



BEA WebLogic Portal™

Guide to Building a Product Catalog

Version 4.0
Document Date: October 2001

Copyright

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

Restricted Rights Legend

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

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

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

Trademarks or Service Marks

BEA, WebLogic, Tuxedo, and Jolt are registered trademarks of BEA Systems, Inc. How Business Becomes E-Business, Operating System for the Internet, Liquid Data, BEA WebLogic E-Business Platform, BEA Builder, BEA Manager, BEA eLink, BEA WebLogic Commerce Server, BEA WebLogic Personalization Server, BEA WebLogic Process Integrator, BEA WebLogic Collaborate, BEA WebLogic Enterprise, BEA WebLogic Server, BEA WebLogic Integration, E-Business Control Center, BEA Campaign Manager for WebLogic, and Portal FrameWork are trademarks of BEA Systems, Inc.

All other trademarks are the property of their respective companies.

Guide to Building a Product Catalog

Document Edition	Date	Software Version
4.0	October 2001	BEA WebLogic Portal 4.0

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 the Product Catalog and Other Commerce Features Are Linked	1-9
Next Step	1-10

2. The Product Catalog Database Schema

The Entity-Relation Diagram	2-2
The Catalog Schema Is Based on Dublin Core Standard	2-4
List of Tables Comprising the Product Catalog	2-5
The Product Catalog Data Dictionary	2-6
The CATALOG_ENTITY Database Table	2-6
The CATALOG_PROPERTY_KEY Database Table	2-6
The CATALOG_PROPERTY_VALUE Database Table	2-7
The WLCS_CATEGORY Database Table	2-8
The WLCS_PRODUCT Database Table	2-12
The WLCS_PRODUCT_CATEGORY Database Table	2-16
The WLCS_PRODUCT_KEYWORD Database Table	2-17
The SQL Scripts Used to Create the Database	2-18

Scripts	2-18
Defined Constraints	2-22

3. Using the 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	3-10
Important Database Considerations	3-10
Using Database-Specific Data Loaders	3-12
Using Third-Party Data Loaders	3-13

4. Catalog Administration Tasks

Starting the Server and the Administration Tools	4-2
Using the Catalog Management Tools	4-3
Changing the Administrator Password	4-4
Loading Data into the Product Catalog	4-7
Adding Categories to the Catalog	4-8
Adding Items to the Catalog	4-14
Controlling the Visibility of Items in the Product Catalog	4-18
Assigning Items to Categories	4-19
What if I Have a Large Amount of Data?	4-19
Using the Administration Tools to Assign Items to Categories	4-20
Editing the Attributes for Categories and Items	4-23
Editing Category Attributes	4-23
Editing Product Item Attributes	4-26
Editing the Availability of an Item	4-29
Determining How Categories and Items are Displayed to Web Site Visitors	4-30
Deleting Items or Removing Items from One or More Categories	4-31
Caching Considerations	4-32
Deleting an Item from the Catalog	4-32
Removing an Item from One or More Categories	4-35
Removing Categories	4-37

Moving Items from One Category to Another Category.....	4-39
Defining Custom Attributes for Items	4-39
Improving Catalog Performance by Optimizing the Catalog Cache	4-41
Considering Hardware Costs Versus the Cost of Dissatisfied Web Site Users	
4-42	
What's in Each Cache Initially?	4-43
Using the wlcs-catalog.properties File	4-43
Location.....	4-44
Some Property Values You Might Modify	4-44
Editing the Catalog Schema Definition.....	4-46

5. The Product Catalog JSP Templates

Introduction	5-3
Accessing the Product Catalog JavaServer Page (JSP) Templates	5-4
On Which JavaServer Page Will My Customers Start?	5-4
Sample Path Through the Product Catalog JSP Templates.....	5-4
JavaServer Pages (JSPs)	5-5
main.jsp Template	5-7
Sample Browser View	5-7
Location in the Directory Structure	5-9
Tag Library Imports	5-9
Java Package Imports.....	5-10
Location in the Default Webflow.....	5-10
Included JSP Templates	5-11
Events.....	5-11
Dynamic Data Display	5-13
Form Field Specification.....	5-18
browse.jsp Template.....	5-19
Sample Browser View	5-21
Location in the Directory Structure	5-24
Tag Library Imports	5-24
Java Package Imports.....	5-24
Location in the Default Webflow.....	5-25
Included JSP Templates	5-25
Events.....	5-39

Dynamic Data Display	5-40
Form Field Specification	5-43
details.jsp Template	5-44
Sample Browser View	5-44
Location in the Directory Structure	5-46
Tag Library Imports	5-46
Java Package Imports	5-47
Location in the Default Webflow	5-47
Included JSP Templates	5-48
Events	5-52
Dynamic Data Display	5-53
Form Field Specification	5-55
search.jsp	5-56
Sample Browser View	5-56
Location in the Directory Structure	5-59
Tag Library Imports	5-59
Java Package Imports	5-59
Location in the Default Webflow	5-60
Included JSP Templates	5-60
Events	5-61
Dynamic Data Display	5-62
Form Field Specification	5-65
searchresults.jsp	5-66
Sample Browser View	5-66
Location in the Directory Structure	5-68
Tag Library Imports	5-68
Java Package Imports	5-69
Location in the Default Webflow	5-69
Included JSP Templates	5-69
Events	5-70
Dynamic Data Display	5-71
Form Field Specification	5-75
Query-Based Search Syntax	5-76
Using Comparison Operators to Construct Queries	5-78
Searchable Catalog Attributes	5-79

Controlling the Number of Search Results	5-80
Input Processors.....	5-82
CatalogIP	5-82
ExpressionSearchIP.....	5-83
GetCategoryIP	5-84
GetProductItemIP	5-85
KeywordSearchIP.....	5-86
MoveAttributeIP.....	5-87
RemoveAttributeIP.....	5-88
Pipeline Components.....	5-89
CatalogPC.....	5-89
GetAncestorsPC	5-90
GetCategoryPC.....	5-91
GetParentPC	5-92
GetProductItemPC	5-93
GetProductItemsPC.....	5-94
GetSubcategoriesPC.....	5-95
MoveAttributePC	5-96
PriceShoppingCartPC.....	5-97
RemoveAttributePC	5-98
SearchPC	5-99

6. Product Catalog JSP Tag Library Reference

Introduction	6-2
The Catalog JSP Tag Library: cat.tld	6-2
<catalog:getProperty>	6-4
Example 1	6-5
Example 2	6-6
<catalog:iterateViewIterator>	6-6
Example 1	6-7
Example 2	6-8
<catalog:iterateThroughView>	6-8
Example 1	6-9
Example 2	6-9
The E-Business JSP Tag Library: eb.tld.....	6-10

<eb:smnav>	6-10
Example.....	6-11

7. Using the API to Extend the Product Catalog

Overview of the Product Catalog API	7-2
Catalog Architecture and Services.....	7-3
Catalog Architecture.....	7-4
Catalog Manager	7-5
Product Item Manager	7-8
Category Manager	7-9
Custom Data Manager.....	7-13
Catalog Query Manager	7-14
Integrating Services with the Catalog Cache.....	7-15
Writing Your Own Catalog Service	7-17
Create New Services.....	7-18
Sample Source Code	7-19
Changes to ejb-jar.xml.....	7-33
Changes to weblogic-ejb-jar.xml.....	7-36

8. Product Catalog Internationalization Support

Support for Multiple Languages.....	8-2
Language and Country Codes	8-2
About the CatalogRequest Object	8-3
Persisting Language Information to the Catalog Database	8-4
Product Items and Categories.....	8-4
Image Support	8-5
Limiting Search Results by Language.....	8-5
Important Note About Currencies	8-6
Using the Catalog Architecture to Maintain Internationalized Product Catalogs ..	8-7
Method 1: Filtering Product Catalog Content	8-7
Method 2: Parsing Language-Specific Data.....	8-8
Two Languages	8-9
Multiple Languages.....	8-9
Method 3: Multiple Product Catalog Instances	8-10

Method 4: Language-Based Service Routing..... 8-12

Index



About This Document

This document explains how to use the Commerce services included in the WebLogic Portal™ product suite to build and customize a Web-based product catalog.

The Product Catalog that ships with WebLogic Portal contains 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 features provide the computing infrastructure and scalability that enables your site to support many concurrent visitors. 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 Product Catalog and the job-based roles of the people who will build and customize the catalog.
- Chapter 2, “The Product Catalog Database Schema,” explains the structure of the included 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 Database Loader,” describes a bulk loader program that you can use to insert, update, or delete large numbers of records into a database. This program can be especially helpful for loading category and item records for your product catalog.
- Chapter 4, “Catalog Administration Tasks,” describes the administration screens used to find, add, edit, or remove categories and items in the Product Catalog.

-
- Chapter 6, “Product Catalog JSP Tag Library Reference,” describes the JSP templates included with the Product Catalog. 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 7, “Using the API to Extend the Product Catalog,” describes the various options available for extending, customizing, or writing third-party integrations for the Product Catalog.
 - Chapter 8, “Product Catalog Internationalization Support,” describes how your product catalog can be localized for visitors from 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 HTML/JSP developers, Java/EJB developers, and system 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.

- **HTML/JSP 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.
- **Java/EJB developers** handle any customization of the JavaServer Pages (JSPs) and might use the included JSP tags (or custom tags) to extend the functionality provided on the pages. Java/EJB developers might also modify or add scriptlets on the pages.
- **System administrators** maintain the data in the catalog, either adding, editing, or removing categories and items from the catalog. **Business analysts** bring their market research to the design table and help the administrators understand the required product items, categories, and pricing.

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 Portal 4.0 documentation (which includes the Commerce services) starts at <http://e-docs.bea.com/wlcs/docs40/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 on the WebLogic Portal 4.0 documentation Home page on the e-docs Web site. A PDF version of this document is also available in the documentation kit on the product CD. You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access the PDFs, open the WebLogic Portal 4.0 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 Portal 4.0 documentation (which includes the Commerce services) 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 Portal documentation.

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

If you have any questions about this version of BEA WebLogic Portal, or if you have problems installing and running BEA WebLogic Portal, 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

Today, Internet-savvy customers have high expectations about their online shopping experience. They love the convenience of finding just about any product item for sale on the Web. Customers 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 customers 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 customers 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:

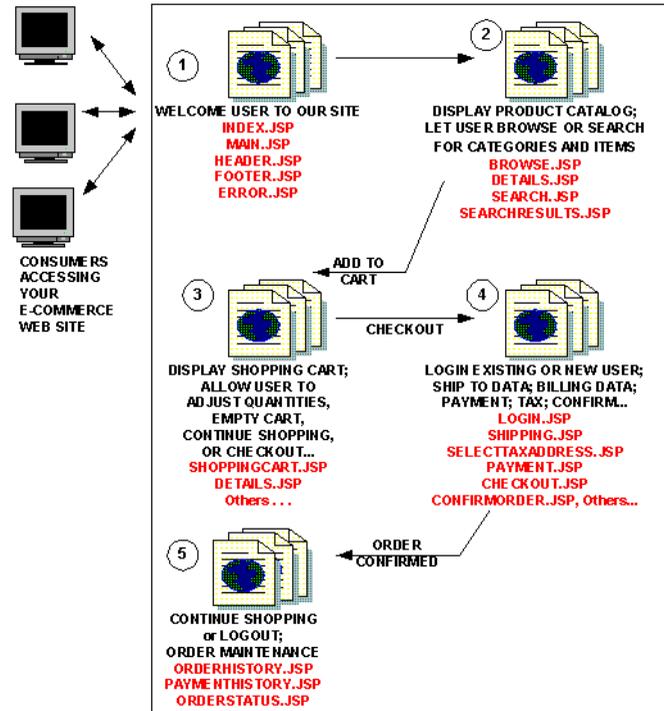
- Established companies with a solid market presence that pre-dates the Internet revolution. These companies are extending their marketing and sales reach by enabling existing and new customers to buy items on their e-commerce Web sites. They are moving from “bricks and mortar” to “bricks and clicks.”
- 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 JavaBeans (EJBs) 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

1 Introduction to the Product Catalog

develop the standardized database resources, EJB programming expertise, and JSPs that together will provide the commonly expected Web site functions shown in Figure 1-1.

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 e-commerce site by eager customers.

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 Commerce services that are included in the BEA WebLogic Portal product suite!

What Does the Product Catalog Provide?

As part of the Commerce services, the 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 Database Schema,” of this document.

- **A bulk loader program called DBLoader** that takes a data input file and populates a database. For example, you can add large volumes of product item and category records to your product catalog 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 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.

Figure 1-2 Sample Administration Screen

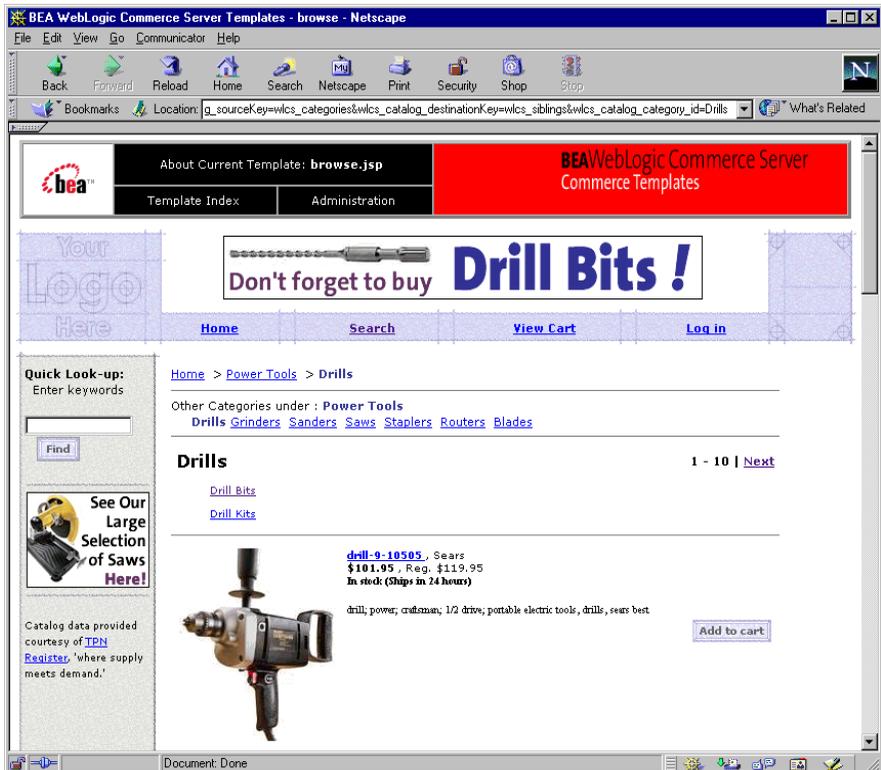


Administrators should read Chapter 4, “Catalog Administration Tasks,” in this document.

- **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 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 the Product Catalog.

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,” in this document. For related details about the JSPs used with order processing (shopping cart, shipping, tax, checkout, and so on), see *Guide to Managing Purchases and Processing Orders*. For related details about the JSPs used to register customers, see *Guide to Registering Customers and Managing Customer Services*.

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

Chapter 7, “Using the API to Extend the Product Catalog,” describes the various options available for extending, customizing, or writing third-party integrations

for the Product Catalog. The API is also described in the 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 product catalog 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.

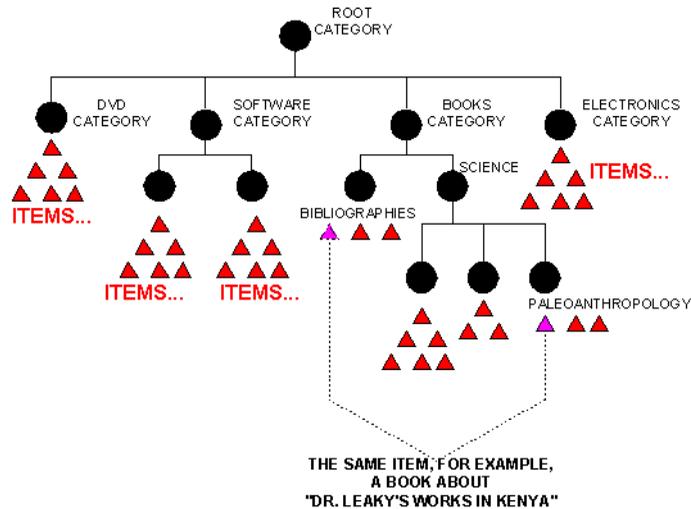
Chapter 8, “Product Catalog Internationalization Support,” provides more detail about how you can internationalize your product catalog.

Catalog Hierarchy

Categories in the Product Catalog exist in a hierarchy, as illustrated in Figure 1-4.

Note: The example shown in the figure deviates from the sample data that is seen when you run the Product Catalog JSP 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? 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.

- **HTML/JSP 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.
- **Java/EJB developers** handle any customization of the JavaServer Pages (JSPs) and might use the included JSP tags (or custom tags) to extend the functionality provided on the pages. Java/EJB developers might also modify or add scriptlets on the pages.
- **System administrators** maintain the data in the catalog, either adding, editing, or removing categories and items from the catalog. **Business analysts** bring their market research to the design table and help the administrators 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.

How the Product Catalog and Other Commerce Features Are Linked

The Product Catalog is only a portion of the Commerce services included in this release. Related features are implemented by the order processing and customer registration packages. When a customer on your Web site decides to click the Add to Cart button (or equivalent) on one of the catalog pages, by default the customer is directed to the shopping cart page of the order processing package.

Once the customer clicks Add to Cart, a series of JavaServer Pages (JSPs) under the control of background processors are employed to guide the customer through the process of entering the information required to complete the order.

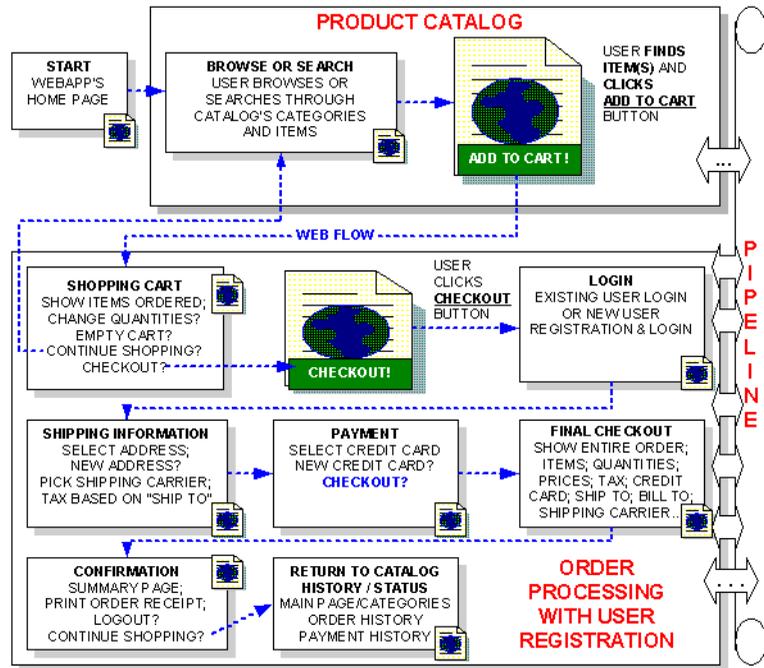
You can modify the actual sequence of pages, or **Webflow**, by editing a centralized Webflow configuration file. A default Webflow file is included with the Commerce services. The advantage here is that the flow of the Web site (that is, what page to go to next) is not hardwired into each page, but 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 elements within a configuration file. Also, you do not have to restart the WebLogic Server instance in which your applications are running, allowing for dynamic site management.

Underlying all this processing for both catalog management and order management is another important feature called the Commerce services Pipeline. The **Pipeline** is a mechanism for binding together a sequence of services into a single named service. While the JSPs 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 contains elements that describe the execution of a series of business methods.

It is therefore easy to change a business process (such as checking an order status) by adding or removing elements in the Pipeline configuration file. Such modifications do not require any programming.

Figure 1-5 shows the link between the Product Catalog and order processing. 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 customer has selected for purchase to the order fulfillment package, which will handle the order.

Figure 1-5 Link Between Catalog and Order Fulfillment



For details about configuring a site's Webflow (and Pipelines), see *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline*. For details about order processing, see *Guide to Managing Purchases and Processing Orders*. For details about customer registration, see *Guide to Registering Customers and Managing Customer Services*.

Next Step

We suggest you read Chapter 2, "The Product Catalog Database 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, and to adding new data to the catalog.

2 The Product Catalog Database Schema

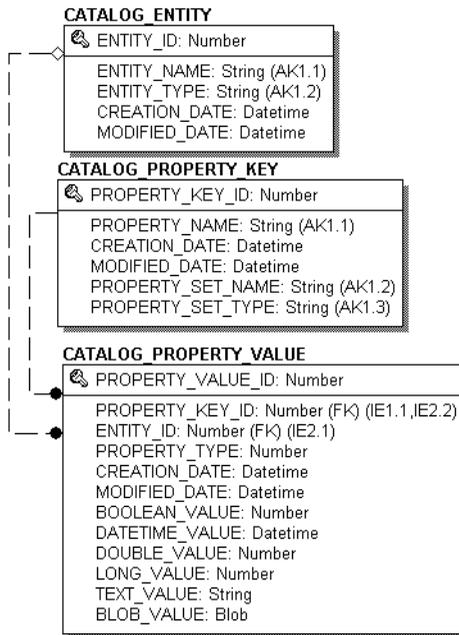
This topic documents the database schema for the Commerce services Product Catalog. This topic includes the following sections:

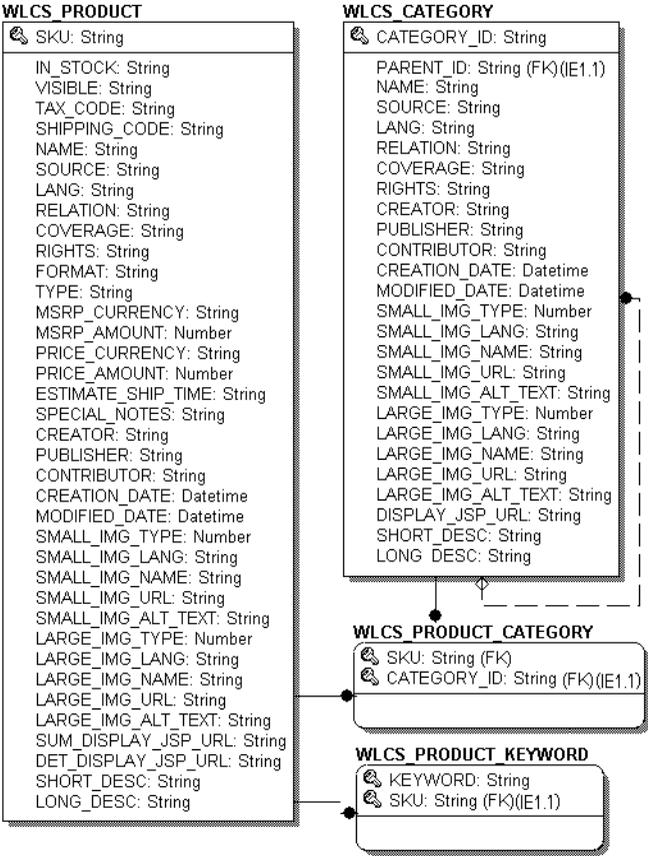
- The Entity-Relation Diagram
- The Catalog Schema Is Based on Dublin Core Standard
- List of Tables Comprising the Product Catalog
- The Product Catalog Data Dictionary
- The SQL Scripts Used to Create the Database
- Defined Constraints

The Entity-Relation Diagram

Figure 2-1 shows the logical Entity-Relation diagram for the Commerce services core Product Catalog tables in the Commerce database. See the subsequent sections in this topic for information about the data type syntax.

Figure 2-1 Entity-Relation Diagram for the Core Product Catalog Tables





The Catalog Schema Is Based on Dublin Core Standard

The metadata for items in Commerce services 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>.

List of Tables Comprising the Product Catalog

The Commerce services Product Catalog is comprised of the following tables. In this list, the tables are sorted by functionality:

- The CATALOG_ENTITY Database Table
- The CATALOG_PROPERTY_KEY Database Table
- The CATALOG_PROPERTY_VALUE Database Table
- The WLCS_CATEGORY Database Table
- The WLCS_PRODUCT Database Table
- The WLCS_PRODUCT_CATEGORY Database Table
- The WLCS_PRODUCT_KEYWORD Database Table

The Product Catalog Data Dictionary

In this section, the Commerce services schema tables are arranged alphabetically as a data dictionary.

The CATALOG_ENTITY Database Table

Table 2-1 describes the metadata for the Commerce services CATALOG_ENTITY table in the Commerce database. This table stores unique identification numbers for configurable entities.

The Primary Key is ENTITY_ID.

Table 2-1 CATALOG_ENTITY Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
ENTITY_ID	NUMBER (15)	NOT NULL	PK—a unique, system-generated number used as a record identifier.
ENTITY_NAME	VARCHAR (200)	NOT NULL	The name of the entity.
ENTITY_TYPE	VARCHAR (100)	NOT NULL	The type of entity (e.g., User, Group, etc.)
CREATION_DATE	DATE	NOT NULL	The time and date the record was created.
MODIFIED_DATE	DATE	NOT NULL	The time and date the record was last modified.

The CATALOG_PROPERTY_KEY Database Table

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

The Primary Key is PROPERTY_KEY_ID.

