails</code>

Table 5-2 briefly describes each of the Pipelines from Table 5-1, as they are defined in the `pipeline.properties` file. For more information about individual Pipeline components, see “Pipeline Components” on page 5-87.

Table 5-2 main.jsp Pipelines

Pipeline	Description
<code>RefreshOrderHistory</code>	Contains <code>RefreshOrderHistoryPC</code> and is not transactional.
<code>RefreshPaymentHistory</code>	Contains <code>RefreshPaymentHistory</code> and is not transactional.
<code>GetBrowseDetails</code>	Contains <code>GetCategoryPC</code> , <code>GetParentPC</code> , <code>GetSubcategoriesPC</code> , <code>MoveAttributePC</code> , <code>GetCategoryPC</code> , <code>GetAncestorsPC</code> , <code>GetSubcategoriesPC</code> , <code>GetProductItemsPC</code> and is not transactional.

Dynamic Data Display

One purpose of the `main.jsp` template is to decide which version of the left column to display (the one with just the Quick Look-up or the one with links to customer-specific data). This is accomplished on the `main.jsp` template using a combination of Pipeline JSP tags and the WebLogic Personalization Server's User Management JSP tags.

First, the `getPipelineProperty` Pipeline JSP tag obtains the `USER_NAME` attribute from the Pipeline session. Table 5-3 provides more detailed information on this attribute.

Table 5-3 `main.jsp` Pipeline Session Attributes

Attribute	Type	Description
<code>PipelineSessionConstants.USER_NAME</code>	<code>java.lang.String</code>	The customer's username, if available.

Listing 5-4 illustrates how this attribute is obtained from the Pipeline session using the `getPipelineProperty` Pipeline JSP tag.

Listing 5-4 Obtaining the `USER_NAME` Attribute

```
<pipeline:getPipelineProperty
  propertyName="<%=PipelineSessionConstants.USER_NAME%>"
  returnName="userName"
  returnType="String" />
```

Note: For more information on the Pipeline JSP tags, see [BEA WebLogic Commerce Server Webflow and Pipeline Management](#).

Next, the `main.jsp` template checks to see if there is a value assigned to the username. If so, the `getProfile` User Management JSP tag is used to set the customer profile (context) for which the customer information should be retrieved, and the left column is displayed with links to customer-specific data. Otherwise, just the Quick Look-up is shown. This functionality is shown in Listing 5-5.

Listing 5-5 Displaying the Left Column of main.jsp

```

<% if (userName != null && userName.length() != 0) { %>
<!-- Tag to get customer profile -->
<um:getProfile
  profileKey="<%=request.getRemoteUser()%>"
  profileType="WLCS_Customer" />
<p><font color="#567856"><b>Welcome<br>
<um:getPropertyAsString propertyName="firstName" />
<um:getPropertyAsString propertyName="lastName" /></b></font></p>
<p>This is BEA WebLogic Commerce Server</p>
<p class="commentary">
<a href="<%= WebflowJSPHelper.createWebflowURL(pageContext,
  "main.jsp", "link(logout)", true) %>">Logout</a></p>
<hr size="1">
<p><font color="#567856"><b>Customer Service:</b></font></p>
<p class="commentary">
<a href="<%= WebflowJSPHelper.createWebflowURL(pageContext,
  "main.jsp", "link(viewOrderHistory)", true)%>">View order
  history</a></p>
<p class="commentary">
<a href="<%= WebflowJSPHelper.createWebflowURL(pageContext,
  "main.jsp", "link(viewPaymentHistory)", true) %>">View payment
  history</a></p>
<hr size="1">
<% } %>
<%@ include file="/commerce/catalog/includes/quicksearch.jsp" %>

```

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.

5 The Product Catalog JSP Templates and Tag Library

However, the primary purpose of the `main.jsp` template is to dynamically display product catalog data by category. This is accomplished using a combination of Pipeline and Catalog JSP tags.

First, the `getPipelineProperty` Pipeline JSP tag obtains the `CATALOG_CATEGORY` and `CATALOG_CATEGORIES` attributes from the Pipeline session. Table 5-4 provides more detailed information on these attributes.

Table 5-4 `main.jsp` Pipeline Session Attributes

Attribute	Type	Description
<code>PipelineSessionConstants.CATALOG_CATEGORY</code>	<code>com.beasys.commerce.ebusiness.catalog.Category</code>	Contains the root category for the product catalog.
<code>PipelineSessionConstants.CATALOG_CATEGORIES</code>	<code>com.beasys.commerce.ebusiness.catalog.ViewIterator</code>	Contains the top-level categories for the product catalog.

Listing 5-6 illustrates how these attributes are obtained from the Pipeline session using the `getPipelineProperty` Pipeline JSP tag.

Listing 5-6 Obtaining the `CATALOG_CATEGORY` and `CATALOG_CATEGORIES` Attributes

```
<pipeline:getPipelineProperty
  propertyName="<%=PipelineSessionConstants.CATALOG_CATEGORY%"
  returnName="topCategory"
  returnType="com.beasys.commerce.ebusiness.catalog.Category"
  attributeScope="<%=PipelineConstants.REQUEST_SCOPE%" />

<pipeline:getPipelineProperty
  propertyName="<%=PipelineSessionConstants.CATALOG_CATEGORIES%"
  returnName="subcategories"
  returnType="com.beasys.commerce.ebusiness.catalog.ViewIterator"
  attributeScope="<%= PipelineConstants.REQUEST_SCOPE %%" />
```

Note: For more information on the Pipeline JSP tags, see [BEA WebLogic Commerce Server Webflow and Pipeline Management](#).

Next, a string containing common browse parameters for the page is created, as shown in Listing 5-7. These parameters are used to establish context for the page (that is, a knowledge of previous activity) and provide the Pipelines with appropriate information during their subsequent executions.

Listing 5-7 Creating a String With Common Browse Parameters

```
<p class="head1">Store Catalog</p>
  <ul type="square">

<%-- Declare a String containing common browse parameters --%>

  <%! static final String commonParameters = "&" +
    HttpRequestConstants.CATALOG_SOURCE_KEY + "=" +
    PipelineSessionConstants.CATALOG_CATEGORIES + "&" +
    HttpRequestConstants.CATALOG_DESTINATION_KEY +
    "=wlcs_siblings&"; %>
```

Lastly, the `iterateViewIterator` Catalog JSP tag is used to iterate through all the categories (one at a time). The context for the page is captured by appending values to the previously established browse parameters, and the `getProperty` Catalog JSP tag is used to list the name of each category on the `main.jsp` template.

Listing 5-8 Displaying the Contents of the Product Catalog

```
<catalog:iterateViewIterator
  iterator="<%=subcategories%>" id="currentCategory"
  returnType="com.beasys.commerce.ebusiness.catalog.Category">

<% String browseParameters = commonParameters +
  HttpRequestConstants.CATALOG_CATEGORY_ID + "=" +
  currentCategory.getKey().getIdentifier(); %>

<div class="head2">

  <li>

    <a href="<%=WebflowJSPHelper.createWebflowURL(pageContext,
      "main.jsp", "link(browse)", browseParameters, true) %>">

    <catalog:getProperty object="<%=currentCategory%>"
      propertyName="Name" />
```

5 *The Product Catalog JSP Templates and Tag Library*

```
        </a><br>
    </div>
</catalog:iterateViewIterator>
</ul>
```

Note: For more information about the Catalog JSP tags, see “The Catalog JSP Tag Library: cat.tld” on page 5-97.

Form Field Specification

No form fields are used in the `main.jsp` template.

browse.jsp Template

In the hierarchical product catalog, the `browse.jsp` template can take on two different forms, both of which are used by customers to browse for product items. These forms are as follows:

- The product catalog reads a unique `category.jsp` template from the database for the current category being browsed, and includes it into the `browse.jsp` template. The `category.jsp` presents the current level of available categories as hyperlinks. Additionally, a hyperlinked list of sibling categories is made available beneath the Quick Look-up search field in the left column (the current category is highlighted). An ancestor category navigation bar (such as [Home](#) → [D](#)) is also displayed at the top of the page. Figure 5-7 shows a screen shot of this `browse.jsp` template form.

Figure 5-7 `browse.jsp` Template - D Subcategory Display



- Once a customer reaches the end of the catalog hierarchy (that is, when a category contains product items instead of more subcategories), the catalog reads the appropriate number of `itemsummary.jsp` templates from the database and includes them into the `category.jsp` template. In this form of the

5 *The Product Catalog JSP Templates and Tag Library*

`browse.jsp` template, the `itemsummary.jsp` templates replace the hyperlinked list of categories. The left column still contains the current category (highlighted), and the ancestor category navigation bar is also shown at the top of the page. Figure 5-8 illustrates this nesting of JSP templates within the `browse.jsp` template.

Figure 5-8 Hierarchical Relationship of browse.jsp and Other JSPs

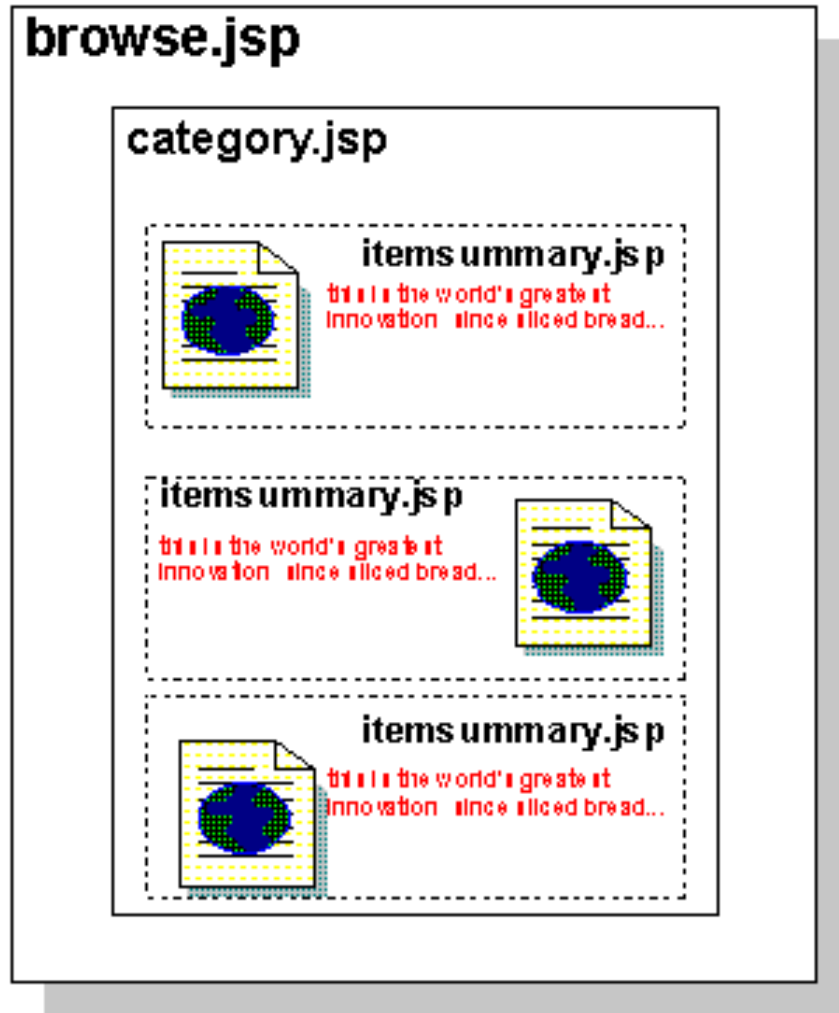
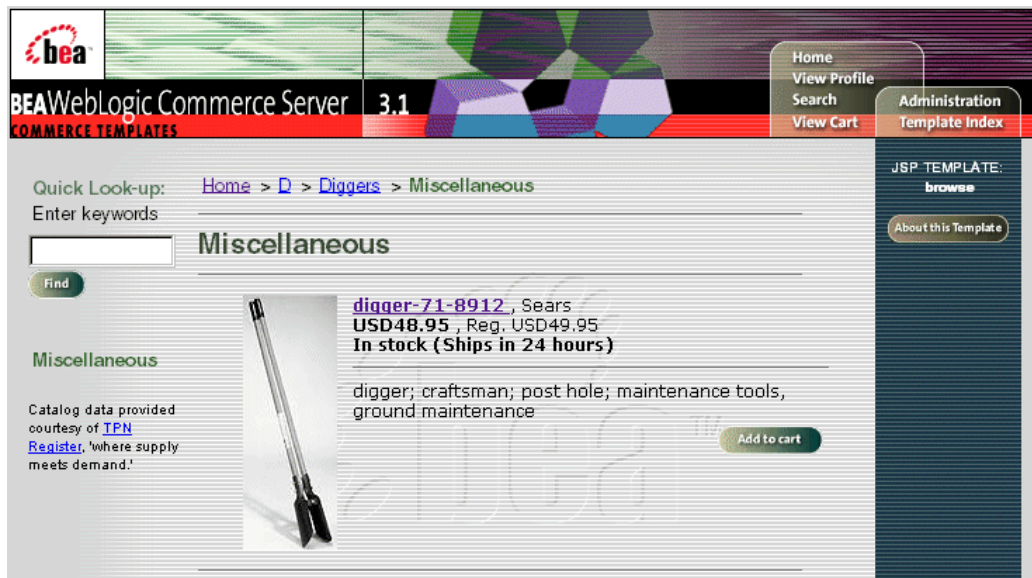


Figure 5-9 shows a screen shot of this `browse.jsp` template form, with one included `itemssummary.jsp`.

Figure 5-9 browse.jsp Template - D > Diggers > Miscellaneous Category With Item Summary Display



Note: Each item in the catalog can be assigned a different item summary JSP, allowing you to customize the layout for each type of item. This assignment can be made on the catalog’s administration screen. For more information, see the section “How Are Categories and Items Displayed to the Web Site User?” on page 4-36.

Sample Browser View

Figure 5-10 shows an annotated version of a `browse.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 5-10 Annotated browse.jsp Template



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

1. The page header is created from an import of the `header.jsp` template. This is standard across most of the JSP templates provided by WebLogic Commerce Server. The import call is:

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

The `header.jsp` file creates the top banner and reserves space for the left-side column. The contents of the column are determined by other processing.

2. Region 2 of the `browse.jsp` template contains the `quicksearch.jsp` template that is included. This template provides a keyword-search feature. The include statement is:

5 The Product Catalog JSP Templates and Tag Library

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

3. Region 3 of the `browse.jsp` template contains the results of processing that checks for sibling categories and displays them with hyperlinks to those categories. In this example, the Pack category does not have any sibling categories.
4. Region 4 of the `browse.jsp` template includes `navigation.jsp`, which builds a hierarchical browse list for the catalog's categories from the top-level Home category down to the current category. The call to include this JSP is:

```
<jsp:include page="/commerce/catalog/includes/navigation.jsp"
flush="true"/>
```

5. Regions 5 and 6 of the `browse.jsp` template shows the generated results of processing with the Webflow and Pipeline mechanisms that found the values for this category (presented in the included `category.jsp` template). The `category.jsp` template, in turn, obtains the values that return the item summary data (displayed by the `itemsummary.jsp` templates) for each product item in this category.
6. Region 7 of the `browse.jsp` template contains the included `footer.jsp` template. The include statement is:

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

The `footer.jsp` file consists of the horizontal footer at the bottom of the page (not shown in the figure), plus the right-side vertical column that describes the name of the current template and links to its *About* information. In the `footer.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 `browse.jsp` 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\catalog\
browse.jsp (Windows)
$WL_COMMERCE_HOME/server/webapps/wlcs/commerce/catalog/
browse.jsp (UNIX)
```

Tag Library Imports

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

```
<%@ taglib uri="pipeline.tld" prefix="pipeline" %>
<%@ taglib uri="cat.tld" prefix="catalog" %>
```

Note: For more information on the Pipeline JSP tags, see *BEA WebLogic Commerce Server Webflow and Pipeline Management*. For more information about the Catalog JSP tags, see “The Catalog JSP Tag Library: `cat.tld`” on page 5-97.

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

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

Location in the Default Webflow

The `browse.jsp` template is displayed (with the included `category.jsp` template) when a customer clicks on a link for one of the categories shown on the `main.jsp` template. The `browse.jsp` template is redisplayed (with different content using the included `category.jsp` template) each time a customer clicks on a subcategory link. It is also displayed when a customer selects a sibling link from the left-side column. The `browse.jsp` continues to be displayed until the customer arrives at item summaries (shown by the `category.jsp` template’s included `itemsummary.jsp` templates). From there, the customer can choose to view more details about an item (which loads the `details.jsp` template), or add the item to their shopping cart (which loads the `shoppingcart.jsp` template).

Customers can also still enter keywords and click the Find button to perform a Quick Look-up of a particular product item or category (which loads the `searchresults.jsp` template). If the customer is logged into the site, the customer can also choose to logout (which loads the generic version of the `main.jsp` template), view their order history (which loads the `orderhistory.jsp` template), or view their payment history (which loads the `paymenthistory.jsp` template).

Note: For more information about the default Webflow, see Figure 5-4.

Included JSP Templates

The following JSP templates are included into the `browse.jsp` template:

- `header.jsp`, which creates the top banner, and also includes the `leftside.jsp` template; the `leftside.jsp` template reserves column space for generated content that is displayed in `main.jsp`.
- `quicksearch.jsp`, which provides a keyword-based search tool for finding product items via keywords that have already been assigned.
- `columnbreak.jsp`, which creates space (a vertical column) between the left-side column and the main display area.
- `navigation.jsp`, which builds a hierarchical browse list for the catalog's categories from the top-level Home category down to the current category. For more information about the `navigation.jsp` template, see "About the Included `navigation.jsp` Template" on page 5-33.
- `category.jsp`, which displays links to the current category's subcategories, and also includes the `itemsummary.jsp` template (if particular items are available). For more details about the `category.jsp` template, see "About the Included `category.jsp` Template" on page 5-35. For more details about the `itemsummary.jsp` template, see "About the Included `itemsummary.jsp` Template" on page 5-39.
- `footer.jsp`, which creates a horizontal footer at the bottom of the page, and also includes the `rightside.jsp` template. The `rightside.jsp` file describes (for the benefit of you and your development team) the name of the current template and links to its *About* information.

About the Included navigation.jsp Template

The `navigation.jsp` template (included in the `browse.jsp` template) is responsible for generating the ancestor category navigation bar that is shown at the top of the page. The `navigation.jsp` template utilizes Pipeline, Catalog, and the WebLogic Personalization Server's Utility JSP tags to generate this content.

First, the `getPipelineProperty` Pipeline JSP tag is used to obtain the current category (that is, the `CATALOG_CATEGORY` attribute) from the Pipeline session, as shown in Listing 5-9.

Listing 5-9 Obtaining the CATALOG_CATEGORY Attribute

```
<pipeline:getPipelineProperty
  propertyName="<%=PipelineSessionConstants.CATALOG_CATEGORY%>"
  returnName="category"
  returnType="com.beasys.commerce.ebusiness.catalog.Category"
  attributeScope="<%= PipelineConstants.REQUEST_SCOPE %>"/>
```

Note: For more information on the Pipeline JSP tags, see [BEA WebLogic Commerce Server Webflow and Pipeline Management](#).

Next, the `<es>` JSP tag (one of the WebLogic Personalization Server Utility JSP tags) is used to ensure that the conditions under which the ancestor category navigation bar is displayed are appropriate. If so, the category's ancestors are obtained from the Pipeline session (again using the `getPipelineProperty` Pipeline JSP tag).

Listing 5-10 Establishing Conditional Display of the Navigation Bar and Obtaining the Category's Ancestors

```
<!-- Only output the navigation bar if a current category exists in
the PipelineSession --%>

<es:notNull item="<%=category%>">

  <!-- Get the category's ancestors from the PipelineSession --%>

  <pipeline:getPipelineProperty
    propertyName="<%=PipelineSessionConstants.CATALOG_ANCESTORS%>"
    returnName="ancestors"
```

5 The Product Catalog JSP Templates and Tag Library

```
returnType="com.beasys.commerce.ebusiness.catalog.Category[]"
attributeScope="<%=PipelineConstants.REQUEST_SCOPE%"/>
```

Note: For more information on the Pipeline JSP tags, see *BEA WebLogic Commerce Server Webflow and Pipeline Management*. For more information on the WebLogic Personalization Server's Utility JSP tags, see "JSP Tag Reference" in the *BEA WebLogic Personalization Server* documentation.

Browse parameters for the `navigation.jsp` template are concatenated into a single string, and a link is created for each category ancestor. These parameters are used to establish context for the page (that is, a knowledge of previous activity) and provide the Pipelines with appropriate information during their subsequent executions. In the case of the last ancestor, a link to the main catalog page is also created. This is accomplished with the `<es>` WebLogic Personalization Server Utility JSP tag and the `getProperty` Catalog JSP tag, as shown in Listing 5-11.

Listing 5-11 Generating the Hierarchical Category Navigation Bar

```
<%-- Declare a String containing common browse parameters --%>

<%! static final String commonParameters = "&" +
    HttpRequestConstants.CATALOG_SOURCE_KEY + "=" +
    PipelineSessionConstants.CATALOG_CATEGORIES + "&" +
    HttpRequestConstants.CATALOG_DESTINATION_KEY +
    "=wlcs_siblings&"; %>

<%-- Iterate through all the category's ancestors, creating a
browse link for each --%>

<es:forEachInArray id="ancestor"
    type="com.beasys.commerce.ebusiness.catalog.Category"
    array="<%=ancestors%>" counterId="i">

    <%-- Add a link to the main catalog page in the case of the last
    ancestor --%>

        <% if (i.intValue() == 0) { %>
            <p><a href="<%= WebflowJSPHelper.createWebflowURL
                (pageContext, "navigation.jsp", "link(home)", true)
                %>">Home</a>

            <%-- Otherwise, link to the browse page for the current
            ancestor --%>
```



```

<% } else { %>
  <a href="<%= WebflowJSPHelper.createWebflowURL
    (pageContext, "navigation.jsp", "link(browse)",
    commonParameters + HttpRequestConstants.CATALOG_CATEGORY_ID
    + "=" + ancestor.getKey().getIdentifier(), true) %>">

    <catalog:getProperty object="<%=ancestor%>"
      propertyName="Name" /></a>

<% } %>

  &nbsp;<b>&gt;</b>

</es:forEachInArray>

<!-- Insert the category name -->

  <b><font color="#33632B">
    <catalog:getProperty object="<%=category%>"
      propertyName="Name" />
  </font></b></p>

</es:NotNull>

```

Note: For more information about the WebLogic Personalization Server's Utility JSP tags, see ["JSP Tag Reference"](#) in the *BEA WebLogic Personalization Server* documentation. For more information about the Catalog JSP tags, see ["The Catalog JSP Tag Library: cat.tld"](#) on page 5-97.

About the Included category.jsp Template

The `category.jsp` template (included in the `browse.jsp` template) provides a standardized format for the display of hyperlinked subcategories. In the WebLogic Commerce Server Web application, this format is a simple list. However, you can always modify the template to use a different format.

The `category.jsp` template utilizes Pipeline and Catalog JSP tags to generate the specialized content displayed in the `browse.jsp` template. This is accomplished by first obtaining the current catalog category and its subcategories from the Pipeline session using the `getPipelineProperty` Pipeline JSP tag, as shown in Listing 5-12.

Listing 5-12 Obtaining the CATALOG_CATEGORY and CATALOG_CATEGORIES Attributes

```
<!-- Get the current category from the PipelineSession -->

<pipeline:getPipelineProperty
  propertyName="<%=PipelineSessionConstants.CATALOG_CATEGORY%"
  returnName="category"
  returnType="com.beasys.commerce.ebusiness.catalog.Category"
  attributeScope="<%=PipelineConstants.REQUEST_SCOPE%" />