Table 2-2 CATALOG_PROPERTY_KEY Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
PROPERTY_KEY_ID	NUMBER(15)	NOT NULL	PK—a unique, system-generated number used as a record identifier.
PROPERTY_NAME	VARCHAR(100)	NOT NULL	The name of the property (formerly PROPERTY_NAME from the WLCS_PROP_ID table).
CREATION_DATE	DATE	NOT NULL	The time and date the record was created.
MODIFIED_DATE	DATE	NOT NULL	The time and date the record was last modified.
PROPERTY_SET_NAME	VARCHAR(100)	NULL	The name of the property set (formerly the SCOPE_NAME from WLCS_PROP_ID).
PROPERTY_SET_TYPE	VARCHAR(100)	NULL	The type of property set (for example, USER)

The CATALOG_PROPERTY_VALUE Database Table

Table 2-3 describes the metadata for the Commerce services CATALOG_PROPERTY_VALUE table in the Commerce database. This table stores Boolean, timestamp, float, integer, text, and user-defined (object) property values that are associated with configurable entities.

See “Defined Constraints” on page 2-23 for information about the constraint defined for this table.

The Primary Key is PROPERTY_VALUE_ID.

Table 2-3 CATALOG_PROPERTY_VALUE Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
PROPERTY_VALUE_ID	NUMBER(15)	NOT NULL	PK—a unique, system-generated number used as a record identifier.

2 The Product Catalog Database Schema

Table 2-3 CATALOG_PROPERTY_VALUE Table Metadata (Continued)

Column Name	Data Type	Null Value	Description and Recommendations
PROPERTY_KEY_ID	NUMBER (15)	NOT NULL	A system-generated value and foreign key to the PROPERTY_KEY column.
ENTITY_ID	NUMBER (15)	NOT NULL	A system-generated value and foreign key to the ENTITY column.
PROPERTY_TYPE	NUMBER (1)	NOT NULL	Valid entries are: 0=Boolean, 1=Integer, 2=Float, 3=Text, 4=Date and Time, 5=User-Defined (BLOB)
CREATION_DATE	DATE	NOT NULL	The time and date the record was created.
MODIFIED_DATE	DATE	NOT NULL	The time and date the record was last modified.
BOOLEAN_VALUE	NUMBER (1)	NULL	The value for each boolean property identifier.
DATETIME_VALUE	DATE	NULL	The value for each date and time property identifier.
DOUBLE_VALUE	NUMBER	NULL	The value associated with each float property identifier.
LONG_VALUE	NUMBER (20)	NULL	The value associated with the integer property.
TEXT_VALUE	VARCHAR (254)	NULL	The value associated with the text property.
BLOB_VALUE	BLOB	NULL	The value associated with the user-defined property.

The WLCS_CATEGORY Database Table

Table 2-4 describes the metadata for the Commerce services 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.

See “Defined Constraints” on page 2-23 for information about the constraint defined for this table.

The Primary Key is CATEGORY_ID.

Table 2-4 WLCS_CATEGORY Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
CATEGORY_ID	VARCHAR(20)	NOT NULL	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)	NULL	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)	NULL	The name of the category in the product catalog.
SOURCE	VARCHAR(30)	NULL	A reference to a category from which the present category is derived.

2 The Product Catalog Database Schema

Table 2-4 WLCS_CATEGORY Table Metadata (Continued)

Column Name	Data Type	Null Value	Description and Recommendations
LANG	VARCHAR (30)	NULL	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)	NULL	A reference to a related category.
COVERAGE	VARCHAR (30)	NULL	The extent or scope of the content of the category.
RIGHTS	VARCHAR (30)	NULL	Information about rights held in and over the category.
CREATOR	VARCHAR (50)	NULL	An entity primarily responsible for making the content of the category.
PUBLISHER	VARCHAR (50)	NULL	An entity responsible for making the category available.
CONTRIBUTOR	VARCHAR (50)	NULL	An entity responsible for making contributions to the content of the category.
CREATION_DATE	DATE	NULL	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.

Table 2-4 WLCS_CATEGORY Table Metadata (Continued)

Column Name	Data Type	Null Value	Description and Recommendations
MODIFIED_DATE	DATE	NULL	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 the Commerce services. 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	NUMBER (3)	NULL	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)	NULL	The language of the thumbnail image for the category. For related information, see the description of the LANG column.
SMALL_IMG_NAME	VARCHAR(50)	NULL	The name of the thumbnail image for the category.
SMALL_IMG_URL	VARCHAR(254)	NULL	The URL of the thumbnail image for the category.
SMALL_IMG_ALT_TEXT	VARCHAR(254)	NULL	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.

2 The Product Catalog Database Schema

Table 2-4 WLCS_CATEGORY Table Metadata (Continued)

Column Name	Data Type	Null Value	Description and Recommendations
LARGE_IMG_TYPE	NUMBER (3)	NULL	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)	NULL	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)	NULL	The name of the full-size image for the category.
LARGE_IMG_URL	VARCHAR (254)	NULL	The URL of the full-size image for the category.
LARGE_IMG_ALT_TEXT	VARCHAR (254)	NULL	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 (254)	NULL	The URL to the JSP used to display the category. For example: <code>/commerce/catalog/includes/ category.jsp</code>
SHORT_DESC	VARCHAR (50)	NULL	A short description of the content of the category.
LONG_DESC	VARCHAR (254)	NULL	A long description of the content of the category.

The WLCS_PRODUCT Database Table

Table 2-5 describes the metadata for the Commerce services 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.

The Primary Key is SKU.

Table 2-5 WLCS_PRODUCT Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
SKU	VARCHAR(40)	NOT NULL	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	VARCHAR(1)	NULL	A flag to indicate whether the product item is in stock. 0 equates to false, 1 equates to true.
VISIBLE	VARCHAR(1)	NULL	Indicates whether the item should be displayed to the user. Enter 1 if visible or 0 if not visible. If not specified in the database, the default is 1. See “Controlling the Visibility of Items in the Product Catalog” on page 4-18 for important information about this field.
TAX_CODE	VARCHAR(10)	NULL	The code used by the TAXWARE system to identify the specific tax category to which this item belongs.
SHIPPING_CODE	VARCHAR(10)	NULL	The code used by the shipping company for this item.
NAME	VARCHAR(100)	NULL	A name given to the product item.
SOURCE	VARCHAR(30)	NULL	A reference to another product item from which the present item is derived.

2 The Product Catalog Database Schema

Table 2-5 WLCS_PRODUCT Table Metadata (Continued)

Column Name	Data Type	Null Value	Description and Recommendations
LANG	VARCHAR (30)	NULL	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)	NULL	A reference to a related product item.
COVERAGE	VARCHAR (30)	NULL	The extent or scope of the content of the product item.
RIGHTS	VARCHAR (30)	NULL	Information about rights held in and over the item.
FORMAT	VARCHAR (30)	NULL	The physical or digital manifestation of the item.
TYPE	VARCHAR (30)	NULL	The nature or genre of the content of the item.
MSRP_CURRENCY	VARCHAR (30)	NULL	The currency type of the manufacturer's recommended price.
MSRP_AMOUNT	NUMBER (16 , 4)	NULL	The manufacturer's recommended price.
PRICE_CURRENCY	VARCHAR (30)	NULL	The currency type of our catalog price for this item.
PRICE_AMOUNT	NUMBER (16 , 4)	NULL	Our current price for this item in the catalog.
ESTIMATE_SHIP_TIME	VARCHAR (100)	NULL	Inventory: number of days/weeks before the item can be shipped.
SPECIAL_NOTES	VARCHAR (100)	NULL	Inventory related message to display with the item.

Table 2-5 WLCS_PRODUCT Table Metadata (Continued)

Column Name	Data Type	Null Value	Description and Recommendations
CREATOR	VARCHAR(50)	NULL	An entity primarily responsible for making the content of the product item.
PUBLISHER	VARCHAR(50)	NULL	An entity responsible for making the product item available.
CONTRIBUTOR	VARCHAR(50)	NULL	An entity responsible for making contributions to the content of the product item.
CREATION_DATE	DATE	NULL	A date associated with an event in the life cycle of the product item. 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	DATE	NULL	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 the Commerce services. 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	NUMBER(3)	NULL	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)	NULL	The language of the thumbnail image for the item. For related information, see the description of the LANG column.
SMALL_IMG_NAME	VARCHAR(50)	NULL	The name of the thumbnail image for the item.

2 The Product Catalog Database Schema

Table 2-5 WLCS_PRODUCT Table Metadata (Continued)

Column Name	Data Type	Null Value	Description and Recommendations
SMALL_IMG_URL	VARCHAR (254)	NULL	The URL of the thumbnail image for the category.
SMALL_IMG_ALT_TEXT	VARCHAR (254)	NULL	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.
LARGE_IMG_TYPE	NUMBER (3)	NULL	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)	NULL	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)	NULL	The name of the full-size image for the item.
LARGE_IMG_URL	VARCHAR (254)	NULL	The URL of the full-size image for the item.
LARGE_IMG_ALT_TEXT	VARCHAR (254)	NULL	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 (254)	NULL	The URL to the JSP used to display the item in summary form. For example: <code>/commerce/catalog/includes/itemssummary.jsp</code>

Table 2-5 WLCS_PRODUCT Table Metadata (Continued)

Column Name	Data Type	Null Value	Description and Recommendations
DET_DISPLAY_JSP_URL	VARCHAR(254)	NULL	The URL to the JSP used to display the item in detailed form. For example: /commerce/catalog/includes/itemdetails.jsp
SHORT_DESC	VARCHAR(254)	NULL	A short description of the content of the product item.
LONG_DESC	VARCHAR(2000)	NULL	A long description of the content of the product item.

The WLCS_PRODUCT_CATEGORY Database Table

Table 2-6 describes the metadata for the Commerce services WLCS_PRODUCT_CATEGORY table in the Commerce database. This table is used to join categories and items.

See “Defined Constraints” on page 2-23 for information about the constraint defined for this table.

The Primary Keys are SKU and CATEGORY_ID.

Table 2-6 WLCS_PRODUCT_CATEGORY Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
SKU	VARCHAR(40)	NOT NULL	A unique identifier (the “Stock Keeping Unit,” or SKU) for an item.
CATEGORY_ID	VARCHAR(20)	NOT NULL	A unique identifier for a category.

The WLCS_PRODUCT_KEYWORD Database Table

Table 2-7 describes the metadata for the Commerce services 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.

See "Defined Constraints" on page 2-23 for information about the constraint defined for this table.

The Primary Keys are KEYWORD and SKU.

Table 2-7 WLCS_PRODUCT_KEYWORD Table Metadata

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

The SQL Scripts Used to Create the Database

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

Scripts

Regardless of your database, execute one of the following to generate the necessary database objects for the modules desired (WebLogic Portal, WebLogic Personalization Server, Commerce services, Campaign services and Sample Portal):

- `PORTAL_HOME\db\create_all.bat` (Windows)
- `PORTAL_HOME/db/create_all.sh` (UNIX)

The following are the various directories underneath `WL_COMMERCE_HOME/db` (as seen in a UNIX environment):

`PORTAL_HOME/db/cloudscape/351`

`PORTAL_HOME/db/oracle/817`

Note: In this documentation, `PORTAL_HOME` is used to designate the directory where the product is installed.

Each of the databases supported have the same number of scripts in each of their sub-directories. The scripts are listed and described in Table 10-15 below.

Table 2-8 The Scripts Supporting the Databases

Script Name	Description
<code>create_all.bat</code>	Windows script used to connect to the database and create the necessary database objects for the modules desired (e.g., WebLogic Portal, WebLogic Personalization Server, Commerce services, Campaign services and Sample Portal)

2 The Product Catalog Database Schema

Table 2-8 The Scripts Supporting the Databases (Continued)

Script Name	Description
<code>create_all.sh</code>	Unix script used to connect to the database and create the necessary database objects for the modules desired (e.g., WebLogic Portal, WebLogic Personalization Server, Commerce services, Campaign services and Sample Portal)
<code>campaign_create_fkeys.sql</code>	SQL script used to create all foreign keys associated with the Campaign services.
<code>campaign_create_indexes.sql</code>	SQL script used to create all indexes associated with the Campaign services.
<code>campaign_create_tables.sql</code>	SQL script used to create all tables associated with the Campaign services.
<code>campaign_create_triggers.sql</code>	SQL script used to create all database triggers associated with the Campaign services.
<code>campaign_create_views.sql</code>	SQL script used to create all views associated with the Campaign services.
<code>campaign_drop_constraints.sql</code>	SQL script used to drop all constraints (other than foreign keys) associated with the Campaign services.
<code>campaign_drop_fkeys.sql</code>	SQL script used to drop all foreign key constraints associated with the Campaign services.
<code>campaign_drop_indexes.sql</code>	SQL script used to drop all indexes associated with the Campaign services.
<code>campaign_drop_tables.sql</code>	SQL script used to drop all tables associated with the Campaign services.
<code>campaign_drop_views.sql</code>	SQL script used to drop all views associated with the Campaign services.
<code>p13n_create_fkeys.sql</code>	SQL script used to create all foreign keys associated with the WebLogic Personalization Server.
<code>p13n_create_indexes.sql</code>	SQL script used to create all indexes associated with the WebLogic Personalization Server.
<code>p13n_create_tables.sql</code>	SQL script used to create all tables associated with the WebLogic Personalization Server.
<code>p13n_create_triggers.sql</code>	SQL script used to create all database triggers associated with the WebLogic Personalization Server.

Table 2-8 The Scripts Supporting the Databases (Continued)

Script Name	Description
<code>p13n_create_views.sql</code>	SQL script used to create all views associated with the WebLogic Personalization Server.
<code>p13n_drop_constraints.sql</code>	SQL script used to drop all constraints (other than foreign keys) associated with the WebLogic Personalization Server.
<code>p13n_drop_fkeys.sql</code>	SQL script used to drop all foreign key constraints associated with the WebLogic Personalization Server.
<code>p13n_drop_indexes.sql</code>	SQL script used to drop all indexes associated with the WebLogic Personalization Server.
<code>p13n_drop_tables.sql</code>	SQL script used to drop all tables associated with the WebLogic Personalization Server.
<code>p13n_drop_views.sql</code>	SQL script used to drop all views associated with the WebLogic Personalization Server.
<code>portal_create_fkeys.sql</code>	SQL script used to create all foreign keys associated with the WebLogic Portal.
<code>portal_create_indexes.sql</code>	SQL script used to create all indexes associated with the WebLogic Portal.
<code>portal_create_tables.sql</code>	SQL script used to create all tables associated with the WebLogic Portal.
<code>portal_create_triggers.sql</code>	SQL script used to create all database triggers associated with the WebLogic Portal.
<code>portal_create_views.sql</code>	SQL script used to create all views associated with the WebLogic Portal.
<code>portal_drop_constraints.sql</code>	SQL script used to drop all constraints (other than foreign keys) associated with the WebLogic Portal.
<code>portal_drop_fkeys.sql</code>	SQL script used to drop all foreign key constraints associated with the WebLogic Portal.
<code>portal_drop_indexes.sql</code>	SQL script used to drop all indexes associated with the WebLogic Portal.
<code>portal_drop_tables.sql</code>	SQL script used to drop all tables associated with the WebLogic Portal.
<code>portal_drop_views.sql</code>	SQL script used to drop all views associated with the WebLogic Portal.

2 The Product Catalog Database Schema

Table 2-8 The Scripts Supporting the Databases (Continued)

Script Name	Description
<code>sample_portal_create_fkeys.sql</code>	SQL script used to create all foreign keys associated with the Sample Portal.
<code>sample_portal_create_indexes.sql</code>	SQL script used to create all indexes associated with the Sample Portal.
<code>sample_portal_create_tables.sql</code>	SQL script used to create all tables associated with the Sample Portal.
<code>sample_portal_create_triggers.sql</code>	SQL script used to create all database triggers associated with the Sample Portal.
<code>sample_portal_create_views.sql</code>	SQL script used to create all views associated with the Sample Portal.
<code>sample_portal_drop_constraints.sql</code>	SQL script used to drop all constraints (other than foreign keys) associated with the Sample Portal.
<code>sample_portal_drop_fkeys.sql</code>	SQL script used to drop all foreign key constraints associated with the Sample Portal.
<code>sample_portal_drop_indexes.sql</code>	SQL script used to drop all indexes associated with the Sample Portal.
<code>sample_portal_drop_tables.sql</code>	SQL script used to drop all tables associated with the Sample Portal.
<code>sample_portal_drop_views.sql</code>	SQL script used to drop all views associated with the Sample Portal.
<code>wlcs_create_fkeys.sql</code>	SQL script used to create all foreign keys associated with the Commerce services.
<code>wlcs_create_indexes.sql</code>	SQL script used to create all indexes associated with the Commerce services.
<code>wlcs_create_tables.sql</code>	SQL script used to create all tables associated with the Commerce services.
<code>wlcs_create_triggers.sql</code>	SQL script used to create all database triggers associated with the Commerce services.
<code>wlcs_create_views.sql</code>	SQL script used to create all views associated with the Commerce services.
<code>wlcs_drop_constraints.sql</code>	SQL script used to drop all constraints (other than foreign keys) associated with the Commerce services.

Table 2-8 The Scripts Supporting the Databases (Continued)

Script Name	Description
<code>wlcs_drop_fkeys.sql</code>	SQL script used to drop all foreign key constraints associated with the Commerce services.
<code>wlcs_drop_indexes.sql</code>	SQL script used to drop all indexes associated with the Commerce services.
<code>wlcs_drop_tables.sql</code>	SQL script used to drop all tables associated with the Commerce services.
<code>wlcs_drop_views.sql</code>	SQL script used to drop all views associated with the Commerce services.

Defined Constraints

Various constraints are defined and used in the Product Catalog database schema. These constraints can be found in the following scripts:

`wlcs_create_fkeys.sql`—contains the Foreign Keys

`wlcs_create_tables.sql`—contains the Check Constraints

Table 2-9 Constraints Defined on Product Catalog Database Tables

Table Name	Constraints
CATALOG_PROPERTY_VALUE	<p>Column—ENTITY_ID Constraint—FK1_CAT_PROP_V Constraint Type—FOREIGN KEY Ensures that each CATALOG_PROPERTY_VALUE references an existing CATALOG_ENTITY via the ENTITY_ID column.</p> <p>Column—PROPERTY_KEY_ID Constraint—FK2_CAT_PROP_V Constraint Type—FOREIGN KEY Ensures that each CATALOG_PROPERTY_VALUE references an existing CATALOG_PROPERTY_KEY via the PROPERTY_KEY_ID column.</p> <p>Column—BOOLEAN_VALUE Constraint—CC1_CAT_PROP_V Constraint Type—CHECK Ensures the value of the BOOLEAN_VALUE column is either 0 (false) or 1 (true).</p>
WLCS_CATEGORY	<p>Column—CATEGORY_ID Constraint—FK1_CATEGORY Constraint Type—FOREIGN KEY Ensures that each PARENT_ID references an existing WLCS_CATEGORY via the CATEGORY_ID column.</p>
WLCS_PRODUCT_CATEGORY	<p>Column—CATEGORY_ID Constraint—FK1_PRODUCT_CAT Constraint Type—FOREIGN KEY Ensures that each CATEGORY_ID references an existing WLCS_CATEGORY via the CATEGORY_ID column.</p> <p>Column—SKU Constraint—FK2_PRODUCT_CAT Constraint Type—FOREIGN KEY Ensures that each SKU references an existing WLCS_PRODUCT via the SKU column.</p>
WLCS_PRODUCT_KEYWORD	<p>Column—SKU Constraint—FK1_PRODUCT_KEY Constraint Type—FOREIGN KEY Ensures that each SKU references an existing WLCS_PRODUCT via the SKU column.</p>

2 *The Product Catalog Database Schema*

3 Using the Database Loader

The Commerce services include a DBLoader program that you can use to bulk load data into any table in a database, which is especially helpful for tasks like populating your product catalog. While you could use the browser-based Administration Tools 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 Commerce 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 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 Commerce services DBLoader program loads data that you provide in a text file into a 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: column names for that table
- Third row: data types for the columns 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 database.

For example, the header line might contain:

```
130 , WLCS_PRODUCT
```

Second Row

The second row identifies the table column names into which you are loading data. You must include the primary key column or columns in the input file. Preface each primary key column name with an asterisk (*). Apart from primary keys in tables, all other columns are defaulted as `NULL`. Thus, you may omit column 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 column being loaded. See Chapter 2, “The Product Catalog Database 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 , VARCHAR , 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 , VARCHAR , 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` column 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 column in the input record by including the delimiter character (by default, a comma) in the correct position (matching the position of the columns you listed in line 2, the column names). For example:

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

In the previous example a value for the fourth identified column (`EST_SHIP_TIME`) was not specified. This condition is fine because this column is not a primary key for the database record. The column’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 column, you should enter `NULL` in the correct position for the column in that record. Do not enclose `NULL` in quotes; enclosing the word `'NULL'` in quotes will cause the column to be stored as a string.

The `dbloader.properties` File

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

This file resides in the `PORTAL_HOME` directory. `PORTAL_HOME` is the directory where you installed WebLogic Portal.

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. The default driver is the Cloudscape JDBC driver that ships with WebLogic.
<code>connection</code>	<code>jdbc:cloudscape:Commerce</code>	Database name where loaded data should go. <code>Commerce</code> is the name of the default database that ships with the WebLogic Portal. The location of this database is specified by the system property: <code>cloudscape.system.home</code> .
<code>dblogin</code>	<code>None</code>	The database username. The default Cloudscape database does not require a username to be specified. The login name must have read/write privileges on the affected tables.

Property Name	Default Value	Description
<code>dbpassword</code>	None	The database user password. The default Cloudscape database does not require a user password to be specified.
<code>delimiter</code>	,	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 (<code>LONG_DESC</code>). Choose another character, such as the circumflex (^) as a delimiter.
<code>timestampable</code>	<code>WLCS_CATEGORY,</code> <code>WLCS_PRODUCT</code>	Identifies the database tables to which DBLoader will track updates (for these two tables). The column name is fixed in the schema provided by the Commerce services. However, if you are using DBLoader for other tables (not WLCS tables), you can specify other column names of your own.
<code>timestampfield</code>	<code>MODIFIED_DATE</code>	Specifies the column in the <code>WLCS_CATEGORY</code> and <code>WLCS_PRODUCT</code> tables that identifies the last time this record in the table was modified. The value of the column specified is used by DBLoader to learn when the most recent update was made in each record in the tables identified in the <code>timestampable</code> property. The column name is fixed in the schema provided by the Commerce services. However, if you are using DBLoader for other tables (not WLCS tables), you can specify other column names of your own.
<code>commitTxn</code>	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.