<!-- Get the subcategories from the PipelineSession -->

<pipeline:getPipelineProperty
  propertyName="<%=PipelineSessionConstants.CATALOG_CATEGORIES%"
  returnName="subcategories"
  returnType="com.beasys.commerce.ebusiness.catalog.ViewIterator"
  attributeScope="<%=PipelineConstants.REQUEST_SCOPE%" />
```

Note: For more information on the Pipeline JSP tags, see [BEA WebLogic Commerce Server Webflow and Pipeline Management](#).

Next, a string containing common browse parameters for the page is created. These parameters are used to establish context for the page (that is, a knowledge of previous activity) and provide the Pipelines with appropriate information during their subsequent executions. The `category.jsp` template then displays the current category name using the `getProperty` Catalog JSP tag, as shown in Listing 5-13.

Listing 5-13 Generating Browse Parameters and Inserting the Category Name

```
<%! static final String commonParameters = "&" +
  HttpRequestConstants.CATALOG_SOURCE_KEY + "=" +
  PipelineSessionConstants.CATALOG_CATEGORIES + "&" +
  HttpRequestConstants.CATALOG_DESTINATION_KEY +
  "=wlcs_siblings&"; %>

<p class="head1">
  <catalog:getProperty object="<%=category%">"propertyName="Name"/>
</p>
```

Note: For more information about the Catalog JSP tags, see “The Catalog JSP Tag Library: cat.tld” on page 5-97.

Any subcategories associated with the category are then displayed as hyperlinks that allow the customer to browse further. This is done using using the `iterateViewIterator` Catalog JSP tag, as shown in Listing 5-14.

Listing 5-14 Displaying Hyperlinked Subcategories

```
<center>
<table cellpadding="3" border="0" width="90%">

  <catalog:iterateViewIterator
    iterator="<%=subcategories%>"
    id="subcategory"
    returnType="com.beasys.commerce.ebusiness.catalog.Category">

  <tr><td width="30%" valign="top">
    <p class="tabletext">
      <a href="<%=WebflowJSPHelper.createWebflowURL(pageContext,
        "category.jsp", "link(browse)", commonParameters +
        HttpRequestConstants.CATALOG_CATEGORY_ID + "=" +
        subcategory.getKey().getIdentifier(), true) %>">

      <catalog:getProperty object="<%=subcategory%>"
        propertyName="Name" /></a></p>

    </td></tr>

  </catalog:iterateViewIterator>

</table>
</center>
```

Note: For more information about the Catalog JSP tags, see “The Catalog JSP Tag Library: cat.tld” on page 5-97.

If there are any individual items at this level in the hierarchy, the `category.jsp` template retrieves them using the `getPipelineProperty` Pipeline JSP tag, and sets the view. The view is basically a pointer that indicates the location in the complete list of results where we want to start displaying information. This processing is shown in Listing 5-15.

5 The Product Catalog JSP Templates and Tag Library

Listing 5-15 Obtaining Individual Product Items and Setting the View

```
<pipeline:getPipelineProperty
  propertyName="<%=PipelineSessionConstants.CATALOG_ITEMS%>"
  returnName="items"
  returnType="com.beasys.commerce.ebusiness.catalog.ViewIterator"
  attributeScope="<%=PipelineConstants.REQUEST_SCOPE%>" />

<% String viewIndexString = (String)request.getParameter(HttpRequestConstants.
  CATALOG_VIEW_INDEX); %>

<% if (viewIndexString == null) { viewIndexString = "0"; } %>

<% int viewIndex = Integer.valueOf(viewIndexString).intValue(); %>
```

Note: For more information on the Pipeline JSP tags, see [BEA WebLogic Commerce Server Webflow and Pipeline Management](#).

Lastly, if individual product items were obtained, the `category.jsp` template iterates through each item using the `iterateThroughView` and `getProperty` Catalog JSP tags, and includes an `itemsummary.jsp` template to display the information related to each item, as shown in Listing 5-16.

Listing 5-16 Iterating Through and Displaying Product Item Information

```
<catalog:iterateThroughView iterator="<%=items%>" id="item"
  returnType="com.beasys.commerce.ebusiness.catalog.ProductItem"
  viewIndex="<%= viewIndex %>">

<%-- Add the required parameters for the included JSP to the request --%>

  <% request.setAttribute("product_item", item); %>
  <% request.setAttribute("details_link", "details"); %>

<%-- Get the summary JSP from the current product item --%>

  <catalog:getProperty object="<%=item%>" propertyName="Jsp"
    getterArgument="<%= new Integer(ProductItem.SUMMARY_DISPLAY_JSP_INDEX) %>"
    id="summaryJsp" returnType="com.beasys.commerce.ebusiness.catalog.JspInfo"/>

<%-- Included the summary JSP --%>

  <jsp:include page="<%=summaryJsp.getUrl()%>" flush="true"/>
```

```
</catalog:iterateThroughView>
```

Note: For more information about the Catalog JSP tags, see “The Catalog JSP Tag Library: cat.tld” on page 5-97. For more information about the `itemsummary.jsp` template, see the following section.

About the Included `itemsummary.jsp` Template

The `itemsummary.jsp` template (included in the `category.jsp` template) provides a standardized format for the display of specific product items. In the WebLogic Commerce Server Web application, this format contains an image, a link to more details about the item (that is, to the `details.jsp` template), some brief information about the item, and an Add to Cart button. However, you can always modify the template to use a different format.

The `itemsummary.jsp` template utilizes a combination of Pipeline, Catalog, and the WebLogic Personalization Server’s Utility JSP tags to generate the specialized content displayed for each item within the `category.jsp` template. This is accomplished by first obtaining all required parameters from the request object, and then obtaining the current catalog category (that is, the `CATALOG_CATEGORY` attribute) from the Pipeline session using the `getPipelineProperty` Pipeline JSP tag, as shown in Listing 5-17.

Listing 5-17 Obtaining Request Object Parameters and the `CATALOG_CATEGORY` Attribute

```
<% ProductItem productItem =
    (ProductItem)request.getAttribute("product_item"); %>

<% String detailsLink =
    (String)request.getAttribute("details_link"); %>

<%-- Get the current category from the PipelineSession --%>

<pipeline:getPipelineProperty
    propertyName="<%=PipelineSessionConstants.CATALOG_CATEGORY%>"
    returnName="category"
    returnType="com.beasys.commerce.ebusiness.catalog.Category"
    attributeScope="<%=PipelineConstants.REQUEST_SCOPE%>" />
```

5 The Product Catalog JSP Templates and Tag Library

Note: For more information on the Pipeline JSP tags, see *BEA WebLogic Commerce Server Webflow and Pipeline Management*.

Next, each piece of descriptive information for the item is displayed, using the `getProperty` Catalog JSP tag and the `<es>` WebLogic Personalization Server Utility JSP tag, as shown in Listing 5-18.

Listing 5-18 Displaying Product Item Information

```
<%-- Add the small image --%>

<catalog:getProperty object="<%=productItem%>"
  propertyName="Image"
  getterArgument="<%=new Integer(ProductItem.SMALL_IMAGE_INDEX)%>"
  id="smallImage"
  returnType="com.beasys.commerce.ebusiness.catalog.ImageInfo"/>

<td valign="top">
  
</td>

<td align="left" valign="top" width="90%">

  <%-- Add the item name and creator --%>
  <%-- Create the details link --%>

  <% String detailsUrl = null; %>
  <es:isNull item="<%=category%>">

    <% detailsUrl = WebflowJSPHelper.createWebflowURL(pageContext,
      "itemssummary.jsp", "link(" + detailsLink + ")", "&" +
      HttpRequestConstants.CATALOG_ITEM_SKU + "=" +
      productItem.getKey().getIdentifier(), true);%>

  </es:isNull>

  <es:notNull item="<%=category%>">

    <% detailsUrl = WebflowJSPHelper.createWebflowURL(pageContext,
      "itemssummary.jsp", "link(" + detailsLink + ")", "&" +
      HttpRequestConstants.CATALOG_ITEM_SKU + "=" +
      productItem.getKey().getIdentifier() + "&" +
      HttpRequestConstants.CATALOG_CATEGORY_ID + "=" +
      category.getKey().getIdentifier(), true);%>

  </es:notNull>
```

```

<div class="tabletext"><b>
  <a href="<%=detailsUrl%>">
    <catalog:getProperty object="<%=productItem%>"
      propertyName="Name" />
  </a>

</b>,

  <catalog:getProperty object="<%=productItem%>"
    propertyName="Creator" />

</div>

<%-- Add the item price --%>

<div class="tabletext"><b>

  <catalog:getProperty object="<%=productItem%>"
    propertyName="CurrentPrice" id="price"
    returnType="com.beasys.commerce.axiom.units.Money" />

  <%= price.getCurrency() %>
  <%= WebflowJSPHelper.priceFormat(price.getValue()) %>

</b>, Reg.

  <catalog:getProperty object="<%=productItem%>"
    propertyName="Msrp" id="msrp"
    returnType="com.beasys.commerce.axiom.units.Money" />

  <%= msrp.getCurrency() %>
  <%= WebflowJSPHelper.priceFormat(msrp.getValue()) %>

</b></div>

<%-- Add inventory information --%>

<catalog:getProperty object="<%=productItem%>"
  propertyName="Availability" id="inventory"
  returnType="com.beasys.commerce.ebusiness.catalog.InventoryInfo"
  />

<div class="tabletext"><b>

  <% if (inventory.getInStock()) { %>
    In stock (<%=inventory.getShippingTime()%>)
  <% } else { %>
    Out of stock.
  <% } %>

</b></div>

```

5 The Product Catalog JSP Templates and Tag Library

```
<!-- Add a short description of the item -->

<hr size="1"><div class="tabletext">

  <catalog:getProperty object="<%=productItem%>"
    propertyName="Description"
    getterArgument="<%=new Integer
      (CatalogItem.SHORT_DESCRIPTION_INDEX)%>" />

</div>

<!-- Add the 'Add to Cart' link -->

<div align="right">

  <a href="<%=WebflowJSPHelper.createWebflowURL(pageContext,
    "itemssummary.jsp", "link(add)", "&" +
    HttpRequestConstants.CATALOG_ITEM_SKU + "=" +
    productItem.getKey().getIdentifier(), true) %>">
    " alt="Add To Shopping Cart" border="0"
      vspace="4"></a>

</div>

</td>
```

Note: For more information about the Catalog JSP tags, see “The Catalog JSP Tag Library: cat.tld” on page 5-97. For more information about the WebLogic Personalization Server’s Utility JSP tags, see “[JSP Tag Reference](#)” in the *BEA WebLogic Personalization Server* documentations.

Events

Every time a customer clicks a link or a button on the `browse.jsp` template, 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 5-5 provides information about these events and the business logic they invoke.

Table 5-5 browse.jsp (and included category.jsp) Events

Event	Webflow Response(s)
link(browse)	BrowseCategory (IP) MoveSiblingResults (IP) GetBrowseDetails

Because the `category.jsp` template also includes the `itemsummary.jsp` template, the events shown in Table 5-6 are also considered part of the `browse.jsp` template.

Table 5-6 itemsummary.jsp Events

Event	Webflow Response(s)
link(details)	GetProductItemDetails (IP)
link(add)	AddProductItemToShoppingCart (IP)

Table 5-7 briefly describes each of the Pipelines from Table 5-5, as they are defined in the `pipeline.properties` file. For more information about individual Pipeline components, see “Pipeline Components” on page 5-87.

Table 5-7 browse.jsp Pipelines

Pipeline	Description
GetBrowseDetails	Contains <code>GetCategoryPC</code> , <code>GetParentPC</code> , <code>GetSubcategoriesPC</code> , <code>MoveAttributePC</code> , <code>GetCategoryPC</code> , <code>GetAncestorsPC</code> , <code>GetSubcategoriesPC</code> , <code>GetProductItemsPC</code> and is not transactional.

Dynamic Data Display

The primary purpose of the `browse.jsp` template is to dynamically display content based on the hyperlinked path the customer chooses to follow. As previously described, this is accomplished mostly through the included `category.jsp` and `itemsummary.jsp` templates. (For more information, see “About the Included `category.jsp` Template” on page 5-35 and “About the Included `itemsummary.jsp` Template” on page 5-39, respectively.)

However, there is still some dynamic data that is handled solely by the `browse.jsp` template; that is, the list of sibling categories in the left column. First, the `getPipelineProperty` Pipeline JSP tag retrieves the `CATALOG_CATEGORY` and `wlcs_siblings` attributes from the Pipeline session. Table 5-8 provides more detailed information on these attributes.

Table 5-8 `browse.jsp` Pipeline Session Attributes

Attribute	Type	Description
<code>PipelineSessionConstants.CATALOG_CATEGORY</code>	<code>com.beasys.commerce.ebusiness.catalog.Category</code>	Contains the root category for the product catalog.
<code>wlcs_siblings</code>	<code>com.beasys.commerce.ebusiness.catalog.ViewIterator</code>	Contains the siblings for a given category.

Listing 5-19 illustrates how these attributes are retrieved from the Pipeline session using the `getPipelineProperty` Pipeline JSP tag, along with the category siblings.

Listing 5-19 Obtaining the `CATALOG_CATEGORY` and `wlcs_siblings` Attributes

```
<!-- sibling categories (or subcategories) with the current
category highlighted to indicate state -->

<%-- Get the current category from the PipelineSession --%>

<pipeline:getPipelineProperty
  propertyName="<%=PipelineSessionConstants.CATALOG_CATEGORY%>"
  returnName="category"
  returnType="com.beasys.commerce.ebusiness.catalog.Category"
  attributeScope="<%= PipelineConstants.REQUEST_SCOPE %>" />
```

```
<%-- Get the siblings from the PipelineSession --%>

<pipeline:getPipelineProperty
  propertyName="wlcs_siblings"
  returnName="siblings"
  returnType="com.beasys.commerce.ebusiness.catalog.ViewIterator"
  attributeScope="<%= PipelineConstants.REQUEST_SCOPE %>" />
```

Note: For more information on the Pipeline JSP tags, see [BEA WebLogic Commerce Server Webflow and Pipeline Management](#).

Next, a string containing common browse parameters for the page is created. These parameters are used to establish context for the page (that is, a knowledge of previous activity) and provide the Pipelines with appropriate information during their subsequent executions. The `iterateViewIterator` and `getPropertyCatalog` JSP tags are then used to iterate through the siblings and create browse hyperlinks for each of them (if appropriate), as shown in Listing 5-20. This activity happens prior to any calls to the included `category.jsp` and/or `itemsummary.jsp` templates.

Listing 5-20 Establishing Common Browse Parameters and Creating Browse Links for Categories

```
<%-- Declare a String containing common browse parameters --%>

<%! static final String commonParameters = "&" +
HttpRequestConstants.CATALOG_SOURCE_KEY + "=" +
PipelineSessionConstants.CATALOG_CATEGORIES + "&" +
HttpRequestConstants.CATALOG_DESTINATION_KEY + "=wlcs_siblings&";
%>

<%-- Iterate through all siblings, creating a browse link for each.
--%>

<p>&nbsp;</p>

<catalog:iterateViewIterator iterator="<%=siblings%>" id="sibling"
  returnType="com.beasys.commerce.ebusiness.catalog.Category">

<%-- Just highlight the category name if the current sibling is the
current category --%>

<% if (sibling.getKey().equals(category.getKey())) { %>
  <p><b><font color="#33632B">
```

5 The Product Catalog JSP Templates and Tag Library

```
<catalog:getProperty object="<%=sibling%" propertyName="Name"/>
</font></b></p>

<% } else { %>

<!-- Otherwise, link to the browse page for the current sibling -->

<p><a href="<%=WebflowJSPHelper.createWebflowURL(pageContext,
"browse.jsp", "link(browse)", commonParameters +
HttpRequestConstants.CATALOG_CATEGORY_ID + "=" +
sibling.getKey().getIdentifier(), true) %>">

<catalog:getProperty object="<%=sibling%"
propertyName="Name"/></a></p>

<% } %>

</catalog:iterateViewIterator>
```

Note: For more information about the Catalog JSP tags, see “The Catalog JSP Tag Library: cat.tld” on page 5-97.

Form Field Specification

No form fields are used in the `browse.jsp` template, nor in the `browse.jsp` template’s included `category.jsp` or `itemsummary.jsp` templates.

details.jsp Template

The brief description presented for each product item on the generated `itemsummary.jsp` templates includes a hyperlink to the item. When customers click this link, the browser loads the `details.jsp` template, which customer can use to view more detailed information about the item. Although the WebLogic Commerce Server Web application presents the same information on the `itemsummary.jsp` template and the `details.jsp` template, you can use this page separation to customize the content for your customers.

Because the name of the detailed display JSP is loaded from the database (that is, it does not have to always be `details.jsp`), you can have different display JSPs for different product catalog items. For example, you can provide custom display JSPs to include seasonal or promotional text. The product catalog administrator simply needs to switch between the detailed display JSPs (once they have been tested) using the Administration Tools.

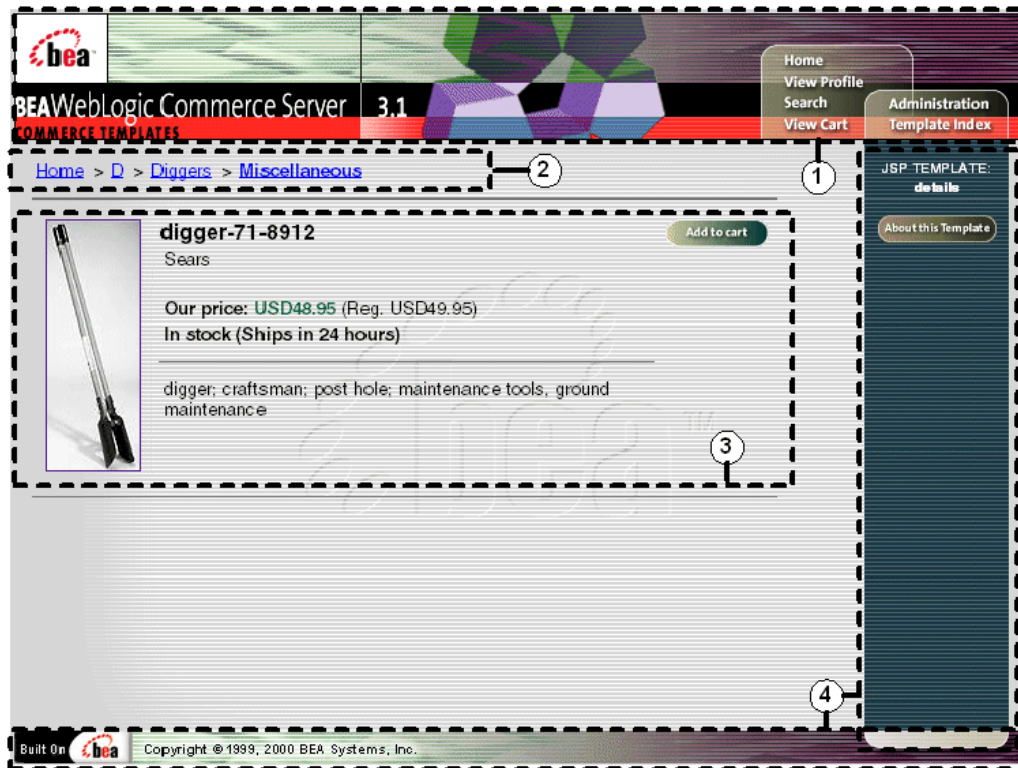
Notes: For more information about the `itemsummary.jsp` template, see “About the Included `itemsummary.jsp` Template” on page 5-39.

For more information about the product catalog Administration Tools, see Chapter 4, “Catalog Administration Tasks.”

Sample Browser View

Figure 5-11 shows an annotated version of a `details.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 5-11 Annotated details.jsp Template



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

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

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

2. Region 2 of the details.jsp template includes navigation.jsp, which builds a hierarchical browse list for the catalog's categories from the top-level Home category down to the current category. The call to include this JSP is:

```
<jsp:include page="/commerce/catalog/includes/navigation.jsp" flush="true"/>
```

3. Region 3 of the `details.jsp` template shows the results generated from processing with the Webflow and Pipeline mechanisms that found the detailed information for this particular product item (presented in the included `itemdetails.jsp` template).
4. Region 4 of the `details.jsp` template contains the included `footer2.jsp` template. The include statement is:

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

The `footer2.jsp` file 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 `details.jsp` 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\catalog\  
details.jsp (Windows)  
$WL_COMMERCE_HOME/server/webapps/wlcs/commerce/catalog/  
details.jsp (UNIX)
```

Tag Library Imports

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

```
<%@ taglib uri="pipeline.tld" prefix="pipeline" %>  
<%@ taglib uri="cat.tld" prefix="catalog" %>
```

Note: For more information on the Pipeline JSP tags, see [BEA WebLogic Commerce Server Webflow and Pipeline Management](#). For more information about the Catalog JSP tags, see “The Catalog JSP Tag Library: `cat.tld`” on page 5-97.

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

5 The Product Catalog JSP Templates and Tag Library

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

Java Package Imports

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

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

Location in the Default Webflow

The `details.jsp` template is displayed when a customer clicks on a link for a particular product item shown on an `itemsummary.jsp` template (part of the `browse.jsp` or `search.jsp` templates). From the `details.jsp` template, the customer can choose to browse back up the ancestor category navigation bar (which loads the `browse.jsp` template) or add the product item to their shopping cart (which loads the `shoppingcart.jsp` template).

Note: For more information about the default Webflow, see Figure 5-4.

Included JSP Templates

The following JSP templates are included into the `details.jsp` template:

- `header2.jsp`, which creates the top banner, and also includes the `leftside.jsp` template; the `leftside.jsp` template reserves column space for generated content that is displayed in `details.jsp`.
- `navigation2.jsp`, which builds a hierarchical browse list for the catalog's categories from the top-level Home category down to the current category (only if previous page was the `browse.jsp` template). The `navigation2.jsp` template is similar to the `navigation.jsp` template, except that it also adds a link for the top-level category. For more information about the `navigation.jsp` template, see "About the Included `navigation.jsp` Template" on page 5-33.

- `itemdetails.jsp`, which displays the detailed information about the selected product item. For more details about the `itemdetails.jsp` template, see “About the Included `itemdetails.jsp` Template” on page 5-51.
- `footer2.jsp`, which creates a horizontal footer at the bottom of the page, and also includes the `rightside.jsp` template. The `rightside.jsp` file describes (for the benefit of you and your development team) the name of the current template and links to its *About* information.

About the Included `itemdetails.jsp` Template

The `itemdetails.jsp` template (included in the `details.jsp` template) provides a standardized format for the display of specific item information. In the WebLogic Commerce Server Web application, this format is the same as what is shown in the `itemsummary.jsp` templates, with some exceptions. In the `itemdetails.jsp` template, for example, there is a link to a larger version of the image, and no link to the product item itself. (For more information about `itemsummary.jsp` templates, see “About the Included `itemsummary.jsp` Template” on page 5-39.) Remember, you can always modify the template to use a different format that better suits your requirements.

Events

Every time a customer clicks a link or button on a JSP, it is considered an event. Events trigger particular responses in the default Webflow that allow customers 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.

Because the `details.jsp` template includes the `itemdetails.jsp` template (which is used to display most of the information), Table 5-9 provides information about the only event for the included `itemdetails.jsp` template, and the business logic it invokes.

Table 5-9 `itemdetails.jsp` Events

Event	Webflow Response(s)
<code>link(add)</code>	<code>AddProductItemToShoppingCart (IP)</code> <code>AddProductItemToShoppingCart</code> <code>RefreshSavedList</code>

5 The Product Catalog JSP Templates and Tag Library

Table 5-10 briefly describes each of the Pipelines from Table 5-9, as they are defined in the `pipeline.properties` file. For more information about individual Pipeline components, see “Pipeline Components” on page 5-87.