3 Using the Database Loader

Property Name	Default Value	Description
encoding	Not specified in the <code>dbloader.properties</code> file; therefore, the default is the Java 2 SDK's platform default.	<p>Sets the multibyte character encoding type. The property value supplied can be UCS2 or UTF8.</p> <p>When writing data into and reading data out of the database, 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.</p> <p>However, if you need to write/read data to/from the database that is encoded differently than your system's native encoding, you will have to explicitly perform the translation. For more information, see "Important Database Considerations" on page 3-10.</p>

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

Listing 3-1 Sample `dbloader.properties` File

```
# THE PROPERTIES FILE FOR BEA CUSTOM DATA LOADER
#####
# Examples of using JDBC drivers and establishing connections to your database
# 'Commerce' and 'CATALOG' are examples of database names

# Cloudscape driver connects to the sample database shipped with WebLogic
jdbcdriver=COM.cloudscape.core.JDBCdriver
connection=jdbc:cloudscape:Commerce
dblogin=none
dbpassword=none

# WebLogic jDriver for Oracle (OCI Driver). Uncomment to use:
#jdbcdriver=weblogic.jdbc.oci.Driver
#connection=jdbc:weblogic:oracle:sturney
#dblogin=FLAGSTAFF
#dbpassword=FLAGSTAFF

# Delimiter used in your data file
delimiter=,

# Character used to specify the primary key in your data file.
# This value will default to '*' if not specified.
primarykeyidentifier=*
```

```
# Table and column name for autoinsertion of current DATE.
timestamptable=wlcs_category
timestampfield=modified_date

# Allowable encoding values:
# If encoding=UCS2, unicode encoding used is Unicode
# If encoding=UTF8, unicode encoding used is UTF8
# If no encoding is specified, it's default to the platform encoding of the JDK
# For any other value specified, it is reset to default
# Note:
# 1. If encoding is UCS2, no change is needed to database or sql API
# (accessor/mutator)using WLS OCI driver.
# 2. If encoding is UTF8, no change is needed to database or mutator sql API.
# However, you have to getBinaryStream() with WLS OCI driver or Oracle thin driver.
# Cloudscape doesn't have proper sql API to get multibyte codes out of Cloudview
encoding=UTF8

# Committing transactions to the database after parsing every N number of data
# records. If the value is less than or equal to one, it'll be committing every
# transaction.
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:

- `PORTAL_HOME\bin\win32` (Windows)
- `PORTAL_HOME/bin/unix` (UNIX)

Note: `PORTAL_HOME` is the directory where you installed WebLogic Portal.

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`.

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

If you are running the WebLogic Portal server with Oracle, then the drawback might be a slower performance for the time the data is being loaded into the database.

Note: You might want to back up 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 the corresponding input filename 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 Commerce services 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 filename, 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 filename, 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 columns affected by the load (as specified in the second line of the input data file) against the number of input columns in each record. Because the column delimiter is a comma (by default), this character is not allowed in a string input column. If extra commas are supplied inadvertently, such as punctuation in a `LONG_DESC` (Long Description) column, 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 column values. Or select a different delimiter character and specify it in the `dbloader.properties` file. For more information, see “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 data and referential integrity between tables with the use of constraints. For example, the primary key constraint on `WLCS_PRODUCT` and `WLCS_CATEGORY`, or the foreign key constraint on `WLCS_PRODUCT_CATEGORY`.

Primary keys and unique indexes prevent the possibility of placing duplicate entries in the table. Foreign key constraints ensure referential integrity by making certain that the parent key already exists before allowing the child record to be written to the database. For example, a foreign key constraint exists between the `WLCS_PRODUCT_CATEGORY` table and its parent tables, the `WLCS_PRODUCT` and `WLCS_CATEGORY` table. This ensures that each `SKU` value and each `CATEGORY_ID` value in the `WLCS_PRODUCT_CATEGORY` table have corresponding entries in their respective parent tables (`WLCS_PRODUCT` and `WLCS_CATEGORY`). For example, the category record with a `CATEGORY_ID` value

of "Men's Apparel" in the `WLCS_PRODUCT_CATEGORY` table requires that an entry exist in `WLCS_CATEGORY`, otherwise, an exception will be thrown. At the same time, a SKU of "Khakis" must exist in `WLCS_PRODUCT`.

Note: For every `WLCS_PRODUCT` and `WLCS_CATEGORY` table entry, a corresponding entry in the `CATALOG_ENTITY` table must also be made.

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 Database 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 and higher can convert several files encoding into Unicode 2.0 and higher. For details, see <http://java.sun.com/products/jdk/1.2/docs/guide/internat/encoding.doc.html>.

What this means to your development group:

- a. When writing data into and reading data out of the database, 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, see <http://java.sun.com/products/jdk/1.2/docs/guide/internat/encoding.doc.html>.
- c. If you need to write/read data to/from the database 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/>.

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 Database 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, 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, 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, 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, see <http://www.eti.com>.

4 Catalog Administration Tasks

The Commerce services included in the WebLogic Portal product suite provide administrators with command-line scripts, property files, and browser-based Administration Tools. The Administration Tools allow you to manage the behavior and content of your product catalog.

For catalog administration, the tasks include:

- Starting the Server and the Administration Tools
- Using the Catalog Management Tools
- 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 Product Catalog
- Assigning Items to Categories
- Editing the Attributes for Categories and Items
- Editing the Availability of an Item
- Determining How Categories and Items are Displayed to Web Site Visitors
- Deleting Items or Removing Items from One or More Categories
- Removing Categories

- Moving Items from One Category to Another Category
- Defining 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 `PORTAL_HOME` is used to represent the directory in which you installed the WebLogic Portal software.

Starting the Server and the Administration Tools

The browser-based Administration Tools run as a Web application in the WebLogic Server environment. Therefore, the WebLogic Portal server must be running before you can use the Administration Tools.

When you start the server for a Web application, you are passing 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, the Commerce services include a `StartCommerce.bat` (Windows) or `StartCommerce.sh` (UNIX) script. Detailed instructions for using these scripts can be found in the “Starting and Shutting Down a Server” topic in the *Deployment Guide*.

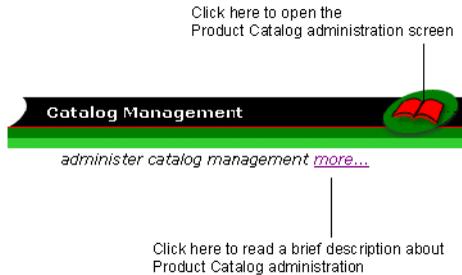
Once the WebLogic Portal server is started, you need to start the Administration Tools. For detailed instructions on starting the Administration Tools, see “Accessing the WebLogic Portal Administration Tools” in the *WebLogic Portal Architectural Overview* documentation.

Using the Catalog Management Tools

Once you have started the Administration Tools and are viewing the home page for the tools, you can click the Catalog Management graphic to begin using the Catalog Management tools.

Figure 4-1 identifies the two types of links on the Catalog Management graphic.

Figure 4-1 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 4-2.

Figure 4-2 Main Catalog Management Screen



Table 4-1 summarizes the catalog management functional areas.

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.

Changing the Administrator Password

The initial account information for the Administrator account supplied with the Commerce services is:

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

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

1. Open the Administration Tools Home Page, as described in “Accessing the WebLogic Portal Administration Tools” in the *WebLogic Portal Architectural Overview* documentation. The server must be running. One way to access the Administration Tools Home Page is to enter the URL in your browser:

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

2. When you are prompted to log in, enter the current password.
3. On the Administration Tools Home Page, click the User Management icon, as shown in Figure 4-3.

Figure 4-3 User Management Icon on Main Administration Screen



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

Figure 4-4 Users Link on the User Management Screen



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

Figure 4-5 Searching for the Administrator Account



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

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

Figure 4-6 Administrator Link in the User Account Search Results



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 4-7.

Figure 4-7 Edit Button on Users: administrator Screen



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

Figure 4-8 Entering the New Password

The screenshot shows the BEA 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 three buttons labeled "home", a book icon, and a question mark icon on the right. Below the header is a dark green 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 header, there is a line of text: "Enter changes to the user information then click Save." The form contains three fields: "Username:" with the value "administrator", "Password:" with a masked password field and a red asterisk to its right, and "Verify Password:" with a masked password field and a red asterisk to its right. At the bottom right of the form are two buttons labeled "back" and "save".

Warning: Make sure to 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 Tools, a step that requires the current administrator password. If you forget the account’s password, you will have to recreate the Commerce database.

After entering the new password, click the Save button. Then click the Home button at the top of the screen to return to the main administration screen.

Loading Data into the Product Catalog

The Commerce services include a DBLoader program that can bulk load data into the product catalog (or any) database. While you could use the Administration Tools to add new items 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 when loading legacy data from an existing database into the Commerce 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.

For details, see Chapter 3, “Using the 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 Database Loader,” or by using the Catalog Management Administration Tools. This section explains how to use the Administration Tools.

The steps are as follows:

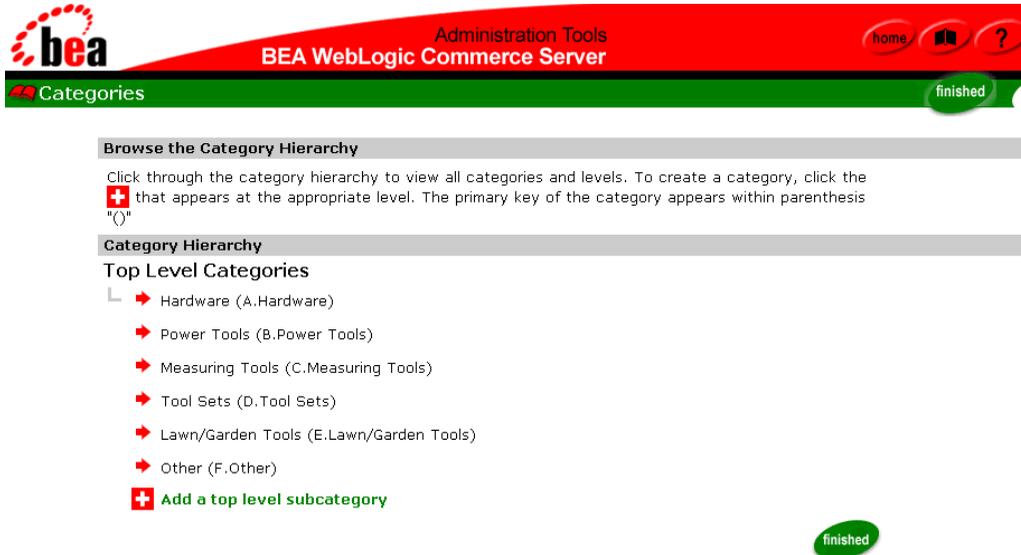
1. Make sure the WebLogic Portal server is running and that the Administration Tools are started. See “Starting the Server and the Administration Tools” on page 4-2 for more information.
2. On the Administration Tools Home Page, click the Catalog Management graphic.
3. On the main Catalog Management screen, click the Create button on the Categories graphic, as shown in Figure 4-9.

Figure 4-9 Create Button on Categories Graphic



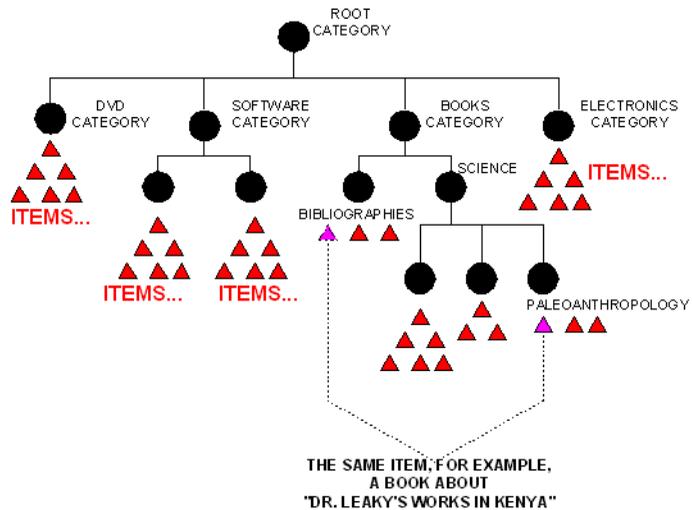
On the Categories screen, you can either add a new category at the current level in the catalog hierarchy, or you can click an existing category name to create a subcategory. Figure 4-10 shows the Categories screen.

Figure 4-10 Portion of Categories Screen When the Create Button Is Clicked



As you will recall from “Catalog Hierarchy” on page 1-6, the categories in the product catalog are organized as a hierarchy as shown in Figure 4-11.

Figure 4-11 Product Catalog Hierarchy



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

4. 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 top level subcategory”. For example, assume that you want to add a “Wood and Lumber” category because the electronic store will now offer different types of specialty hardwoods such as Ash. After you create the Wood and Lumber category, you will then create an “Ash” subcategory.
5. Figure 4-12 and Figure 4-13 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 4-12 Top Portion of the Create New Subcategory Screen



Create New Subcategory of Category : 'ROOT'

Enter the appropriate information then click Create. Fields marked * are required.

Category Identifier*:	<input type="text"/>	Unique category identifier.
Category Name*:	<input type="text"/>	Name of the category.
Short Description*:	<input type="text"/>	Short description associated with the category.
Long Description:	<input type="text"/>	Long description of the category.
Display JSP URL:	<input type="text"/>	JSP associated with the display of this category.
Creator:	<input type="text"/>	Person who created this category.
Publisher:	<input type="text"/>	Publisher of the category.
Contributor:	<input type="text"/>	Entity responsible for contributing to the category.
Creation Date:	<input type="text"/>	Creation date for the resource.
Modified Date:	<input type="text"/>	Modification date for the resource.
Language:	<input type="text"/>	Language of the intellectual content of the resource.
Relation:	<input type="text"/>	Reference to a related resource.
Rights:	<input type="text"/>	Information about rights held in and over the resource.
Coverage:	<input type="text"/>	Extent or scope of the resource content.
Source:	<input type="text"/>	Reference to a resource from which the present resource is derived.

Figure 4-13 Bottom Portion of the Create New Subcategory Screen

Images for the Category

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

- Enter the required fields, which are indicated by the asterisk (*). The required fields are:
 - Category Identifier (also the primary key of the WLCS_CATEGORY table in the Commerce database). In this example, enter Wood.
 - Category Name. In this example, enter Wood and Lumber.

- **Short Description.** Enter a short description such as “hardwood, exterior, moulding, construction grade”.

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 `WLCS_CATEGORY` Database Table” on page 2-9.

7. After you enter the field values, click the Create button to create the new category. (Or, click the Back button to cancel the current create category operation.)
8. When the new category has been created, a confirmation message is displayed on the refreshed Categories screen. Figure 4-14 shows the confirmation message after creating the new category.

Figure 4-14 Confirmation Screen After New Category Is 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, the text "Administration Tools" and "BEA WebLogic Commerce Server" in the center, and "home", a folder icon, and a question mark icon on the right. Below the header is a green bar with the text "Categories". A red message box displays "Category : Wood and Lumber created successfully." Below this is a grey bar with the text "Create New Subcategory of Category : 'ROOT'". Underneath is a grey box with the instruction "Enter the appropriate information then click Create. Fields marked * are required." The form contains three fields: "Category Identifier*" with the value "Wood" and the label "Unique category identifier.", "Category Name*" with the value "Wood and Lumber" and the label "Name of the category.", and "Short Description*" with the value "hardwood, exterior, mouldings, construction grade" and the label "Short description associated with the category."

9. 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 new category. Figure 4-15 shows the new Wood and Lumber category under the catalog’s top-level root.

Figure 4-15 Newly Added Category Near Bottom of Category Screen

- Other (F.Other)
- Wood and Lumber (Wood)
- +** Add a top level subcategory



10. 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 a Hardwood subcategory under the Wood and Lumber category. To do this, click the red right-arrow next to the Wood and Lumber category. Figure 4-16 shows the updated screen.

Figure 4-16 Updated Screen While Adding a Subcategory

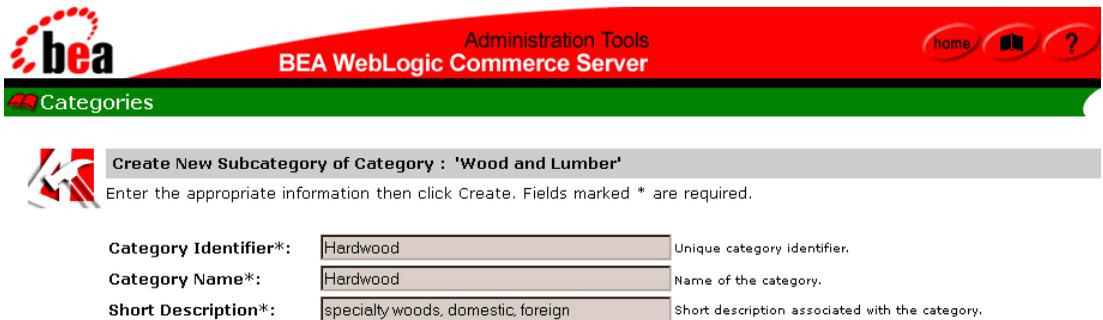
- Other (F.Other)
- Wood and Lumber (Wood)
- +** Add a subcategory to Wood and Lumber
- +** Add a top level subcategory



The icon next to the Wood and Lumber 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, click the plus-sign icon next to the text, Add a subcategory to Wood and Lumber.

11. Figure 4-17 shows the top portion of the next screen: Create New Subcategory below Category : 'Wood and Lumber'. On this screen, we have entered data into the first three fields.

Figure 4-17 Creating a New Subcategory



The screenshot shows the BEA WebLogic Commerce Server Administration Tools interface. The top navigation bar is red with the BEA logo on the left and 'Administration Tools BEA WebLogic Commerce Server' in the center. On the right, there are 'home', a book icon, and a question mark icon. Below the navigation bar is a green bar with the text 'Categories'. The main content area has a grey header that reads 'Create New Subcategory of Category : 'Wood and Lumber''. Below the header, there is a text prompt: 'Enter the appropriate information then click Create. Fields marked * are required.' The form contains three rows of input fields:

Category Identifier*:	<input type="text" value="Hardwood"/>	Unique category identifier.
Category Name*:	<input type="text" value="Hardwood"/>	Name of the category.
Short Description*:	<input type="text" value="specialty woods, domestic, foreign"/>	Short description associated with the category.

Click the Create button to create the new subcategory Hardwood under the Wood and Lumber category.

Repeat the previous steps to create additional categories and subcategories.

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 Database Loader,” or by using the Administration Tools. This section explains how to use the Administration Tools.

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 4-19.

The steps are as follows:

1. Make sure the WebLogic Portal server is running and that the Administration Tools are started. See “Starting the Server and the Administration Tools” on page 4-2 for more information.
2. On the main administration screen, click the Catalog Management graphic.
3. On the main Catalog Management screen, click the Create button on the Items graphic, as shown in Figure 4-18.

Figure 4-18 Create Button on Items Graphic



4. The Create New Item screen is displayed. Figure 4-19 and Figure 4-20 shows the top portion and bottom portion of the screen.

Figure 4-19 Top Portion of the Create New Item Screen



Create a New Item

Enter the appropriate information then click Create.

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 style="height: 30px;" 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 4-20 Bottom Portion of Create New Item Screen

Modified Date:  Modification date for the resource.
Language: A language of the intellectual content of the resour
Relation: A reference to a related resource.
Rights: Information about rights held in and over the resou
Coverage: The extent or scope of the content of the resource.
Source: A reference to a resource from which the present resource is derived.

Keywords for the Item

Keywords:  

Images for the Item

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

back
create

5. 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 WLCS_PRODUCT Database Table” on page 2-13.

Note: See “Controlling the Visibility of Items in the Product Catalog” on page 4-18 for important information about the Visible check box.

- For example, assume that you need to add a new product item, an “Ash-1x3”. Figure 4-21 shows the top portion of the screen as it appears when information is being added about this item.

Figure 4-21 Adding a New Product Item



Create a New Item

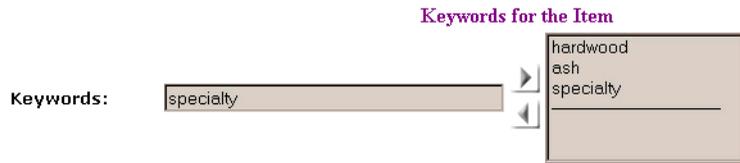
Enter the appropriate information then click Create.

SKU*:	<input type="text" value="9-MY9467"/>	The Stock Keeping Unit for the item.
Item Name*:	<input type="text" value="Ash-1x3"/>	The name of the item.
Short Description*:	<input type="text" value="Solid dried true 1x3"/>	Short description associated with the item.
Visible:	<input checked="" type="checkbox"/>	Include or exclude the item from search results.
Long Description:	<input type="text" value="Kiln dried and pre-final milling dimensions for greater flexibility of use."/>	Long description of the item.
Price Amount:	<input type="text" value="4.50 per linear foot"/>	The selling price of the item.
Price Currency:	<input type="text" value="USD"/>	The currency for the selling price of the item.
MSRP Amount:	<input type="text" value="5.50 per linear foot"/>	The manufacturer suggested retail price of this item.
MSRP Currency:	<input type="text" value="USD"/>	The currency for the msrp of this item.

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 visitors 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 Tools 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.

Figure 4-22 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 4-22 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 “Using the `wlcs-catalog.properties` File” on page 4-43.
7. 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.)
 8. 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 Product 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 visitors to your deployed Web site are *browsing* for an item (that is, clicking through the catalog’s hierarchy of categories and subcategories), by default they will see the item even if you left the Visible check box unchecked.

To make an item invisible to the results of keyword searches of the catalog, in the current release you must:

- Mark the item as invisible (set or leave the Visible check box unchecked)
- Orphan the item by removing it from all categories

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 of the catalog, the default visitor's `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 Tools.

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 Tools, you can assign each item to one or more categories. This section describes how to use the Administration Tools 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 Tools 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 `WLCS_PRODUCT_CATEGORY` table maps the `CATEGORY_ID` primary key from the `WLCS_CATEGORY` table to the `SKU` primary key for the `WLCS_PRODUCT` table.

For related information, see:

- “The `WLCS_PRODUCT_CATEGORY` Database Table” on page 2-17.
- Chapter 3, “Using the Database Loader.”

Using the Administration Tools to Assign Items to Categories

To assign items to categories in the Administration Tools, follow these steps:

1. Make sure the WebLogic Portal server is running and that the Administration Tools are started. See “Starting the Server and the Administration Tools” on page 4-2 for more information.
2. On the main administration screen, click the Catalog Management graphic.
3. On the main Catalog Management screen, click the underlined Categories link.
4. 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.

5. 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 Wood and Lumber category, then clicked the underlined Hardwood link. (These sample categories were added to the catalog in the previous section, “Adding Categories to the Catalog” on page 4-8.)
6. Figure 4-23 shows the top portion of a sample Editing Category: Hardwood screen. Notice the link near the top of the display: To modify the items assigned to this category, click [here](#).

Figure 4-23 Sample Editing Category Screen with Modify Link

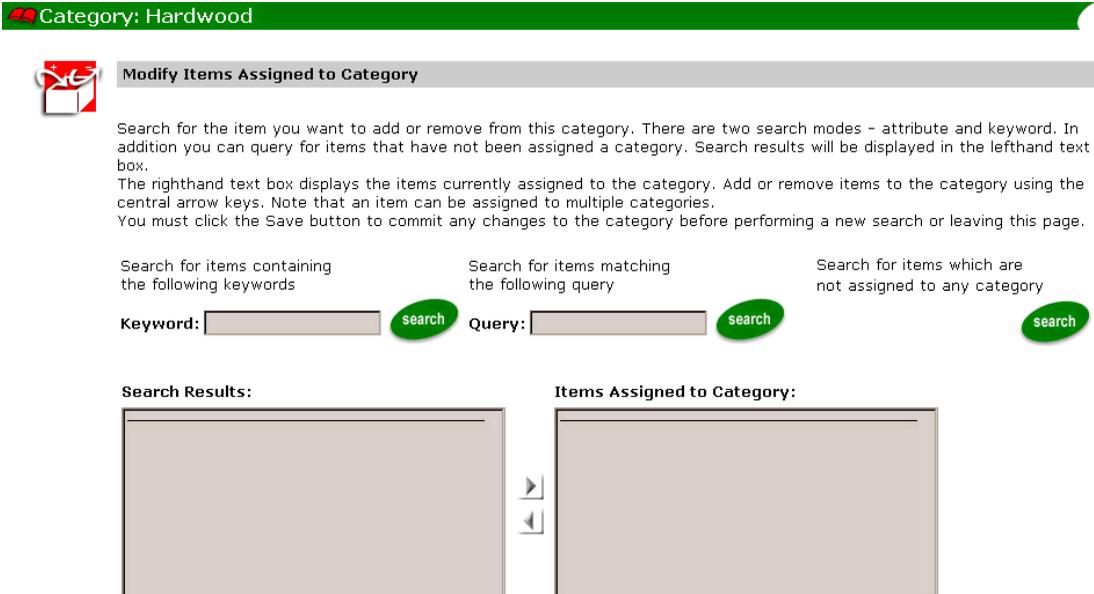
Editing Category : Hardwood

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

Category Identifier:	Hardwood	Unique category identifier.
Category Name*:	Hardwood	Name of the category.
Short Description*:	specialty, domestic, foreign	Short description associated

- 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 4-24. In this particular screen, there are currently no items assigned to the Hardwood category.

Figure 4-24 Modify Items Assigned to Category Screen



- The Items Assigned to Category text box shows the items that are already in this category. (In this example so far, there are no items in the Hardwood category.) You can search for the item you want to add or remove via three modes: keyword, query-based, or orphaned-items (uncategorized items). The search results are displayed on the left-side text box. To add an item to the category, move the item to the right-side text box 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 Ash-1x3 item to the Hardwood category. (This item was added to the sample data in the catalog in the section “Adding Items to the Catalog” on page 4-14.) You can search for the keywords

related to the item. Figure 4-25 shows a sample Modify Items Assigned to Category screen where the ash keyword was entered, the search results found the item, and the results were displayed in the left-side Search Results box.

Figure 4-25 Sample Keyword Search Results on Modify Items Assigned to Category Screen

The screenshot displays the 'Modify Items Assigned to Category' interface. At the top, there are three search options: 'Search for items containing the following keywords', 'Search for items matching the following query', and 'Search for items which are not assigned to any category'. The first option is selected, and the 'Keyword' field contains 'ash'. Below this, there are two search buttons labeled 'search'. The 'Search Results' box on the left contains the text '(9-MY9467), Ash-1x3'. To the right of this box are two arrow buttons (right and left). The 'Items Assigned to Category' box on the right is currently empty.

10. To add one or more items to the current category, select it or them in the left-side text box and then click the right-arrow button. The item(s) move(s) to the right-side text box. 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, a confirmation message is displayed. In this example, the message was: Category 'Hardwood' has been updated.

11. 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 'A*'
!(price > 100) || (msrpAmount >= 300)
modifiedDate < now
```

When the search results are displayed in the left-side text box, use the right-arrow button to move the appropriate items into the right-side Category Items text box. 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. For

details about the query-based search syntax, see “Query-Based Search Syntax” on page 5-76.

12. To find a product item that is no longer associated with any categories in the catalog, but is still present in the catalog, click the Search button located under the text “Search for items which are not assigned to any category”.
13. 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 text box and then click the right-arrow button. This step will move the items to the right-wide text box. 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.

Editing the Attributes for Categories and Items

You can use the Catalog Management Administration Tools 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 WebLogic Portal server is running and that the Administration Tools are started. See “Starting the Server and the Administration Tools” on page 4-2 for more information.
2. On the main administration screen, click the Catalog Management graphic.
3. On the main Catalog Management screen, click the underlined *Categories* button on the Categories graphic.
4. On the main Categories screen, you can find the category you want to modify by either:

4 Catalog Administration Tasks

- 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 appears in parentheses next to the category name. You can assign an identifier when you create a new category, but once created, you cannot edit the identifier. Also, categories are created sequentially. The newest category is always displayed last in the list that the visitor sees on your site. Figure 4-26 shows a portion of the resulting Categories screen after we entered the category identifier `hardwood` in the search field and clicked the Search button. In the figure, notice how the found category name is shown in red type.