Table 5-10 `itemdetails.jsp` Pipelines

Pipeline	Description
<code>AddProductItemToShoppingCart</code>	Contains <code>GetProductItemPC</code> and <code>AddProductItemToShoppingCartPC</code> , and is not transactional.
<code>RefreshSavedList</code>	Contains <code>RefreshSavedListPC</code> and is not transactional.

Dynamic Data Display

The primary purpose of the `details.jsp` template is to dynamically display content about a customer-selected the product item. As previously described, this is accomplished mostly through the included `itemdetails.jsp` template. (For more information, see “About the Included `itemdetails.jsp` Template” on page 5-51.)

However, there is still some dynamic data that is handled solely by the `details.jsp` template. After an include to `navigation2.jsp` (which generates the ancestor category navigation bar at the top of the page), the `details.jsp` template obtains the `CATALOG_ITEM` and `CATALOG_CATEGORY` attributes from the Pipeline session. Table 5-11 provides more detailed information on these attributes.

Table 5-11 `details.jsp` Pipeline Session Attributes

Attribute	Type	Description
<code>PipelineSessionConstants.CATALOG_CATEGORY</code>	<code>com.beasys.commerce.ebusiness.catalog.Category</code>	Contains the root category for the product catalog.
<code>PipelineSessionConstants.CATALOG_CATEGORIES</code>	<code>com.beasys.commerce.ebusiness.catalog.ViewIterator</code>	Contains the top-level categories for the product catalog.

Listing 5-21 illustrates how these attributes are obtained from the Pipeline session using the `getPipelineProperty` Pipeline JSP tag. It also shows how the required request parameters for the included JSP are added to the request. This activity happens prior to any calls to the included `itemdetails.jsp` template.

Listing 5-21 Obtaining the CATALOG_CATEGORY and CATALOG_CATEGORIES Attributes

```
<%-- Get the item from the PipelineSession --%>

<pipeline:getPipelineProperty
  propertyName="<%=PipelineSessionConstants.CATALOG_ITEM%>"
  returnName="item"
  returnType="com.beasys.commerce.ebusiness.catalog.ProductItem"
  attributeScope="<%= PipelineConstants.REQUEST_SCOPE %>" />

<%-- Get the category from the PipelineSession --%>

<pipeline:getPipelineProperty
  propertyName="<%=PipelineSessionConstants.CATALOG_CATEGORY%>"
  returnName="category"
  returnType="com.beasys.commerce.ebusiness.catalog.Category"
  attributeScope="<%= PipelineConstants.REQUEST_SCOPE %>" />

<%-- Add the required parameters for the included JSP to the request --%>

<% request.setAttribute("product_item", item); %>
<% request.setAttribute("category", category); %>

<%-- Get the summary JSP from the current product item --%>

<catalog:getProperty object="<%=item%>"
  propertyName="Jsp"
  getterArgument="<%=new Integer(ProductItem.DETAILED_DISPLAY_JSP_INDEX)%>"
  id="detailJsp"
  returnType="com.beasys.commerce.ebusiness.catalog.JspInfo" />
```

Note: For more information on the Pipeline JSP tags, see *BEA WebLogic Commerce Server Webflow and Pipeline Management*. For more information about the Catalog JSP tags, see “The Catalog JSP Tag Library: cat.tld” on page 5-97.

Form Field Specification

No form fields are used in the `details.jsp` template, nor in the `details.jsp` template’s included `itemdetails.jsp` template.

search.jsp

The `search.jsp` template displays a form field that allows users to perform advanced searches on the product catalog. Searches are performed using Boolean expressions; results are displayed below the search area, using included summary JSPs.

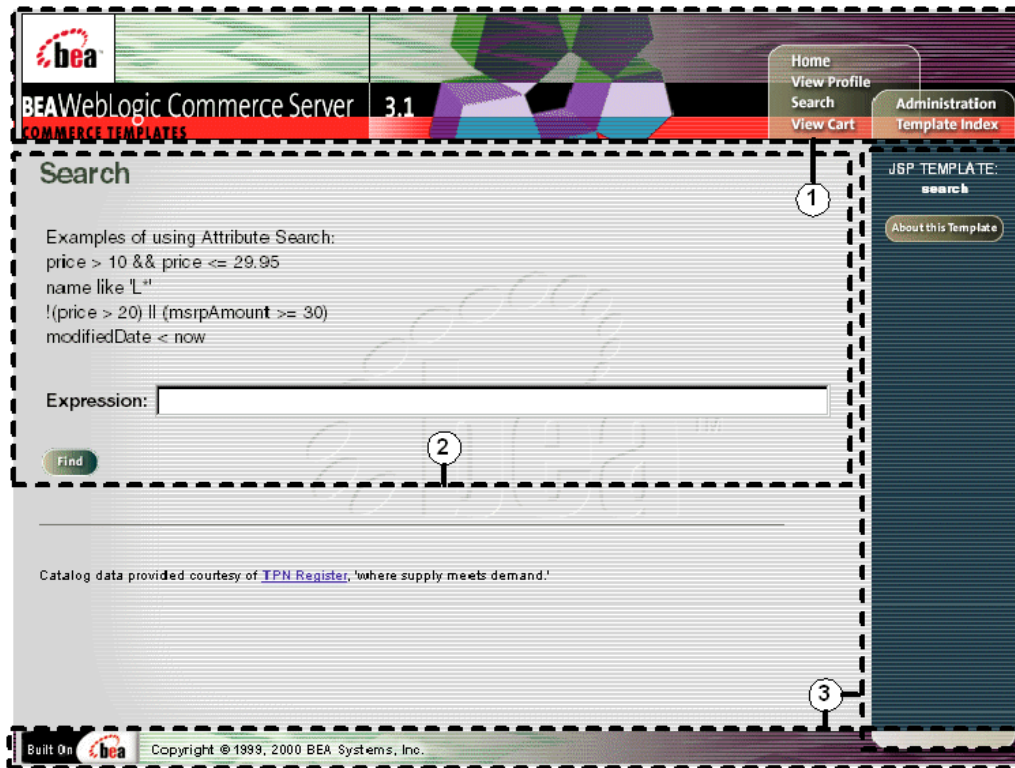
Notes: It is not expected that end-users will search a product catalog using a free form, text-based query syntax. The free form syntax input field is provided to illustrate the power of the search functionality. You should customize the `search.jsp` template to include drop-down lists with the attributes that are appropriate for your business or product items. For example, companies selling books might have edit fields that corresponded to Author Name, ISBN, Price, and so on. The contents of these fields would then be converted into a search expression and passed to the catalog system for processing.

For more information about using the `search.jsp` to perform searches and a description of the syntax for a search expression, refer to “Query-based Search Syntax” on page 5-74.

Sample Browser View

Figure 5-12 shows an annotated version of the `search.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 5-12 Annotated search.jsp Template

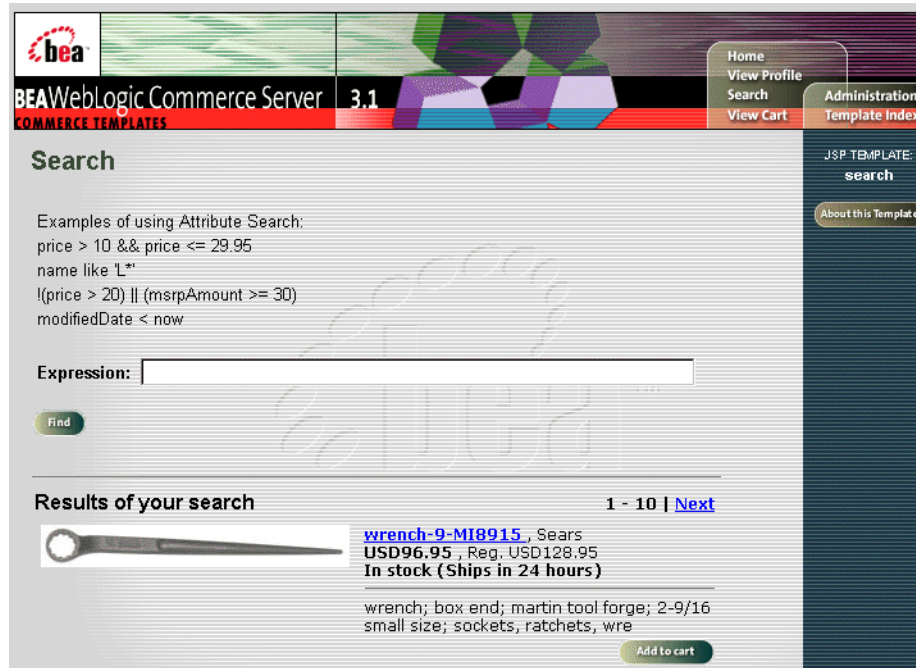


The numbers in the following list refer to the numbered regions in the figure:

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


```
<%@ include file="/commerce/includes/header2.jsp" %>
```
2. Region 2 of the `search.jsp` template is the search area. It provides examples of searches using Boolean expressions, and provides the form field in which customers can enter their search criteria. Following the execution of a search, this region also includes a search results section, as shown in Figure 5-13.

Figure 5-13 The search.jsp Template With Search Results



3. Region 3 of `search.jsp` contains the included `footer2.jsp` template. The include call statement is:

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

The `footer2.jsp` file 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 `search.jsp` 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\catalog\  
search.jsp (Windows)  
$WL_COMMERCE_HOME/server/webapps/wlcs/commerce/catalog/  
search.jsp (UNIX)
```

Tag Library Imports

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

```
<%@ taglib uri="pipeline.tld" prefix="pipeline" %>  
<%@ taglib uri="cat.tld" prefix="catalog" %>  
<%@ taglib uri="weblogic.tld" prefix="wl" %>
```

Note: For more information on the Pipeline JSP tags, see [BEA WebLogic Commerce Server Webflow and Pipeline Management](#). For more information about the Catalog JSP tags, see “The Catalog JSP Tag Library: `cat.tld`” on page 5-97. For more information on the WebLogic 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 `search.jsp` template uses Java classes in the following packages and therefore includes these import statements:

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

Location in the Default Webflow

The `search.jsp` template is displayed any time a customer clicks the Search button located in the top banner of most pages. When a Boolean search is submitted, the `search.jsp` template is reloaded (with included `itemsummary.jsp` templates for each resulting item). From any of the included `itemsummary.jsp` templates, customers can view details about the item (which loads the `details.jsp` template) or

add the item to their shopping cart (which loads the `shoppingcart.jsp` template). Because search results are viewed in groups of 10 by default, the customer may also be able to click Previous/Next links that reload the `search.jsp` template with new content.

Note: For more information about the default Webflow, see Figure 5-4.

Included JSP Templates

The following JSP templates are included into the `search.jsp` template:

- `header2.jsp`, which creates the top banner, and also includes the `leftside.jsp` template; the `leftside.jsp` template reserves column space for generated content that is displayed in `details.jsp`.
- `itemsummary.jsp`, which displays the detailed information about each resulting product item. For more details about the `itemsummary.jsp` template, see “About the Included `itemsummary.jsp` Template” on page 5-39.
- `footer2.jsp`, which creates a horizontal footer at the bottom of the page, and also includes the `rightside.jsp` template. The `rightside.jsp` file 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 link or button on a JSP, it is considered an event. Events trigger particular responses in the default Webflow that allow customers 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 5-12 provides information about the events for the `search.jsp` template, and the business logic they invoke.

Table 5-12 `search.jsp` Events

Event	Webflow Response(s)
--	NewSearch (IP) NewSearch
<code>link(search)</code>	ExpressionSearch (IP) ExpressionSearch

Note: The `NewSearch` input processor and Pipeline are not triggered by an event on the `search.jsp` template. Rather, they are executed when customers click the Search button in the top banner. These mechanisms reset the search results prior to display of the `search.jsp` template.

Because the `search.jsp` template also includes the `itemsummary.jsp` template, the events shown in Table 5-13 are also considered part of the `search.jsp` template.

Table 5-13 `itemsummary.jsp` Events

Event	Webflow Response(s)
<code>link(details)</code>	<code>GetProductItemDetails (IP)</code>
<code>link(add)</code>	<code>AddProductItemToShoppingCart (IP)</code>

Table 5-14 briefly describes each of the Pipelines from Table 5-12, as they are defined in the `pipeline.properties` file. For more information about individual Pipeline components, see “Pipeline Components” on page 5-87.

Table 5-14 `search.jsp` Pipelines

Pipeline	Description
<code>NewSearch</code>	Contains <code>RemoveAttributePC</code> and is not transactional.
<code>ExpressionSearch</code>	Contains <code>SearchPC</code> and is not transactional.

Dynamic Data Display

One purpose of the `search.jsp` template is to present customers with information about the product items that resulted from their search, which customers can then browse. This is accomplished using a combination of Pipeline and Catalog JSP tags.

First, the `getPipelineProperty` Pipeline JSP tag is used to obtain the `CATALOG_SEARCH_RESULTS` attribute from the Pipeline session. Table 5-15 provides more detailed information on this attribute.

Table 5-15 `search.jsp` Pipeline Session Attributes

Attribute	Type	Description
<code>PipelineSessionConstants.CATALOG_SEARCH_RESULTS</code>	<code>com.beasys.commerce.ebusiness.catalog.ViewIterator</code>	Contains the results of the customer's search.

Listing 5-22 illustrates how this attribute is obtained from the Pipeline session using the `getPipelineProperty` Pipeline JSP tag.

Listing 5-22 Obtaining the `CATALOG_SEARCH_RESULTS` Attribute

```
<pipeline:getPipelineProperty
  propertyName="<%=PipelineSessionConstants.CATALOG_SEARCH_RESULTS%>"
  returnName="results"
  returnType="com.beasys.commerce.ebusiness.catalog.ViewIterator" />
```

Note: For more information on the Pipeline JSP tags, see [BEA WebLogic Commerce Server Webflow and Pipeline Management](#).

Then, the `search.jsp` template sets the view. The view is basically a pointer that indicates the location in the complete list of results where we want to start displaying information. This processing is shown in Listing 5-23.

Listing 5-23 Setting the View of the Search Results

```

<% if (results != null && results.size() > 0) { %>
  <% String viewIndexString =
    (String)request.getParameter(HttpRequestConstants.CATALOG_VIEW_INDEX); %>

  <% if (viewIndexString == null) { viewIndexString = "0"; } %>
  <% int viewIndex = Math.min(Integer.valueOf(viewIndexString).intValue(),
    results.getViewCount() - 1); %>
  <% results.gotoViewAt(viewIndex); %>

```

Next, if the search results require more than one view, a navigation bar (containing Previous and Next links, as well as text showing the results currently being viewed) is generated, as shown in Listing 5-24.

Listing 5-24 Generating the View Navigation Bar

```

<table border="0" width="90%">
<tr>

<td align="left" valign="top"><p class="head2">Results of your search</p></td>

<td align="right" valign="bottom">
  <p class="tabletext"><b>

    <!-- Add previous link -->

    <% if (results.hasPreviousView()) { %>
      <a href="<%= WebflowJSPHelper.createWebflowURL(pageContext, "search.jsp",
        "link(search)","&" + HttpRequestConstants.CATALOG_VIEW_INDEX + "=" +
        (viewIndex - 1), true) %>">Previous</a> |
    <% } %>

    <!-- Add current view indicies -->

    <% if (results.size() > 1) { %>
      <%= results.getCurrentView().getFirstIndex() %> - <%= results.
        getCurrentView().getLastIndex() %>
    <% } %>

    <!-- Add next link -->

    <% if (results.hasNextView()) { %>
      | <a href="<%= WebflowJSPHelper.createWebflowURL(pageContext, "search.jsp",
        "link(search)","&" + HttpRequestConstants.CATALOG_VIEW_INDEX + "=" +

```

5 The Product Catalog JSP Templates and Tag Library

```
        (viewIndex + 1), true) %>">Next</a>
    <% } %>

</b>
</td>

</tr>
</table>
```

Lastly, the `iterateThroughView` Catalog JSP tag is used to iterate through the product items that are in the current view. The required parameters for the included JSP are added to the request, and the `getProperty` Catalog JSP tag obtains the correct `itemssummary.jsp` templates for inclusion into the `search.jsp` template. This processing is shown in Listing 5-25.

Listing 5-25 Obtaining and Displaying the Product Item Summaries

```
<%-- Iterate through the items in the current view, including the summary JSP for
each --%>

<catalog:iterateThroughView iterator="<%=results%>" id="item"
returnType="com.beasys.commerce.ebusiness.catalog.ProductItem"
viewIndex="<%= viewIndex %>">

    <%-- Add the required parameters for the included JSP to the request --%>

    <% request.setAttribute("product_item", item); %>
    <% request.setAttribute("details_link", "itemdetails"); %>

    <%-- Get the summary JSP from the current product item --%>

    <catalog:getProperty object="<%=item%>" propertyName="Jsp"
getterArgument="<%= new Integer(ProductItem.SUMMARY_DISPLAY_JSP_INDEX) %>"
id="summaryJsp"
returnType="com.beasys.commerce.ebusiness.catalog.JspInfo"/>

    <%-- Included the summary JSP --%>

    <jsp:include page="<%= summaryJsp.getUrl() %>" flush="true"/>

</catalog:iterateThroughView>
```

Note: For more information about the Catalog JSP tags, see “The Catalog JSP Tag Library: cat.tld” on page 5-97.

Form Field Specification

The primary purpose of the `search.jsp` template is to allow customers to enter their search criteria into an HTML form field. It is also used to pass needed information to the Webflow.

The form fields used in the `search.jsp` template, and a description for each of these form fields are listed in Table 5-16.

Table 5-16 `search.jsp` Form Fields

Parameter Name	Type	Description
<code>HttpServletRequest.CATALOG_VIEW_SIZE</code>	Hidden	Optional parameter that allows you to specify the number of items shown in a result view.
<code>HttpServletRequest.CATALOG_SEARCH_STRING</code>	Text	The form field into which customers will enter their search criteria.
<code>HttpServletRequest.CATALOG_SOURCE_KEY</code>	Hidden	Used to determine whether results are from a new search or an iteration through an existing search.

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

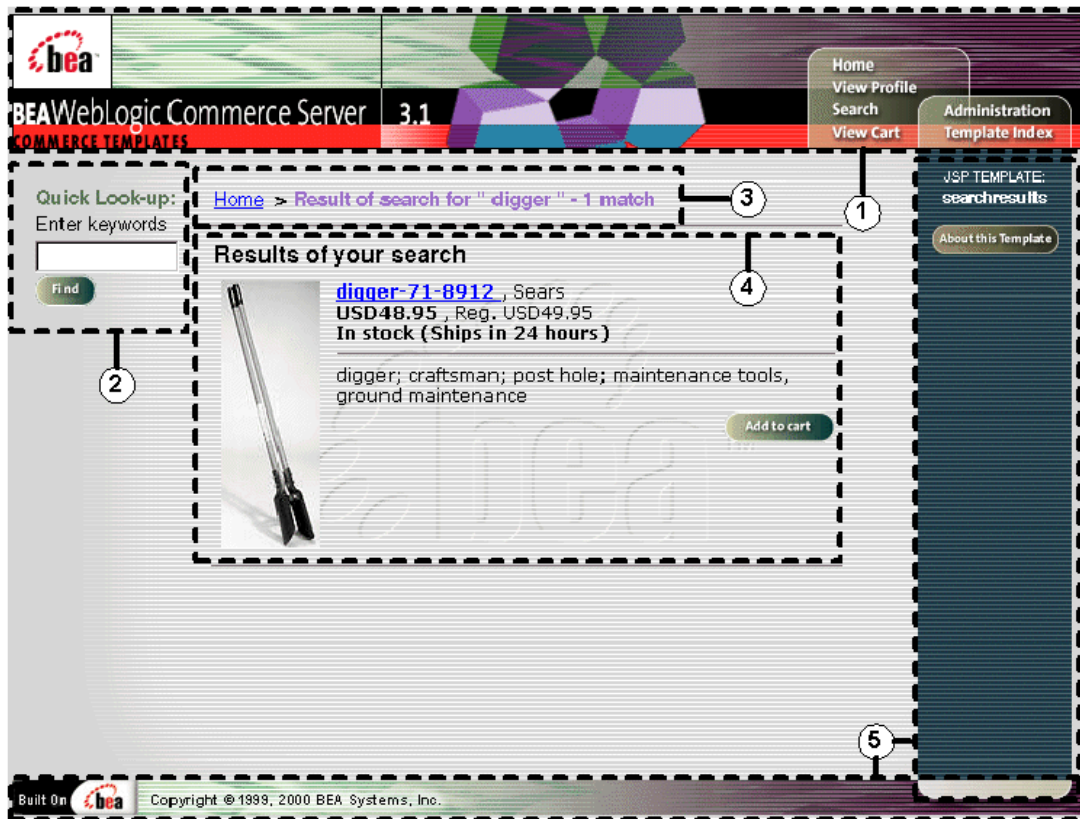
searchresults.jsp

The `searchresults.jsp` template displays results from a keyword search that is launched from the Quick Look-up text field. The Quick Look-up is available in the left-side column of any page on the site.

Sample Browser View

Figure 5-14 shows an annotated version of the `searchresults.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 5-14 Annotated searchresults.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 `header.jsp` template. This is standard across many of the JSP templates provided by WebLogic Commerce Server. The import call is:

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

2. Region 2 of the `searchresults.jsp` template contains the `quicksearch.jsp` template that is included. This template provides a keyword-search feature. The include statement is:

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

3. Region 3 provides customers with a link back to the home page (that is, the `main.jsp` template) and some information about how many matches to their search criteria were identified. Both are presented in a format that is similar to the ancestor category navigation bar that appears on the `browse.jsp` template.

Note: For more information about the `browse.jsp` template, see “`browse.jsp` Template” on page 5-25.

4. Region 4 of the `searchresults.jsp` template shows the results generated from processing with the Webflow and Pipeline mechanisms that found the detailed information for each resulting product item (presented in the included `itemsummary.jsp` template).
5. Region 5 of the `searchresults.jsp` template contains the included `footer.jsp` template. The include statement is:

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

The `footer.jsp` file 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 `footer.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 `searchresults.jsp` 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\catalog\  
searchresults.jsp (Windows)  
$WL_COMMERCE_HOME/server/webapps/wlcs/commerce/catalog/  
searchresults.jsp (UNIX)
```

Tag Library Imports

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


```
<%@ taglib uri="pipeline.tld" prefix="pipeline" %>
<%@ taglib uri="cat.tld" prefix="catalog" %>
<%@ taglib uri="es.tld" prefix="es" %>
```

Note: For more information on the Pipeline JSP tags, see [BEA WebLogic Commerce Server Webflow and Pipeline Management](#). For more information about the Catalog JSP tags, see “The Catalog JSP Tag Library: cat.tld” on page 5-97. For more information on the WebLogic Personalization Server Utility 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 `searchresults.jsp` template uses Java classes in the following packages and therefore includes these import statements:

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

Location in the Default Webflow

Customers arrive at the `searchresults.jsp` template after they enter a keyword in the Quick Look-up text field (located in the left-side column of every page) and click the Find button. From here, customers can navigate back to the `main.jsp` template using the Home link at the top of the page. Customers can also choose to view more details about a particular item shown in the results list (which loads the `details.jsp` template), or add the product item to their shopping cart (which loads the `shoppingcart.jsp` template). Finally, customers can also choose to perform another search by using the Quick Look-up again.

Note: For more information about the default Webflow, see Figure 5-4.

Included JSP Templates

The following JSP templates are included into the `searchresults.jsp` template:

5 The Product Catalog JSP Templates and Tag Library

- `header.jsp`, which creates the top banner, and also includes the `leftside.jsp` template; the `leftside.jsp` template reserves column space for generated content that is displayed in `searchresults.jsp`.
- `quicksearch.jsp`, which provides a keyword-based search tool for finding product items via keywords that have already been assigned.
- `columnbreak.jsp`, which creates space (a vertical column) between the left-side column and the main display area.
- `itemsummary.jsp`, which displays the detailed information about each resulting product item. For more details about the `itemsummary.jsp` template, see “About the Included `itemsummary.jsp` Template” on page 5-39.
- `footer.jsp`, which creates a horizontal footer at the bottom of the page, and also includes the `rightside.jsp` template. The `rightside.jsp` file 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 link or button on a JSP, it is considered an event. Events trigger particular responses in the default Webflow that allow customers 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 5-17 provides information about the events for the `searchresults.jsp` template, and the business logic they invoke.

Table 5-17 `searchresults.jsp` Events

Event	Webflow Response(s)
<code>link(home)</code>	<code>GetTopCategories (IP)</code> <code>GetTopCategories</code>
<code>link(quicksearch)</code>	<code>KeywordSearch (IP)</code> <code>KeywordSearch</code>

Because the `searchresults.jsp` template also includes the `itemsummary.jsp` template, the events shown in Table 5-18 are also considered part of the `search.jsp` template.

Table 5-18 itemssummary.jsp Events

Event	Webflow Response(s)
link(details)	GetProductItemDetails (IP)
link(add)	AddProductItemToShoppingCart (IP)

Table 5-19 briefly describes each of the Pipelines from Table 5-17, as they are defined in the pipeline.properties file. For more information about individual Pipeline components, see “Pipeline Components” on page 5-87.

Table 5-19 search.jsp Pipelines

Pipeline	Description
GetTopCategories	Contains GetCategoryPC and GetSubcategoriesPC and is not transactional.
KeywordSearch	Contains SearchPC and is not transactional.

Dynamic Data Display

The primary purpose of the searchresults.jsp template is to present customers with information about the product items that resulted from their search, which customers can then browse through. This is accomplished using a combination of Pipeline and Catalog JSP tags.

First, the getPipelineProperty Pipeline JSP tag is used to obtain the CATALOG_QUERY and CATALOG_SEARCH_RESULTS attributes from the Pipeline session. Table 5-20 provides more detailed information on these attributes.

Table 5-20 searchresults.jsp Pipeline Session Attributes

Attribute	Type	Description
PipelineSessionConstants .CATALOG_QUERY	com.beasys.commerce.ebusiness .catalog.service.query. KeywordQuery	Contains the customer’s search criteria.

5 The Product Catalog JSP Templates and Tag Library

Table 5-20 searchresults.jsp Pipeline Session Attributes

Attribute	Type	Description
PipelineSessionConstants .CATALOG_SEARCH_RESULTS	com.beasys.commerce.ebusiness .catalog.ViewIterator	Contains the results of the customer's search.

Listing 5-26 illustrates how this attribute is obtained from the Pipeline session using the `getPipelineProperty` Pipeline JSP tag.

Listing 5-26 Obtaining the CATALOG_QUERY and CATALOG_SEARCH_RESULTS Attributes

```
<pipeline:getPipelineProperty
propertyName="<%= PipelineSessionConstants.CATALOG_QUERY %>"
returnName="query"
returnType="com.beasys.commerce.ebusiness.catalog.service.query.KeywordQuery"/>

<pipeline:getPipelineProperty
propertyName="<%= PipelineSessionConstants.CATALOG_SEARCH_RESULTS %>"
returnName="results"
returnType="com.beasys.commerce.ebusiness.catalog.ViewIterator" />
```

Note: For more information on the Pipeline JSP tags, see [BEA WebLogic Commerce Server Webflow and Pipeline Management](#).

Next, the navigation bar at the top of the page (containing the Home link and number of matches) is constructed using the `<es>` WebLogic Personalization Server Utility JSP tag, as shown in Listing 5-27.

Listing 5-27 Constructing the Top Navigation Bar

```
<p>
<a href="<%= WebflowJSPHelper.createWebflowURL(pageContext,
"searchresults.jsp", "link(home)", true) %>">Home</a>

&nbsp;<b>&gt;</b>

<b>
<font color="#9C77DE">Result of search for "
```

```

<es:forEachInArray id="keyword" type="java.lang.String"
  array="<%= query.getKeywords() %>">
  <%= " " + keyword %>
</es:forEachInArray>"
- <%= results.size() %> match<% if (results.size() > 1 ||
results.size() == 0) { %>es<% } %>

</font>
/b>

</p>

<hr size="1" width="90%" align="left">

```

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

Then, the `searchresults.jsp` template sets the view. The view is basically a pointer that indicates the location in the complete list of results where we want to start displaying information. This processing is shown in Listing 5-28.

Listing 5-28 Setting the View of the Search Results

```

<% if (results != null && results.size() > 0) { %>
  <% String viewIndexString =
  (String)request.getParameter(HttpRequestConstants.CATALOG_VIEW_INDEX); %>

  <% if (viewIndexString == null) { viewIndexString = "0"; } %>
  <% int viewIndex = Math.min(Integer.valueOf(viewIndexString).intValue(),
  results.getViewCount() - 1); %>
  <% results.gotoViewAt(viewIndex); %>

```

Next, if the search results require more than one view, a navigation bar (containing Previous and Next links, as well as text showing the results currently being viewed) is generated, as shown in Listing 5-29.

5 The Product Catalog JSP Templates and Tag Library

Listing 5-29 Generating the View Navigation Bar

```
<table border="0" width="90%">
<tr>

<td align="left" valign="top"><p class="head2">Results of your search</p></td>

<td align="right" valign="bottom">
  <p class="tabletext"><b>

    <%-- Add previous link --%>

    <% if (results.hasPreviousView()) { %>
      <a href="<%= WebflowJSPHelper.createWebflowURL(pageContext, "search.jsp",
        "link(search)","&" + HttpRequestConstants.CATALOG_VIEW_INDEX + "=" +
        (viewIndex - 1), true) %>">Previous</a> |
    <% } %>

    <%-- Add current view indices --%>

    <% if (results.size() > 1) { %>
      <%= results.getCurrentView().getFirstIndex() %> - <%= results.
        getCurrentView().getLastIndex() %>
    <% } %>

    <%-- Add next link --%>

    <% if (results.hasNextView()) { %>
      | <a href="<%= WebflowJSPHelper.createWebflowURL(pageContext, "search.jsp",
        "link(search)","&" + HttpRequestConstants.CATALOG_VIEW_INDEX + "=" +
        (viewIndex + 1), true) %>">Next</a>
    <% } %>

  </b>
</td>

</tr>
</table>
```

Lastly, the `iterateThroughView` Catalog JSP tag is used to iterate through the product items that are in the current view. The required parameters for the included JSP are added to the request, and the `getProperty` Catalog JSP tag obtains the correct `itemssummary.jsp` templates for inclusion into the `searchresults.jsp` template. This processing is shown in Listing 5-30.

Listing 5-30 Obtaining and Displaying the Product Item Summaries

```
<%-- Iterate through the items in the current view, including the summary JSP for
each --%>

<catalog:iterateThroughView iterator="<%=results%>" id="item"
  returnType="com.beasys.commerce.ebusiness.catalog.ProductItem"
  viewIndex="<%= viewIndex %>">

  <%-- Add the required parameters for the included JSP to the request --%>

  <% request.setAttribute("product_item", item); %>
  <% request.setAttribute("details_link", "itemdetails"); %>

  <%-- Get the summary JSP from the current product item --%>

  <catalog:getProperty object="<%=item%>" propertyName="Jsp"
    getterArgument="<%= new Integer(ProductItem.SUMMARY_DISPLAY_JSP_INDEX) %>"
    id="summaryJsp"
    returnType="com.beasys.commerce.ebusiness.catalog.JspInfo"/>

  <%-- Included the summary JSP --%>

  <jsp:include page="<%= summaryJsp.getUrl() %>" flush="true"/>

</catalog:iterateThroughView>
```

Note: For more information about the Catalog JSP tags, see “The Catalog JSP Tag Library: cat.tld” on page 5-97.

Form Field Specification

No form fields are used in the `searchresults.jsp` template.

Query-based Search Syntax

Search queries within the BEA WebLogic Commerce Server product use a syntax similar to the SQL string syntax that supports basic Boolean-type comparison expressions, including nested parenthetical queries. In general, the syntax includes a metadata property name, a comparison operator, and a literal value.

The basic query uses the following syntax:

```
attribute_name comparison_operator literal_value
```

Note: Consult the [Javadoc](#) API documentation on `com.beasys.commerce.util.ExpressionHelper` for more information about the query syntax.

Several constraints apply to queries constructed using this syntax:

- String literals must be enclosed in single quotes, as shown below:
 - `'WebLogic Server'`
 - `'football'`
- Date literals can be created via a simplistic `toDate` method that takes one or two `String` arguments (enclosed in single quotes). The first, if two arguments are supplied, is the `SimpleDateFormat` format string; the second argument is the date string. If only one argument is supplied, it should include the date string in the `MM/dd/yyyy HH:mm:ss z` format (also enclosed in single quotes), as shown below:
 - `toDate('EE dd MMM yyyy HH:mm:ss z', 'Thr 06 Apr 2000 16:56:00 MDT')`
 - `toDate('02/23/2000 13:57:43 MST')`
- Use the `toProperty` method to compare properties whose names include spaces or other special characters. In general, use `toProperty` when the property name does not comply with the Java variable-naming convention that uses alphanumeric characters, as shown below:
 - `toProperty('My Property') = 'Content'`
- To include a scope into the property name, use either `scope.propertyName` or the `toProperty` method with two arguments, as shown below:

- `toProperty ('myScope', 'myProperty')`

Note: The reference document management system ignores property scopes.

- Use `\` along with the appropriate character(s) to create an escape sequence that includes special characters in string literals, as shown below:

- `toProperty ('My Property\'s Contents') = 'Content'`

- Additionally, use Java-style unicode escape sequences to embed non-ASCII characters in string literals, as shown below:

- Description like ``*\u65e5\u672c\u8a9e*``

Notes: The query syntax can only contain ASCII and extended ASCII characters (0-255).

Use `ExpressionHelper.toStringLiteral` to convert an arbitrary string to a fully quoted and escaped string literal, which can then be placed in a query.

- The `now` keyword—only used on the literal value side of the expression—refers to the current date and time.
- Boolean literals are either `true` or `false`.
- Numeric literals consist of the numbers themselves without any text decoration (like quotation marks). The system supports scientific notation in the forms (for example, `1.24e4` and `1.24E-4`).
- An exclamation mark (!) can be placed at an opening parenthesis to negate an expression, as shown below:
 - `!(size >= 256)`
- The Boolean `and` operator is represented by the literal `&&`, as shown below:
 - `author == 'james' && age < 55`
- The Boolean `or` operator is represented by the literal `||`, as shown below:
 - `creationDate > now || expireDate < now`

The following examples illustrate full expressions:

Example 1:

```
((color='red' && size <=1024) || (keywords contains 'red' && creationDate < now))
```

Example 2:

```
creationDate > toDate ('MM/dd/yyyy HH:mm:ss', '2/22/2000 14:51:00')
&& expireDate <= now && mimetype like 'text/*'
```

Using Comparison Operators to Construct Queries

To support advanced searching, the system allows construction of nested Boolean queries incorporating comparison operators. The following table summarizes the comparison operators available for each metadata type.

Operator Type	Characteristics
Boolean (==, !=)	Boolean attributes support an equality check against <code>Boolean.TRUE</code> or <code>Boolean.FALSE</code> .
Numeric (==, !=, >, <, >=, <=)	Numeric attributes support the standard equality, greater than, and less than checks against a <code>java.lang.Number</code> .
Text (==, !=, >, <, >=, <=, like)	Text strings support standard equality checking (case sensitive), plus lexicographical comparison (less than or greater than). In addition, strings can be compared using wildcard pattern matching (that is, the <i>like</i> operator), similar to the SQL LIKE operator or DOS prompt file matching. In this situation, the wildcards will be * (asterisk) for match any and ? (question mark) for match single. Interval matching (for example, using []) is not supported. To match * or ? exactly, the quote character will be \ (backslash).
Datetime (==, !=, >, <, >=, <=)	Date/time attributes support standard equality, greater than, and less than checks against a <code>java.sql.Timestamp</code> .

Operator Type	Characteristics
Multi-valued Comparison Operators (contains, containsall)	<p>Multi-valued attributes support a <i>contains</i> operator that takes an object of the attribute's subtype and checks that the attribute's value contains it. Additionally, multi-valued attributes support a <i>containsall</i> operator, which takes another collection of objects of the attribute's subtype and checks that the attribute's value contains all of them.</p> <p>Single-valued operators applied to a multi-valued attribute should cause the operator to be applied over the attribute's collection of values. Any value that matches the operator and operand should return <code>true</code>. For example, if the multi-valued text attribute <i>keywords</i> has the values <i>BEA</i>, <i>Computer</i>, and <i>WebLogic</i> and the operand is <i>BEA</i>, then the <code><</code> operator returns <code>true</code> (<i>BEA</i> is less than <i>Computer</i>), the <code>></code> operator returns <code>false</code> (<i>BEA</i> is not greater than any of the values), and the <code>==</code> operator returns <code>true</code> (<i>BEA</i> is equal to <i>BEA</i>).</p>
User Defined Comparison Operators	Currently, no operators can be applied to a user-defined attribute.

Note: The search parameters and expression objects support negation (using `!`) of expressions via a bit flag.

Searchable Catalog Attributes

You can base your searches on the following attributes of the product catalog:

- `sku`
- `identifier`
- `name`
- `shortDesc`
- `shortDescription`
- `description`
- `creator`
- `publisher`
- `contributor`
- `creationDate`
- `source`

- lang
- language
- relation
- coverage
- rights
- format
- type
- inStock
- msrpCurrency
- msrpAmount
- priceCurrency
- priceAmount
- price
- estimateShipTime
- shipTime
- specialNotes
- notes
- taxCode
- shippingCode
- modifiedDate

Note: Some of the attributes have several aliases (for example, `shortDesc` and `shortDescription`, `lang` and `language`) but refer to a single attribute.

Controlling the Number of Search Results

The number of items returned from a keyword- or attribute-based search is controlled using the following APIs:

- `getMaxSearchResults()`
- `setMaxSearchResults(int max)`

These are methods on the `com.beasys.commerce.ebusiness.catalog.service.query.KeywordQuery` and `ProductItemQuery` interfaces.

If an unlimited number of search results is desired, use:

```
setMaxSearchResults( CatalogQuery.ALL_RESULTS );
```

Limiting the number of search results returned prevents potentially very large result sets from being moved from the database to the JSP container. If the search query name like '*' is executed, all the items in the database will be returned. It may be desirable to limit the size of the result set to a suitably large number such as 1,000.

By default, the results from queries are not limited. As shown in Listing 5-31, the default search result size is controlled by the `catalog.searchresults.size` property in the `wlcs-catalog.properties` file. The file resides in the `WL_COMMERCE_HOME\classes` directory, where `WL_COMMERCE_HOME` is the directory in which you installed the WebLogic Commerce Server software.

Listing 5-31 Using the `wlcs-catalog.properties` File to Control the Default Search Result Size

```
#####  
#   Maximum search results returned by the catalog  
#  
#   You can dynamically change the searchresults by using the  
#   get/set MaxSearchResults methods on the CatalogQuery object  
#  
#   Set the value to -1 to return all results by the search engine  
#####  
catalog.searchresults.size=-1
```

Input Processors

This section provides a brief description of each input processor associated with the Product Catalog JSP templates.

CatalogIP

Class Name	<code>com.beasys.commerce.ebusiness.catalog.webflow.CatalogIP</code>
Description	Base <code>InputProcessor</code> for all Catalog-related <code>InputProcessors</code> . This abstract class contains global Catalog HTTP request parameter extraction and validation.
Required <code>HttpServletRequest</code> Parameters	None
Optional <code>HttpServletRequest</code> Parameters	<code>HttpRequestConstants.CATALOG_VIEW_SIZE</code>
Required Pipeline Session Attributes	None
Updated Pipeline Session Attributes	<code>PipelineSessionConstants.CATALOG_VIEW_SIZE</code> (Request scope) only if <code>CATALOG_VIEW_SIZE</code> is given in the <code>HttpServletRequest</code> .
Removed Pipeline Session Attributes	None
Validation	Verifies that the <code>CATALOG_VIEW_SIZE</code> parameter is specified appropriately. Because view size is an optional parameter, no action is taken if it is not specified.

Exceptions	ProcessingException, thrown if the CATALOG_VIEW_SIZE is invalid.
-------------------	--

GetProductItemIP

Class Name	com.beasys.commerce.ebusiness.catalog.webflow. GetProductItemIP
Description	Creates a ProductItemKey based on a product item SKU HTTP request parameter and adds it to the Pipeline session.
Required HttpServletRequest Parameters	HttpRequestConstants.CATALOG_ITEM_SKU
Optional HttpServletRequest Parameters	None
Required Pipeline Session Attributes	None
Updated Pipeline Session Attributes	PipelineSessionConstants.CATALOG_ITEM_KEY (Request scope)
Removed Pipeline Session Attributes	None
Validation	Verifies that the CATALOG_ITEM_SKU parameter is valid.
Exceptions	ProcessingException, thrown if the CATALOG_ITEM_SKU parameter is invalid.

GetCategoryIP

Class Name	com.beasys.commerce.ebusiness.catalog.webflow. GetCategoryIP
Description	Creates a CategoryKey based on a category ID HTTP request parameter and adds it to the Pipeline session. If no such parameter is supplied, the CategoryKey for the root category is added.
Required HttpServletRequest Parameters	HttpRequestConstants.CATALOG_CATEGORY_ID
Optional HttpServletRequest Parameters	None
Required Pipeline Session Attributes	None
Updated Pipeline Session Attributes	PipelineSessionConstants.CATALOG_CATEGORY_KEY (Request scope)
Removed Pipeline Session Attributes	None
Validation	Validates that the CATALOG_CATEGORY_ID parameter is valid.
Exceptions	ProcessingException, thrown if the CATALOG_CATEGORY_ID parameter is invalid.

KeywordSearchIP

Class Name	com.beasys.commerce.ebusiness.catalog.webflow. KeywordSearchIP
Description	Creates a KeywordQuery based on a keyword search HTTP request parameter and adds it to the Pipeline session.
Required HttpServletRequest Parameters	None
Optional HttpServletRequest Parameters	HttpRequestConstants.CATALOG_SEARCH_STRING
Required Pipeline Session Attributes	PipelineSessionConstants.CATALOG_QUERY. Required only if the CATALOG_SEARCH_STRING parameter does not exist.
Updated Pipeline Session Attributes	PipelineSessionConstants.CATALOG_QUERY
Removed Pipeline Session Attributes	PipelineSessionConstants.CATALOG_SEARCH_RESULTS, only if the CATALOG_SEARCH_STRING parameter exists.
Validation	Verifies that the CATALOG_SEARCH_STRING parameter is valid.
Exceptions	InvalidSessionStateException, thrown if the CATALOG_SEARCH_STRING parameter does not exist and the PipelineSessionConstants.CATALOG_SEARCH_RESULTS Pipeline session attribute has expired. ProcessingException, thrown if the CATALOG_SEARCH_RESULTS parameter is invalid.

ExpressionSearchIP

Class Name	<code>com.beasys.commerce.ebusiness.catalog.webflow.ExpressionSearchIP</code>
Description	Creates a <code>ProductItemQuery</code> based on a search expression HTTP request parameter and adds it to the Pipeline session.
Required <code>HttpServletRequest</code> Parameters	None
Optional <code>HttpServletRequest</code> Parameters	<code>HttpRequestConstants.CATALOG_SEARCH_STRING</code>
Required Pipeline Session Attributes	<code>PipelineSessionConstants.CATALOG_QUERY</code> . Required only if the <code>CATALOG_SEARCH_STRING</code> parameter does not exist.
Updated Pipeline Session Attributes	<code>PipelineSessionConstants.CATALOG_QUERY</code>
Removed Pipeline Session Attributes	<code>PipelineSessionConstants.CATALOG_SEARCH_RESULTS</code> if the <code>CATALOG_SEARCH_STRING</code> parameter exists.
Validation	Verifies that the <code>CATALOG_SEARCH_STRING</code> parameter is a valid expression.
Exceptions	<p><code>InvalidSessionStateException</code>, thrown if the <code>CATALOG_SEARCH_STRING</code> parameter does not exist and the <code>PipelineSessionConstants.CATALOG_SEARCH_RESULTS</code> Pipeline session attribute has expired.</p> <p><code>ProcessingException</code>, thrown if the <code>CATALOG_SEARCH_STRING</code> parameter is invalid.</p>

MoveAttributeIP

Class Name	com.beasys.commerce.ebusiness.catalog.webflow. MoveAttributeIP
Description	Sets Pipeline session attributes when moving a Pipeline session attribute value from a source attribute to a destination attribute. The source and destination attributes are specified as HTTP request parameters.
Required HttpServletRequest Parameters	HttpRequestConstants.CATALOG_SOURCE_KEY HttpRequestConstants.CATALOG_DESTINATION_KEY
Optional HttpServletRequest Parameters	None
Required Pipeline Session Attributes	None
Updated Pipeline Session Attributes	PipelineSessionConstants.CATALOG_SOURCE_KEY (Request scope) PipelineSessionConstants.CATALOG_DESTINATION_KEY (Request scope)
Removed Pipeline Session Attributes	None
Validation	Verifies that the CATALOG_SOURCE_KEY and the CATALOG_DESTINATION_KEY parameters are valid strings.
Exceptions	ProcessingException, thrown if either the source key or destination key is invalid.

RemoveAttributeIP

Class Name	<code>com.beasys.commerce.ebusiness.catalog.webflow.RemoveAttributeIP</code>
Description	Sets Pipeline session attributes for removing a Pipeline session attribute value from a source attribute. The source attribute is specified as an HTTP request parameter.
Required HttpServletRequest Parameters	<code>HttpRequestConstants.CATALOG_SOURCE_KEY</code>
Optional HttpServletRequest Parameters	None
Required Pipeline Session Attributes	None
Updated Pipeline Session Attributes	<code>PipelineSessionConstants.CATALOG_SOURCE_KEY</code> (Request scope)
Removed Pipeline Session Attributes	None
Validation	Verifies that the <code>CATALOG_SOURCE_KEY</code> parameter is valid.
Exceptions	<code>ProcessingException</code> , thrown if the <code>CATALOG_SOURCE_KEY</code> parameter is invalid.

Pipeline Components

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

CatalogPC

Class Name	com.beasys.commerce.ebusiness.catalog.pipeline.CatalogPC
Description	Base PipelineComponent for all Catalog-related PipelineComponents. This abstract class contains Catalog-related Pipeline utility methods.
Required Pipeline Session Attributes	None
Optional Pipeline Session Attributes	None
Updated Pipeline Session Attributes	None
Removed Pipeline Session Attributes	None
Type	Java object
JNDI Name	None
Exceptions	None

GetCategoryPC

Class Name	com.beasys.commerce.ebusiness.catalog.pipeline. GetCategoryPC
Description	Retrieves a <code>Category</code> based upon the <code>CategoryKey</code> in the Pipeline session <code>CATALOG_CATEGORY_KEY</code> attribute. The resultant <code>Category</code> is placed into the Pipeline session as the <code>CATALOG_CATEGORY</code> attribute.
Required Pipeline Session Attributes	<code>PipelineSessionConstants.CATALOG_CATEGORY_KEY</code> (Request scope)
Optional Pipeline Session Attributes	None
Updated Pipeline Session Attributes	<code>PipelineSessionConstants.CATALOG_CATEGORY</code> (Request scope)
Removed Pipeline Session Attributes	None
Type	Java object
JNDI Name	None
Exceptions	<code>PipelineFatalException</code> on Catalog finder error, Catalog general error, EJB create error, or JNDI lookup error.