Figure 4-26 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

- ▶ [Hardware \(A.Hardware\)](#) **X**
- ▶ [Power Tools \(B.Power Tools\)](#) **X**
- ▶ [Measuring Tools \(C.Measuring Tools\)](#) **X**
- ▶ [Tool Sets \(D.Tool Sets\)](#) **X**
- ▶ [Lawn/Garden Tools \(E.Lawn/Garden Tools\)](#) **X**
- ▶ [Other \(F.Other\)](#) **X**
- ◀ [Wood and Lumber \(Wood\)](#) **X**
 - ▶ **Hardwood (Hardwood)** **X**

finished

- 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.
5. When the category you want to modify is displayed, click the name of the category. For example, Figure 4-27 shows a portion of the resulting screen when we clicked the `Hardwood (Hardwood)` category.

Figure 4-27 Editing Category: Chargers Sample Screen

Editing Category : Hardwood

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

Category Identifier:	<input type="text" value="Hardwood"/>	Unique category identifier.
Category Name*:	<input type="text" value="Hardwood"/>	Name of the category.
Short Description*:	<input type="text" value="specialty, domestic, foreign"/>	Short description associated with the category.
Long Description:	<input type="text"/>	Long description of the category.
Display JSP URL:	<input type="text" value="/commerce/catalog/includes/category.jsp"/>	JSP associated with the display of this category.

- On this screen, you can add, change, or remove the category’s 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 2-9 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.

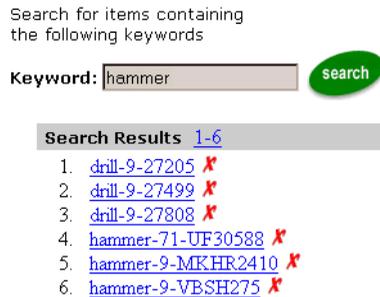
- After you edit the attributes for the category, click the Save button to commit your changes to the database. Or, to exit the screen without committing your changes, click the Back button.

Editing Product Item Attributes

To edit the attributes for a product item, follow these steps:

1. Make sure the WebLogic Portal server is running and that the Administration Tools are started. See “Starting the Server and the Administration Tools” on page 4-2 for more information.
2. On the main administration screen, click the Catalog Management graphic.
3. On the main Catalog Management screen, click the underlined `Items` button on the Items graphic.
4. On the Item Search screen, you can find the product item by either entering one of its keywords, or by entering a query-based search expression, or by searching for orphaned items. For example, Figure 4-28 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 4-28 Sample Keyword Search Results for Product Items



5. 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 'H*'
!(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 about the query-based search syntax, see “Query-Based Search Syntax” on page 5-76.

Figure 4-29 shows the results of a search for `price > 10 && price <= 50`.

Figure 4-29 Sample Query-Based Search to Find a Product Item

Search for items containing the following keywords

Keyword:

[search](#)

Search for items matching the following query

Query:

[search](#)

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

1. [bit set-9-SK84209](#) ✖
2. [caliper-9-38739](#) ✖
3. [level-9-39909](#) ✖
4. [pliers set-9-45011](#) ✖
5. [pliers set-9-45017](#) ✖
6. [regulator-9-16022](#) ✖
7. [shears-9-WSW5](#) ✖
8. [shovel-71-UF44600](#) ✖
9. [tape-9-82576](#) ✖
10. [wrench set-9-800215](#) ✖

6. After you find the item you want to modify, click the underlined item name to edit its attributes. Figure 4-30 shows the resulting screen when we click the search result for `wrench Set-9-800215`.

Figure 4-30 Sample Initial Edit Item Attributes Screen

Item: 9-800215 finished

Display Property Set attributes for this item: [search](#)

Item Core Attributes edit

Item Name: **wrench set-9-800215**

Short Description: **wrench set; open end; craftsman; 7 pieces; industrial; sockets, ratchets, wrench**

Custom Attributes

finished

- Click the red and white Edit button. Figure 4-31 shows a portion of a resulting Edit Item Information screen. (Notice how the price is greater than 10 and less than or equal to 50, from the prior search that returned this as a matching item.)

Figure 4-31 Sample Edit Item Information Screen

SKU:	9-800215	The Stock Keeping Unit for the item.
Item Name*:	wrench set-9-800215	The name of the item.
Short Description*:	wrench set open end; craftsman; 7 pieces; in	Short description associated with the item.
Visible:	<input checked="" type="checkbox"/>	Include or exclude the item from search results.
Long Description:	wrench set; open end; craftsman; 7 pieces; industrial; sockets,	Long description of the item.
Price Amount:	41.95	The selling price of the item.
Price Currency:	USD	The currency for the selling price of the item.
MSRP Amount:	47.95	The manufacturer suggested retail price of this item.
MSRP Currency:	USD	The currency for the msrp of this item.
Tax Code:	73212	The Tax Code for the item.

- 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 WLCS_PRODUCT Database Table” on page 2-13 for details about the item’s fields. Also see “Controlling the Visibility of Items in the Product Catalog” on page 4-18 for important related information.
- Before you save any changes on the Edit Item Information screen, set the item’s inventory status on the Item Inventory screen. For information about this setting, see “Editing the Availability of an Item.”
- 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, the message “The item was updated successfully” is displayed.

Editing the Availability of an Item

In the current release, the Administration Tools include an inventory setting. 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 4-31 in the previous section showed the top portion of an Edit Item Information screen, which included the link, “For inventory information, click here.” Figure 4-32 shows the Item Inventory screen that is displayed when you click this link.

Figure 4-32 Sample Item Inventory Screen

Items

Item Inventory

Enter the appropriate information then click Save.

Item Name: wrench set-9-800215

In Stock: Yes No

Shipping Time: Ships in 24 hours

Comments:

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 visitor of your Web site on the item details screen) to indicate the shipping time. You can also enter additional comments about the item’s inventory.

Note: In the current release, there is no automated inventory-count calculation.

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

Determining How Categories and Items are Displayed to Web Site Visitors

You can control the format and content of category and item data that is displayed to the Web site's visitors by setting the Display URLs field for each category or item. The Product Catalog includes the URLs 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. The `wlcs-catalog.properties` file is located in the `commerce_system.jar` file, which resides in the `PORTAL_HOME/lib` directory. 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/itemsummary.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.

You can change these Display JSP defaults in `wlcs-catalog.properties`. You also can overwrite the default 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 columns in the Commerce database:

- `DISPLAY_JSP_URL` in the `WLCS_CATEGORY` table
- `DET_DISPLAY_JSP_URL` and `SUM_DISPLAY_JSP_URL` columns 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 HTML/JSP developer about the type of summary or detail pages to be presented. For example, the `details.jsp` template is coded to return the detail JSP for items, as shown in Listing 4-1.

Listing 4-1 Excerpt From the `details.jsp` Template

```
<!-- 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"/>

<!-- Included the detail JSP -->
<jsp:include page="<%= detailJsp.getUrl() %>" flush="true"/>
.
.
.
```

Deleting Items or Removing Items from One or More Categories

As described earlier in this topic, you can assign an item to one or more categories. If necessary, you could use the Administration Tools to:

- Delete the item entirely from the catalog.
- Remove the item from one category while keeping its assignment 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, the item record will automatically be removed from the ProductItemCache or CategoryCache. This step ensures that subsequent Web site visitors get a valid view of the available, categorized items. For more information about caching, see “Improving Catalog Performance by Optimizing the Catalog Cache” on page 4-41.

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 “Using Caches” in the *Performance Tuning Guide*, or the WebLogic Server Administration Console’s online help.

Deleting an Item from the Catalog

To permanently delete an item from the catalog database via the Administration Tools, 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:

Deleting Items or Removing Items from One or More Categories

1. Make sure the WebLogic Portal server is running and that the Administration Tools are started. See “Starting the Server and the Administration Tools” on page 4-2 for more information.
2. On the main administration screen, click the Catalog Management graphic.
3. On the main Catalog Management screen, click the underlined `Items` link.
4. 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 4-33 shows the Search results on the refreshed Item Search screen after searching for the keyword `cabinet`.

Figure 4-33 Sample Search for an Item Using Cabinet Keyword

Search for items containing
the following keywords

Keyword:

Search Results 1-6

1. [cabinet-9-65021](#) ✖
2. [cabinet-9-65022](#) ✖
3. [cabinet-9-65065](#) ✖
4. [cabinet-9-65459](#) ✖
5. [cabinet-9-FB8500ST](#) ✖
6. [cabinet-9-WLSTX400](#) ✖

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.

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

Note: The query expression `name like` is case sensitive. For example, Figure 4-34 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 “Query-Based Search Syntax” on page 5-76.

Figure 4-34 Sample Query-Based Search Results Screen

Search for items containing the following keywords

Search for items matching the following query

Keyword:

Query:

Search Results [1-6](#)

1. [cabinet-9-65021](#)
2. [cabinet-9-65022](#)
3. [cabinet-9-65065](#)
4. [cabinet-9-65459](#)
5. [cabinet-9-FB8500ST](#)
6. [cabinet-9-WLSTX400](#)

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 4-35 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 4-35 Sample Search Results for Uncategorized Items

Search for items which are not assigned to any category

- Search Results 1-6
1. [wrench-9-IR1077XP](#) X
 2. [wrench-9-CP628](#) 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 “Using the Administration Tools to Assign Items to Categories” on page 4-20.

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 “Deleting an Item from the Catalog” on page 4-32.

The steps to remove an item from a category are covered in the following procedure. Due to their similarity, the sample screens that would duplicate the ones shown in “Using the Administration Tools to Assign Items to Categories” on page 4-20 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 WebLogic Portal server is running and that the Administration Tools are started. See “Starting the Server and the Administration Tools” on page 4-2 for more information.
2. On the main administration screen, click the Catalog Management graphic.
3. On the main Catalog Management screen, click the underlined *Categories* link.
4. In the Catalog hierarchy display, click the category or subcategory from which you want to remove the item.

Note: To expand a current category and reveal its subcategories (if any), 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.

5. When the category or subcategory name is shown in the hierarchy, click its underlined link.
6. On the Editing Category: <category name> screen, click the following link: *To modify the items assigned to this category, click here.*
7. Figure 4-36 shows a sample screen. In this example, assume that we need to remove one of the pneumatic hammer items from the following category:

Root → Hardware → Storage and Cabinets → Cabinets

Figure 4-36 Removing an Item from a Category



Modify Items Assigned to Category

Search for the item you want to add or remove from this category. There are two search modes - attribute and keyword. In addition you can query for items that have not been assigned a category. Search results will be displayed in the lefthand text box.

The righthand text box displays the items currently assigned to the category. Add or remove items to the category using the central arrow keys. Note that an item can be assigned to multiple categories.

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

Search for items containing the following keywords

Keyword:

search

Search for items matching the following query

Query:

search

Search for items which are not assigned to any category

search

Search Results:

Items Assigned to Category:

(9-65021), cabinet-9-65021
(9-65022), cabinet-9-65022
(9-65065), cabinet-9-65065
(9-65459), cabinet-9-65459
(9-FB8500ST), cabinet-9-FB8500ST
(9-WLSTX400), cabinet-9-WLSTX400

Navigation arrows: left arrow, right arrow, up arrow, down arrow. A mouse cursor is hovering over the left arrow.

back

save

On the screen, the second item in the Items Assigned to Category text box has been highlighted, and the cursor is hovering over the left arrow.

8. Click the left arrow to remove the item from the category.
9. 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, repeat steps 3 through 9 to remove the item from the next category.

Removing Categories

To remove a category, find the category or subcategory via the Administration Tools 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.

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.

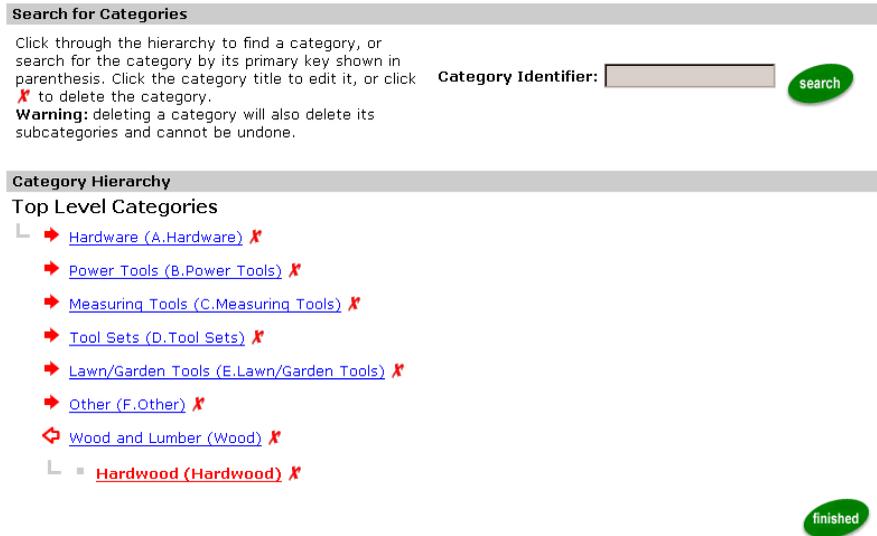
Also, product items that are associated with removed categories may be orphaned (unless they belong to another category in the hierarchy). Orphaned items are allowed to remain in the catalog, and can be reassigned later to one or more categories. The caveat here is that by deleting a category, you may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

To remove a category from the catalog's hierarchy, follow these steps:

1. Make sure the WebLogic Portal server is running and that the Administration Tools are started. See "Starting the Server and the Administration Tools" on page 4-2 for more information.
2. On the main administration screen, click the Catalog Management graphic.
3. On the main Catalog Management screen, click the underlined *Categories* button on the Categories graphic.
4. 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 is a letter of the alphabet followed by the category name.
 - Or by browsing through the hierarchy of categories and subcategories. Click the solid, red, right-facing arrows to expand a category and view its subcategories.

Figure 4-37 shows a portion of the resulting Categories screen after searching for the category identifier `hardwood`. Notice how the found category name is shown in red type.

Figure 4-37 Sample Category Search Result



5. When the category you want to remove is displayed, click the red X icon to the right of the category name. 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 deleted categories may be orphaned unless they belong to another category. Orphaned items will remain in the catalog, and can be reassigned later to one or more categories. Remember that by deleting a category, you may inadvertently remove 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 “Using the Administration Tools to Assign Items to Categories” on page 4-20.
- Remove the item from an existing category, as described in “Removing an Item from One or More Categories” on page 4-35.

Defining 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 attributes in the Product Catalog database tables.

Warning: The default Product Catalog attributes that are provided by the Commerce services are based on the 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 administrator maps new custom attributes onto the default Dublin Core attributes. For related information about the category and item caches, see “Improving Catalog Performance by Optimizing the Catalog Cache” on page 4-41.

The Commerce services use 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 run time, these properties can be accessed or modified using the standard `getProperty` or `setProperty` interfaces of `ConfigurableEntity`.

WebLogic Portal provides administrative tools that allow the custom attributes of categories and items to be edited. You can create and manage property sets using the E-Business Control Center. Select the Site Infrastructure tab in the Explorer window, then click the Catalog Structure icon. You define property sets for custom attributes of categories and items in your product catalog the same way you define property sets for other purposes. For more information, and instructions on how to accomplish this, see “Using the E-Business Control Center” in the “Creating and Managing Property Sets” topic of the *Guide to Building Personalized Applications* documentation.

You therefore have a number of options for customizing the Product Catalog:

- **Property Set Definition**

Defining Property Sets for custom attributes is a run-time 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 Commerce database 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 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 7, “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

The Commerce services provide 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

Both of these caches use the common caching framework that “Using Caches” in the *Performance Tuning Guide* describes. They are configured by MBeans, which provides the following features:

- You can use the Administration Console (or `application-config.xml`) to modify the cache configuration. Your modifications are immediately deployed, and the cache configurations persist when you restart a server.
- In a cluster, the Administration Server propagates modifications to cache configurations to all instances of the cache on all nodes.
- The flush command removes all items from all instances of the cache on all nodes in the cluster.
- The invalidate command invalidates a specific item in all instances of the cache on all nodes in the cluster.

As the administrator, your primary tasks related to the setup and maintenance of the caches are as follows:

- Enable or disable each cache. Each cache is enabled by default.
Note: BEA recommends that you always leave the CategoryCache enabled; if your catalog has highly volatile category data, use a lower time-to-live (TTL) value to ensure that the data is refreshed at appropriate intervals.
- Adjust the size of each cache. That is, the maximum number of category and item records in each cache.

- Adjust the time-to-live (TTL) for each cache. When the TTL expires, the cache invalidates the entry for the item or category. Then as new item records are accessed by customers (items retrieved from the database), a copy of each record is added to the cache.
- Remove or invalidate items in the cache.
- View percentage statistics about the hit or miss rate of each cache.
- Reset the statistics.
- Monitor the caching statistics to understand whether the values that you select initially are too low or too high. 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.

For information on how to tune, monitor, and manage caches, refer to “Configuring Caches” and “Monitoring and Managing Caches” under “Using Caches” in the *Performance Tuning Guide*.

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 uses the setting in `application-config.xml` to configure and instantiate the `CategoryCache` and `ProductItemCache`. But those values simply set the behavior and potential size of the separate caches. Each cache is initially empty.

As visitors start accessing the site's pages and requests for category or product item data are received, the Commerce services first check 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 visitor 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 visitor and displayed.

If you are interested in the architectural view of how the catalog service managers use cache, see "Catalog Architecture and Services" on page 7-3.

Using the `wlcs-catalog.properties` File

In the default Product Catalog configuration, the `wlcs-catalog.properties` file serves two purposes. The first part of the file defines values for internationalization string constants and constant configuration 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 4-44. To learn how to modify the names of tables, columns, and so on, see "Editing the Catalog Schema Definition" on page 4-46.

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 located in the `commerce_system.jar` file, which resides in the `PORTAL_HOME\lib` directory.

Note: `PORTAL_HOME` is the directory in which you installed the WebLogic Portal software.

Some Property Values You Might Modify

The `wlcs-catalog.properties` file contains over 1000 lines of name/value properties. Table 4-2 lists a subset of the properties to introduce you to the type of information contained within 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 4-2 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> columns in the database.
<code>catalog.jsp.default.iterator.size</code>	<code>10</code> Sets the default size of the <code>ViewIterators</code> created by <code>Pipeline Components</code> . 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>	<code>-1</code> Sets the maximum search results returned by the catalog. <code>-1</code> means unlimited search result size. You can dynamically change the search results by using the <code>get/set MaxSearchResults</code> methods on the <code>CatalogQuery</code> object.

4 *Catalog Administration Tasks*

Table 4-2 Summary of Values in the `wlcs-catalog.properties` File (Continued)

Property	Value and Description
<code>catalog.jsp.keyword.convert</code>	<code>lower</code> Used by the Administration pages to determine whether keywords should be case converted. Possible values are: <ul style="list-style-type: none">■ <code>lower</code>■ <code>upper</code>■ <code>none</code>
<code>catalog.category.jsp.default</code>	<code>applications/wlcsApp/wlcs/commerce/catalog/includes/category.jsp</code> Sets the default JSP file used to display a category.
<code>catalog.item.jsp.default.summary</code>	<code>applications/wlcsApp/wlcs/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>applications/wlcsApp/wlcs/commerce/catalog/includes/itemdetails.jsp</code> Sets the default JSP file used to display an item's details page.
<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.

Table 4-2 Summary of Values in the *wlcs-catalog.properties* File (Continued)

Property	Value and Description
Messages:	
The <i>wlcs-catalog.properties</i> file includes numerous properties for catalog-related messages. A few examples:	
<code>user.message.error.format.date.log</code>	= The date format is invalid. Please enter a valid date.
<code>user.message.error.duplicate.user</code>	= A category with the specified identifier already exists. Category identifiers must be unique within the catalog.
<code>user.message.search.invalid.expression.user</code>	= 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:	
<ul style="list-style-type: none">■ General messages■ Category messages■ Item messages■ Search messages■ Tag messages■ Error messages	
See the <i>wlcs-catalog.properties</i> 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 Product Catalog.

The schema and persistence definition section of the *wlcs-catalog.properties* file are referenced by the Tier 2 JDBC catalog Service Providers.

Warning: The configuration parameters listed in this section are subject to change. Take extreme care when editing the 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 Product Catalog to be deployed, bound to distinct tables or columns.

Note: For more information about multiple Product Catalog instances, see “Method 3: Multiple Product Catalog Instances” on page 8-10.

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 4-2.

Listing 4-2 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 Product Catalog Tier 2 persistence mechanism.

Notes: It is not currently possible to add or remove attributes by editing the schema file. New attributes (schema additions) are best 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

The Commerce services included in the WebLogic Portal product suite provide JavaServer Page (JSP) templates and JSP tags that implement commonly used Web-based product catalog features. The Product Catalog JSP templates allow your visitors 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 Commerce services (`wlcs`) Web application passes data about the selected product items to the order management JSPs. For information about the order processing JSPs, see the *Guide to Managing Purchases and Processing Orders*.

This topic includes the following sections:

- Introduction
 - Accessing the Product Catalog JavaServer Page (JSP) Templates
 - On Which JavaServer Page Will My Customers Start?
 - Sample Path Through the Product Catalog JSP Templates
- JavaServer Pages (JSPs)
 - `main.jsp` Template
 - `browse.jsp` Template
 - `details.jsp` Template
 - `search.jsp`
 - `searchresults.jsp`

- Query-Based Search Syntax
 - Using Comparison Operators to Construct Queries
 - Searchable Catalog Attributes
 - Controlling the Number of Search Results
- Input Processors
 - CatalogIP
 - ExpressionSearchIP
 - GetCategoryIP
 - GetProductItemIP
 - KeywordSearchIP
 - MoveAttributeIP
 - RemoveAttributeIP
- Pipeline Components
 - CatalogPC
 - GetAncestorsPC
 - GetCategoryPC
 - GetParentPC
 - GetProductItemPC
 - GetProductItemsPC
 - GetSubcategoriesPC
 - MoveAttributePC
 - PriceShoppingCartPC
 - RemoveAttributePC
 - SearchPC

Note: In this topic, the environment variable `PORTAL_HOME` is used to represent the directory in which you installed the WebLogic Portal software.

Introduction

The JavaServer Page (JSP) templates and JSP tags included in the Commerce services allow you to easily customize the presentation of the Product Catalog. Some important points about the Product Catalog JSP templates (that you should be aware of before proceeding in this topic) are as follows:

- 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 Database 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 Product Catalog integrates with the Webflow mechanism, which automatically selects the appropriate JSP for displaying a particular category or product item. The Webflow is set by elements in a centralized Webflow configuration file, as explained in the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline* 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. For more information about the Product Catalog JSP tags, see Chapter 6, “Product Catalog JSP Tag Library Reference.”

The remainder of this introduction provides you with information relevant to viewing the Product Catalog and learning how it functions. Once you understand how the Product Catalog works, you may want to use it as a starting point for building your own product catalog.

Accessing the Product Catalog JavaServer Page (JSP) Templates

The Commerce services JSP templates and all the supporting Java packages that comprise the Product Catalog are configured to run as a Web application on the WebLogic Server. Therefore, you can access the sample Product Catalog (as well as the other JSP templates that comprise the order fulfillment and customer services) by accessing the `wlcs` Web application. For more information about how to access Web applications, see the “URLs for Accessing Web Applications” and “The Reference Web Applications, Enterprise Applications, and Domain” topics in the *WebLogic Portal Architectural Overview* documentation.

On Which JavaServer Page Will My Customers Start?

Because the `wlcs` Web application utilizes the Webflow mechanism, there is some behind-the-scenes processing taking place to direct customers to the initial Product Catalog JavaServer Page (JSP). Detailed information on this processing can be located in “Understanding How Webflow Is Invoked From a URL” in the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline* documentation.

Sample Path Through the Product Catalog JSP Templates

The Product Catalog includes a number of JSP templates that span the start-to-finish experience for a Web site customer 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.

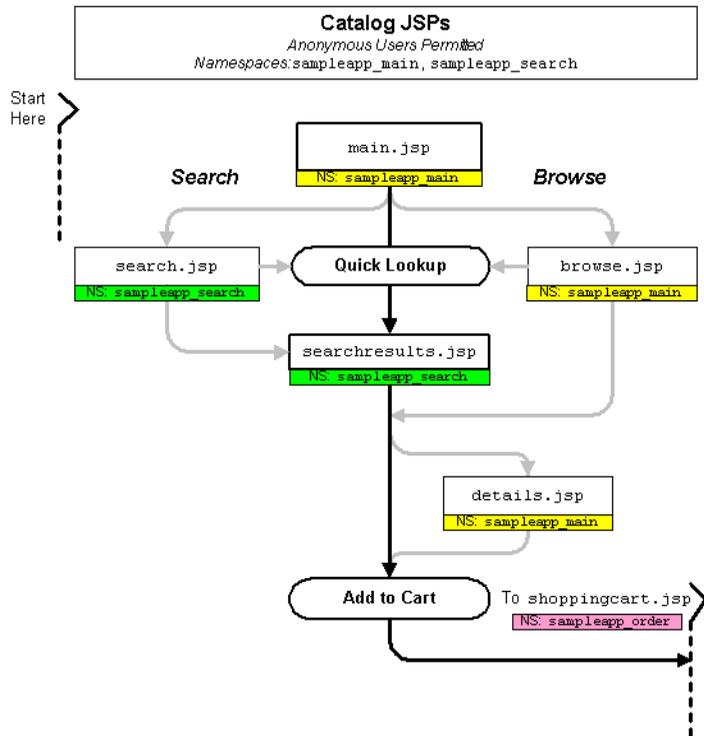
JavaServer Pages (JSPs)

The `wlcs` Web application contains a number of JavaServer Pages (JSPs) that allow your customers to view a catalog's categories and product items. You can use 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 Commerce services (`wlcs`) Web application and a listing of their locations in the directory structure, see the *Summary of JSP Templates* documentation.