GetProductItemPC

Class Name	com.beasys.commerce.ebusiness.catalog.pipeline. GetProductItemPC
Description	Retrieves a <code>ProductItem</code> based upon the <code>ProductItemKey</code> contained in the Pipeline session <code>CATALOG_ITEM_KEY</code> attribute. The resultant <code>ProductItem</code> is placed into the Pipeline session as the <code>CATALOG_ITEM</code> attribute.
Required Pipeline Session Attributes	<code>PipelineSessionConstants.CATALOG_ITEM_KEY</code> (Request scope)
Optional Pipeline Session Attributes	None
Updated Pipeline Session Attributes	<code>PipelineSessionConstants.CATALOG_ITEM</code> (Request scope)
Removed Pipeline Session Attributes	None
Type	Java object
JNDI Name	None
Exceptions	<code>PipelineFatalException</code> on Catalog finder error, Catalog general error, Catalog create error, or JNDI lookup error.

GetParentPC

Class Name	com.beasys.commerce.ebusiness.catalog.pipeline. GetParentPC
Description	Retrieves the parent Category of the category contained in the Pipeline session CATALOG_CATEGORY attribute. The resultant Category is placed into the Pipeline session as the CATALOG_CATEGORY attribute.
Required Pipeline Session Attributes	PipelineSessionConstants.CATALOG_CATEGORY (Request scope)
Optional Pipeline Session Attributes	None
Updated Pipeline Session Attributes	PipelineSessionConstants.CATALOG_CATEGORY (Request scope)
Removed Pipeline Session Attributes	None
Type	Java object
JNDI Name	None
Exceptions	PipelineFatalException on Catalog finder error, Catalog general error, Catalog create error, or JNDI lookup error.

GetAncestorsPC

Class Name	com.beasys.commerce.ebusiness.catalog.pipeline. GetAncestorsPC
Description	Retrieves the ancestor Categories of the Category contained in the Pipeline session CATALOG_CATEGORY attribute. The resultant Category array is placed into the Pipeline session as the CATALOG_ANCESTORS attribute.
Required Pipeline Session Attributes	PipelineSessionConstants.CATALOG_CATEGORY (Request scope)
Optional Pipeline Session Attributes	None
Updated Pipeline Session Attributes	PipelineSessionConstants.CATALOG_ANCESTORS (Request scope)
Removed Pipeline Session Attributes	None
Type	Java object
JNDI Name	None
Exceptions	PipelineFatalException on Catalog finder error, Catalog general error, Catalog create error, or JNDI lookup error.

GetProductItemsPC

Class Name	com.beasys.commerce.ebusiness.catalog.pipeline. GetProductItemsPC
Description	Retrieves the ProductItems associated with the Category contained in the Pipeline session CATALOG_CATEGORY attribute. The resultant ViewIterator of ProductItems is placed into the Pipeline session as the CATALOG_ITEMS attribute. The view size of the resultant ViewIterator may be specified with the optional CATALOG_VIEW_SIZE Pipeline session attribute.
Required Pipeline Session Attributes	PipelineSessionConstants.CATALOG_CATEGORY (Request scope)
Optional Pipeline Session Attributes	PipelineSessionConstants.CATALOG_VIEW_SIZE (Request scope)
Updated Pipeline Session Attributes	PipelineSessionConstants.CATALOG_ITEMS (Request scope)
Removed Pipeline Session Attributes	None
Type	Java object
JNDI Name	None
Exceptions	PipelineFatalException on Catalog finder error, Catalog general error, Catalog create error, or JNDI lookup error.

GetSubcategoriesPC

Class Name	com.beasys.commerce.ebusiness.catalog.pipeline. GetSubcategoriesPC
Description	Retrieves the sub-categories of the Category in the Pipeline session CATALOG_CATEGORY attribute. The resultant ViewIterator of Categories is placed into the Pipeline session as the CATALOG_CATEGORIES attribute. The view size of the resultant ViewIterator may be specified with the optional CATALOG_VIEW_SIZE Pipeline session attribute.
Required Pipeline Session Attributes	PipelineSessionConstants.CATALOG_CATEGORY (Request scope)
Optional Pipeline Session Attributes	PipelineSessionConstants.CATALOG_VIEW_SIZE (Request scope)
Updated Pipeline Session Attributes	PipelineSessionConstants.CATALOG_CATEGORIES (Request scope)
Removed Pipeline Session Attributes	None
Type	Java object
JNDI Name	None
Exceptions	PipelineFatalException on Catalog finder error, Catalog general error, EJB create error, or JNDI lookup error.

MoveAttributePC

Class Name	com.beasys.commerce.ebusiness.catalog.pipeline. MoveAttributePC
Description	Moves a Pipeline session attribute value from a source attribute to a destination attribute. The source and destination attributes are specified by the CATALOG_SOURCE_KEY and CATALOG_DESTINATION_KEY Pipeline session attributes.
Required Pipeline Session Attributes	PipelineSessionConstants.CATALOG_SOURCE_KEY (Request scope) PipelineSessionConstants.CATALOG_DESTINATION_KEY (Request scope)
Optional Pipeline Session Attributes	None
Updated Pipeline Session Attributes	Both request and session scoped Pipeline session attributes keyed by the CATALOG_DESTINATION_KEY Pipeline session attribute.
Removed Pipeline Session Attributes	Both request and session scoped Pipeline session attributes keyed by the CATALOG_SOURCE_KEY Pipeline session attribute.
Type	Java object
JNDI Name	None
Exceptions	None

RemoveAttributePC

Class Name	com.beasys.commerce.ebusiness.catalog.pipeline. RemoveAttributePC
Description	Removes a Pipeline session attribute value from a source attribute. The source attribute is specified by the CATALOG_SOURCE_KEY Pipeline session attributes.
Required Pipeline Session Attributes	PipelineSessionConstants.CATALOG_SOURCE_KEY (Request scope)
Optional Pipeline Session Attributes	None
Updated Pipeline Session Attributes	None
Removed Pipeline Session Attributes	Both request and session scoped Pipeline session attributes keyed by the CATALOG_SOURCE_KEY Pipeline session attribute.
Type	Java object
JNDI Name	None
Exceptions	None

SearchPC

Class Name	<code>com.beasys.commerce.ebusiness.catalog.pipeline.SearchPC</code>
Description	Performs a Catalog query based upon the <code>CatalogQuery</code> in the Pipeline session <code>CATALOG_QUERY</code> attribute. The resultant <code>ViewIterator</code> of <code>ProductItems</code> is placed into the Pipeline session as the <code>CATALOG_SEARCH_RESULTS</code> attribute. The view size of the resultant <code>ViewIterator</code> may be specified with the optional <code>CATALOG_VIEW_SIZE</code> Pipeline session attribute.
Required Pipeline Session Attributes	<code>PipelineSessionConstants.CATALOG_QUERY</code> (Request scope)
Optional Pipeline Session Attributes	<code>PipelineSessionConstants.CATALOG_VIEW_SIZE</code> (Request scope)
Updated Pipeline Session Attributes	<code>PipelineSessionConstants.CATALOG_SEARCH_RESULTS</code>
Removed Pipeline Session Attributes	None
Type	Java object
JNDI Name	None
Exceptions	<code>PipelineFatalException</code> on Catalog finder error, Catalog general error, EJB create error, or JNDI lookup error.

The Catalog JSP Tag Library: cat.tld

Table 5-21 summarizes the tags that comprise the WebLogic Commerce Server Product Catalog JSP Tag Library. To use the functionality provided by a catalog tag, you must import the `cat.tld` tag library into your JSP file, as follows:

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

These tags are used in the JSP templates that comprise the default Product Catalog. You can add or remove tags in your use of the JSP templates to match your specific formatting requirements.

The tag elements always start with `<catalog:` and are followed by the type of operation and one or more parameters. The operation, such as `getProperty`, always follows `<catalog:` without a space or breaking line. You do include a space between the tag element name and its parameters. Each parameter uses an equal sign and the parameter's value is enclosed in double quotes. End each tag with the forward slash, followed by the closing angle bracket: `/>`.

Table 5-21 JSP Tag Library

Tag	Description
<code><catalog:getProperty></code>	Retrieves a property for display from a specified <code>ProductItem</code> or <code>Category</code> . Either explicit or implicit properties may be retrieved.
<code><catalog:iterateViewIterator></code>	Iterates a specified <code>ViewIterator</code> . The <code>ViewIterator</code> may be iterated either by <code>View</code> (one <code>View</code> per iteration) or by contained <code>Catalog</code> item (one <code>ProductItem</code> or <code>Category</code> per iteration).
<code><catalog:iterateThroughView></code>	Iterates a specified <code>ViewIterator</code> through the <code>ProductItems</code> or <code>Categories</code> contained within a specified <code>View</code> .

Subsequent sections in this chapter describe the tags in more detail.

Note: Items enclosed within brackets [] are optional.

The <catalog:getProperty> Tag

Use the <catalog:getProperty> tag to retrieve a property for display from either a `ProductItem` or `Category`. The property can either be an explicit property (a property that can be retrieved using a `get` method on the Catalog item) or an implicit property (a property available through the `ConfigurableEntity` `getProperty` methods on the Catalog item). The tag first checks to see if the specified property can be retrieved as an explicit property. If it cannot, the specified property is retrieved as an implicit property.

Format

```
<catalog:getProperty
  object="<%= objectReference %>"
  propertyName="propertyName"
  [getterArgument="<%= getterArgumentReference %>"]
  [id="newInstance"]
  [returnType="returnType"]
/>
```

Attributes

`object`
Denotes a reference to a `ProductItem` or `Category` object that must be presented in the form `<%= objectReference %>`.

`propertyName`
Name of the property to retrieve. If the property is explicit, it may be one of the following values shown in Table 5-22.

Table 5-22 `propertyName` Values

Property Name	Catalog Item Type
"contributor coverage creationDate creator description image key language modifiedDate name publisher relation rights source"	Catalog Item (common properties)

Table 5-22 propertyName Values

Property Name	Catalog Item Type
"jsp"	Category
"availability currentPrice format jsp msrp shippingCode taxCode type visible"	ProductItem

getterArgument

Denotes a reference to an object supplied as an argument to an explicit property getter method. The object must be presented in the form `<%= getterArgumentReference %>`.

id

If the `id` attribute is supplied, the value of the retrieved property will be available in the variable name to which `id` is assigned. Otherwise, the value of the property is inlined.

returnType

If the `id` attribute is supplied, declares the type of the variable specified by the `id` attribute.

Example

The following example retrieves the Detail JSP information from an existing `ProductItem`:

```
<catalog:getProperty
object="<%= item %>"
    propertyName="Jsp"
    getterArgument=
        "<%= new Integer(ProductItem.DETAILED_DISPLAY_JSP_INDEX) %>"
    id="detailJspInfo"
returnType="com.beasys.commerce.ebusiness.catalog.JspInfo"
/>
```

The <catalog:iterateViewIterator> Tag

Use this tag to iterate through a `ViewIterator`. A `ViewIterator` is an iterator over a potentially large collection of remote data that is broken up into a series of fixed sized Views. `ViewIterators` are returned from all Catalog service API methods that may potentially return a large set of `ProductItems` or `Categories`. This tag allows you to iterate the `ViewIterator` one item (`ProductItem` or `Category`) at a time (the default behavior) or by an entire `View` (fixed size set of `ProductItems` or `Categories`) at a time. It is important to note that this tag does not reset the state of the `ViewIterator` upon completion.

Format

```
<catalog:iterateViewIterator
  iterator="<%= iteratorReference %>"
  id="newInstance"
  [returnType="returnType"]
  [iterateByView="{true|false}"]>
</catalog:iterateViewIterator/>
```

Attributes

iterator
Denotes a reference to a `ViewIterator` object that must be presented in the form `<%= iteratorReference %>`.

id
The value of the current iterated object will be available in the variable name to which the `id` is assigned.

returnType
Declares the type of the variable specified by the `id` attribute. Defaults to `java.lang.Object`. If `iterateByView` is `true`, the type is assumed to be `com.beasys.commerce.ebusiness.catalog.View`.

iterateByView
Specifies whether to iterate the `ViewIterator` by `View` or by `Catalog` item. If not specified, the `ViewIterator` will be iterated by `Catalog` item.

Examples

The following example displays the keys of all Categories in a ViewIterator:

```
<catalog:iterateViewIterator
  iterator="<%= myIterator %>"
  id="category"
  returnType="com.beasys.commerce.ebusiness.catalog.Category">
  <%= category.getKey().toString() %>
</catalog:iterateViewIterator>
```

The following example displays all the Views contained within a ViewIterator:

```
<catalog:iterateViewIterator
  iterator="<%= myIterator %>"
  id="view"
  returnType="com.beasys.commerce.ebusiness.catalog.ViewIterator"
  iterateByView="true">
  <%= view.toString() %>
</catalog:iterateViewIterator>
```

The `<catalog:iterateThroughView>` Tag

Use this tag to iterate through a `View` of a specified `ViewIterator`. The tag will iterate the `View` one `Catalog` item at a time until the end of the `View` is reached. If you do not specify a specific `View` (by index) through which to iterate, the current `View` of the `ViewIterator` is used. It is important to note that this tag does not reset the state of the `ViewIterator` upon completion.

Format

```
<catalog:iterateThroughView
  iterator="<%= iteratorReference %>"
  id="newInstance"
  [returnType="returnType"]
  [viewIndex="<%= viewIndexIntegerReference %>"]>
</catalog:iterateThroughView>
```

Attributes

`iterator`
Denotes a reference to a `ViewIterator` object that must be presented in the form `<%= iteratorReference %>`

`id`
The value of the current iterated object will be available in the variable name to which the `id` is assigned.

`returnType`
Declares the type of the variable specified by the `id` attribute. Defaults to `java.lang.Object`.

`viewIndex`
Specifies the index of the `View` (relative to the start of the `ViewIterator`) through which to iterate. The referenced object must be presented in the form `<%= viewIndexIntegerReference %>`.

Examples

The following example displays the keys of all the `ProductItems` contained in the current `View` of a specified `ViewIterator`:

```
<catalog:iterateThroughView
    iterator="<%= myIterator %>"
    id="item"
    returnType="com.beasys.commerce.ebusiness.catalog.ProductItem">
<%= item.getKey().toString() %>
</catalog:iterateThroughView>
```

The following example displays the keys of all the `ProductItems` contained in the first `View` of a specified `ViewIterator`:

```
<catalog:iterateThroughView
    iterator="<%= myIterator %>"
    id="item"
    returnType="com.beasys.commerce.ebusiness.catalog.ProductItem"
    viewIndex="new Integer(0)">
    <%= item.getKey().toString() %>
</catalog:iterateThroughView>
```

5 *The Product Catalog JSP Templates and Tag Library*

6 Using the API to Extend the Product Catalog

This chapter describes the various options available for extending, customizing, or writing third-party integrations for the WebLogic Commerce Server product catalog. The catalog defines interfaces for services that are required to access and administer an electronic product catalog. The architecture is built on Java 2 Enterprise Edition (J2EE) standards-based components and BEA WebLogic Server.

In addition an implementation of the services is provided which defines an electronic product catalog that uses JDBC as a persistence mechanism.

Note: The descriptions in this chapter assume that you are an experienced EJB developer.

This topic includes the following sections:

- Overview of the Product Catalog API
- Catalog Architecture and Services
 - Catalog Architecture
 - Catalog Manager
 - Product Item Manager
 - Category Manager
 - Custom Data Manager
 - Catalog Query Manager
- The Catalog Cache
- Writing Your Own Catalog Service

Note: In this chapter, the environment variable `WL_COMMERCE_HOME` is used to represent the directory in which you installed the WebLogic Commerce Server software.

Overview of the Product Catalog API

The product catalog API package structure is organized as follows:

`com.beasys.commerce.ebusiness.catalog` is the main end-user package for Product Catalog development. It contains all the commonly accessed classes for accessing both Product Items and Categories.

`com.beasys.commerce.ebusiness.catalog.loader` contains the classes necessary to support the command-line Product Catalog database bulk loader, DBLoader. This loader allows you to easily and quickly import data into the Product Catalog from simple character separated value files.

`com.beasys.commerce.ebusiness.catalog.pipeline` contains all Pipeline Components that facilitate accessing the Product Catalog from JavaServer Pages.

`com.beasys.commerce.ebusiness.catalog.service` contains the base services on top of which all pluggable Product Catalog services are implemented. Additionally, this packaged contains several sub-packages for managing Product Items and Categories, searching the Product Catalog, and providing custom attribute support.

`com.beasys.commerce.ebusiness.catalog.service.category` contains the classes that define a pluggable service to manage the Categories and hierarchical structure of the Product Catalog.

`com.beasys.commerce.ebusiness.catalog.service.data` contains the classes that define a pluggable service to manage the custom attributes for Product Items and Categories within the Product Catalog.

`com.beasys.commerce.ebusiness.catalog.service.item` contains the classes that define a pluggable service to manage the Product Items within the Product Catalog.

`com.beasys.commerce.ebusiness.catalog.service.query` contains the classes that define a pluggable service to perform powerful searching of the Product Catalog. The Product Items within the Catalog can be searched using keywords or boolean search expressions across their attributes.

`com.beasys.commerce.ebusiness.catalog.sql` contains the classes that provide a database persistence model for the Product Catalog. Industry standard JDBC and SQL are used to ensure compatibility with a wide range of databases.

`com.beasys.commerce.ebusiness.catalog.util` contains the classes that provide utility methods for the Product Catalog.

`com.beasys.commerce.ebusiness.catalog.webflow` contains all Input Processors that facilitate accessing the Product Catalog from JavaServer Pages.

Catalog Architecture and Services

The WebLogic Commerce Server product catalog architecture divides the functionality of the product catalog into five functional areas, each of which requires an implementation of an associated Product Catalog server interface. The five services are:

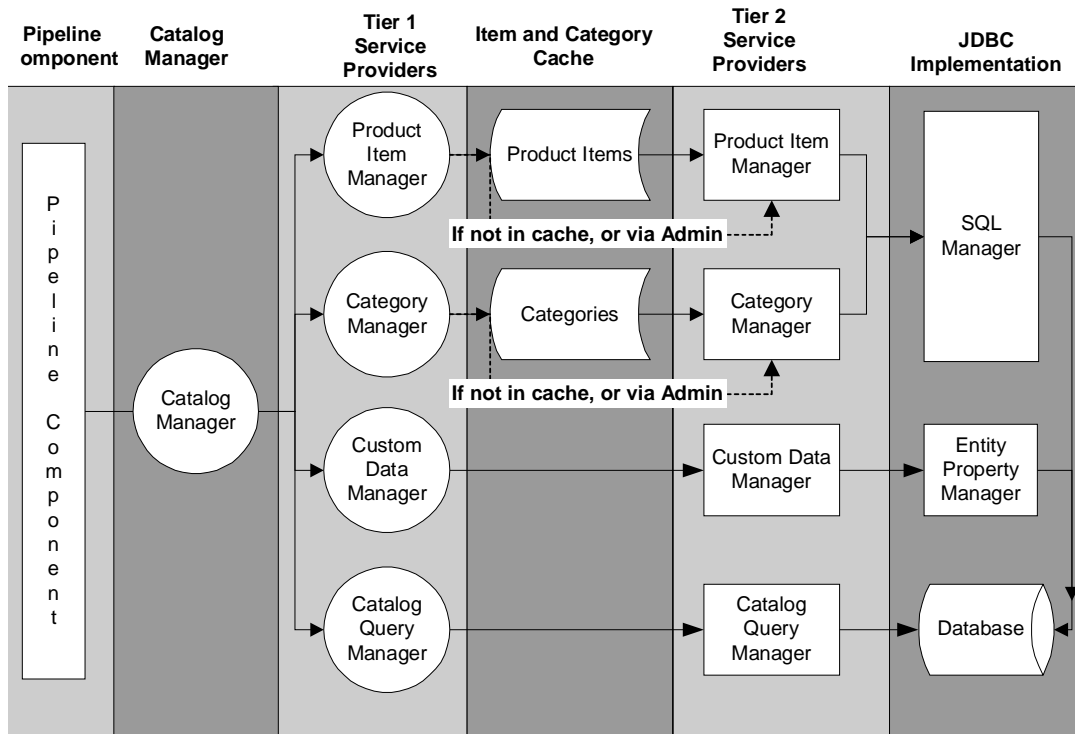
- Catalog Manager
- Product Item Manager
- Category Manager
- Custom Data Manager
- Catalog Query Manager

All services are implemented using J2EE-compliant stateless session EJBs. These EJBs separate the functionality of the catalog into discrete, pluggable components.

Catalog Architecture

Figure 6-1 illustrates the catalog architecture.

Figure 6-1 Product Catalog Architecture



For example, the process of displaying a product item in a user's browser involves the following phases:

1. The Web browser opens the JavaServer Page (JSP) that is running in an instance of the WebLogic Server.
2. One or more Catalog Pipeline Components are executed. For example, `GetProductItemPC` is executed when a user views a product item.
3. The Pipeline Component finds the `CatalogManager` stateless session EJB.
4. The Pipeline Component requests a service from the `CatalogManager` (such as the service provided by the `ProductItemManager`) and receives a stateless session EJB that implements the `ProductItemManager` interface.

5. The Pipeline Component calls a method on the `ProductItemManager` interface. For example, `getItem(12345)` – where 12345 is the unique identifier for a product item, in the form of a Stock Keeping Unit (SKU) number.
6. The catalog services analyze the incoming request and the in-memory cache. If the request can be satisfied using in-memory cached data, the cached data is returned. Otherwise, a service provider is selected (based upon deployment settings) that can handle the request, and the corresponding method is invoked on the service (which must also implement the `ProductItemManager` interface). The return value from the service is added to the cache and the return value is propagated back to the Pipeline Component.
7. The Pipeline Component then adds the result of the Catalog request to the Pipeline Session.
8. The JSP uses the WebLogic Commerce Server tag libraries to extract the results of the Catalog request from the Pipeline Session and formats the results as HTML.

Catalog Manager

The Catalog Manager will not typically require customization. Its main purpose is to provide a single point of access to the other catalog services. Table 6-1 shows the method summary for the `CatalogManager` interface.

Table 6-1 Method Summary for the `CatalogManager` Interface

Return Type	Method Signature
<code>CatalogRequest</code>	<code>createAdminCatalogRequest()</code> Creates a <code>CatalogRequest</code> with administrative user access permissions.
<code>CatalogRequest</code>	<code>createCatalogRequest()</code> Creates a <code>CatalogRequest</code> with default user access permissions.
<code>CatalogQueryManager</code>	<code>getCatalogQueryManager(CatalogRequest request)</code> Returns the <code>CatalogQueryManager</code> catalog service.

6 Using the API to Extend the Product Catalog

Table 6-1 Method Summary for the CatalogManager Interface

CategoryManager	getCategoryManager(CatalogRequest request) Returns the CategoryManager catalog service.
CustomDataManager	getCustomDataManager(CatalogRequest request) Returns the CustomDataManager catalog service.
ProductItemManager	getProductItemManager(CatalogRequest request) Returns the ProductItemManager catalog service.
void	onRemoveItem(CatalogRequest request, CatalogItemKey item) Callback method.

The JNDI names of the EJBs returned from the `CatalogManager` methods are defined in the `weblogic-ejb-jar.xml` deployment descriptor for the `CatalogManager`, as shown in Listing 6-1.

Note: You can find the `ejb-jar.xml` and `weblogic-ejb-jar.xml` deployment descriptor files in the `ebusiness.jar` file, which is in `WL_COMMERCE_HOME\lib`.

Listing 6-1 CatalogManager Deployment Descriptor

```
<weblogic-enterprise-bean>
  <ejb-name>com.beasys.commerce.ebusiness.catalog.CatalogManager</ejb-name>
  <reference-descriptor>
    <ejb-reference-description>
      <ejb-ref-name>ejb/ProductItemManager</ejb-ref-name>
      <jndi-name>
        com.beasys.commerce.ebusiness.catalog.service.item.ProductItemManager
      </jndi-name>
    </ejb-reference-description>
  </reference-descriptor>
```

```
<ejb-ref-name>ejb/CategoryManager</ejb-ref-name>
<jndi-name>
com.beasys.commerce.ebusiness.catalog.service.category.CategoryManager
</jndi-name>
</ejb-reference-description>
<ejb-reference-description>
  <ejb-ref-name>ejb/CatalogQueryManager</ejb-ref-name>
  <jndi-name>
com.beasys.commerce.ebusiness.catalog.service.query.CatalogQueryManager
  </jndi-name>
</ejb-reference-description>
<ejb-reference-description>
  <ejb-ref-name>ejb/CustomDataManager</ejb-ref-name>
  <jndi-name>
com.beasys.commerce.ebusiness.catalog.service.data.CustomDataManager
  </jndi-name>
</ejb-reference-description>
<ejb-reference-description>
  <ejb-ref-name>ejb/CatalogManager</ejb-ref-name>
  <jndi-name>
com.beasys.commerce.ebusiness.catalog.CatalogManager
  </jndi-name>
</ejb-reference-description>
</reference-descriptor>
<jndi-name>
com.beasys.commerce.ebusiness.catalog.CatalogManager
</jndi-name>
```

</weblogic-enterprise-bean>

Product Item Manager

The Product Item Manager is responsible for creating, getting, updating, and deleting items within the catalog. Table 6-2 shows the method summary for the `ProductItemManager` interface.

Table 6-2 Method Summary for the ProductItemManager Interface

Return Type	Method Signature
void	<code>createItem(CatalogRequest request, ProductItem product)</code> Creates a new product item.
ProductItem	<code>getItem(CatalogRequest request, ProductItemKey productKey)</code> Returns the product item with the specified key.
int	<code>getItemCount(CatalogRequest request)</code> Returns the number of product items in the product catalog.
ProductItemKey[]	<code>getItemKeys(CatalogRequest request, int beginIndex, int endIndex)</code> Returns an array over all existing product item keys within the specified ordered range.
ViewIterator	<code>getItems(CatalogRequest request, int viewSize)</code> Returns a ViewIterator over all existing product items.
ProductItem[]	<code>getItems(CatalogRequest request, ProductItemKey[] productKeys)</code> Returns the product items with the given product item keys.
java.lang.String[]	<code>getKeywords(CatalogRequest request, ProductItemKey productKey)</code> Returns the keywords associated with a given product item.

Table 6-2 Method Summary for the ProductItemManager Interface

Return Type	Method Signature
void	<pre>setKeywords(CatalogRequest request, ProductItemKey productKey, java.lang.String[] keywords)</pre> Sets the keywords for a given product item.
void	<pre>removeItem(CatalogRequest request, ProductItemKey productKey)</pre> Removes a product item.
void	<pre>updateItem(CatalogRequest request, ProductItem product)</pre> Updates a product item.

Category Manager

The Category Manager is responsible for managing the hierarchical structure of the electronic product catalog. It defines the interface that allows the hierarchy to be created and modified, as well as the mapping of items into categories to be managed.

Table 6-3 shows the method summary for the `CategoryManager` interface.

Table 6-3 Method Summary of the CategoryManager Interface

Return Type	Method Signature
void	<pre>addItem(CatalogRequest request, CategoryKey categoryKey, ProductItemKey itemKey)</pre> Adds an item to the specified category.
void	<pre>createCategory(CatalogRequest request, CategoryKey parentKey, Category category)</pre> Creates a subcategory within the supplied parent category.

6 Using the API to Extend the Product Catalog

Table 6-3 Method Summary of the CategoryManager Interface (Continued)

Return Type	Method Signature
Category[]	<code>getAncestors(CatalogRequest request, CategoryKey categoryKey)</code> Returns the ancestors of the specified category in ascending order.
Category[]	<code>getCategories(CatalogRequest request, CategoryKey[] categoryKeys)</code> Returns the categories with the given category keys.
ViewIterator	<code>getCategories(CatalogRequest request, int viewSize)</code> Returns a <code>ViewIterator</code> over all existing categories.
Category	<code>getCategory(CatalogRequest request, CategoryKey categoryKey)</code> Returns the category with the given category key.
int	<code>getCategoryCount(CatalogRequest request)</code> Returns the total number of categories in the product catalog.
CategoryKey[]	<code>getCategoryKeys(CatalogRequest request, int beginIndex, int endIndex)</code> Returns an array of all existing category keys within the specified ordered range.
int	<code>getItemCount(CatalogRequest request, CategoryKey categoryKey)</code> Returns the number of product items associated with the specified category.
ProductItemKey[]	<code>getItemKeys(CatalogRequest request, CategoryKey categoryKey, int beginIndex, int endIndex)</code> Returns an array of all product item keys of the specified category within the specified ordered range.
ViewIterator	<code>getItems(CatalogRequest request, CategoryKey categoryKey, int viewSize)</code> Returns a <code>ViewIterator</code> over all product items of the specified category.

Table 6-3 Method Summary of the CategoryManager Interface (Continued)

Return Type	Method Signature
int	<p><code>getOrphanedItemCount(CatalogRequest request)</code></p> <p>Returns the number of orphaned items in the catalog. An orphaned item (uncategorized) is an item that does not belong to any categories.</p>
ProductItemKey[]	<p><code>getOrphanedItemKeys(CatalogRequest request, int beginIndex, int endIndex)</code></p> <p>Returns an array of all existing orphaned item keys within the specified ordered range. An orphaned item (uncategorized) is an item that does not belong to any categories.</p>
ViewIterator	<p><code>getOrphanedItems(CatalogRequest request, int viewSize)</code></p> <p>Returns a <code>ViewIterator</code> over all existing orphaned items. An orphaned item (uncategorized) is an item that does not belong to any categories.</p>
Category	<p><code>getParent(CatalogRequest request, CategoryKey categoryKey)</code></p> <p>Returns the parent of the specified category.</p>
Category	<p><code>getRootCategory(CatalogRequest request)</code></p> <p>Returns the root category.</p>
int	<p><code>getSiblingCount(CatalogRequest request, CategoryKey categoryKey)</code></p> <p>Returns the number of siblings associated with the specified category.</p>
CategoryKey[]	<p><code>getSiblingKeys(CatalogRequest request, CategoryKey categoryKey, int beginIndex, int endIndex)</code></p> <p>Returns an array of all sibling keys of the specified category within the specified ordered range.</p>
ViewIterator	<p><code>getSiblings(CatalogRequest request, CategoryKey categoryKey, int viewSize)</code></p> <p>Returns a <code>ViewIterator</code> over all siblings of the specified category.</p>

6 Using the API to Extend the Product Catalog

Table 6-3 Method Summary of the CategoryManager Interface (Continued)

Return Type	Method Signature
ViewIterator	<code>getSubCategories(CatalogRequest request, CategoryKey categoryKey, int viewSize)</code> Returns a ViewIterator over all subcategories of the specified category.
int	<code>getSubCategoryCount(CatalogRequest request, CategoryKey categoryKey)</code> Returns the number of subcategories associated with the specified category.
CategoryKey[]	<code>getSubCategoryKeys(CatalogRequest request, CategoryKey categoryKey, int beginIndex, int endIndex)</code> Returns an array of all sub category keys of the specified category within the specified ordered range.
void	<code>moveCategory(CatalogRequest request, CategoryKey categoryKey, CategoryKey newParentKey)</code> Moves the specified category under the specified parent.
void	<code>removeCategory(CatalogRequest request, CategoryKey categoryKey)</code> Removes the specified category.
void	<code>removeItem(CatalogRequest request, CategoryKey categoryKey, ProductItemKey itemKey)</code> Removes an item from the specified category.
void	<code>updateCategory(CatalogRequest request, Category category)</code> Updates an existing category.

Custom Data Manager

The Custom Data Manager defines an interface that allows custom attributes (attributes not defined in the `ProductItem` interface) to be persisted for Product Items. The `getProperty` and `setProperty` Configurable Entity methods on Categories and Product Items use the Custom Data Manager service to allow a client to retrieve and set customer attributes.

Table 6-4 shows the method summary for the `CustomDataManager` interface.

Table 6-4 Method Summary for the CustomDataManager Interface

Return Type	Method Signature
<code>java.util.Map</code>	<pre>getProperties(CatalogRequest request, CatalogItemKey itemKey)</pre> Retrieve all the property values.
<code>java.util.Map</code>	<pre>getProperties(CatalogRequest request, CatalogItemKey itemKey, java.lang.String namespace)</pre> Retrieve all the property values within a namespace.
<code>java.lang.Object</code>	<pre>getProperty(CatalogRequest request, CatalogItemKey itemKey, java.lang.String namespace, java.lang.String key, java.lang.Object defaultValue)</pre> Retrieve the value associated with the named key.
<code>void</code>	<pre>removeProperties(CatalogRequest request, CatalogItemKey itemKey)</pre> Remove all the properties for an item.
<code>java.lang.Object</code>	<pre>removeProperty(CatalogRequest request, CatalogItemKey itemKey, java.lang.String namespace, java.lang.String key)</pre> Remove the property associated with the named key.

Table 6-4 Method Summary for the CustomDataManager Interface (Continued)

Return Type	Method Signature
void	<pre>setProperty(CatalogRequest request, CatalogItemKey itemKey, java.lang.String namespace, java.lang.String key, java.lang.Object value)</pre> Associate the specified value with the named key.

Catalog Query Manager

The Catalog Query Manager is responsible for searching the Catalog for Product Items. It currently defines two types of catalog search: keyword search and query-based search.

The keyword search is a search of the keywords associated with a product item. Query-based search allows a complex Boolean expression on any of the item attributes to be evaluated.

Table 6-5 shows the method summary for the `CatalogQueryManager` interface.

Table 6-5 Method Summary for the CatalogQueryManager Interface

Return Type	Method Signature
<code>ProductItemKey[]</code>	<pre>search(CatalogRequest request, CatalogQuery query)</pre> Returns the keys of product items that met the criteria of the supplied catalog query object.
<code>ViewIterator</code>	<pre>search(CatalogRequest request, CatalogQuery query, int viewSize)</pre> Returns a <code>ViewIterator</code> over all product items that met the criteria of the supplied catalog query object.

For related information, see the section “Query-based Search Syntax” on page 5-74.

The Catalog Cache

The catalog architecture includes a powerful caching mechanism for items and categories within the product catalog. Integrators can choose between integrating services in front of the cache or behind the cache. Currently the `ProductItemManager` and `CategoryManager` benefit from the caching architecture, as illustrated earlier in this chapter in Figure 6-1.

Replacing the JNDI name of a bean in the `CatalogManager`'s deployment descriptor will replace a service in front of the cache. The service will have to implement its own caching mechanism or forgo the benefits of caching.

The services defined by BEA, specified in the deployment descriptor for the `CatalogManager`, implement the caching for access to items and categories. The following beans query the cache and returned cached data if available; otherwise they delegate to the beans specified in their deployment descriptors:

```
com.beasys.commerce.ebusiness.catalog.service.item.ProductItemManager
com.beasys.commerce.ebusiness.catalog.service.category.CategoryManager
```

By editing the deployment descriptors for the `ProductItemManager` and `CategoryManager` beans, the functionality of the product catalog can be extended behind the cache. This enables developers to concentrate on the persistence model for the catalog without worrying about the caching architecture. For example, in Listing 6-2, you could replace the current delegate service provider class (`JdbcCategoryManager`) with the name of a new session bean that implements the `CategoryManager` interface. This listing is from the `ejb-jar.xml` deployment descriptor file (platform independent).

Listing 6-2 CategoryManager Deployment Descriptor

```
<session>
  <ejb-name>
    com.beasys.commerce.ebusiness.catalog.service.category.CategoryManager
  </ejb-name>
  <home>
    com.beasys.commerce.ebusiness.catalog.service.category.CategoryManagerHome
  </home>
  <remote>com.beasys.commerce.ebusiness.catalog.service.category.CategoryManager
</remote>
```

6 Using the API to Extend the Product Catalog

```
<ejb-class>
com.beasys.commerce.ebusiness.catalog.service.category.CategoryManagerImpl
</ejb-class>
<session-type>Stateless</session-type>
<transaction-type>Container</transaction-type>

<!-- one specifies the delegateName to tell the Bridge component (the one
      used by the catalog manager which ejb to delegate to. That way, one
      can change delegates by changing the env-entry...
-->

<env-entry>
  <env-entry-name>delegateName</env-entry-name>
  <env-entry-type>java.lang.String</env-entry-type>
  <env-entry-value>ejb/JdbcCategoryManager</env-entry-value>
</env-entry>

<ejb-ref>
  <ejb-ref-name>ejb/JdbcCategoryManager</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <home>
com.beasys.commerce.ebusiness.catalog.service.category.JdbcCategoryManagerHome
  </home>
  <remote>
    com.beasys.commerce.ebusiness.catalog.service.category.JdbcCategoryManager
  </remote>
</ejb-ref>

<ejb-ref>
  <ejb-ref-name>ejb/CatalogManager</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>

  <home>
    com.beasys.commerce.ebusiness.catalog.CatalogManagerHome
  </home>
  <remote>
    com.beasys.commerce.ebusiness.catalog.CatalogManager
  </remote>
</ejb-ref>
</session>
```

Again, the previous listing is from the `ejb-jar.xml` deployment descriptor file. You would need to make corresponding changes in the `weblogic-ejb-jar.xml` file. The `ejb-jar.xml` file (platform independent) and `weblogic-ejb-jar.xml` file (platform specific) file are packaged in the `ebusiness.jar` file. (This JAR file can be found in the `WL_COMMERCE_HOME\lib` directory, where `WL_COMMERCE_HOME` is the directory in which you installed WebLogic Commerce Server.) For related information, see the *WebLogic Server EJB Reference* documentation at

http://www.weblogic.com/docs51/classdocs/API_ejb/EJB_reference.html, and the *WebLogic Server Deployment Guides* at <http://www.weblogic.com/docs51/techdeploy/index.html>.

Writing Your Own Catalog Service

This section describes the steps required to implement Product Catalog services, by way of an example. In this example, we replace the `JdbcProductItemManager` and the `JdbcCatalogQueryManager` with non-JDBC based implementations. Both provide simple (that is, not suitable for production) implementations based around storing items in memory and serializing them to (and from) disk.

Also outlined in this section are the changes to the Catalog Services deployment description that are required to plug in the new service implementation. Because these new services reside “behind” the catalog caching mechanism (see the Tier 2 portion of Figure 6-1), the new services can take advantage of the powerful caching features of the WLCS catalog.

To implement the new services, the general steps are as follows:

1. Create the new services
2. Compile the new services
3. Adjust the service deployment descriptor
4. Deploy the new services

Note: Steps 1 and 3 are described in the remainder of this chapter. For information about steps 2 and 4, please refer to the *BEA WebLogic Server Deployment Guides* at <http://www.weblogic.com/docs51/techdeploy/index.html>.

The following topics are covered in this section:

- Create New Services
- Changes to `ejb-jar.xml`
- Changes to `weblogic-ejb-jar.xml`

6 Using the API to Extend the Product Catalog

The `ejb-jar.xml` and `weblogic-ejb-jar.xml` files are packaged in the `ebusiness.jar` file, which can be found in the `WL_COMMERCE_HOME\lib` directory, where `WL_COMMERCE_HOME` is the directory in which you installed WebLogic Commerce Server.

You can find all the source code shown in this section in the `WL_COMMERCE_HOME/src/examples/catalog/file` directory. (The updated deployment descriptors are not in this directory, however.)

Warning: This section assumes that you are familiar with building and deploying EJBs. This section also describes modifications to WebLogic Commerce Server deployment JAR files; therefore, it is important that you first back up all files and JAR libraries that you intend to modify. You should work in a new directory, such as `WL_COMMERCE_HOME/src/myexamples/`, and when you are ready to deploy, change the path to the `weblogic.ejb.deploy` value in the `weblogic.properties` file. For example, in the following:

```
weblogic.ejb.deploy=\
D:/WebLogicCommerceServer3.1/lib/foundation.jar,\
D:/WebLogicCommerceServer3.1/lib/axiom.jar,\
D:/WebLogicCommerceServer3.1/lib/ebusiness.jar,\
.
.
.
```

You could change this in `WL_COMMERCE_HOME/weblogic.properties` to:

```
weblogic.ejb.deploy=\
D:/WebLogicCommerceServer3.1/lib/foundation.jar,\
D:/WebLogicCommerceServer3.1/lib/axiom.jar,\
D:/WebLogicCommerceServer3.1/lib/my-ebusiness.jar,\
.
.
.
```

Note: BEA may post build scripts for the examples in this section on the BEA Developer Center at <http://developer.bea.com/>, after the 3.1 release. Please check the BEA Developer Center periodically for news about downloading the build scripts. Any announcement about the build scripts will also be posted on the WebLogic Commerce Server documentation Web site at <http://edocs.bea.com/wlc310s/index.htm>.

Create New Services

The first step in creating a new catalog service is to implement the corresponding Stateless Session EJB service API. Some of the files are optional, as explained in the following summary. After the summary, sample source code is provided for the implementation files, `FileCatalogQueryManagerImpl.java` and `FileProductItemManagerImpl.java`. Again, you can find this source code in the following directory:

`WL_COMMERCE_HOME/src/examples/catalog/file`

■ `FileCatalogQueryManager.java`

This is the remote interface for the `FileCatalogQueryManager` session bean. The new remote interface is not required, and the remote interface for the bean specified in `ejb-jar.xml` should remain `CatalogQueryManager`.

■ `FileCatalogQueryManagerHome.java`

The Home for the new service is not required, as access to the bean should always be through the environment of the `CatalogManager` or one of the Tier 1 service providers' environments.

■ `FileCatalogQueryManagerImpl.java`

The implementation file for the new Tier 2 service. It implements a file-based Product Catalog search engine.

■ `FileProductItemManager.java`

This is the remote interface for the `FileProductItemManager` session bean. The new remote interface is not required, and the remote interface for the bean specified in `ejb-jar.xml` should remain `ProductItemManager`.

■ `FileProductItemManagerHome.java`

The Home for the new service is not required, as access to the bean should always be through the environment of the `CatalogManager` or one of the Tier 1 service providers' environments.

■ `FileProductItemManagerImpl.java`

The implementation file for the new Tier 2 service contains the new functionality desired. It implements a file-based Product Item management service.

Sample Source Code

Listing 6-3 show sample implementation source code for:

- FileProductItemManagerImpl.java
- FileCatalogQueryManagerImpl.java

After you install WebLogic Commerce Server, these files can be found in the `WL_COMMERCE_HOME/src/examples/catalog/file` directory.

In the following listing, the bold typeface is used to direct your attention to the most relevant lines of code.

Listing 6-3 FileProductItemManagerImpl.java

```
/*
 * B E A   S Y S T E M S
 *
 * C O M M E R C E   C O M P O N E N T S
 *
 * Copyright (c) 1997-2000 BEA Systems, Inc.
 *
 * All Rights Reserved. Unpublished rights reserved under the copyright laws
 * of the United States. The software contained on this media is proprietary
 * to and embodies the confidential technology of BEA Systems, Inc. The
 * possession or receipt of this information does not convey any right to disclose
 * its contents, reproduce it, or use, or license the use, for manufacture or
 * sale, the information or anything described therein. Any use, disclosure, or
 * reproduction without BEA System's prior written permission is strictly
 * prohibited.
 *
 *
 * $Header:$
 */

package com.beasys.commerce.ebusiness.catalog.examples.file;
import com.beasys.commerce.foundation.*;
import com.beasys.commerce.util.*;
import java.util.*;
import java.rmi.*;
import javax.ejb.*;
import javax.naming.*;
//$Import$_Begin ----- CUSTOM CODE -----
import com.beasys.commerce.ebusiness.catalog.*;
import com.beasys.commerce.ebusiness.catalog.service.*;
```

```

import com.beasys.commerce.ebusiness.catalog.service.item.*;
import java.io.*;
//$Import$_End ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
/**
 *
 * @see
com.beasys.commerce.ebusiness.catalog.service.item.FileProductItemManager
 * @see
com.beasys.commerce.ebusiness.catalog.service.item.FileProductItemManagerHome
 */
public class FileProductItemManagerImpl extends
com.beasys.commerce.ebusiness.catalog.service.CatalogServiceImpl
    //$Implements$_Begin ----- CUSTOM CODE -----
        // USER CHANGES: Add interfaces that are implemented here
    //$Implements$_End ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
{

    //$AdditionalAttributeDeclarations$_Begin ----- CUSTOM CODE
    // -----
    private Hashtable            itemTable = null;
    private Hashtable            keywordTable = null;

    //$AdditionalAttributeDeclarations$_End ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    public FileProductItemManagerImpl( )
    {
        super( );
        //$Constructor$_Begin ----- CUSTOM CODE -----
        //$Constructor$_End ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    }
    private void loadData()
    {
        if( itemTable == null || keywordTable == null )
        {
            try
            {
                FileInputStream istream = new FileInputStream( "items.bin" );
                ObjectInputStream objIn = new ObjectInputStream( istream );
                itemTable = (Hashtable) objIn.readObject();
                keywordTable = (Hashtable) objIn.readObject();
            }
            catch( Exception e )
            {
                e.printStackTrace();
            }

            if( itemTable == null )
                itemTable = new Hashtable();
        }
    }
}

```

6 Using the API to Extend the Product Catalog

```
        if( keywordTable == null )
            keywordTable = new Hashtable();
    }
}
private void saveData()
{
    try
    {
        FileOutputStream ostream = new FileOutputStream( "items.bin" );
        ObjectOutputStream objOut = new ObjectOutputStream( ostream );
        objOut.writeObject( itemTable );
        objOut.writeObject( keywordTable );
    }
    catch( Exception e )
    {
        e.printStackTrace();
    }
}
public void ejbCreate( ) throws CreateException
{
    super.ejbCreate( );
    //$EjbCreate$_Begin ----- CUSTOM CODE -----
    // read all the items from the input stream
    loadData();
    //$EjbCreate$_End  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
}
public void ejbPostCreate( ) throws CreateException
{
    super.ejbPostCreate( );

    //$EjbPostCreate$_Begin ----- CUSTOM CODE -----
    //$EjbPostCreate$_End  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
}

public void ejbActivate( ) throws EJBException
{
    super.ejbActivate( );

    //$EjbActivate$_Begin ----- CUSTOM CODE -----
    //$EjbActivate$_End  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
}

public void ejbPassivate( ) throws EJBException
{
    super.ejbPassivate( );

    //$EjbPassivate$_Begin ----- CUSTOM CODE -----
    //$EjbPassivate$_End  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
}
}
```

```

public void ejbRemove( ) throws EJBException
{
    super.ejbRemove( );
    //$EjbRemove$_Begin ----- CUSTOM CODE -----

    saveData();

    itemTable = null;
    keywordTable = null;

    //$EjbRemove$_End ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
}

public void setSessionContext( SessionContext ctx ) throws EJBException
{
    super.setSessionContext( ctx );

    //$SetSessionContext$_Begin ----- CUSTOM CODE -----
    //$SetSessionContext$_End   ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
}

/**
 * Returns the number of product items in the product catalog.
 * @param request The catalog request object
 * @return The number of product items in the product catalog.
 * @throws CatalogException on general error.
 */
public int getItemCount( CatalogRequest request ) throws CatalogException
{
    //$Method int getItemCount(CatalogRequest request)$_Begin -----
CUSTOM CODE -----
    return itemTable.size();
    //$Method int getItemCount(CatalogRequest request)$_End
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
}

/**
 * Returns the product item with the specified key.
 * @param request The catalog request object.
 * @param productKey The key of the target product.
 * @throws CatalogFinderException if the product item could not be found.
 * @throws CatalogException on general error.
 */
public ProductItem getItem( CatalogRequest request, ProductItemKey productKey
) throws CatalogFinderException, CatalogException
{
    //$Method ProductItem getItem(CatalogRequest request, ProductItemKey
productKey)$_Begin ----- CUSTOM CODE -----