Figure 5-1 illustrates the JSP templates that participate in the Product Catalog portion of the Webflow. Notice that the Product Catalog JSPs belong to the `sampleapp_main` and `sampleapp_search` Webflow namespaces.

Figure 5-1 Flow of Catalog JSP Templates



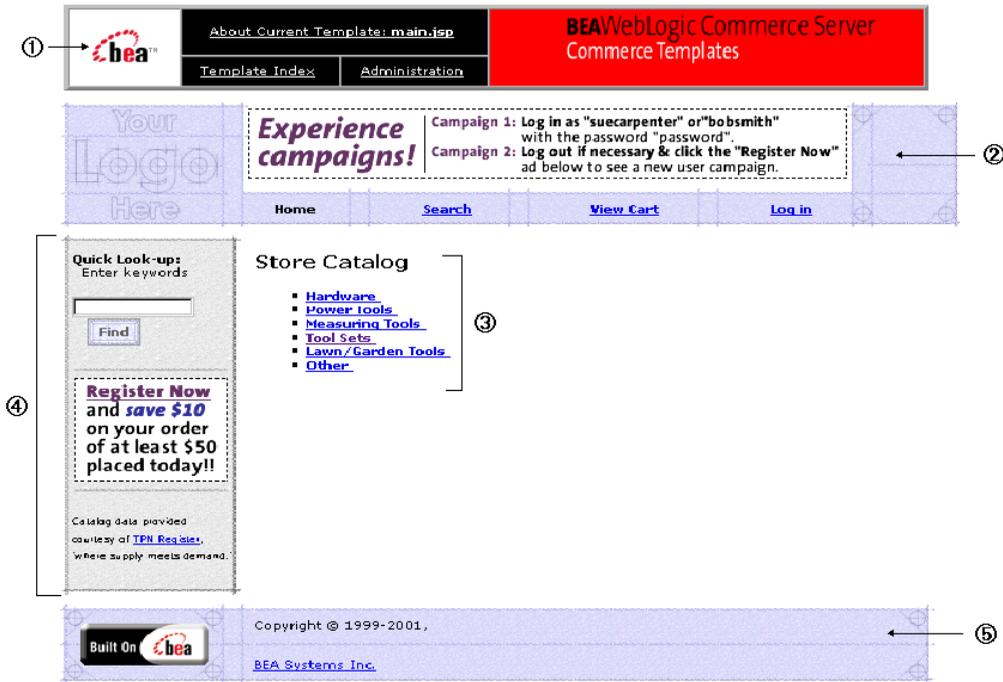
main.jsp Template

The `main.jsp` template is the default home page for the Product Catalog. Depending on whether or not customers are logged in, the `main.jsp` template displays different content in the left-side bar.

Sample Browser View

Figure 5-2 shows a version of the `main.jsp` template.

Figure 5-2 The main.jsp Template Before Customer Login



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

1. The admin banner is created from an import of the `admin.inc` template. The banner provides links to the current template's *About* information, the JSP template index, and the Administration Tools page. The import call is:

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

You should remove the `admin.inc` template from the production pages before you move them to your live server.

2. The page header is created from an import of the `header.inc` template. This is standard across most of the JSP templates included with the Commerce services. The import call is:

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

The `header.inc` file creates the top banner and reserves space for the left-side column. The contents of the left-side column are determined by other processing (see item 4 in this list).

3. The content in region 3 on the `main.jsp` template is generated by a series of Webflow 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 Catalog and Webflow JSP tags display each category name in a hyperlinked list.
4. The content shown in region 4 depends on whether the customer is logged in. (In Figure 5-2, the customer had not logged in.) The `leftside.inc` template is always included into the `main.jsp` template. The include statement is:

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

If the customer is logged in, the customer is considered a registered customer, welcomed with a greeting, and presented with links to view their profile, order history, and payment history. In addition, the customer can choose to Logout of their account. Figure 5-3 shows only the left-side column when a customer is logged in.

Figure 5-3 Left-Column of main.jsp When the Customer Is Logged In



5. The `main.jsp` template's content in region 5 contains the included `footer.inc` template. The include statement is:

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

The `footer.inc` file consists of the horizontal footer at the bottom of the page.

Location in the Directory Structure

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

```
%PORTAL_HOME%\applications\wlcsApp\wlcs\commerce\main.jsp
```

(Windows)

```
$PORTAL_HOME/applications/wlcsApps/wlcs/commerce/main.jsp (UNIX)
```

Tag Library Imports

The `main.jsp` template uses Webflow, Catalog, and the WebLogic Personalization Server's User Management, Personalization, E-Commerce Services, and Content Management JSP tags. Therefore, the template includes the following JSP tag libraries:

```
<%@ taglib uri="webflow.tld" prefix="webflow" %>
<%@ taglib uri="cat.tld" prefix="catalog" %>
<%@ taglib uri="um.tld" prefix="um" %>
<%@ taglib uri="pz.tld" prefix="pz" %>
<%@ taglib uri="es.tld" prefix="es" %>
<%@ taglib uri="cm.tld" prefix="cm" %>
```

Notes: For more information on the Webflow JSP tags, see the “Webflow JSP Tag Library Reference” in the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline* documentation. For more information about the Catalog JSP tags, see “The Catalog JSP Tag Library: cat.tld” on page 6-2. For more information on the WebLogic Personalization Server’s JSP tags, see “Personalization Server JSP Tag Library Reference” in the *Guide to Building Personalized Applications* documentation.

These files reside in the following directory for the Commerce services (wlcs) Web application:

```
%PORTAL_HOME%\applications\wlcsApp\wlcs\WEB-INF (Windows)
$PORTAL_HOME/applications/wlcsApp/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.HttpRequestConstants" %>
<%@ page import="com.beasys.commerce.webflow.PipelineSessionConstants" %>
<%@ page import="com.beasys.commerce.ebusiness.catalog.Category" %>
<%@ page import="com.beasys.commerce.ebusiness.catalog.ViewIterator" %>
<%@ page import="com.bea.pl3n.content.ContentHelper" %>
```

Location in the Default Webflow

The `main.jsp` template is the first page you or your customers will see upon starting the Commerce services (wlcs) Web application, and is part of the Webflow’s `sampleapp_main` namespace. 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 log out (which loads a different version of the `main.jsp` template), view their customer profile

(which loads the `viewprofile.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 5-1.

Included JSP Templates

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

- `admin.inc`, which creates the admin banner that provides links to the current template's *About* information, the JSP template index, and the Administration Tools page.
- `stylesheet.inc`, which is a cascading stylesheet that defines global paragraph and text styles for the site.
- `header.inc`, which creates the top banner and reserves space for the left-side column. The contents of the left-side column are determined by processing in the included `leftside.inc`.
- `leftside.inc`, which provides a keyword-based search tool for finding product items via keywords that have already been assigned, and additional links if customers are logged in.
- `footer.inc`, which creates a horizontal footer at the bottom of the page.

Events

Every time a customer clicks a link or a button on the `main.jsp` template (or those displayed by the included file `leftside.inc`), 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 Webflow response(s) they invoke.

Note: Descriptions are provided for Pipelines only.

Table 5-1 main.jsp Events

Event	Webflow Response(s)	Description
link.browse	browseCategory (IP) moveSiblingResults (IP)	--
	getBrowseDetails	Contains getCategoryPC, GetParentPC, GetSubcategoriesPC, MoveAttributePC, getCategoryPC2, GetAncestorsPC, GetSubcategoriesPC2, GetProductItemsPC and is not transactional.
link.quicksearch	keywordSearchIP	--
	keywordSearch	Contains SearchPC and is not transactional.
link.viewCustomerProfile	No business logic required. Loads viewprofile.jsp	--
link.logout	LogoutCustomerIP	--
link.viewOrderHistory	refreshOrderHistory	Contains RefreshOrderHistoryPC and is not transactional.
link.viewPaymentHistory	refreshPaymentHistory	Contains RefreshPaymentHistory and is not transactional.

Notes: For more information about individual Input Processors, see “Input Processors” on page 5-82. For more information about individual Pipeline Components, see “Pipeline Components” on page 5-89.

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). To accomplish this, the included `header.inc` file first obtains the customer’s profile, which may be null if the customer is not logged in. Listing 5-1 illustrates how a customer profile is obtained using the `getProfile` User Management JSP tag in the included `header.inc` file.

Listing 5-1 Obtaining the Customer Profile

```
<%-- Get the username --%>
<%-- if the user is not null, get the customer's profile --%>

<%
if (request.getRemoteUser() != null) {
%>

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

<% } %>
```

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

Next, the included `leftside.inc` file attempts to obtain the customer’s username from the customer profile. If the customer’s username is not equal to null, the additional links for the left column are displayed, as shown in Listing 5-2. Otherwise, just the Quick Look-up is shown.

information on the Webflow JSP tags, see the “Webflow JSP Tag Library Reference” in the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline* documentation.

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 Webflow and Catalog JSP tags.

First, the `getProperty` Webflow JSP tag obtains the `CATALOG_CATEGORY` and `CATALOG_CATEGORIES` properties from the Pipeline Session. Table 5-2 provides more detailed information on these properties.

Table 5-2 main.jsp Pipeline Session Properties

Property	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 5-3 illustrates how these properties are obtained from the Pipeline Session using the `getProperty` Webflow JSP tag.

Listing 5-3 Obtaining the `CATALOG_CATEGORY` and `CATALOG_CATEGORIES` Properties

```
<!-- Get the top category from the PipelineSession. -->

<webflow:getProperty id="topCategory"
  property="<%=PipelineSessionConstants.CATALOG_CATEGORY%>"
  type="com.beasys.commerce.ebusiness.catalog.Category"
  scope="request" namespace="sampleapp_main" />

<!-- Get the subcategories of the top category from the
PipelineSession. -->

<webflow:getProperty id="subcategories"
  property="<%=PipelineSessionConstants.CATALOG_CATEGORIES%>"
  type="com.beasys.commerce.ebusiness.catalog.ViewIterator"
  scope="request" namespace="sampleapp_main" />
```

Note: For more information on the Webflow JSP tags, see “Webflow JSP Tag Library Reference” in the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline* documentation.

Next, a string containing common browse parameters for the page is created, as shown in Listing 5-4. 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 5-4 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 + "=wlc_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, as shown in Listing 5-5.

Listing 5-5 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 + "=" +
```

```
        java.net.URLEncoder.encode(currentCategory.getKey().
        getIdentifier()); %>

    <li>
        <a href="<webflow:createWebflowURL event="link.browse"
        namespace="sampleapp_main"
        extraParams="<%=browseParameters%"/>">
        <b><catalog:getProperty object="<%= currentCategory %>"
        propertyName="Name"/></b>
        </a>

        <br>

    </catalog:iterateViewIterator>
```

Notes: For more information about the Catalog JSP tags, see “The Catalog JSP Tag Library: cat.tld” on page 6-2. For more information on the Webflow JSP tags, see “Webflow JSP Tag Library Reference” in the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline* documentation.

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 subcategories as hyperlinks. Additionally, a hyperlinked list of sibling categories is made available above the main category. An ancestor category navigation bar (such as Home → Power Tools) is also displayed at the top of the page. Figure 5-4 shows a screen shot of this `browse.jsp` template form.

Figure 5-4 `browse.jsp` Template – Power Tools Subcategories

The screenshot displays a web page layout for a product catalog. At the top left is the BEA logo. A navigation bar contains links for "About Current Template: `browse.jsp`", "Template Index", and "Administration". A red banner on the right reads "BEA WebLogic Commerce Server Commerce Templates". Below this is a promotional banner for "Storage Boxes!" with an image of a storage box. A navigation bar below the banner contains links for "Home", "Search", "View Cart", and "Log in". On the left side, there is a "Quick Look-up:" section with a search input field and a "Find" button. Below that is a "Don't Forget Extension Cords!" advertisement with a "click here" link. At the bottom left, it says "Catalog data provided courtesy of TPN". The main content area shows the breadcrumb "Home > Power Tools" and a list of "Other Categories under : Home" including "Hardware", "Power Tools", "Measuring Tools", "Tool Sets", "Lawn/Garden Tools", and "Other". Under the "Power Tools" heading, there is a list of subcategories: "Drills", "Grinders", "Sanders", "Saws", "Staplers", "Routers", and "Blades".

- 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 `browse.jsp` template, the `itemsummary.jsp` templates replace the hyperlinked

list of categories. A list of sibling categories still displays the current category (highlighted), and the ancestor category navigation bar is also shown at the top of the page. Figure 5-5 illustrates this nesting of JSP templates within the `browse.jsp` template.

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

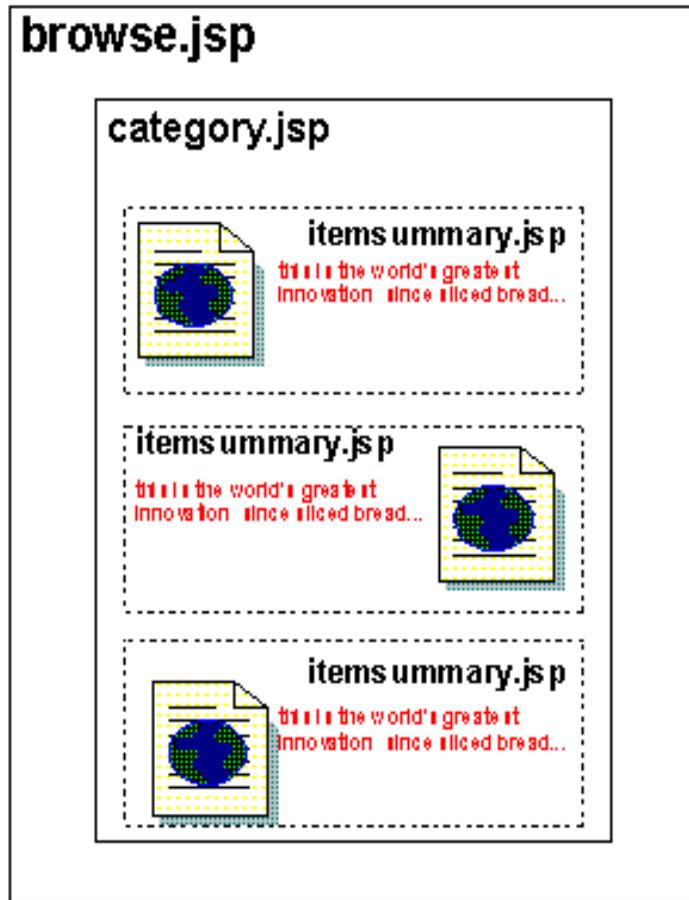


Figure 5-6 shows a screen shot of this `browse.jsp` template form, with one included `itemsummary.jsp`.

Figure 5-6 `browse.jsp` Template – Power Tools → Drills Category with Item Summary Display

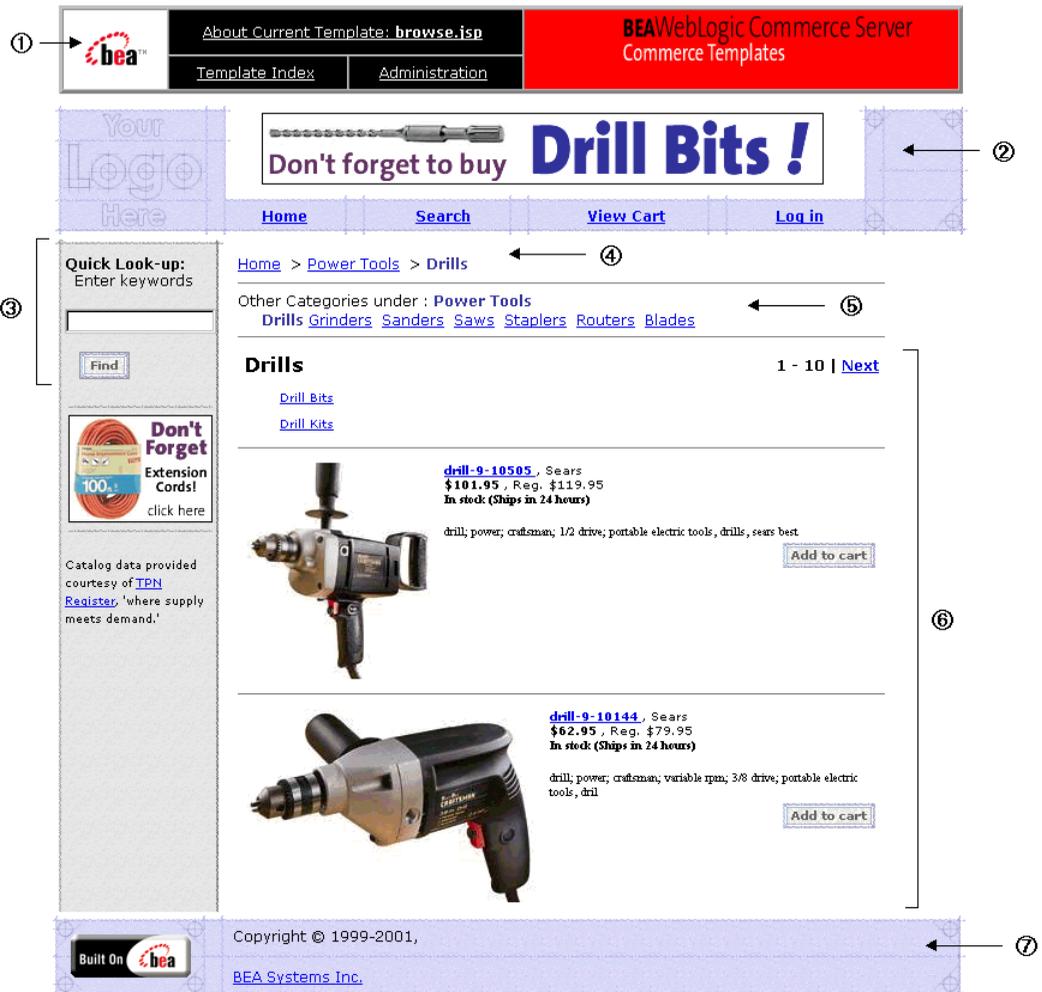
The screenshot displays a web application interface for a product catalog. At the top, there is a navigation bar with a logo on the left, a menu with 'About Current Template: browse.jsp', 'Template Index', and 'Administration', and a red banner on the right that reads 'BEA WebLogic Commerce Server Commerce Templates'. Below the navigation bar is a main banner area with a placeholder for a logo and a large advertisement for 'Drill Bits!' featuring an image of a drill bit. A secondary navigation bar contains links for 'Home', 'Search', 'View Cart', and 'Log in'. The main content area shows a breadcrumb trail 'Home > Power Tools > Drills' and a list of other categories under 'Power Tools'. The 'Drills' section is highlighted, showing a product listing for a drill with a price of \$101.95 (regular \$119.95) and an 'Add to cart' button. A sidebar on the left contains a 'Quick Look-up' search box and a 'Don't Forget Extension Cords!' advertisement.

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 “Determining How Categories and Items are Displayed to Web Site Visitors” on page 4-30.

Sample Browser View

Figure 5-7 shows a version of a `browse.jsp` template.

Figure 5-7 The browse.jsp Template



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

1. The admin banner is created from an import of the `admin.inc` template. The banner provides links to the current template's *About* information, the JSP template index, and the Administration Tools page. The import call is:

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

You should remove the `admin.inc` template from the production pages before you move them to your live server.

2. The page header is created from an import of the `header.inc` template. This is standard across most of the JSP templates provided by Commerce services. The import call is:

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

The `header.inc` file creates the top banner and reserves space for the left-side column. The contents of the left-side column are determined by other processing (see “`main.jsp` Template” on page 5-7 for more information).

3. Region 3 of the `browse.jsp` template contains the `leftside.inc` template that is included. This template provides a keyword-search feature. The include statement is:

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

4. Region 4 of the `browse.jsp` template includes `navigation.jsp`, which builds an ancestor category navigation bar 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. Region 5 of the `browse.jsp` template contains the results of processing that checks for sibling categories and displays them with hyperlinks to those categories.
6. Region 6 of the `browse.jsp` template shows the generated results of processing with the Webflow mechanism, which located 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 included `itemsummary.jsp` templates) for each product item in this category.
7. Region 7 of the `browse.jsp` template contains the included `footer.inc` template. The include statement is:

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

The `footer.inc` file consists of the horizontal footer at the bottom of the page.

Location in the Directory Structure

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

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

Tag Library Imports

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

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

Notes: For more information on the Webflow JSP tags, see “Webflow JSP Tag Library Reference” in the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline* documentation. For more information about the Catalog JSP tags, see “The Catalog JSP Tag Library: `cat.tld`” on page 6-2.

These files reside in the following directory for the Commerce services (`wlcs`) Web application:

```
%PORTAL_HOME%\applications\wlcsApp\wlcs\WEB-INF (Windows)  
$PORTAL_HOME/applications/wlcsApp/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.ebusiness.catalog.*" %>  
<%@ page import="com.beasys.commerce.webflow.HttpRequestConstants" %>  
<%@ page import="com.beasys.commerce.webflow.PipelineSessionConstants" %>
```

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 part of the `sampleapp_main` namespace, and 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 list above the category. The `browse.jsp` continues to be displayed until the customer arrives at item summaries (displayed 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 log out (which loads the generic version of the `main.jsp` template), view their customer profile (which loads the `viewprofile.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 5-1.

Included JSP Templates

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

- `admin.inc`, which creates the admin banner that provides links to the current template's *About* information, the JSP template index, and the Administration Tools page.
- `stylesheet.inc`, which is a cascading stylesheet that defines global paragraph and text styles for the site.
- `header.inc`, which creates the top banner and reserves space for the left-side column. The contents of the left-side column are determined by processing in the included `leftside.inc`.
- `leftside.inc`, which provides a keyword-based search tool for finding product items via keywords that have already been assigned, and additional links if customers are logged in.

- `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 5-26.
- `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 5-29. For more details about the `itemsummary.jsp` template, see “About the Included `itemsummary.jsp` Template” on page 5-35.
- `footer.inc`, which creates a horizontal footer at the bottom of the page.

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 Webflow, Catalog, and the WebLogic Personalization Server's Utility JSP tags to generate this content.

First, the `getProperty` Webflow JSP tag is used to obtain the current category (that is, the `CATALOG_CATEGORY` property) from the Pipeline Session, as shown in Listing 5-6.

Listing 5-6 Obtaining the `CATALOG_CATEGORY` Property

```
<webflow:getProperty id="category"
property="<%=PipelineSessionConstants.CATALOG_CATEGORY%>"
type="com.beasys.commerce.ebusiness.catalog.Category"
scope="request" namespace="sampleapp_main" />
```

Note: For more information on the Webflow JSP tags, see “Webflow JSP Tag Library Reference” in the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline* documentation.

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 `getProperty` Webflow JSP tag).

Listing 5-7 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 --%>

<webflow:getProperty id="ancestors"
  property="<%=PipelineSessionConstants.CATALOG_ANCESTORS%>"
  type="com.beasys.commerce.ebusiness.catalog.Category[]"
  scope="request" namespace="sampleapp_main" />
```

Notes: For more information on the WebLogic Personalization Server's Utility JSP tags, see "Personalization Server JSP Tag Library Reference" in the *Guide to Building Personalized Applications* documentation. For more information on the Webflow JSP tags, see "Webflow JSP Tag Library Reference" in the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline* 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 5-8.

Listing 5-8 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="<webflow:createWebflowURL event="link.home"
        namespace="sampleapp_main"/>">Home</a>

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

<% } else {
    String xtraParams = commonParameters +
        HttpRequestConstants.CATALOG_CATEGORY_ID + "=" +
        java.net.URLEncoder.encode(ancestor.getKey().getIdentifier());
    %>

    <a href="<webflow:createWebflowURL event="link.browse"
        namespace="sampleapp_main" extraParams="<%= xtraParams %>"
        true/>">

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

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

</es:forEachInArray>

<%-- Insert the category name --%>

<b><font color="#333399">
<catalog:getProperty object="<%= category %>"
    propertyName="Name"/></font></b></p>
```

```
</es:notNull>
```

Notes: For more information on the WebLogic Personalization Server's Utility JSP tags, see "Personalization Server JSP Tag Library Reference" in the *Guide to Building Personalized Applications* documentation. For more information on the Webflow JSP tags, see "Webflow JSP Tag Library Reference" in the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline* documentation. For more information about the Catalog JSP tags, see "The Catalog JSP Tag Library: cat.tld" on page 6-2.

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 Commerce services (wlcS) 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 Webflow 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 `getProperty` Webflow JSP tag, as shown in Listing 5-9.

Listing 5-9 Obtaining the CATALOG_CATEGORY and CATALOG_CATEGORIES Properties

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

<webflow:getProperty id="category"
  property="<%=PipelineSessionConstants.CATALOG_CATEGORY%>"
  type="com.beasys.commerce.ebusiness.catalog.Category"
  scope="request" namespace="sampleapp_main" />

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

<webflow:getProperty id="subcategories"
  property="<%=PipelineSessionConstants.CATALOG_CATEGORIES%>"
  type="com.beasys.commerce.ebusiness.catalog.ViewIterator"
  scope="request" namespace="sampleapp_main" />
```

Note: For more information on the Webflow JSP tags, see “Webflow JSP Tag Library Reference” in the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline* documentation.

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. If there are any individual items at this level in the hierarchy, the `category.jsp` template retrieves them using the `getProperty` Webflow JSP tag, and then displays the current category name using the `getProperty` Catalog JSP tag, as shown in Listing 5-10.

Listing 5-10 Generating Browse Parameters, Obtaining Product Items, and Inserting the Category Name

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

<webflow:getProperty id="items"
    property="<%=PipelineSessionConstants.CATALOG_ITEMS%>"
    type="com.beasys.commerce.ebusiness.catalog.ViewIterator"
    scope="request" namespace="sampleapp_main" />

<%-- Insert the category name --%>

<table border="0" width="90%" cellpadding="3">
<tr>
    <td align="left">
        <p class="head1"><catalog:getProperty
            object="<%= category %>" propertyName="Name" /></p>
    </td>
```

Notes: For more information on the Webflow JSP tags, see “Webflow JSP Tag Library Reference” in the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline* documentation. For more information about the Catalog JSP tags, see “The Catalog JSP Tag Library: cat.tld” on page 6-2.

Following display of the category name, the `category.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 5-11.

Listing 5-11 Setting the View

```
<%  
  
// Goto the correct view within the ViewIterator  
  
String viewIndexString = (String)request.getParameter  
    (HttpRequestConstants.CATALOG_VIEW_INDEX);  
  
if (viewIndexString == null) { viewIndexString = "0"; }  
    int viewIndex =  
        Math.min(Integer.valueOf(viewIndexString).intValue(),  
            items.getViewCount() - 1);  
  
if (viewIndex > 0) { items.gotoViewAt(viewIndex); }  
    String myParams = null;  
  
%>
```

The `category.jsp` template now counts the number of `itemsummary.jsp` files used (the number of items), and inserts a numeric range of items across from the category name. For example, if there are nine items on the page, the `category.jsp` template inserts 1 - 9 to indicate that the page contains items one through nine. If there are enough items to create more than one page (for example, more than 10 items), the template inserts Previous and/or Next hyperlinks to navigate to the previous or next page of items, depending on where the visitor is. This is shown in Listing 5-12.

Listing 5-12 Setting the Number of Items, Previous and Next Links

```
<%-- Add "Next" and "Previous" top navigation --%>

<% if (items.size() > 0) { %>

    <td align="right">

        <%-- Add "Previous" link --%>

        <% if (items.hasPreviousView()) {
            myParams = commonParameters +
                HttpRequestConstants.CATALOG_CATEGORY_ID + "=" +
                java.net.URLEncoder.encode(category.getKey().
                    getIdentifier()) + "&" +
                HttpRequestConstants.CATALOG_VIEW_INDEX + "=" + (viewIndex -
                    1);
        %>

        <b><a href="<webflow:createWebflowURL event="link.browse"
            namespace="sampleapp_main" extraParams="<%= myParams %>"
            />">Previous</a> | </b>

        <% } %>

        <%-- Add current view indices --%>

        <% if (items.size() > 1) { %>
            <b><%= items.getCurrentView().getFirstIndex() %> - <%=
                items.getCurrentView().getLastIndex() %></b>
        <% } %>

        <%-- Add "Next" link --%>

        <% if (items.hasNextView()) {
            myParams = commonParameters +
                HttpRequestConstants.CATALOG_CATEGORY_ID + "=" +
                java.net.URLEncoder.encode(category.getKey().
                    getIdentifier()) + "&" +
                HttpRequestConstants.CATALOG_VIEW_INDEX + "=" + (viewIndex +
                    1);
        %>

        <b> | <a href="<webflow:createWebflowURL event="link.browse"
            namespace="sampleapp_main" extraParams="<%= myParams %>"
            />">Next</a></b>

    </td>
```

```
<% } %>
<% } %>
```

Note: For more information on the Webflow JSP tags, see “Webflow JSP Tag Library Reference” in the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline* documentation.

Any subcategories associated with the category are then displayed as hyperlinks that allow the customer to browse further. This is done using the `iterateViewIterator` Catalog JSP tag, as shown in Listing 5-13.

Listing 5-13 Displaying Hyperlinked Subcategories

```
<!-- Subcategory list -->

<!-- Iterate through all subcategories, creating a browse link for
each. -->

<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">

      <% myParams = commonParameters +
      HttpRequestConstants.CATALOG_CATEGORY_ID + "=" +
      java.net.URLEncoder.encode(subcategory.getKey().
      getIdentifier());
      %>

      <a href="<webflow:createWebflowURL event="link.browse"
      namespace="sampleapp_main" extraParams="<%= myParams %>"
      />">

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

    </p>
  </td></tr>
```

```
        </catalog:iterateViewIterator>
    </table>
</center>

<hr size="1" width="90%" align="left">
```

Notes: For more information about the Catalog JSP tags, see “The Catalog JSP Tag Library: cat.tld” on page 6-2. For more information on the Webflow JSP tags, see “Webflow JSP Tag Library Reference” in the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline* documentation.

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 `itemssummary.jsp` template to display the information related to each item, as shown in Listing 5-14.

Listing 5-14 Iterating Through and Displaying Product Item Information

```
<%-- Iterate through all items in the current view and display
summary information about each --%>

<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"/>

    <%-- Include the summary JSP --%>

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

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

Notes: For more information about the Catalog JSP tags, see “The Catalog JSP Tag Library: cat.tld” on page 6-2. For more information on the Webflow JSP tags, see “Webflow JSP Tag Library Reference” in the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline* documentation.

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 Commerce services (wlcS) 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 uses a combination of Webflow, 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` property) from the Pipeline Session using the `getProperty` Webflow JSP tag, as shown in Listing 5-15.

Listing 5-15 Obtaining Request Object Parameters and the `CATALOG_CATEGORY` Property

```
<!-- Get all required parameters from the request object --%>

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

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

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

<webflow:getProperty id="category"
property="<%=PipelineSessionConstants.CATALOG_CATEGORY%>"
```

```
type="com.beasys.commerce.ebusiness.catalog.Category"  
scope="request" namespace="sampleapp_main" />
```

Note: For more information on the Webflow JSP tags, see “Webflow JSP Tag Library Reference” in the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline* documentation.

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 5-16. The display of the item is also tracked.

Listing 5-16 Displaying and Tracking Product Item Information

```
<%-- Fire Behavior Tracking display event --%>  
  
<%-- 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">  
  " align="left">  
</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,  
      "sampleapp_main", "itemssummary.jsp", "link." + detailsLink,  
      null, HttpRequestConstants.CATALOG_ITEM_SKU + "=" +  
      productItem.getKey().getIdentifier() + "&" +  
      HttpRequestConstants.DOCUMENT_TYPE + "=" + detailsLink,  
      true, false);
```

```

    %>
</es:isNull>

<es:NotNull item="<%= category %">
    <%
        detailsUrl = WebflowJSPHelper.createWebflowURL(pageContext,
            "sampleapp_main", "itemssummary.jsp", "link." + detailsLink,
            null, HttpRequestConstants.CATALOG_ITEM_SKU + "=" +
            productItem.getKey().getIdentifier() + "&" +
            HttpRequestConstants.CATALOG_CATEGORY_ID + "=" +
            category.getKey().getIdentifier() + "&" +
            HttpRequestConstants.DOCUMENT_TYPE + "=" + detailsLink,
            true, false);
    %>
</es:NotNull>

<div class="tabletext">
<b>

<!-- we will fire off a clickProductEvent when the user clicks on
the product details to see this product -->

    <productTracking:clickProductEvent id="url" documentId="<%=
        productItem.getName() %">
        sku="<%= productItem.getKey().getIdentifier() %"> />

    <% detailsUrl = detailsUrl + "&" + url; %>

    <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"/>

```

```
<il8n:getMessage bundleName="/commerce/currency" messageName="<%=
price.getCurrency() %>"/><%=
WebflowJSPHelper.priceFormat(price.getValue()) %>

</b>, Reg.

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

<il8n:getMessage bundleName="/commerce/currency" messageName="<%=
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>

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

<br><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">
<%
String tmpParams = HttpRequestConstants.CATALOG_ITEM_SKU + "=" +
productItem.getKey().getIdentifier();
%>

<a href="<webflow:createWebflowURL event="link.add"
namespace="sampleapp_order" extraParams="<%=tmpParams%>" />">" alt="Add To
Shopping Cart" border="0" vspace="4"></a>

</div>
</td>

```

Notes: For more information about the Catalog JSP tags, see “The Catalog JSP Tag Library: cat.tld” on page 6-2. For more information on the WebLogic Personalization Server’s Utility JSP tags, see “Personalization Server JSP Tag Library Reference” in the *Guide to Building Personalized Applications* documentation. For more information on the Webflow JSP tags, see “Webflow JSP Tag Library Reference” in the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline* documentation.

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 5-3 provides information about these events and the Webflow response(s) they invoke.

Note: Descriptions are provided for Pipelines only.

Table 5-3 `browse.jsp` (and included `category.jsp`) Events

Event	Webflow Response(s)	Description
<code>link.browse</code>	<code>browseCategory (IP)</code> <code>moveSiblingResults (IP)</code>	--

Table 5-3 browse.jsp (and included category.jsp) Events

Event	Webflow Response(s)	Description
	getBrowseDetails	Contains getCategoryPC, getParentPC, GetSubcategoriesPC, MoveAttributePC, getCategoryPC2, GetAncestorsPC, GetSubcategoriesPC2, GetProductItemsPC and is not transactional.

Because the `category.jsp` template also includes the `itemsummary.jsp` template, the events shown in Table 5-4 are also considered part of the `browse.jsp` template.

Table 5-4 itemsummary.jsp Events

Event	Webflow Response(s)	Description
link.details	getProductItemDetails (IP)	--
link.add	addProductItemToShoppingCart (IP)	--

Notes: The `link.details` event for this JSP is dynamically generated.

For more information about individual Input Processors, see “Input Processors” on page 5-82. For more information about individual Pipeline Components, see “Pipeline Components” on page 5-89.

Dynamic Data Display

The primary purpose of the `browse.jsp` template is to dynamically display content based on the hyperlinked path a 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 5-29 and “About the Included `itemsummary.jsp` Template” on page 5-35.)

However, there is still some dynamic data that is handled solely by the `browse.jsp` template; that is, the list of sibling categories above the category name. To accomplish this, the `browse.jsp` template first calls the included `navigation.jsp` template to determine whether or not to display an ancestor category navigation bar, as shown in Listing 5-17.

Listing 5-17 Determining Whether to Display an Ancestor Category Navigation Bar

```
<!-- ancestor category navigation bar -->

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

Note: For detailed information about the `navigation.jsp` template, see “About the Included `navigation.jsp` Template” on page 5-26.

After the include of `navigation.jsp`, the `browse.jsp` template uses the `getProperty` Webflow JSP tag to retrieve the `CATALOG_ANCESTORS` and `wlcs_siblings` properties from the Pipeline Session. Table 5-5 provides more detailed information on these properties.

Table 5-5 `browse.jsp` Pipeline Session Properties

Property	Type	Description
<code>PipelineSessionConstants.CATALOG_ANCESTORS</code>	<code>com.beasys.commerce.ebusiness.catalog.Category</code>	Contains the ancestors a given category.
<code>wlcs_siblings</code>	<code>com.beasys.commerce.ebusiness.catalog.ViewIterator</code>	Contains the siblings for a given category.

Listing 5-18 illustrates how these properties are retrieved from the Pipeline Session using the `getProperty` Webflow JSP tag.

Listing 5-18 Obtaining the `CATALOG_ANCESTORS` and `wlcs_siblings` Properties

```
<!-- get the sibling categories -->

<hr size="1" width="90%" align="left">

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

<webflow:getProperty id="ancestors"
property="<%=PipelineSessionConstants.CATALOG_ANCESTORS%>"
```

```
type="com.beasys.commerce.ebusiness.catalog.Category[]"
scope="request" namespace="sampleapp_main" />.
.
.
<!-- Get the siblings from the PipelineSession -->
<webflow:getProperty id="siblings" property="wlcs_siblings"
type="com.beasys.commerce.ebusiness.catalog.ViewIterator"
scope="request" namespace="sampleapp_main" />
```

Note: For more information on the Webflow JSP tags, see “Webflow JSP Tag Library Reference” in the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline* documentation.

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 `getProperty` Catalog JSP tags are then used to iterate through the siblings and create browse hyperlinks for each of them (if appropriate), as shown in Listing 5-19. This activity happens prior to any calls to the included `category.jsp` and/or `itemsummary.jsp` templates.

Listing 5-19 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.
-->
<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())) { %>
  <b><font color="#333399"><catalog:getProperty object="<%=
  sibling %>" propertyName="Name"/></font></b>

<% } else { %>

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

  <%
  String extraParams = commonParameters +
  HttpServletRequestConstants.CATALOG_CATEGORY_ID + "=" +
  java.net.URLEncoder.encode(sibling.getKey().getIdentifier());
  %>

  <a href="<webflow:createWebflowURL event="link.browse"
  extraParams="<%= extraParams %>" namespace="sampleapp_main"
  />"><catalog:getProperty object="<%= sibling %>"
  propertyName="Name"/>
  </a>&nbsp;

<% } %>

</catalog:iterateViewIterator>

```

Note: For more information about the Catalog JSP tags, see “The Catalog JSP Tag Library: cat.tld” on page 6-2. For more information on the Webflow JSP tags, see “Webflow JSP Tag Library Reference” in the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline* documentation.

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. (For more information about the `itemsummary.jsp` template, see “About the Included `itemsummary.jsp` Template” on page 5-35.) When customers click this link, the browser loads the `details.jsp` template, which customer can use to view more detailed information about the item.

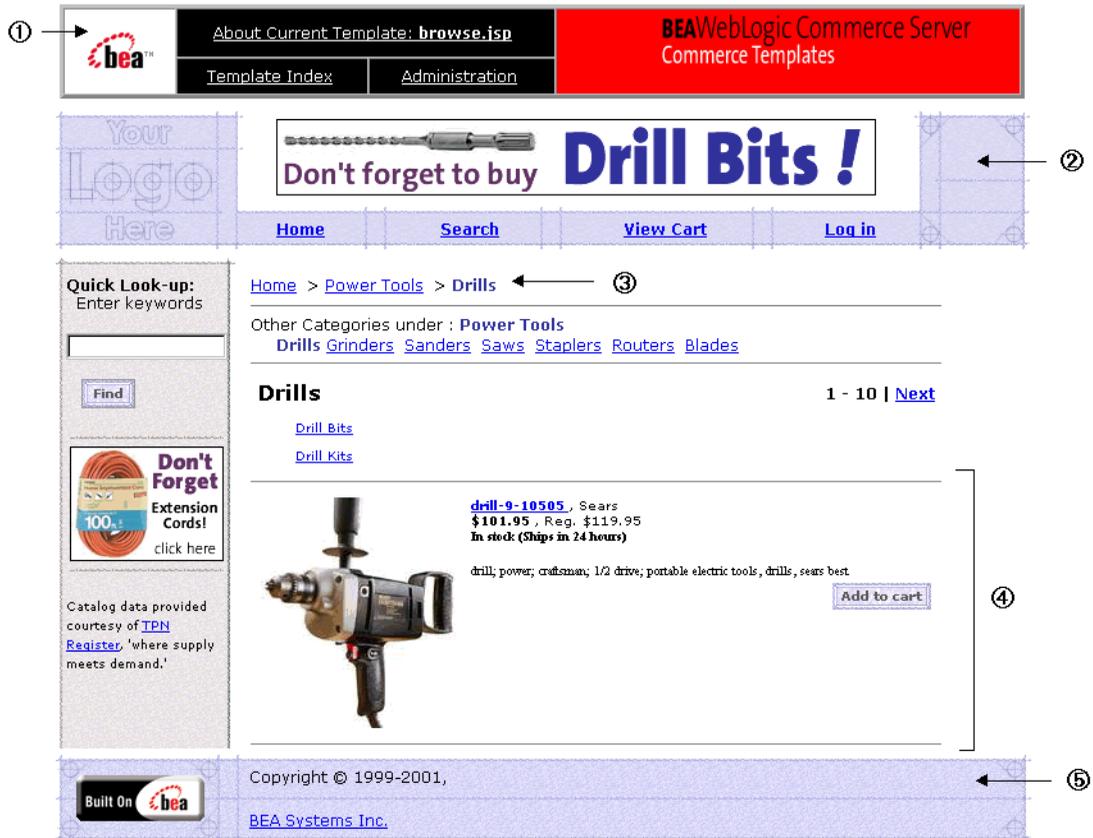
Although the Commerce services (`wlcs`) Web application presents the same information on the `itemsummary.jsp` template and the `details.jsp` template, you can use this page separation to customize or localize the content for your customers. For more information about localization and internationalization using these JSPs, see “Method 2: Parsing Language-Specific Data” on page 8-8.

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 text. The Product Catalog administrator simply needs to switch between the detailed display JSPs (once they have been tested) using the Administration Tools. For more information about the Product Catalog Administration Tools, see Chapter 4, “Catalog Administration Tasks.”

Sample Browser View

Figure 5-8 shows a version of a `details.jsp` template.

Figure 5-8 The details.jsp Template



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

1. The admin banner is created from an import of the `admin.inc` template. The banner provides links to the current template's *About* information, the JSP template index, and the Administration Tools page. The import call is:

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

You should remove the `admin.inc` template from the production pages before you move them to your live server.

2. The page header is created from an import of the `header.inc` template. This is standard across most of the JSP templates provided by Commerce services. The import call is:

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

The `header.inc` file creates the top banner and reserves space for the left-side column. The contents of the left-side column are determined by other processing (see “main.jsp Template” on page 5-7 for more information).

3. Region 3 of the `details.jsp` template is created by the included `navigation2.jsp`, which conditionally displays an ancestor category navigation bar in the form of breadcrumbs. The call to include this JSP is:

```
<jsp:include page="/commerce/catalog/includes/navigation2.jsp"
  flush="true"/>
```

4. Region 4 of the `details.jsp` template shows the results generated from processing with the Webflow mechanism, which located the detailed information for this particular product item (presented in the included `itemdetails.jsp` template).
5. Region 5 of the `details.jsp` template contains the included `footer.inc` template. The include statement is:

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

The `footer.inc` file consists of the horizontal footer at the bottom of the page.

Location in the Directory Structure

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

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

Tag Library Imports

The `details.jsp` template uses Webflow, Catalog, and Product Tracking JSP tags. Therefore, the template includes the following JSP tag libraries:

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

Notes: For more information on the Webflow JSP tags, see the “Webflow JSP Tag Library Reference” in the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline* documentation. For more information about the Catalog JSP tags, see “The Catalog JSP Tag Library: cat.tld” on page 6-2. For more information about the Product Tracking JSP Tags, see “JSP Tag Library Reference for Events and Behavior Tracking” in the *Guide to Events and Behavior Tracking* documentation.

These files reside in the following directory for the Commerce services (wlcs) Web application:

```
%PORTAL_HOME%\applications\wlcsApp\wlcs\WEB-INF (Windows)
$PORTAL_HOME/applications/wlcsApp/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.ebusiness.catalog.*" %>
<%@ page import="com.beasys.commerce.webflow.PipelineSessionConstants" %>
<%@ page import="com.bea.commerce.ebusiness.tracking.events.DisplayProductEvent"
%>
```

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). The `details.jsp` is part of the `sampleapp_main` namespace. 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 5-1.

Included JSP Templates

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

- `admin.inc`, which creates the admin banner that provides links to the current template's *About* information, the JSP template index, and the Administration Tools page.
- `stylesheet.inc`, which is a cascading stylesheet that defines global paragraph and text styles for the site.
- `header.inc`, which creates the top banner and reserves space for the left-side column. The contents of the left-side column are determined by processing in the included `leftside.inc`.
- `leftside.inc`, which provides a keyword-based search tool for finding product items via keywords that have already been assigned, and additional links if customers are logged in.
- `navigation2.jsp`, which builds an ancestor category navigation bar from the top-level Home category down to the current category (under specific conditions). 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 `navigation2.jsp` template, see “About the Included `navigation2.jsp` Template” on page 5-48.
- `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.
- `footer.inc`, which creates a horizontal footer at the bottom of the page.

About the Included `navigation2.jsp` Template

The `navigation2.jsp` template (included in the `details.jsp` template) is responsible for generating the ancestor category navigation bar that is shown at the top of the page. The `navigation2.jsp` template utilizes Webflow, Catalog, and the WebLogic Personalization Server's Utility JSP tags to generate this content.

First, the `getProperty` Webflow JSP tag is used to obtain the current category (that is, the `CATALOG_CATEGORY` property) from the Pipeline Session, as shown in Listing 5-20.

Listing 5-20 Obtaining the CATALOG_CATEGORY Property

```
<webflow:getProperty id="category"
property="<%=PipelineSessionConstants.CATALOG_CATEGORY%>"
type="com.beasys.commerce.ebusiness.catalog.Category"
scope="request" namespace="sampleapp_main" />
```

Note: For more information on the Webflow JSP tags, see the “Webflow JSP Tag Library Reference” in the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline* documentation.

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 `getProperty` Webflow JSP tag).

Listing 5-21 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 --%>

<webflow:getProperty id="ancestors"
property="<%=PipelineSessionConstants.CATALOG_ANCESTORS%>"
type="com.beasys.commerce.ebusiness.catalog.Category[]"
scope="request" namespace="sampleapp_main" />
```

Notes: For more information on the WebLogic Personalization Server’s Utility JSP tags, see “Personalization Server JSP Tag Library Reference” in the *Guide to Building Personalized Applications* documentation. For more information on

the Webflow JSP tags, see the “Webflow JSP Tag Library Reference” in the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline* documentation.

Browse parameters for the `navigation2.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 5-8.

Listing 5-22 Generating the Ancestor 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="<webflow:createWebflowURL event="link.home"
            namespace="sampleapp_main" />">Home</a>

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

    <% } else {

        theParams = commonParameters + HttpRequestConstants.CATALOG_CATEGORY_ID +
            "=" + java.net.URLEncoder.encode(ancestor.getKey().getIdentifier());

    %>
```

```

<a href="<webflow:createWebflowURL event="link.browse"
namespace="sampleapp_main" extraParams="<%= theParams %>" />">

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

<% } %>

&nbsp;<b>&gt;</b>
</es:forEachInArray>
<!-- Insert the category link -->
<%
theParams = commonParameters + HttpRequestConstants.CATALOG_CATEGORY_ID + "=" +
java.net.URLEncoder.encode(category.getKey().getIdentifier());
%>
<b>
<a href="<webflow:createWebflowURL event="link.browse"
namespace="sampleapp_main" extraParams="<%= theParams %>" />">
<catalog:getProperty object="<%= category %>" propertyName="Name"/></a></b>
</es:notNull>

```

Notes: For more information on the WebLogic Personalization Server’s Utility JSP tags, see “Personalization Server JSP Tag Library Reference” in the *Guide to Building Personalized Applications* documentation. For more information on the Webflow JSP tags, see the “Webflow JSP Tag Library Reference” in the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline* documentation. For more information about the Catalog JSP tags, see “The Catalog JSP Tag Library: cat.tld” on page 6-2.

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 Commerce services (wlcS) Web application, this format is the same as what is shown in the `itemsummary.jsp` template, 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` template, see

“About the Included `itemssummary.jsp` Template” on page 5-35.) 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 5-6 provides information about the only event for the included `itemdetails.jsp` template, and the Webflow response(s) it invokes.

Note: Descriptions are provided for Pipelines only.

Table 5-6 `itemdetails.jsp` Events

Event	Webflow Response(s)	Description
<code>link.add</code>	<code>addProductItemToShoppingCart (IP)</code>	--
	<code>addProductItemToShoppingCart</code>	Contains <code>GetProductItemPC</code> and <code>AddProductItemToShoppingCartPC</code> , and <code>PriceShoppingCartPC</code> and is not transactional.
	<code>refreshSavedList</code>	Contains <code>RefreshSavedListPC</code> and is not transactional.

Notes: For more information about individual Input Processors, see “Input Processors” on page 5-82. For more information about individual Pipeline Components, see “Pipeline Components” on page 5-89.

Dynamic Data Display

The primary purpose of the `details.jsp` template is to dynamically display content about a customer-selected product item. To accomplish this, the `details.jsp` template first calls the included `navigation2.jsp` template to determine whether or not to display an ancestor category navigation bar, as shown in Listing 5-23.

Listing 5-23 Determining Whether to Display an Ancestor Category Navigation Bar

```
<!-- breadcrumbs; ONLY if from MAIN, CUSTOMERMAIN, OR BROWSE; NOT
if from SEARCH or SEARCHRESULTS -->

<jsp:include page="/commerce/catalog/includes/navigation2.jsp"
flush="true"/>

<hr size="1" width="90%" align="left">

<!-- end conditional breadcrumbs -->
```

Note: For detailed information about the `navigation2.jsp` template, see “About the Included `navigation.jsp` Template” on page 5-26.

After the include of `navigation2.jsp`, the `details.jsp` template obtains the `CATALOG_ITEM` and `CATALOG_CATEGORY` properties from the Pipeline Session. Table 5-7 provides more detailed information on these properties.

Table 5-7 `details.jsp` Pipeline Session Properties

Property	Type	Description
<code>PipelineSessionConstants.CATALOG_ITEM</code>	<code>com.beasys.commerce.ebusiness.catalog.ProductItem</code>	Contains the product item.
<code>PipelineSessionConstants.CATALOG_CATEGORY</code>	<code>com.beasys.commerce.ebusiness.catalog.Category</code>	Contains the root category for the Product Catalog.