```

6 Using the API to Extend the Product Catalog

```
        ProductItem item = (ProductItem) itemTable.get( productKey );

        if( item == null )
            throw new CatalogFinderException( productKey );

        return item;
    //Method ProductItem getItem(CatalogRequest request, ProductItemKey
productKey)$_End  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    }

    /**
     * Returns the product items with the given product item keys.
     * @param request The catalog request object.
     * @param keys The keys of the target product items.
     * @returns The product items with the given product item keys.
     * @throws CatalogFinderException if a product item with a given key does not
     exist.
     * @throws CatalogException on general error.
     */
    public ProductItem[] getItems( CatalogRequest request, ProductItemKey[]
productKeys ) throws CatalogFinderException,CatalogException
    {
        //Method ProductItem[] getItems(CatalogRequest request, ProductItemKey[]
productKeys)$_Begin  ----- CUSTOM CODE -----
        ProductItem[] itemArray = new ProductItem[ productKeys.length ];

        for( int n = 0; n < productKeys.length; n++ )
            itemArray[n] = getItem( request, productKeys[n] );

        return itemArray;
    //Method ProductItem[] getItems(CatalogRequest request, ProductItemKey[]
productKeys)$_End  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    }

    /**
     * Returns an array over all existing product item keys within the specified
     ordered range.
     * @param request The catalog request object.
     * @param beginIndex The lower bound index for returned product item keys.
     * @param endIndex The upper bound index for returned product item keys.
     * @return An array of the product item keys.
     * @throws CatalogException on general error.
     */
    public ProductItemKey[] getItemKeys( CatalogRequest request, int beginIndex,
int endIndex ) throws CatalogException
    {
        //Method ProductItemKey[] getItemKeys(CatalogRequest request, int
beginIndex, int endIndex)$_Begin  ----- CUSTOM CODE -----
        Enumeration keysEnum = itemTable.keys();
```

```

LinkedList keyList = new LinkedList();

int index = 0;

while( keysEnum.hasMoreElements() != false && index < endIndex )
{
    ProductItemKey key = (ProductItemKey) keysEnum.nextElement();

    if( index >= beginIndex && index < endIndex && key != null )
        keyList.add( key );
}

return (ProductItemKey[]) keyList.toArray( new ProductItemKey[0] );
//Method ProductItemKey[] getItemKeys(CatalogRequest request, int
beginIndex, int endIndex)$_End ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
}

/**
 * Returns a ViewIterator over all existing product items.
 * @param request The catalog request object.
 * @param viewSize The view size of the returned ViewIterator.
 * @return A ViewIterator over all existing product items.
 * @throws CatalogException on general error.
 */
public ViewIterator getItems( CatalogRequest request, int viewSize ) throws
CatalogException,RemoteException
{
    //Method ViewIterator getItems(CatalogRequest request, int
viewSize)$_Begin ----- CUSTOM CODE -----
    int numItems = getItemCount( request );
    ProductItemIterator iterator = new ProductItemIterator(
getCatalogManagerJndiName( ), request, numItems, viewSize );
    return iterator;
    //Method ViewIterator getItems(CatalogRequest request, int viewSize)$_End
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
}

/**
 * Returns the keywords associated with a given product item.
 * @param request The catalog request object.
 * @param productKey The key of the target product.
 * @return The keywords associated with the given product item.  If there are
no keywords associated with the item, a zero length <code>String</code>
array is returned.
 * @throws CatalogFinderException if the product item could not be found.
 * @throws CatalogException on general error.
 * @throws SQLException on database access error.
 */

```

6 Using the API to Extend the Product Catalog

```
    public String[] getKeywords( CatalogRequest request, ProductItemKey productKey
) throws CatalogFinderException,CatalogException
    {
        //$Method String[] getKeywords(CatalogRequest request, ProductItemKey
productKey)$_Begin ----- CUSTOM CODE -----
        String[] stringArray = (String[]) keywordTable.get( productKey );

        if( stringArray == null )
            stringArray = new String[0];

        return stringArray;
        //$Method String[] getKeywords(CatalogRequest request, ProductItemKey
productKey)$_End ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    }

    /**
    * Sets the keywords for a given product item.
    * @param request The catalog request object.
    * @param productKey The key of the target product.
    * @param keywords The keywords to associate with the given product item.
    * @throws CatalogFinderException if the product item could not be found.
    * @throws CatalogException on general error.
    * @throws SQLException on database access error.
    */
    public void setKeywords( CatalogRequest request, ProductItemKey productKey,
String[] keywords ) throws CatalogFinderException,CatalogException
    {
        //$Method void setKeywords(CatalogRequest request, ProductItemKey
productKey, String[] keywords)$_Begin ----- CUSTOM CODE -----
        validateAuthorization( request, CatalogRequest.ADMINISTRATION );

        keywordTable.put( productKey, keywords );
        saveData();

        //$Method void setKeywords(CatalogRequest request, ProductItemKey
productKey, String[] keywords)$_End ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    }

    /**
    * Creates a new product item.
    * @param request The catalog request object.
    * @param product The product item to persist.
    * @throws CatalogCreateException if the product item could not be created.
    * @throws CatalogException on general error.
    */
    public void createItem( CatalogRequest request, ProductItem product ) throws
CatalogCreateException,CatalogException
    {
        //$Method void createItem(CatalogRequest request, ProductItem
```



```

product)$_Begin ----- CUSTOM CODE -----
    validateAuthorization( request, CatalogRequest.ADMINISTRATION );
    itemTable.put( (ProductItemKey) product.getKey(), product );
    saveData();

    //$Method void createItem(CatalogRequest request, ProductItem product)$_End
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
}

/**
 * Updates a product item.
 * @param request The catalog request object.
 * @param product The product item to update.
 * @throws CatalogFinderException if the product item could not be found.
 * @throws CatalogException on general error.
 */
public void updateItem( CatalogRequest request, ProductItem product ) throws
CatalogFinderException,CatalogException
{
    //$Method void updateItem(CatalogRequest request, ProductItem
product)$_Begin ----- CUSTOM CODE -----
    validateAuthorization( request, CatalogRequest.ADMINISTRATION );
    itemTable.put( (ProductItemKey) product.getKey(), product );
    saveData();

    //$Method void updateItem(CatalogRequest request, ProductItem product)$_End
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
}

/**
 * Removes a product item.
 * @param request The catalog request object.
 * @param product The product item to remove.
 * @throws CatalogRemoveException if the product item could not be removed.
 * @throws CatalogFinderException if the product item could not be found.
 * @throws CatalogException on general error.
 */
public void removeItem( CatalogRequest request, ProductItemKey productKey )
throws CatalogRemoveException,CatalogFinderException,CatalogException
{
    //$Method void removeItem(CatalogRequest request, ProductItemKey
productKey)$_Begin ----- CUSTOM CODE -----
    validateAuthorization( request, CatalogRequest.ADMINISTRATION );
    itemTable.remove( productKey );
    saveData();

    //$Method void removeItem(CatalogRequest request, ProductItemKey
productKey)$_End    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
}

```

6 Using the API to Extend the Product Catalog

```
}
```

Listing 6-4 shows the source code for `FileCatalogQueryManagerImpl.java`.

Listing 6-4 FileCatalogQueryManagerImpl.java

```
/*
 * B E A   S Y S T E M S
 *
 * C O M M E R C E   C O M P O N E N T S
 *
 * Copyright (c) 1997-2000 BEA Systems, Inc.
 *
 * All Rights Reserved. Unpublished rights reserved under the copyright laws
 * of the United States. The software contained on this media is proprietary
 * to and embodies the confidential technology of BEA Systems, Inc. The
 * possession or receipt of this information does not convey any right to disclose
 * its contents, reproduce it, or use, or license the use, for manufacture or
 * sale, the information or anything described therein. Any use, disclosure, or
 * reproduction without BEA System's prior written permission is strictly
 * prohibited.
 *
 *
 * $Header:$
 */

package com.beasys.commerce.ebusiness.catalog.examples.file;

import com.beasys.commerce.foundation.*;
import com.beasys.commerce.util.*;

import java.util.*;
import java.rmi.*;
import javax.ejb.*;
import javax.naming.*;

// $Import$ _Begin ----- CUSTOM CODE -----
import java.sql.Connection;
import java.sql.SQLException;
import com.beasys.commerce.ebusiness.catalog.*;
import com.beasys.commerce.foundation.expression.Criteria;
import com.beasys.commerce.foundation.expression.Logical;
import com.beasys.commerce.foundation.expression.Expression;
import com.beasys.commerce.util.ExpressionHelper;
```

```

import com.beasys.commerce.util.TypesHelper;
import com.beasys.commerce.ebusiness.catalog.service.query.*;
import com.beasys.commerce.ebusiness.catalog.service.item.*;
// USER CHANGES: Place additional import statements here
//$Import$_End  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

/**
 *
 *
 * @see
com.beasys.commerce.ebusiness.catalog.service.query.JdbcCatalogQueryManager
 * @see
com.beasys.commerce.ebusiness.catalog.service.query.JdbcCatalogQueryManagerHome
 */
public class FileCatalogQueryManagerImpl extends
com.beasys.commerce.ebusiness.catalog.service.CatalogServiceImpl
    //$Implements$_Begin ----- CUSTOM CODE -----
        // USER CHANGES: Add interfaces that are implemented here
    //$Implements$_End  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
{

    //$AdditionalAttributeDeclarations$_Begin ----- CUSTOM CODE -----
    -----
    //$AdditionalAttributeDeclarations$_End
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

    public FileCatalogQueryManagerImpl()
    {
        super();
        //$Constructor$_Begin ----- CUSTOM CODE -----
        // USER CHANGES: Add constructor code here
        //$Constructor$_End  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    }

    public void ejbCreate() throws CreateException
    {
        super.ejbCreate();

        //$EjbCreate$_Begin ----- CUSTOM CODE -----
        //$EjbCreate$_End  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    }

    public void ejbPostCreate() throws CreateException
    {

```

6 Using the API to Extend the Product Catalog

```
        super.ejbPostCreate();

        //$EjbPostCreate$_Begin ----- CUSTOM CODE -----
        // USER CHANGES: Add custom code here
        //$EjbPostCreate$_End   ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    }

    public void ejbActivate() throws EJBException
    {
        super.ejbActivate();

        //$EjbActivate$_Begin ----- CUSTOM CODE -----
        // USER CHANGES: Add custom code here
        //$EjbActivate$_End   ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    }

    public void ejbPassivate() throws EJBException
    {
        super.ejbPassivate();

        //$EjbPassivate$_Begin ----- CUSTOM CODE -----
        // USER CHANGES: Add custom code here
        //$EjbPassivate$_End   ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    }

    public void ejbRemove() throws EJBException
    {
        super.ejbRemove();
        //$EjbRemove$_Begin ----- CUSTOM CODE -----
        // USER CHANGES: Add custom code here
        //$EjbRemove$_End   ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    }

    public void setSessionContext(SessionContext ctx) throws EJBException
    {
        super.setSessionContext(ctx);

        //$SetSessionContext$_Begin ----- CUSTOM CODE -----
        // USER CHANGES: Add custom code here
        //$SetSessionContext$_End   ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    }

    /**
     * Returns the results of the search performed using the supplied catalog
    query object.
     * @param request The catalog request object.
     * @param catalogQuery The catalog query object.
    */
```

```

    * @return An array of product item keys.
    * @throws CatalogException on general error.
    */
    public ProductItemKey[] search(CatalogRequest request,
com.beasys.commerce.ebusiness.catalog.service.query.CatalogQuery query) throws
CatalogException
    {
        //$Method ProductItemKey[] search(CatalogRequest request,
com.beasys.commerce.ebusiness.catalog.service.query.CatalogQuery query)$_Begin
        ----- CUSTOM CODE -----

        LinkedList resultList = new LinkedList();

        Expression expr = null;

        if (query == null)
            throw new CatalogException("Null query");

        try
        {
            if (query instanceof KeywordQuery)
            {
                String[] keywords = ((KeywordQuery) query).getKeywords();

                if (keywords == null || keywords.length <= 0)
                    throw new CatalogException("Empty keywords");

                // get all the items in the catalog
                ProductItemManager productItemManager =
getCatalogManager().getProductItemManager( request );

                ProductItemKey[] itemKeys = productItemManager.getItemKeys( request,
0, productItemManager.getItemCount( request ) );

                if( itemKeys != null )
                {
                    for( int n = 0; n < itemKeys.length; n++ )
                    {
                        if( itemKeys[n] != null )
                        {
                            String[] itemKeywords = productItemManager.getKeywords(
request, itemKeys[n] );

                                if( itemKeywords != null )
                                {
                                    boolean found = false;
                                    // could be optimized...
                                    for( int i = 0; i < itemKeywords.length && found
== false; i++ )

```



```

viewSize);

        //$Method ViewIterator search(CatalogRequest request,
com.beasys.commerce.ebusiness.catalog.service.query.CatalogQuery query, int
viewSize)$_End      ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    }

    //$AdditionalMethod$_Begin ----- CUSTOM CODE -----
    //$AdditionalMethod$_End      ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
}

```

Changes to ejb-jar.xml

In `ejb-jar.xml`, the first step is to change the name of the delegate Session bean in the environment for the Tier 1 service providers. Occurrences of `JdbcProductItemManager` need to be changed to the name of the new Tier 2 service provider: `FileProductItemManager`. This step is done by modifying the Tier 1 service provider to delegate to the new service implementation by adjusting several EJB deployment settings in the `ejb-jar.xml` and `weblogic-ejb-jar.xml` deployment descriptors. Finally, the modified Tier 1 service provider must be redeployed and the new service implementation deployed.

Warning: Create a backup copy of the file before you modify its contents.

Note: In Listing 6-5, lines that should be removed are shown in *italics*. Lines that should be added are shown in **bold**.

Listing 6-5 Changes to the ejb-jar.xml File

```

<session>
  <ejb-name>com.beasys.commerce.ebusiness.catalog.service.data.CustomDataManager
  </ejb-name>

  <env-entry>
    <env-entry-name>delegateName</env-entry-name>
    <env-entry-type>java.lang.String</env-entry-type>
    <env-entry-value>ejb/JdbcProductItemManager</env-entry-value>
    <env-entry-value>ejb/FileProductItemManager</env-entry-value>
  </env-entry>
  <ejb-ref>

```

6 Using the API to Extend the Product Catalog

```
<ejb-ref-name>ejb/JdbcProductItemManager</ejb-ref-name>
<ejb-ref-name>ejb/FileProductItemManager</ejb-ref-name>
<ejb-ref-type>Session</ejb-ref-type>
<home>
com.beasys.commerce.ebusiness.catalog.service.item.JdbcProductItemManagerHome
</home>
<remote>
  com.beasys.commerce.ebusiness.catalog.service.item.JdbcProductItemManager
</remote>
<home>
com.beasys.commerce.ebusiness.catalog.examples.file.FileProductItemManagerHome
</home>
<remote>
  com.beasys.commerce.ebusiness.catalog.examples.file.FileProductItemManager
</remote>
</ejb-ref>
</ejb-ref>
<session>
<ejb-name>
  com.beasys.commerce.ebusiness.catalog.service.query.CatalogQueryManager
</ejb-name>
<env-entry>
  <env-entry-name>delegateName</env-entry-name>
  <env-entry-type>java.lang.String</env-entry-type>
  <env-entry-value>ejb/JdbcCatalogQueryManager</env-entry-value>
  <env-entry-value>ejb/FileCatalogQueryManager</env-entry-value>
</env-entry>
<ejb-ref>
  <ejb-ref-name>ejb/JdbcCatalogQueryManager</ejb-ref-name>
  <ejb-ref-name>ejb/FileCatalogQueryManager</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <home>
com.beasys.commerce.ebusiness.catalog.service.query.JdbcCatalogQueryManagerHome
</home>
<remote>
  com.beasys.commerce.ebusiness.catalog.service.query.JdbcCatalogQueryManager
</remote>
<home>
com.beasys.commerce.ebusiness.catalog.examples.file.FileCatalogQueryManagerHome
</home>
<remote>
  com.beasys.commerce.ebusiness.catalog.examples.file.FileCatalogQueryManager
</remote>
</ejb-ref>
</ejb-ref>
<session>
<ejb-name>
  com.beasys.commerce.ebusiness.catalog.service.item.JdbcProductItemManager
</ejb-name>
```



```
<ejb-name>
  com.beasys.commerce.ebusiness.catalog.examples.file.FileProductItemManager
</ejb-name>
<home>
  com.beasys.commerce.ebusiness.catalog.service.item.ProductItemManagerHome
</home>
<remote>
  com.beasys.commerce.ebusiness.catalog.service.item.ProductItemManager
</remote>
<ejb-class>
  com.beasys.commerce.ebusiness.catalog.service.item.JdbcProductItemManagerImpl
</ejb-class>
<ejb-class>
  com.beasys.commerce.ebusiness.catalog.examples.file.FileProductItemManagerImpl
</ejb-class>
<session-type>Stateless</session-type>
<transaction-type>Container</transaction-type>
<env-entry>
  <env-entry-name>SchemaFile</env-entry-name>
  <env-entry-type>java.lang.String</env-entry-type>
  <env-entry-value>wlcs-catalog</env-entry-value>
</env-entry>
<env-entry>
  <env-entry-name>SqlManagerClass</env-entry-name>
  <env-entry-type>java.lang.String</env-entry-type>
  <env-entry-value>
    com.beasys.commerce.ebusiness.catalog.sql.JdbcSqlManager
  </env-entry-value>
</env-entry>
<session>
  <ejb-name>
    com.beasys.commerce.ebusiness.catalog.service.query.JdbcCatalogQueryManager
  </ejb-name>
  <ejb-name>
    com.beasys.commerce.ebusiness.catalog.examples.file.FileCatalogQueryManager
  </ejb-name>
  <home>
    com.beasys.commerce.ebusiness.catalog.service.query.CatalogQueryManagerHome
  </home>
  <remote>
    com.beasys.commerce.ebusiness.catalog.service.query.CatalogQueryManager
  </remote>
  <ejb-class>
    com.beasys.commerce.ebusiness.catalog.service.query.JdbcCatalogQueryManagerImpl
  </ejb-class>
  <ejb-class>
    com.beasys.commerce.ebusiness.catalog.examples.file.FileCatalogQueryManagerImpl
  </ejb-class>
```

6 Using the API to Extend the Product Catalog

```
<session-type>Stateless</session-type>
<!-- com.beasys.commerce.ebusiness.catalog.service.item.JdbcProductItemManager
-->
<!-- com.beasys.commerce.ebusiness.catalog.examples.file.FileProductItemManager
-->
<method>
  <ejb-name>
    com.beasys.commerce.ebusiness.catalog.service.item.JdbcProductItemManager
  </ejb-name>
  <ejb-name>
    com.beasys.commerce.ebusiness.catalog.examples.file.FileProductItemManager
  </ejb-name>
  <method-name>getCatalogManager</method-name>
</method>
```

Changes to weblogic-ejb-jar.xml

Listing 6-6 shows the deletions and additions needed in the `weblogic-ejb-jar.xml` file. The `weblogic-ejb-jar.xml` file is packaged in the `ebusiness.jar` file, which can be found in the `WL_COMMERCE_HOME\lib` directory, where `WL_COMMERCE_HOME` is the directory in which you installed WebLogic Commerce Server.

Warning: Create a backup copy of the file before you modify its contents.

Note: In Listing 6-6, lines that should be removed are shown in *italics>*. Lines that should be added are shown in **bold>**.

Listing 6-6 Changes to the weblogic-ejb-jar.xml File

```
<weblogic-enterprise-bean>
  <ejb-name>
    com.beasys.commerce.ebusiness.catalog.service.item.ProductItemManager
  </ejb-name>
  <caching-descriptor>
    <initial-beans-in-free-pool>1</initial-beans-in-free-pool>
  </caching-descriptor>
  <reference-descriptor>
    <ejb-reference-description>
      <ejb-ref-name>ejb/JdbcProductItemManager</ejb-ref-name>
      <jndi-name>
```

```
    com.beasys.commerce.ebusiness.catalog.service.item.JdbcProductItemManager
  </jndi-name>
  <ejb-ref-name>ejb/FileProductItemManager</ejb-ref-name>
  <jndi-name>
    com.beasys.commerce.ebusiness.catalog.examples.file.FileProductItemManager
  </jndi-name>

</ejb-reference-description>

<weblogic-enterprise-bean>
  <ejb-name>
    com.beasys.commerce.ebusiness.catalog.service.query.CatalogQueryManager
  </ejb-name>
  <caching-descriptor>  <!--
    <initial-beans-in-free-pool>5</initial-beans-in-free-pool>  -->
</caching-descriptor>
  <reference-descriptor>
    <ejb-reference-description>

      <ejb-ref-name>ejb/JdbcCatalogQueryManager</ejb-ref-name>
      <jndi-name>
        com.beasys.commerce.ebusiness.catalog.service.query.JdbcCatalogQueryManager
      </jndi-name>
      <ejb-ref-name>ejb/FileCatalogQueryManager</ejb-ref-name>
      <jndi-name>
        com.beasys.commerce.ebusiness.catalog.examples.file.FileCatalogQueryManager
      </jndi-name>

    </ejb-reference-description>

  </ejb-reference-description>

<weblogic-enterprise-bean>

  <ejb-name>
    com.beasys.commerce.ebusiness.catalog.service.item.JdbcProductItemManager
  </ejb-name>
  <ejb-name>
    com.beasys.commerce.ebusiness.catalog.examples.file.FileProductItemManager
  </ejb-name>

  <jndi-name>
    com.beasys.commerce.ebusiness.catalog.service.item.JdbcProductItemManager
  </jndi-name>
  <jndi-name>
    com.beasys.commerce.ebusiness.catalog.examples.file.FileProductItemManager
  </jndi-name>

  <ejb-name>
    com.beasys.commerce.ebusiness.catalog.service.item.JdbcProductItemManager
  </ejb-name>
  <ejb-name>
```

6 Using the API to Extend the Product Catalog

```
com.beasys.commerce.ebusiness.catalog.service.examples.file.FileProductItemManager
</ejb-name>
<weblogic-enterprise-bean>
  <ejb-name>
    com.beasys.commerce.ebusiness.catalog.service.query.JdbcCatalogQueryManager
  </ejb-name>
  <ejb-name>
    com.beasys.commerce.ebusiness.catalog.examples.file.FileCatalogQueryManager
  </ejb-name>
  <jndi-name>
    com.beasys.commerce.ebusiness.catalog.service.query.JdbcCatalogQueryManager
  </jndi-name>
  <jndi-name>
    com.beasys.commerce.ebusiness.catalog.examples.file.FileCatalogQueryManager
  </jndi-name>
  <ejb-name>
    com.beasys.commerce.ebusiness.catalog.service.query.JdbcCatalogQueryManager
  </ejb-name>
  <ejb-name>
    com.beasys.commerce.ebusiness.catalog.examples.file.FileCatalogQueryManager
  </ejb-name>
```

7 Product Catalog Internationalization Support

Internationalization is an important part of the development process for companies doing business in both domestic and overseas markets. However, providing a multilingual product catalog via the Web adds complexity to the internationalization effort and may require additional modifications to the catalog architecture. Although the strings displayed to users can easily be translated by editing various property files (such as `wlcs-catalog.properties`), maintaining dynamic multilingual catalog content and allowing the user to specify a language preference can be more difficult.