Listing 5-24 illustrates how these properties are obtained from the Pipeline Session using the `getProperty` Webflow 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 5-24 Obtaining the CATALOG_ITEM and CATALOG_CATEGORY Properties

```
<!-- Get the item from the PipelineSession -->
<webflow:getProperty id="item"
  property="<%=PipelineSessionConstants.CATALOG_ITEM%>"
  type="com.beasys.commerce.ebusiness.catalog.ProductItem" scope="request"
  namespace="sampleapp_main" />
<!-- Get the category from the PipelineSession -->
<webflow:getProperty id="category"
  property="<%=PipelineSessionConstants.CATALOG_CATEGORY%>"
  type="com.beasys.commerce.ebusiness.catalog.Category" scope="request"
  namespace="sampleapp_main" />
<!-- Add the required parameters for the included JSP to the request -->
<% request.setAttribute("product_item", item); %>
<% request.setAttribute("category", category); %>
```

Note: For more information on the Webflow JSP tags, see the “Webflow JSP Tag Library Reference” in the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline* documentation.

Next, the `details.jsp` template uses the Catalog `getProperty` JSP tag to set up a `JspInfo` object, which is used to include the URL for the `itemdetails.jsp` page. Then the `details.jsp` template displays the product item details (via an `include` of `itemdetails.jsp`) and records that display using the `productTracking` JSP tag, as shown in Listing 5-25.

Listing 5-25 Displaying and Tracking the Product Details

```
<catalog:getProperty object="<%= item %>"
  propertyName="Jsp"
```

```
getterArgument="<%= new
Integer(ProductItem.DETAILED_DISPLAY_JSP_INDEX) %>"
id="detailJsp"
returnType="com.beasys.commerce.ebusiness.catalog.JspInfo"/>

<!-- Include the detail JSP -->

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

<!-- once the product is displayed, fire off a displayProductEvent
-->

<productTracking:displayProductEvent documentId="<%=
item.getName() %>" documentType="<%=
DisplayProductEvent.ITEM_BROWSE %>"
sku="<%= item.getKey().getIdentifier() %>" />
```

Notes: For more information about the Catalog JSP tags, see “The Catalog JSP Tag Library: cat.tld” on page 6-2. For more information about the Product Tracking JSP Tags, see “JSP Tag Library Reference for Events and Behavior Tracking” in the *Guide to Events and Behavior Tracking* documentation.

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 customers 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.

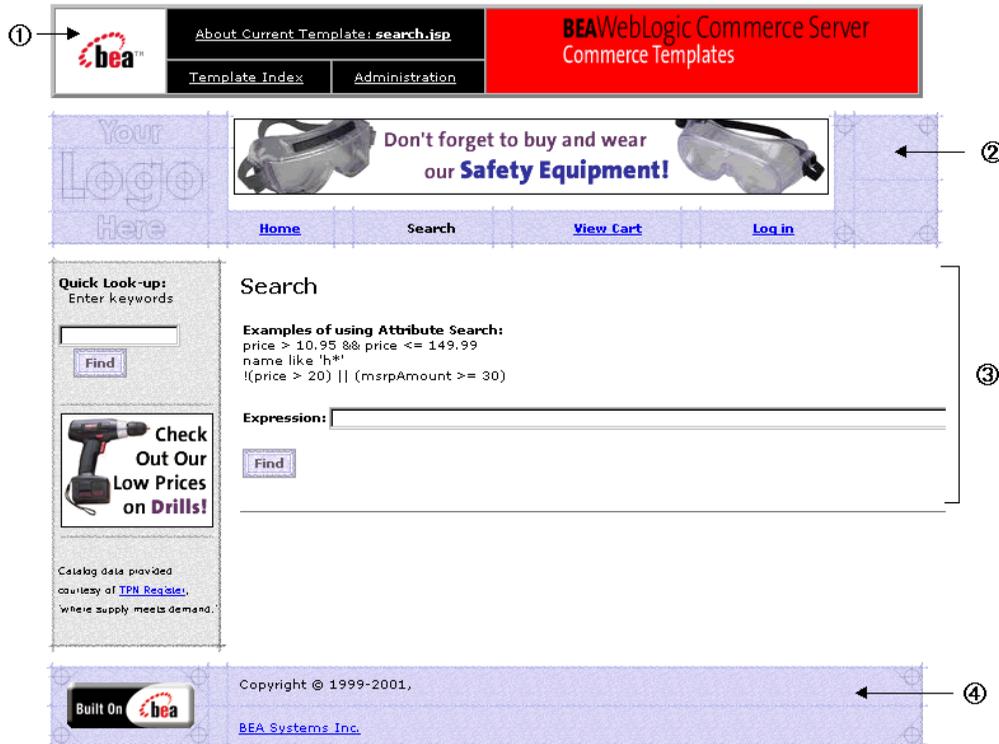
Notes: It is not expected that customers will search a Product Catalog using a free form, text-based query syntax. The free form syntax input field is provided to illustrate the power of the search functionality. You should customize the `search.jsp` template to include drop-down lists with the attributes that are appropriate for your business or product items. For example, companies selling books might have edit fields that correspond 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 5-76.

Sample Browser View

Figure 5-9 shows a version of the `search.jsp` template.

Figure 5-9 The search.jsp Template



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

1. The admin banner is created from an import of the `admin.inc` template. The banner provides links to the current template's *About* information, the JSP template index, and the Administration Tools page. The import call is:

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

You should remove the `admin.inc` template from the production pages before you move them to your live server.

2. The page header is created from an import of the `header.inc` template. This is standard across most of the JSP templates provided by Commerce services. The import call is:

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

The header .inc file creates the top banner and reserves space for the left-side column. The contents of the left-side column are determined by other processing (see “main.jsp Template” on page 5-7 for more information).

3. Region 3 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 5-10.

Figure 5-10 The search.jsp Template with Search Results

The screenshot shows a web page layout for a product catalog. At the top, there is a header with the BEA logo on the left, navigation links for 'About Current Template: search.jsp', 'Template Index', and 'Administration' in the center, and a red banner on the right that reads 'BEA WebLogic Commerce Server Commerce Templates'. Below the header is a blue banner with a 'Your Logo Here' placeholder and a central advertisement for 'Routers!' with a 'Click here to see our full line of powerful Routers!' link. Below this is a navigation bar with links for 'Home', 'Search', 'View Cart', and 'Log in'. The main content area is divided into a left sidebar and a main search area. The sidebar contains a 'Quick Look-up' section with a search input field and a 'Find' button, and an advertisement for 'See Our Large Selection of Saws Here!' with an image of a saw. The main search area has a 'Search' heading, examples of attribute search expressions, an 'Expression:' input field, and a 'Find' button. Below the search form, the 'Results of your search' section displays two products: a 'caliper-9-38739' priced at \$48.95 (regular price \$48.95) and an 'indicator-9-38561' priced at \$61.95 (regular price \$79.95). Both products are marked as 'In stock (Ships in 24 hours)' and have an 'Add to cart' button. The page number '1 - 10' and a 'Next' link are also visible.

4. Region 4 of search.jsp contains the included footer.inc template. The include call statement is:

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

The footer.inc file consists of the horizontal footer at the bottom of the page.

Location in the Directory Structure

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

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

Tag Library Imports

The `search.jsp` template uses Webflow, Catalog, and WebLogic Server Utility JSP tags. Therefore, the template includes the following JSP tag libraries:

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

Notes: For more information on the Webflow JSP tags, see the “Webflow JSP Tag Library Reference” in the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline* documentation. For more information about the Catalog JSP tags, see “The Catalog JSP Tag Library: cat.tld” on page 6-2. For more information on the WebLogic Server Utility JSP tags, see “JSP Tag Reference” in the *Guide to Building Personalized Applications* documentation.

These files reside in the following directory for the Commerce services (`wlcs`) Web application:

```
%PORTAL_HOME%\applications\wlcsApp\wlcs\WEB-INF (Windows)  
$PORTAL_HOME/applications/wlcsApp/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.HttpRequestConstants" %>  
<%@ page import="com.beasys.commerce.webflow.PipelineSessionConstants" %>  
<%@ 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, and is part of the `sampleapp_search` namespace. 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 5-1.

Included JSP Templates

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

- `admin.inc`, which creates the admin banner that provides links to the current template's *About* information, the JSP template index, and the Administration Tools page.
- `stylesheet.inc`, which is a cascading stylesheet that defines global paragraph and text styles for the site.
- `header.inc`, which creates the top banner and reserves space for the left-side column. The contents of the left-side column are determined by processing in the included `leftside.inc`.
- `leftside.inc`, which provides a keyword-based search tool for finding product items via keywords that have already been assigned, and additional links if customers are logged in.
- `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 5-35.
- `footer.inc`, which creates a horizontal footer at the bottom of the page.

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 5-8 provides information about these events and the Webflow response(s) they invoke.

Note: Descriptions are provided for Pipelines only.

Table 5-8 search.jsp Events

Event	Webflow Response(s)	Description
--	newSearch (IP)	--
	newSearch	Contains RemoveAttributePC and is not transactional.
link.search	expressionSearch (IP)	--
	expressionSearch	Contains SearchPC and is not transactional.

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 5-9 are also considered part of the search.jsp template.

Table 5-9 itemsummary.jsp Events

Event	Webflow Response(s)	Description
link.itemdetails	getProductItemDetails (IP)	--
link.add	addProductItemToShoppingCart (IP)	--

Notes: The `link.itemdetails` event for this JSP is dynamically generated.

For more information about individual Input Processors, see “Input Processors” on page 5-82. For more information about individual Pipeline Components, see “Pipeline Components” on page 5-89.

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 Webflow and Catalog JSP tags.

First, the `getProperty` Webflow JSP tag is used to obtain the `CATALOG_SEARCH_RESULTS` property from the Pipeline Session. Table 5-10 provides more detailed information on this property.

Table 5-10 `search.jsp` Pipeline Session Properties

Property	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 5-26 illustrates how this property is obtained from the Pipeline Session using the `getProperty` Webflow JSP tag.

Listing 5-26 Obtaining the `CATALOG_SEARCH_RESULTS` Property

```
<webflow:getProperty id="results"
property="<%=PipelineSessionConstants.CATALOG_SEARCH_RESULTS%"
type="com.beasys.commerce.ebusiness.catalog.ViewIterator" scope="request"
namespace="sampleapp_main" />
```

Note: For more information on the Webflow JSP tags, see the “Webflow JSP Tag Library Reference” in the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline* documentation.

Then, the `search.jsp` template sets the view. The view is 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 5-27.

Listing 5-27 Setting the View of the Search Results

```
<% if (results != null && results.size() > 0) { %>
  <!-- Goto the correct view within the ViewIterator --%>
  <% 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 5-28.

Listing 5-28 Generating the View Navigation Bar

```
<!-- Add search results 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()) {
  String myParams = HttpRequestConstants.CATALOG_VIEW_INDEX + "=" + (viewIndex
    - 1); %>

<a href="<webflow:createWebflowURL namespace="sampleapp_search"
  event="link.search" extraParams="<%=myParams%>" />">Previous</a> |

<% } %>

<!-- Add current view indicies --%>
```

5 The Product Catalog JSP Templates

```
<% if (results.size() > 1) { %>
  <%= results.getCurrentView().getFirstIndex() %> - <%=
  results.getCurrentView().getLastIndex() %>

<% } %>

<!-- Add next link -->

<% if (results.hasNextView()) {
  String myParams2 = HttpRequestConstants.CATALOG_VIEW_INDEX + "=" + (viewIndex
  + 1);
%> |

<a href="<webflow:createWebflowURL namespace="sampleapp_search"
  event="link.search" extraParams="<%=myParams2%>" />">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 5-29.

Listing 5-29 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 6-2.

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 5-11.

Table 5-11 `search.jsp` Form Fields

Parameter Name	Type	Description
<code>HttpServletRequest.CATALOG_VIEW_SIZE</code>	Hidden	Optional parameter that lets you 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 5-11 shows a version of the `searchresults.jsp` template.

Figure 5-11 The searchresults.jsp Template



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

1. The admin banner is created from an import of the `admin.inc` template. The banner provides links to the current template's *About* information, the JSP template index, and the Administration Tools page. The import call is:

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

You should remove the `admin.inc` template from the production pages before you move them to your live server.

2. The page header is created from an import of the `header.inc` template. This is standard across most of the JSP templates provided by Commerce services. The import call is:

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

The `header.inc` file creates the top banner and reserves space for the left-side column. The contents of the left-side column are determined by other processing (see step 3 below and “main.jsp Template” on page 5-7 for more information).

3. Region 3 of the `searchresults.jsp` template contains the `leftside.inc` template that is included. This template provides a keyword-search feature. The include statement is:

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

4. Region 4 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 5-19.

5. Region 5 of the `searchresults.jsp` template shows the results generated from processing with the Webflow mechanism, which located the detailed information for each resulting product item (presented in the included `itemsummary.jsp` template).

6. Region 6 of the `searchresults.jsp` template contains the included `footer.inc` template. The include statement is:

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

The `footer.inc` file consists of the horizontal footer at the bottom of the page.

Location in the Directory Structure

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

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

Tag Library Imports

The `searchresults.jsp` template uses Webflow, Catalog, and the WebLogic Personalization Server Utility JSP tags. Therefore, the template includes the following JSP tag libraries:

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

Notes: For more information on the Webflow JSP tags, see the “Webflow JSP Tag Library Reference” in the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline* documentation. For more information about the Catalog JSP tags, see “The Catalog JSP Tag Library: `cat.tld`” on page 6-2. For more information on the WebLogic Server Utility JSP tags, see “JSP Tag Reference” in the *Guide to Building Personalized Applications* documentation.

These files reside in the following directory for the Commerce services (`wlcs`) Web application:

```
%PORTAL_HOME%\applications\wlcsApp\wlcs\WEB-INF (Windows)  
$PORTAL_HOME/applications/wlcsApp/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.ebusiness.catalog.*" %>
<%@ page import="com.beasys.commerce.webflow.HttpRequestConstants" %>
<%@ page import="com.beasys.commerce.webflow.PipelineSessionConstants" %>
```

Location in the Default Webflow

The `searchresults.jsp` is located in the `sampleapp_search` namespace. 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 5-1.

Included JSP Templates

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

- `admin.inc`, which creates the admin banner that provides links to the current template's *About* information, the JSP template index, and the Administration Tools page.
- `stylesheet.inc`, which is a cascading stylesheet that defines global paragraph and text styles for the site.
- `header.inc`, which creates the top banner and reserves space for the left-side column. The contents of the left-side column are determined by processing in the included `leftside.inc`.
- `leftside.inc`, which provides a keyword-based search tool for finding product items via keywords that have already been assigned, and additional links if customers are logged in.

- `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 5-35.
- `footer.inc`, which creates a horizontal footer at the bottom of the page.

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 these events and the Webflow response(s) they invoke.

Note: Descriptions are provided for Pipelines only.

Table 5-12 `searchresults.jsp` Events

Event	Webflow Response(s)	Description
<code>link.home</code>	No business logic required. Loads <code>index.jsp</code> .	--
<code>link.quicksearch</code>	<code>keywordSearch (IP)</code>	--
	<code>keywordSearch</code>	Contains <code>SearchPC</code> and is not transactional.

Because the `searchresults.jsp` template also includes the `itemsummary.jsp` template, the events shown in Table 5-13 are also considered part of the `searchresults.jsp` template.

Table 5-13 `itemsummary.jsp` Events

Event	Webflow Response(s)	Description
<code>link.itemdetails</code>	<code>getProductItemDetails (IP)</code>	--
<code>link.add</code>	<code>addProductItemToShoppingCart (IP)</code>	--

Notes: The `link.itemdetails` event for this JSP is dynamically generated.

For more information about individual Input Processors, see “Input Processors” on page 5-82. For more information about individual Pipeline Components, see “Pipeline Components” on page 5-89.

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 Webflow and Catalog JSP tags.

First, the `getProperty` Webflow JSP tag is used to obtain the `CATALOG_QUERY` and `CATALOG_SEARCH_RESULTS` properties from the Pipeline Session. Table 5-14 provides more detailed information on these properties.

Table 5-14 `searchresults.jsp` Pipeline Session Properties

Property	Type	Description
<code>PipelineSessionConstants.CATALOG_QUERY</code>	<code>com.beasys.commerce.ebusiness.catalog.service.query.KeywordQuery</code>	Contains the customer’s search criteria.
<code>PipelineSessionConstants.CATALOG_SEARCH_RESULTS</code>	<code>com.beasys.commerce.ebusiness.catalog.ViewIterator</code>	Contains the results of the customer’s search.

Listing 5-30 illustrates how these properties are obtained from the Pipeline Session using the `getProperty` Webflow JSP tag.

Listing 5-30 Obtaining the `CATALOG_QUERY` and `CATALOG_SEARCH_RESULTS` Properties

```
<webflow:getProperty id="query"
property="<%=PipelineSessionConstants.CATALOG_QUERY%>"
type="com.beasys.commerce.ebusiness.catalog.service.query.KeywordQuery"
scope="session" namespace="sampleapp_main" />

<webflow:getProperty id="results"
property="<%=PipelineSessionConstants.CATALOG_SEARCH_RESULTS%>"
```

```
type="com.beasys.commerce.ebusiness.catalog.ViewIterator" scope="request"
namespace="sampleapp_main" />
```

Note: For more information on the Webflow JSP tags, see the “Webflow JSP Tag Library Reference” in the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline* documentation.

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 5-31.

Listing 5-31 Constructing the Top Navigation Bar

```
<% if (results != null && results.size() > 0) { %>
    <p>
        <a href="<webflow:createWebflowURL event="link.home"
            namespace="sampleapp_main" />">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">
    <% } %>
```

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

Then, the `searchresults.jsp` template sets the view. The view is 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 5-32.

Listing 5-32 Setting the View of the Search Results

```
<!-- Found items are summarized here -->
<% if (results != null && results.size() > 0) { %>
  <!-- Goto the correct view within the ViewIterator --%>
  <% 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 5-33.

Listing 5-33 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()) {
```

5 The Product Catalog JSP Templates

```
String xtraParams = HttpRequestConstants.CATALOG_VIEW_INDEX + "=" +
    (viewIndex - 1);

%>

<a href="<webflow:createWebflowURL event="link.quicksearch"
namespace="sampleapp_search" origin="quicksearch.jsp"
extraParams="<%=xtraParams%>" />">Previous</a> |

<% } %>

<!-- Add current view indicies --%>

<% if (results.size() > 1) { %>
<%= results.getCurrentView().getFirstIndex() %> -
<%=results.getCurrentView().getLastIndex() %>

<% } %>

<!-- Add next link --%>

<% if (results.hasNextView()) {

String xtraParams2 = HttpRequestConstants.CATALOG_VIEW_INDEX + "=" +
    (viewIndex + 1);

%>

| <a href="<webflow:createWebflowURL event="link.quicksearch"
namespace="sampleapp_search" origin="quicksearch.jsp"
extraParams="<%=xtraParams2%>" />">Next</a>

<% } %>

</b>

</td>

</tr><tr>

<td colspan="2" align="left" valign="top">
    <hr size="1" align="left">
</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 5-34.

Listing 5-34 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 6-2.

Form Field Specification

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

Query-Based Search Syntax

Search queries within WebLogic Portal 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 ? literally, the escape 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 5-35, the default search result size is controlled by the `catalog.searchresults.size` property in the `wlcs-catalog.properties` file. The file resides in the `PORTAL_HOME\classes` directory, where `PORTAL_HOME` is the directory in which you installed the WebLogic Portal software.

Listing 5-35 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>examples.wlcs.sampleapp.catalog.webflow.CatalogIP</code>
Description	Base InputProcessor for all Catalog-related InputProcessors. This abstract class contains global Catalog HTTP request parameter extraction and validation.
Required HttpServletRequest Parameters	None
Optional HttpServletRequest Parameters	<code>HttpRequestConstants.CATALOG_VIEW_SIZE</code>
Required Pipeline Session Properties	None
Updated Pipeline Session Properties	<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 Properties	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	<code>ProcessingException</code> , thrown if the <code>CATALOG_VIEW_SIZE</code> is invalid.

ExpressionSearchIP

Class Name	<code>examples.wlcs.sampleapp.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 HTTPServletRequest Parameters	None
Optional HTTPServletRequest Parameters	<code>HttpRequestConstants.CATALOG_SEARCH_STRING</code>
Required Pipeline Session Properties	<code>PipelineSessionConstants.CATALOG_QUERY</code> . Required only if the <code>CATALOG_SEARCH_STRING</code> parameter does not exist.
Updated Pipeline Session Properties	<code>PipelineSessionConstants.CATALOG_QUERY</code>
Removed Pipeline Session Properties	<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>ProcessingException</code>, thrown if the <code>CATALOG_SEARCH_STRING</code> parameter does not exist and the <code>PipelineSessionConstants.CATALOG_SEARCH_RESULTS</code> Pipeline Session property has expired.</p> <p><code>ProcessingException</code>, thrown if the <code>CATALOG_SEARCH_STRING</code> parameter is invalid.</p>

GetCategoryIP

Class Name	<code>examples.wlcs.sampleapp.catalog.webflow.GetCategoryIP</code>
Description	Creates a <code>CategoryKey</code> based on a category ID HTTP request parameter and adds it to the Pipeline Session. If no such parameter is supplied, the <code>CategoryKey</code> for the root category is added.
Required HttpServletRequest Parameters	<code>HttpRequestConstants.CATALOG_CATEGORY_ID</code>
Optional HttpServletRequest Parameters	None
Required Pipeline Session Properties	None
Updated Pipeline Session Properties	<code>PipelineSessionConstants.CATALOG_CATEGORY_KEY</code> (Request scope)
Removed Pipeline Session Properties	None
Validation	Validates that the <code>CATALOG_CATEGORY_ID</code> parameter is valid.
Exceptions	<code>ProcessingException</code> , thrown if the <code>CATALOG_CATEGORY_ID</code> parameter is invalid.

GetProductItemIP

Class Name	<code>examples.wlcs.sampleapp.catalog.webflow. GetProductItemIP</code>
Description	Creates a <code>ProductItemKey</code> based on a product item SKU HTTP request parameter and adds it to the Pipeline Session.
Required HttpServletRequest Parameters	<code>HttpRequestConstants.CATALOG_ITEM_SKU</code>
Optional HttpServletRequest Parameters	None
Required Pipeline Session Properties	None
Updated Pipeline Session Properties	<code>PipelineSessionConstants.CATALOG_ITEM_KEY</code> (Request scope)
Removed Pipeline Session Properties	None
Validation	Verifies that the <code>CATALOG_ITEM_SKU</code> parameter is valid.
Exceptions	<code>ProcessingException</code> , thrown if the <code>CATALOG_ITEM_SKU</code> parameter is invalid.

KeywordSearchIP

Class Name	<code>examples.wlcs.sampleapp.catalog.webflow.KeywordSearchIP</code>
Description	Creates a <code>KeywordQuery</code> based on a keyword search HTTP request parameter and adds it to the Pipeline Session.
Required HttpServletRequest Parameters	None
Optional HttpServletRequest Parameters	<code>HttpRequestConstants.CATALOG_SEARCH_STRING</code>
Required Pipeline Session Properties	<code>PipelineSessionConstants.CATALOG_QUERY</code> . Required only if the <code>CATALOG_SEARCH_STRING</code> parameter does not exist.
Updated Pipeline Session Properties	<code>PipelineSessionConstants.CATALOG_QUERY</code>
Removed Pipeline Session Properties	<code>PipelineSessionConstants.CATALOG_SEARCH_RESULTS</code> , only if the <code>CATALOG_SEARCH_STRING</code> parameter exists.
Validation	Verifies that the <code>CATALOG_SEARCH_STRING</code> parameter is valid.
Exceptions	<p><code>ProcessingException</code>, thrown if the <code>CATALOG_SEARCH_STRING</code> parameter does not exist and the <code>PipelineSessionConstants.CATALOG_SEARCH_RESULTS</code> Pipeline Session property has expired.</p> <p><code>ProcessingException</code>, thrown if the <code>CATALOG_SEARCH_RESULTS</code> parameter is invalid.</p>

MoveAttributeIP

Class Name	<code>examples.wlcs.sampleapp.catalog.webflow.MoveAttributeIP</code>
Description	Sets Pipeline Session properties when moving a Pipeline Session property value from a source property to a destination property. The source and destination properties are specified as HTTP request parameters.
Required HTTPServletRequest Parameters	<code>HttpRequestConstants.CATALOG_SOURCE_KEY</code> <code>HttpRequestConstants.CATALOG_DESTINATION_KEY</code>
Optional HTTPServletRequest Parameters	None
Required Pipeline Session Properties	None
Updated Pipeline Session Properties	<code>PipelineSessionConstants.CATALOG_SOURCE_KEY</code> (Request scope) <code>PipelineSessionConstants.CATALOG_DESTINATION_KEY</code> (Request scope)
Removed Pipeline Session Properties	None
Validation	Verifies that the <code>CATALOG_SOURCE_KEY</code> and the <code>CATALOG_DESTINATION_KEY</code> parameters are valid strings.
Exceptions	<code>ProcessingException</code> , thrown if either the source key or destination key is invalid.