Note: For more information about the `wlcs-catalog.properties` file, see “Using the `wlcs-catalog.properties` File” on page 4-55.

To meet these objectives, the BEA WebLogic Commerce Server includes several features that you can utilize to build a multilingual product catalog. These features will help you internationalize your system and render a localized version of each category or item on a Web page, including text descriptions, images, item cost, type of currency, and so on. This topic describes these features in detail.

Note: It is important that you understand the architecture of the product catalog before reading about support for internationalization. For more information about the product catalog architecture, see “Catalog Architecture and Services” on page 6-3.

This topic includes the following sections:

- Support for Multiple Languages

- Language and Country Codes
- About the CatalogRequest Object
- Persisting Language Information to the Catalog Database
- Limiting Search Results by Language
- Using the Catalog Architecture to Maintain Internationalized Product Catalogs
 - Method 1: Filtering Product Catalog Content
 - Method 2: Parsing Language-Specific Data
 - Method 3: Multiple Product Catalog Instances
 - Method 4: Language-Based Service Routing

Support for Multiple Languages

BEA recognizes that an e-commerce Web site may need to support international users. To meet this requirement, a company may want to display their product catalog items in several languages. For example, a Canadian company may display product items to their potential customers in both English and French. Some European companies may support ten or more languages to localize their pages and ensure the success of their e-business. This section explains how you can use the built-in language attribute to support multiple languages.

Language and Country Codes

The best practice language descriptions defined by RFC 1766 include a two letter language code, optionally followed by a two letter country code. (These language codes are obtained from the ISO 639 and ISO 3166 standards, respectively.) For example, a language description of “en-uk” indicates that the language is English and that the country is the United Kingdom.

About the CatalogRequest Object

The first parameter to every product catalog API is a `CatalogRequest` object, which contains a language attribute. The language attribute is a `String` that should conform to the format described in “Language and Country Codes” on page 7-2.

The language attribute of the `CatalogRequest` object informs the product catalog system about the language a user would like to receive. For example, if the catalog contains both an “en-US” and an “en-GB” version of SKU 1001 and the incoming `CatalogRequest` object has specified the language as “en-GB”, then code can be written to return the “en-GB” version of SKU 1001 according to the user's language preference.

An example of setting the language on a `CatalogRequest` object is shown in Listing 7-1.

Listing 7-1 Setting the Language on a CatalogRequest Object

```
CatalogManager cm = null;
// lookup the CatalogManager using JNDI (omitted)

CatalogRequest cr = cm.createCatalogRequest();

cr.setLanguage("en-GB");
```

Once you have set the language attribute, you can use the `CatalogRequest` object in other API calls, as shown in Listing 7-2.

Listing 7-2 Using the CatalogRequest Object

```
final int viewSize = 50;
ViewIterator subIterator = cm.getCategoryManager(cr).
    getSubCategories(cr, CategoryKey.ROOT, viewSize);
```

Persisting Language Information to the Catalog Database

Each product item, category, and product image also has a language attribute. This individualized information is stored in the database and later used to retrieve information based on a user's specified language preference.

Product Items and Categories

Items and categories within the product catalog each have a language attribute as defined in the Dublin Core Standard. The format of the language attribute for product items and categories should conform to that described in "Language and Country Codes" on page 7-2.

Note: For more information about WebLogic Commerce Server and the Dublin Core Standard, see "The Catalog Schema Is Based on Dublin Core Standard" on page 2-4.

An example of creating a new item with a language specification and persisting it to the database is shown in Listing 7-3.

Listing 7-3 Creating and Persisting an Item with a Language Specification

```
MutableProductItem mutItem =
CatalogFactory.createMutableProductItem(new ProductItemKey(
"SKU001"));

// set the language attributes on the item

mutItem.setLanguage("en-GB");
ImageInfo smallImage = mutItem.getImage(SMALL_IMAGE_INDEX);
smallImage.setLanguage("en-GB");
mutItem.setImage(SMALL_IMAGE_INDEX , smallImage);

// create the new item

CatalogManager cm = null;

// lookup the CatalogManager using JNDI (omitted)

// create an administration catalog request object
// (required for write access to the catalog)
```



```
CatalogRequest cr = cm.createAdminCatalogRequest();  
// create the new item using the ProductItemManager  
cm.getProductItemManager().createItem(cr, mutItem);
```

Image Support

As shown in Listing 7-3, each image associated with an item or category also has a language attribute for storing multilingual images. This attribute allows you to provide a description of the product item or category in one language, while displaying an image that corresponds to the version of the item that is available for speakers of that language (that is, the item version for a particular country or region). In this case, the appropriate image would be chosen based on the language specified in the `CatalogRequest` object to ensure that customers are previewing the version of the product for their country.

Note: For more information about the `CatalogRequest` object, see “About the `CatalogRequest` Object” on page 7-3.

Limiting Search Results by Language

The default `CatalogQueryManager Tier 1 Service Provider` provides you with the ability to limit search results by language. To activate this feature, set the language attribute on the `CatalogRequest` object to non-null, using the format described in “Language and Country Codes” on page 7-2. An example is shown in Listing 7-1.

If `CatalogRequest.getLanguage()` is non-null, then search results will be limited to the language specified (exact, case-sensitive matches only). If `CatalogRequest.getLanguage()` is null, then search results are not automatically limited to a language.

The `CatalogRequest.getLanguage()` method originally contains a value set by the `catalog.request.language.default` property of the `wlcs-catalog.properties` file. In the version of this file that ships with the WebLogic Commerce Server product, the `catalog.request.language.default` property is commented out, meaning that the default language is null. This portion of the default `wlcs-catalog.properties` file is shown in Listing 7-4.

Listing 7-4 Default Language for Catalog Requests as Set in the wics-catalog.properties File

```
#####  
#####  
# DEFAULT LANGUAGE FOR CATALOG REQUESTS  
# If this entry is not specified the default language will be set  
# to null.  
#####  
#####  
# catalog.request.language.default=en_US
```

Note: In the current release, the language attribute of the `CatalogRequest` object is only used when searching the catalog (if non-null). The other APIs do not use this attribute and will return catalog items irrespective of that specified in the language attribute.

Additionally, the language attribute can be used in expression-based queries when `CatalogRequest.getLanguage()` is null (for example, "description like *black* && (language like *en* || language == null)").

Using the Catalog Architecture to Maintain Internationalized Product Catalogs

Depending on your business and technical requirements, you can use the catalog architecture in several different ways to maintain internationalized product catalogs. The criteria for selection of an effective method include:

- the amount of JSP coding that will be required from your development team.
- the amount of EJB coding that will be required from your development team.
- whether you want product categories and items for each language stored in a single database table or in multiple tables.
- whether or not you want to enable multilingual searching.

This section describes four methods for maintaining internationalized product catalogs.

Method 1: Filtering Product Catalog Content

The first method for internationalization of product catalogs:

- requires modification to the JSP templates to filter categories and items based on a user's language preference.
- requires no EJB coding.
- persists all product categories and items for each language to a single database table.
- allows for multilingual searching.

If you want to filter product catalog content, each JSP template will require modification. All language versions of a category or product item will be returned from the database, but the JSP template will contain logic that essentially filters the information shown to the user based on their specified language preference. If the language attribute of the category or item does not match that of the user's specified

language, the item or category is not displayed. Else, if the language attribute does match that of the user's selected language, you will want to display that item or category.

Note: If you use this method, be sure you also add code in your JSP templates to display the correct number of returned results (that is, the number of results for the specified language only) to the user.

The content filtering method is simple because it does not require EJB coding or redeployment, and still allows you to utilize the Administration Tools and DBLoader utility as in the out-of-the-box product. Also, if multilingual items are not mapped into non-country specific categories, it is not necessary to filter items in addition to the categories. For example, if the current category has a language attribute value of "en-GB", then you can assume that all the items within this category will also have a language of "en-GB".

Note: For more information about the DBLoader utility available in the WebLogic Commerce Server, see Chapter 3, "Using the Product Catalog Database Loader."

Method 2: Parsing Language-Specific Data

Although the content filtering method previously described is simple, it is not the most elegant method for maintaining internationalized product catalogs. As a more complex but elegant solution, the second method for internationalization of product catalogs:

- requires modification to the JSP template to parse language specific data from items.
- requires no EJB coding.
- persists all product categories and items for each language to a single database table.
- allows for multilingual searching.

Two Languages

Recall that in the product catalog JSP templates, there are two JSPs (containing two separate images) that are responsible for displaying item summary and item detail information. Out of the box, these pages contain very similar content. However, they can be customized to render a product item in two different languages. For example, instead of clicking the View Details link on the item summary page (which loads the item details page with nearly the same information), the user could click a link that allows them to view the item in French instead of English. The parsing logic for extracting the language-specific data from the product item attributes would be invoked from the relevant display JSP.

Note: For more information about the product catalog JSP templates, see Chapter 5, “The Product Catalog JSP Templates and Tag Library.”

Multiple Languages

If it is necessary to display information about product items in more than two languages, a master identifier can be used instead, as a reference (proxy) to several language-specific identifiers. For example, a master identifier of 100 can be thought of as a category and used to locate a number of language-defined subcategories. A language-specific identifier of 100A can contain English versions of a product item, an identifier of 100B can contain German versions, and so on. All but the master identifier should be marked as invisible and orphaned to ensure that the user cannot reach them directly by searching or by browsing the product catalog.

Note: To facilitate multilingual searches, language-specific identifiers should just be marked orphaned and not marked invisible, as the user should be able to reach these items through searches.

To display the categories in several languages, the language-specific identifiers (or subcategories) of the master identifier should be encoded within a field of the master identifier. The display JSP used to render the master identifier should be modified to perform a JSP include of the localized version of the master identifier. To prevent the user from browsing the localized categories, their top-level parent category can be set to something other than root. Alternatively, if the localized categories are required to be accessible from the Administration Tools, the template JSPs used to display the siblings and child categories can be modified to exclude the top-level parent of the localized categories.

Then, within the display JSPs for the product item, the master identifier can be parsed to locate the correct language-specific item (identifier) and its data substituted.

Method 3: Multiple Product Catalog Instances

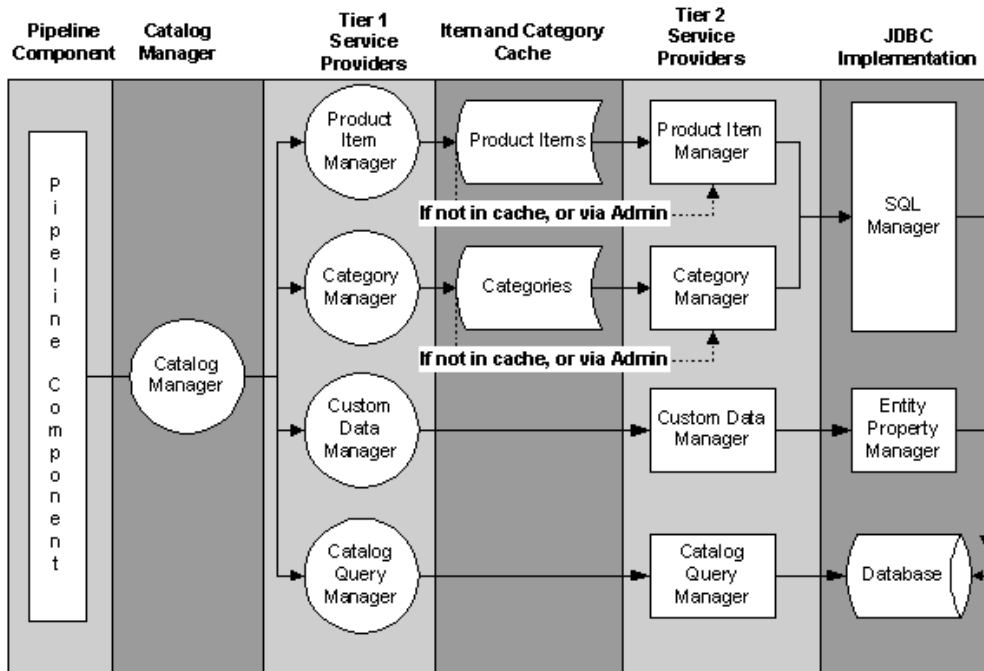
The third method for internationalization of product catalogs:

- requires separate Administration support for additional product catalogs.
- requires no EJB coding.
- persists the product categories and items for each language to a separate database table.
- does not allow for multilingual searching.

Before learning more about the multiple instances method, it is important that you understand what a schema file is and how it can be used for product catalog internationalization.

Figure 7-1 shows the same product catalog architecture that is described in “Catalog Architecture and Services” on page 6-3.

Figure 7-1 Product Catalog Architecture



A schema file is the primary means for deploying multiple product catalogs and is therefore one way to support internationalization. An instance of a Tier 2 Service Provider can be bound to a schema file, which exists in the deployment descriptor for the Tier 2 session bean. The schema file contains the names of the database tables, columns, and SQL statements that the Tier 2 Service Provider will use for persistence.

By deploying multiple instances of the `CatalogManager` session bean (and the Tier 1 and Tier 2 session beans) multiple catalog instances can be deployed. Binding the new Tier 2 session beans to different schema files provides for the maintenance of independent, language-specific catalogs. Each `CatalogManager` should be bound to a JNDI name appropriate for the language of the catalog. For example:

- `catalog.en_US`
- `catalog.fr_FR`

When a catalog function is required, a `CatalogManager` should be looked up using the appropriate JNDI name.

Note: The WebLogic Commerce Server Administration Tools work against a `CatalogManager` instance bound to the JNDI name `com.beasys.commerce.ebusiness.catalog.CatalogManager`, which is not customizable. Therefore, organizations using the multiple instances method will need to administer additional catalogs via SQL or develop their own GUI tools.

Although the two product catalogs are totally independent, the category and item identifiers between catalogs should be the same to ensure that data can be easily updated using the `DBLoader` utility.

Note: For more information about the `DBLoader` utility available in the WebLogic Commerce Server, see Chapter 3, “Using the Product Catalog Database Loader.”

Method 4: Language-Based Service Routing

The fourth method for internationalization of product catalogs:

- requires separate Administration support for additional product catalogs.
- requires developers to create a new, derived `CatalogManager` session bean.
- persists the product categories and items for each language to a separate database table.
- does not allow for multilingual searching.

Instead of creating multiple instances, redeploying the Services, and binding the Services to different schema files as described in “Method 3: Multiple Product Catalog Instances” on page 7-10, you can simply create and redeploy a new `CatalogManager` session bean to return an instance of a Tier 1 Service based on the language specified in the `CatalogRequest` object.

The new `CatalogManager` session bean should extend the existing `CatalogManager` to examine the language attribute of the incoming `CatalogRequest` object. By deriving a new class from the `CatalogManagerImpl` object and overriding the Service accessor method, you can easily implement language-based service routing, as shown in the following example:

```
public ProductItemManager getProductItemManager (CatalogRequest
request)
```


Using the Catalog Architecture to Maintain Internationalized Product Catalogs

The overridden `getProductItemManager()` method will return an instance of a `ProductItemManager` session bean, which delegates to a `JdbcProductItemManager` session bean. The `JdbcProductItemManager`, in turn, is bound to a schema file for the language required.

Note: Using the language-based service routing method, the WebLogic Commerce Server's Administration Tools will work as expected, because there is still only one `CatalogManager` instance.

7 *Product Catalog Internationalization Support*

Index

A

- adding categories 4-12
- adding items 4-18
- administration tasks
 - adding categories 4-12
 - adding items 4-18
 - assigning items to categories 4-23
 - catalog cache settings 4-48
 - changing administrator password 4-8
 - controlling visibility of items 4-22
 - deleting categories 4-43
 - deleting items 4-37
 - editing attributes
 - categories 4-27
 - items 4-27
 - editing availability of items 4-34
 - introduction 4-1
 - moving items 4-46
 - starting the Admin tool 4-4
 - starting the server 4-2
 - wlcs-catalog.properties file 4-55
- assigning items to categories 4-23
- attribute-based search syntax 5-74
- attributes
 - custom
 - for items 4-46
 - language 7-3
 - and expression-based queries 7-6
 - CatalogRequest object 7-3, 7-12
 - image 7-5
 - product items and categories 7-4

- availability of items
 - editing 4-34

B

- BEA, contacting xiv
- browse.jsp template 5-25
- business logic 5-19, 5-42, 5-51, 5-58, 5-68

C

- cache
 - catalog 4-48
- cat.tld tag library 5-97
- catalog
 - adding categories 4-12
 - adding items 4-18
 - administration 4-1
 - API overview 6-2
 - architecture and services 6-3, 7-1
 - using for internationalization 7-7
 - assigning items to categories 4-23
 - cache 4-48
 - caching settings 4-48
 - database constraints 2-21
 - deleting categories 4-43
 - deleting items 4-37
 - development roles 1-8
 - Dublin Core standard 2-4
 - editing attributes
 - categories 4-27
 - items 4-27

- editing availability of items 4-34
- editing schema definition 4-59
- Entity-Relation diagram 2-1
- getProperty tag 5-98
- hierarchy 1-6
- input processors 5-80
- internationalization
 - images 7-5
 - products and categories 7-4
- introduction 1-1
- iterateThroughView tag 5-102
- iterateViewIterator tag 5-100
- JSP tag library 5-97
- link with order processing 1-9
- moving items 4-46
- multiple instances 7-11
- overview of schema 2-1
- persisting language information 7-4
- pipeline components 5-87
- SQL files 2-21
- using API to extend 6-1
- visibility of items 4-22
- wlcs-catalog.properties file 4-55, 7-1
- writing your own catalog service 6-17
- CatalogIP input processor 5-80
- CatalogManager interface 6-5, 7-11
 - extending 7-12
- CatalogPC pipeline component 5-87
- CatalogQueryManager interface 6-14
 - and internationalization 7-5
- CatalogRequest object
 - language attribute 7-3, 7-12
- categories
 - adding 4-12
 - assigning items to 4-23
 - deleting 4-43
 - displayed to users 4-36
 - editing attributes 4-27
- category.jsp
 - about 5-35
- CategoryManager interface 6-9

- changing the administrator password 4-8
- comparison operators in query 5-76
- constraints
 - defined for catalog tables 2-21
- contacting BEA xiv
- controlling number of search results 5-78
- controlling visibility of items 4-22
- custom attributes
 - for items 4-46
- CustomDataManager interface 6-13

D

- Data Junction 3-13
- data loaders
 - introduction 3-11
 - third party 3-13
- DataStage by Informix 3-13
- DBLoader
 - and multiple catalog instances 7-12
 - database considerations 3-10
 - dbloader.properties files 3-4
 - input file 3-2
 - introduction 3-1
 - log files 3-9
 - running 3-7
 - validations 3-9
- deleting categories 4-43
- deleting items 4-37
- development roles
 - for catalog 1-8
- documentation, where to find it xiii
- Dublin Core standard
 - catalog 2-4

E

- editing attributes
 - categories 4-27
 - items 4-27
- Entity-Relation diagram

- for catalog tables 2-1
- ETI-EXTRACT by ETI 3-13
- event(s)
 - browse.jsp 5-42
 - itemdetails.jsp 5-51
 - main.jsp 5-19
 - search.jsp 5-58
 - searchresults.jsp 5-68
- expression-based search queries 7-6
- ExpressionSearchIP input processor 5-84
- extending catalog
 - using API 6-1

G

- GetAncestorsPC pipeline component 5-91
- GetCategoryIP input processor 5-82
- GetCategoryPC pipeline component 5-88
- GetParentPC pipeline component 5-90
- GetProductItemIP input processor 5-81
- GetProductItemPC pipeline component 5-89
- GetProductItemsPC pipeline component 5-92
- getProperty tag 5-98
- GetSubcategoriesPC pipeline component 5-93

H

- hierarchy
 - catalog 1-6

I

- improving performance
 - catalog cache 4-48
- input processors
 - for catalog JSPs 5-80
- Internationalization
 - and catalog architecture 7-7
 - language and country codes 7-2

- methods
 - filtering content 7-7
 - multiple catalog instances 7-10
 - parsing language-specific data 7-8
 - service routing 7-12
- non-ASCII characters 5-75
- of images 7-5
- of product items and categories 7-4
- product catalog support 7-1
- itemdetails.jsp
 - about 5-51
- items
 - adding 4-18
 - defining custom attributes 4-46
 - deleting 4-37
 - displayed to users 4-36
 - editing attributes 4-27
 - editing availability 4-34
 - moving 4-46
 - visibility of 4-22
- itemsummary.jsp
 - about 5-39
- iterateThroughView tag 5-102
- iterateViewIterator tag 5-100

J

- JSP tag library
 - cat.tld 5-97
- JSP tags
 - getPipelineProperty 5-20, 5-22, 5-44
- JSP templates 5-1
 - overview 5-3

K

- KeywordSearchIP input processor 5-83

L

- language

- attribute
 - and expression-based queries 7-6
 - CatalogRequest object 7-3, 7-12
 - product items and categories 7-4
- limiting search results by 7-5

M

- main.jsp template 5-14
- methods
 - internationalization
 - filtering content 7-7
 - multiple catalog instances 7-10
 - parsing language-specific data 7-8
 - service routing 7-12
- MoveAttributeIP input processor 5-85
- MoveAttributePC pipeline component 5-94
- moving items 4-46
- multiple catalog instances 7-11

O

- object, CatalogRequest
 - language attribute 7-3, 7-12
- optimizing catalog cache 4-48
- overview of JSP templates 5-3

P

- passwords
 - administrator account 4-4
 - changing administrator's 4-8
- performance
 - improving with catalog cache 4-48
- pipeline components
 - for catalog 5-87
- PowerConnect by Informatica 3-13
- printing product documentation xiii
- product catalog
 - architecture 7-1
 - using for internationalization 7-7

- editing schema definition 4-59
- internationalization
 - images 7-5
 - product items and categories 7-4
- introduction 1-1
- multiple instances 7-11
- overview of features 1-3
- persisting language information 7-4
- ProductItemManager interface 6-8

Q

- query
 - comparison operators 5-76
 - expression-based and language attribute 7-6
- query-based search syntax 5-74

R

- related information xiv
- RemoveAttributeIP input processor 5-86
- RemoveAttributePC pipeline component 5-95

S

- schema
 - catalog tables 2-1
 - editing product catalog 4-59
 - file
 - for internationalization 7-11
- search results
 - controlling number of 5-78
 - limiting by language 7-5
- search syntax 5-74
- search.jsp template 5-54
- SearchPC pipeline component 5-96
- searchresults.jsp template 5-64
- server
 - starting 4-2

SQL files
 for catalog 2-21
Start page
 for JSPs 5-4
starting the server 4-2
support, technical xiv

T

templates
 JSP overview 5-3

W

WLCS_CAT_ENTITY_ID database table 2-17
WLCS_CAT_PROP_* database tables 2-17
WLCS_CAT_PROP_BOOLEAN database table 2-19
WLCS_CAT_PROP_DATETIME database table 2-20
WLCS_CAT_PROP_FLOAT database table 2-19
WLCS_CAT_PROP_ID database table 2-18
WLCS_CAT_PROP_INTEGER database table 2-19
WLCS_CAT_PROP_TEXT database table 2-20
WLCS_CAT_PROP_USER_DEFINED database table 2-21
WLCS_CATEGORY database table 2-5
WLCS_PRODUCT database table 2-9
WLCS_PRODUCT_CATEGORY database table 2-15
WLCS_PRODUCT_KEYWORD database table 2-16
wlcs-catalog.properties file 4-55, 7-1