RemoveAttributeIP

Class Name	<code>examples.wlcs.sampleapp.catalog.webflow.RemoveAttributeIP</code>
Description	Sets Pipeline Session properties for removing a Pipeline Session property value from a source property. The source property is specified as an HTTP request parameter.
Required HttpServletRequest Parameters	<code>HttpRequestConstants.CATALOG_SOURCE_KEY</code>
Optional HttpServletRequest Parameters	None
Required Pipeline Session Properties	None
Updated Pipeline Session Properties	<code>PipelineSessionConstants.CATALOG_SOURCE_KEY</code> (Request scope)
Removed Pipeline Session Properties	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	<code>examples.wlcs.sampleapp.catalog.pipeline.CatalogPC</code>
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 Properties	None
Updated Pipeline Session Properties	None
Removed Pipeline Session Properties	None
Type	Java object
JNDI Name	None
Exceptions	None

GetAncestorsPC

Class Name	<code>examples.wlcs.sampleapp.catalog.pipeline.GetAncestorsPC</code>
Description	Retrieves the ancestor <code>Categories</code> of the <code>Category</code> contained in the Pipeline Session <code>CATALOG_CATEGORY</code> property. The resultant <code>Category</code> array is placed into the Pipeline Session as the <code>CATALOG_ANCESTORS</code> property.
Required Pipeline Session Properties	<code>PipelineSessionConstants.CATALOG_CATEGORY</code> (Request scope)
Optional Pipeline Session Properties	None
Updated Pipeline Session Properties	<code>PipelineSessionConstants.CATALOG_ANCESTORS</code> (Request scope)
Removed Pipeline Session Properties	None
Type	Java object
JNDI Name	None
Exceptions	<code>PipelineException</code> on Catalog finder error, Catalog general error, Catalog create error, or JNDI lookup error. This <code>PipelineException</code> is fatal, as specified in the Pipeline XML configuration file.

GetCategoryPC

Note: The `GetCategoryPC2` Pipeline Component simply references the `GetCategoryPC` Pipeline Component, which is described in the following table.

Class Name	<code>examples.wlcs.sampleapp.catalog.pipeline.GetCategoryPC</code>
Description	Retrieves a <code>Category</code> based upon the <code>CategoryKey</code> in the Pipeline Session <code>CATALOG_CATEGORY_KEY</code> property. The resultant <code>Category</code> is placed into the Pipeline Session as the <code>CATALOG_CATEGORY</code> property.
Required Pipeline Session Properties	<code>PipelineSessionConstants.CATALOG_CATEGORY_KEY</code> (Request scope)
Optional Pipeline Session Properties	None
Updated Pipeline Session Properties	<code>PipelineSessionConstants.CATALOG_CATEGORY</code> (Request scope)
Removed Pipeline Session Properties	None
Type	Java object
JNDI Name	None
Exceptions	<code>PipelineException</code> on Catalog finder error, Catalog general error, EJB create error, or JNDI lookup error. This <code>PipelineException</code> is fatal, as specified in the Pipeline XML configuration file.

GetParentPC

Class Name	<code>examples.wlcs.sampleapp.catalog.pipeline.GetParentPC</code>
Description	Retrieves the parent <code>Category</code> of the category contained in the Pipeline Session <code>CATALOG_CATEGORY</code> property. The resultant <code>Category</code> is placed into the Pipeline Session as the <code>CATALOG_CATEGORY</code> property.
Required Pipeline Session Properties	<code>PipelineSessionConstants.CATALOG_CATEGORY</code> (Request scope)
Optional Pipeline Session Properties	None
Updated Pipeline Session Properties	<code>PipelineSessionConstants.CATALOG_CATEGORY</code> (Request scope)
Removed Pipeline Session Properties	None
Type	Java object
JNDI Name	None
Exceptions	<code>PipelineException</code> on Catalog finder error, Catalog general error, Catalog create error, or JNDI lookup error. This <code>PipelineException</code> is fatal, as specified in the Pipeline XML configuration file.

GetProductItemPC

Class Name	<code>examples.wlcs.sampleapp.catalog.pipeline. GetProductItemPC</code>
Description	Retrieves a <code>ProductItem</code> based upon the <code>ProductItemKey</code> contained in the Pipeline Session <code>CATALOG_ITEM_KEY</code> property. The resultant <code>ProductItem</code> is placed into the Pipeline Session as the <code>CATALOG_ITEM</code> property.
Required Pipeline Session Properties	<code>PipelineSessionConstants.CATALOG_ITEM_KEY</code> (Request scope)
Optional Pipeline Session Properties	None
Updated Pipeline Session Properties	<code>PipelineSessionConstants.CATALOG_ITEM</code> (Request scope)
Removed Pipeline Session Properties	None
Type	Java object
JNDI Name	None
Exceptions	<code>PipelineException</code> on Catalog finder error, Catalog general error, Catalog create error, or JNDI lookup error. This <code>PipelineException</code> is fatal, as specified in the Pipeline XML configuration file.

GetProductItemsPC

Class Name	<code>examples.wlcs.sampleapp.catalog.pipeline. GetProductItemsPC</code>
Description	Retrieves the <code>ProductItems</code> associated with the <code>Category</code> contained in the Pipeline Session <code>CATALOG_CATEGORY</code> property. The resultant <code>ViewIterator</code> of <code>ProductItems</code> is placed into the Pipeline Session as the <code>CATALOG_ITEMS</code> property. The view size of the resultant <code>ViewIterator</code> may be specified with the optional <code>CATALOG_VIEW_SIZE</code> Pipeline Session property.
Required Pipeline Session Properties	<code>PipelineSessionConstants.CATALOG_CATEGORY</code> (Request scope)
Optional Pipeline Session Properties	<code>PipelineSessionConstants.CATALOG_VIEW_SIZE</code> (Request scope)
Updated Pipeline Session Properties	<code>PipelineSessionConstants.CATALOG_ITEMS</code> (Request scope)
Removed Pipeline Session Properties	None
Type	Java object
JNDI Name	None
Exceptions	<code>PipelineException</code> on Catalog finder error, Catalog general error, Catalog create error, or JNDI lookup error. This <code>PipelineException</code> is fatal, as specified in the Pipeline XML configuration file.

GetSubcategoriesPC

Note: The `GetSubcategoriesPC2` Pipeline Component simply references the `GetSubcategoriesPC` Pipeline Component, which is described in the following table.

Class Name	<code>examples.wlcs.sampleapp.catalog.pipeline. GetSubcategoriesPC</code>
Description	Retrieves the subcategories of the <code>Category</code> in the Pipeline Session <code>CATALOG_CATEGORY</code> property. The resultant <code>ViewIterator</code> of <code>Categories</code> is placed into the Pipeline Session as the <code>CATALOG_CATEGORIES</code> property. The view size of the resultant <code>ViewIterator</code> may be specified with the optional <code>CATALOG_VIEW_SIZE</code> Pipeline Session property.
Required Pipeline Session Properties	<code>PipelineSessionConstants.CATALOG_CATEGORY</code> (Request scope)
Optional Pipeline Session Properties	<code>PipelineSessionConstants.CATALOG_VIEW_SIZE</code> (Request scope)
Updated Pipeline Session Properties	<code>PipelineSessionConstants.CATALOG_CATEGORIES</code> (Request scope)
Removed Pipeline Session Properties	None
Type	Java object
JNDI Name	None
Exceptions	<code>PipelineException</code> on Catalog finder error, Catalog general error, EJB create error, or JNDI lookup error. This <code>PipelineException</code> is fatal, as specified in the Pipeline XML configuration file.

MoveAttributePC

Class Name	<code>examples.wlcs.sampleapp.catalog.pipeline.MoveAttributePC</code>
Description	Moves a Pipeline Session property value from a source property (attribute) to a destination property. The source and destination property are specified by the <code>CATALOG_SOURCE_KEY</code> and <code>CATALOG_DESTINATION_KEY</code> Pipeline Session property.
Required Pipeline Session Properties	<code>PipelineSessionConstants.CATALOG_SOURCE_KEY</code> (Request scope) <code>PipelineSessionConstants.CATALOG_DESTINATION_KEY</code> (Request scope)
Optional Pipeline Session Properties	None
Updated Pipeline Session Properties	Both request-scoped and session-scoped Pipeline Session properties keyed by the <code>CATALOG_DESTINATION_KEY</code> Pipeline Session property.
Removed Pipeline Session Properties	Both request-scoped and session-scoped Pipeline Session properties keyed by the <code>CATALOG_SOURCE_KEY</code> Pipeline Session property.
Type	Java object
JNDI Name	None
Exceptions	None

PriceShoppingCartPC

Class Name	<code>examples.wlcs.sampleapp.shoppingcart.pipeline.PriceShoppingCartPC</code>
Description	Invokes the Pricing Service to price each of the shopping cart lines and compute the subtotal.
Required Pipeline Session Properties	<code>PipelineSessionConstants.SHOPPING_CART</code>
Optional Pipeline Session Properties	<code>PipelineSessionConstants.USER_NAME</code>
Updated Pipeline Session Properties	None
Removed Pipeline Session Properties	None
Type	Java object
JNDI Name	None
Exceptions	<code>PipelineException</code> on price service JNDI lookup error or remote error. This <code>PipelineException</code> is fatal, as specified in the Pipeline XML configuration file.

RemoveAttributePC

Class Name	<code>examples.wlcs.sampleapp.catalog.pipeline.RemoveAttributePC</code>
Description	Removes a Pipeline Session property (attribute) value from a source property. The source property is specified by the CATALOG_SOURCE_KEY Pipeline Session properties.
Required Pipeline Session Properties	<code>PipelineSessionConstants.CATALOG_SOURCE_KEY</code> (Request scope)
Optional Pipeline Session Properties	None
Updated Pipeline Session Properties	None
Removed Pipeline Session Properties	Both request-scoped and session-scoped Pipeline Session properties keyed by the CATALOG_SOURCE_KEY Pipeline Session property.
Type	Java object
JNDI Name	None
Exceptions	None

SearchPC

Class Name	<code>examples.wlcs.sampleapp.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>PipelineException</code> on Catalog finder error, Catalog general error, EJB create error, or JNDI lookup error. This <code>PipelineException</code> is fatal, as specified in the Pipeline XML configuration file.

6 Product Catalog JSP Tag Library Reference

The Commerce services included in the WebLogic Portal product suite provide 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.

This topic includes the following sections:

- Introduction
- The Catalog JSP Tag Library: cat.tld
 - <catalog:getProperty>
 - <catalog:iterateViewIterator>
 - <catalog:iterateThroughView>
- The E-Business JSP Tag Library: eb.tld
 - <eb:smnav>

Note: In this topic, the environment variable `PORTAL_HOME` is used to represent the directory in which you installed the WebLogic Portal software.

Introduction

The JSP templates and JSP tags included in the Commerce services 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 Database 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 Product Catalog also integrates with the Webflow mechanism, which automatically selects the appropriate JSP for displaying a particular category or product item. The Webflow is set by elements in a centralized Webflow configuration file, as explained in the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline* 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.

The Catalog JSP Tag Library: `cat.tld`

Table 6-1 summarizes the tags that comprise the 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" %>
```

Table 6-1 Catalog 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> .

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 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. However, 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: `/>`.

Subsequent sections in this topic describe the tags in more detail.

Note: In the following tables, the Required column specifies if the attribute is required (yes) or optional (no). In the R/C column, C means that the attribute is a Compile time expression, and R means that the attribute can be either a Request time expression or a Compile time expression.

<catalog:getProperty>

Use the `<catalog:getProperty>` tag (Table 6-2) 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.

Table 6-2 `<catalog:getProperty>`

Tag Attribute	Required	Type	Description	R/C
<code>getterArgument</code>	No	String	Denotes a reference to an object supplied as an argument to an explicit property getter method. May also be used to obtain implicit or custom properties that are defined using the property set framework, in which case the <code>getterArgument</code> would be the scope name for the property set (see second example below). The object must be presented in the form <code><%= getterArgumentReference %></code> and must be a run-time expression.	R
<code>id</code>	No	String	<code>id="newInstance"</code> If the <code>id</code> attribute is supplied, the value of the retrieved property will be available in the variable name to which <code>id</code> is assigned. Otherwise, the value of the property is inlined.	C
<code>object</code>	Yes	Catalog item	Denotes a reference to a <code>ProductItem</code> or <code>Category</code> object that must be presented in the form <code><%= objectReference %></code> .	R
<code>propertyName</code>	Yes	String	<code>propertyName="propertyName"</code> Name of the property to retrieve. If the property is explicit, it may be one of the values shown in Table 6-3.	C

Table 6-2 <catalog:getProperty> (Continued)

Tag Attribute	Required	Type	Description	R/C
returnType	No	String	returnType="returnType" If the id attribute is supplied, declares the type of the variable specified by the id attribute.	C

Table 6-3 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)
"jsp"	Category
"availability currentPrice format jsp msrp shippingCode taxCode type visible"	ProductItem

Example 1

This example retrieves the detail JSP information from an existing ProductItem:

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

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

Example 2

The following example shows how to use the `getterArgument` attribute to obtain an implicit or custom property for a property set/schema with the following characteristics:

- Name: `MyCatalog`
- `PropertyName`: `color`

Note: Because the `getterArgument` must be a run-time expression, we assign `MyCatalog` to a `String` variable and use the variable as the value to the `getterArgument`.

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

<%
String myPropertyName = "MyCatalog";
ProductItem myProductItem = .....; // reference to a ProductItem
%>
<catalog:getProperty
  object="<%=myProductItem%">
  propertyName="color"
  getterArgument="<%=myPropertyName%">
/>
```

<catalog:iterateViewIterator>

Use the `<catalog:iterateViewIterator>` tag (Table 6-4) 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.

Table 6-4 <catalog:iterateViewIterator>

Tag Attribute	Required	Type	Description	R/C
id	Yes	String	id="newInstance" The value of the current iterated object will be available in the variable name to which the id is assigned.	C
iterator	Yes	ViewIterator	Denotes a reference to a ViewIterator object. Must be presented in the form <%= iteratorReference %>.	R
iterateByView	No	String	iterateByView="{true false}" Specifies whether to iterate the ViewIterator by View or by Catalog item. If not specified, the ViewIterator will be iterated by Catalog item.	C
returnType	No	String	returnType="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.	C

Example 1

The following example displays the keys of all Categories in a ViewIterator:

```
<%@ taglib uri="cat.tld" prefix="catalog" %>
<catalog:iterateViewIterator
  iterator="<%= myIterator %>"
  id="category"
  returnType="com.beasys.commerce.ebusiness.catalog.Category">
  <%= category.getKey().toString() %>
</catalog:iterateViewIterator>
```

Example 2

The following example displays all the `Views` contained within a `ViewIterator`:

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

<catalog:iterateViewIterator
    iterator="<%= myIterator %>"
    id="view"
    returnType="com.beasys.commerce.ebusiness.catalog.ViewIterator"
    iterateByView="true">
    <%= view.toString() %>
</catalog:iterateViewIterator>
```

<catalog:iterateThroughView>

The `<catalog:iterateThroughView>` tag (Table 6-5) iterates 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.

Table 6-5 `<catalog:iterateThroughView>`

Tag Attribute	Required	Type	Description	R/C
<code>id</code>	Yes	String	<code>id="newInstance"</code> The value of the current iterated object will be available in the variable name to which the <code>id</code> is assigned.	C
<code>iterator</code>	Yes	<code>ViewIterator</code>	Denotes a reference to a <code>ViewIterator</code> object that must be presented in the form <code><%= iteratorReference %></code>	R
<code>returnType</code>	No	String	<code>returnType="returnType"</code> Declares the type of the variable specified by the <code>id</code> attribute. Defaults to <code>java.lang.Object</code> .	C

Table 6-5 <catalog:iterateThroughView> (Continued)

Tag Attribute	Required	Type	Description	R/C
viewIndex	No	Integer	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 %>.	R

Example 1

The following example displays the keys of all the ProductItems contained in the current View of a specified ViewIterator:

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

<catalog:iterateThroughView
  iterator="<%= myIterator %>"
  id="item"
  returnType="com.beasys.commerce.ebusiness.catalog.ProductItem">
<%= item.getKey().toString() %>
</catalog:iterateThroughView>
```

Example 2

The following example displays the keys of all the ProductItems contained in the first View of a specified ViewIterator:

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

<catalog:iterateThroughView
  iterator="<%= myIterator %>"
  id="item"
  returnType="com.beasys.commerce.ebusiness.catalog.ProductItem"
  viewIndex="new Integer(0)">
  <%= item.getKey().toString() %>
</catalog:iterateThroughView>
```

The E-Business JSP Tag Library: eb.tld

Table 6-6 summarizes the tags that comprise the E-Business JSP Tag Library. To use the functionality provided by an e-business tag, you must import the `eb.tld` tag library into your JSP file, as follows:

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

Table 6-6 E-Business JSP Tag Library

Tag	Description
<code><eb:smnav></code>	Controls the presentation of elements in the list of value objects that are being viewed, and provides links to the previous and next pages.

`<eb:smnav>`

A Scrollable Model is used to retrieve value objects so that only what is viewed is retrieved. The `<eb:smnav>` tag (Table 6-7) allows you to control the presentation of elements in the list of value objects that are being viewed, and provides links to the previous and next pages.

The `<eb:preface` stands for e-business. The Scrollable Model can be use throughout the e-business package to iterate through a list of objects. It can be used in conjunction with transaction, shopping cart, order history, or shipping services.

This tag relies on a Pipeline Session containing a ScrollableModel object on the `PipelineSessionConstants.SCROLLABLE_MODEL` key.

Table 6-7 `<eb:smnav>`

Tag Attribute	Required	Type	Description	R/C
event	No	String	The name of the link configurable in the Webflow as the visitor clicks on Next or Previous.	C

Table 6-7 <eb:smnav> (Continued)

Tag Attribute	Required	Type	Description	R/C
nextstring	No	String	The localized name for Next. Could be as simple as ">".	C
origin	No	String	The current JSP page.	C
pageindex	No	String	The index of the page to display.	R
prevstring	No	String	The localized name for Previous. Could be as simple as "<".	C

Example

The `orderhistory.jsp` that is part of the Commerce services JSP templates allows a visitor to browse page by page over the set of orders placed. Only 10 orders are displayed at a time. To go to the next or to the previous page, the visitor clicks on the "Next" or "Previous" hyperlinks shown by the tag. In this example, if the visitor has 40 orders and is viewing the second page, the tag will be displayed as "Previous | 20-29 | Next".

```
<%@ taglib uri="eb.tld" prefix="eb" %>
.
.
.
<!-- Show the Previous / 10-19 / Next navigation string -->
<eb:smnav origin="orderhistory.jsp" event="link.viewOrderHistory"
prevstring="Previous" nextstring="Next"
pageindex="<%=pageIndexString%" />
```


7 Using the API to Extend the Product Catalog

This topic describes the various options available for extending, customizing, or writing third-party integrations for the 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 that defines an electronic product catalog that uses JDBC as a persistence mechanism.

Note: The descriptions in this topic assume that you are an experienced Java/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
- Integrating Services with the Catalog Cache
- Writing Your Own Catalog Service

Note: In this chapter, the environment variable `PORTAL_HOME` is used to represent the directory in which you installed the WebLogic Portal 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 package contains several subpackages 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 Product Catalog architecture is divided into five functional areas, each of which requires an implementation of an associated Product Catalog server interface. The five services are:

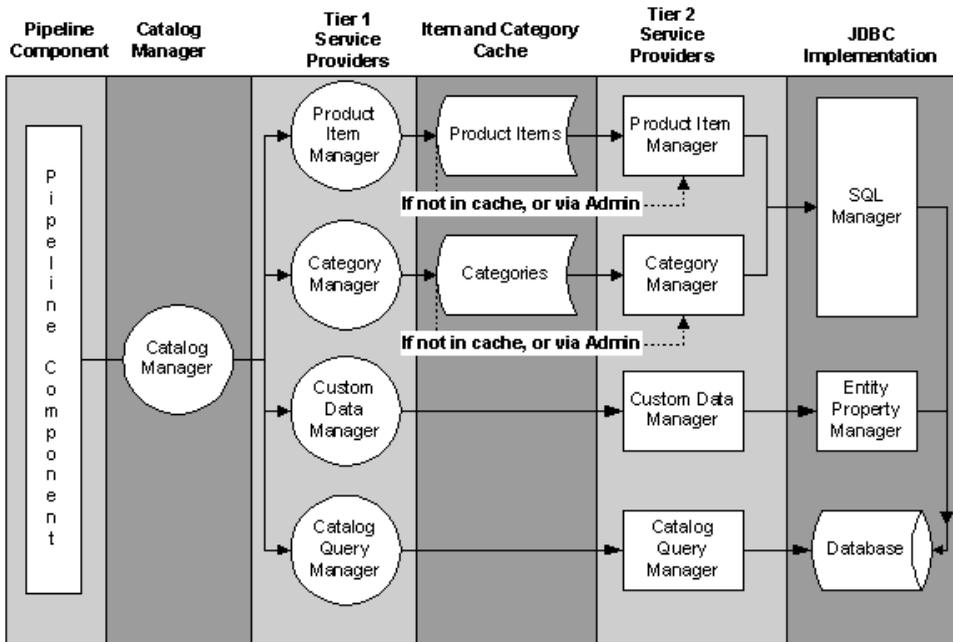
- Catalog Manager
- Product Item Manager
- Category Manager
- Custom Data Manager
- Catalog Query Manager

All services are implemented using J2EE-compliant stateless session Enterprise JavaBeans (EJBs). These EJBs separate the functionality of the Catalog into discrete, pluggable components.

Catalog Architecture

Figure 7-1 illustrates the Product Catalog architecture.

Figure 7-1 Product Catalog Architecture



For example, the process of displaying a product item in a visitor'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 visitor 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. An example is `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. For more information about caching and the Product Catalog, see “Using Caches” in the *Performance Tuning Guide*.
7. The Pipeline Component then adds the result of the Catalog request to the Pipeline Session.
8. The JSP uses the JSP tag libraries to extract the results of the Catalog request from the Pipeline Session and formats the results as HTML.

Note: For more information about Pipelines, Pipeline Components, and the Pipeline Session, see the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline* documentation.

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 7-1 shows the method summary for the `CatalogManager` interface.

Table 7-1 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.

Table 7-1 Method Summary for the CatalogManager Interface (Continued)

CatalogQueryManager	getCatalogQueryManager(CatalogRequest request) Returns the CatalogQueryManager catalog service.
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 7-1.

Note: You can find the ejb-jar.xml and weblogic-ejb-jar.xml deployment descriptor files in the ebusiness.jar file, which is located in the PORTAL_HOME\applications\wlcsApp directory.

Listing 7-1 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>
  </ejb-reference-description>
```

```
<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 7-2 shows the method summary for the `ProductItemManager` interface.

Table 7-2 Method Summary of the `ProductItemManager` Interface

Return Type	Method Signature
<code>void</code>	<code>createItem(CatalogRequest request, ProductItem product)</code> Creates a new product item.
<code>ProductItem</code>	<code>getItem(CatalogRequest request, ProductItemKey productKey)</code> Returns the product item with the specified key.
<code>int</code>	<code>getItemCount(CatalogRequest request)</code> Returns the number of product items in the catalog.
<code>ProductItemKey[]</code>	<code>getItemKeys(CatalogRequest request, int beginIndex, int endIndex)</code> Returns an array over all existing product item keys within the specified ordered range.
<code>ViewIterator</code>	<code>getItems(CatalogRequest request, int viewSize)</code> Returns a <code>ViewIterator</code> over all existing product items.
<code>ProductItem[]</code>	<code>getItems(CatalogRequest request, ProductItemKey[] productKeys)</code> Returns the product items with the given product item keys.
<code>java.lang.String[]</code>	<code>getKeywords(CatalogRequest request, ProductItemKey productKey)</code> Returns the keywords associated with a given product item.
<code>void</code>	<code>setKeywords(CatalogRequest request, ProductItemKey productKey, java.lang.String[] keywords)</code> Sets the keywords for a given product item.

Table 7-2 Method Summary of the ProductItemManager Interface (Continued)

Return Type	Method Signature
void	removeItem(CatalogRequest request, ProductItemKey productKey) Removes a product item.
void	updateItem(CatalogRequest request, ProductItem product) 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 7-3 shows the method summary for the CategoryManager interface.

Table 7-3 Method Summary of the CategoryManager Interface

Return Type	Method Signature
void	addItem(CatalogRequest request, CategoryKey categoryKey, ProductItemKey itemKey) Adds an item to the specified category.
void	createCategory(CatalogRequest request, CategoryKey parentKey, Category category) Creates a subcategory within the supplied parent category.
Category[]	getAncestors(CatalogRequest request, CategoryKey categoryKey) Returns the ancestors of the specified category in ascending order.
Category[]	getCategories(CatalogRequest request, CategoryKey[] categoryKeys) Returns the categories with the given category keys.

Table 7-3 Method Summary of the CategoryManager Interface (Continued)

Return Type	Method Signature
ViewIterator	<pre>getCategories(CatalogRequest request, int viewSize)</pre> <p>Returns a ViewIterator over all existing categories.</p>
Category	<pre>getCategory(CatalogRequest request, CategoryKey categoryKey)</pre> <p>Returns the category with the given category key.</p>
int	<pre>getCategoryCount(CatalogRequest request)</pre> <p>Returns the total number of categories in the product catalog.</p>
CategoryKey[]	<pre>getCategoryKeys(CatalogRequest request, int beginIndex, int endIndex)</pre> <p>Returns an array of all existing category keys within the specified ordered range.</p>
int	<pre>getItemCount(CatalogRequest request, CategoryKey categoryKey)</pre> <p>Returns the number of product items associated with the specified category.</p>
ProductItemKey[]	<pre>getItemKeys(CatalogRequest request, CategoryKey categoryKey, int beginIndex, int endIndex)</pre> <p>Returns an array of all product item keys of the specified category within the specified ordered range.</p>
ViewIterator	<pre>getItems(CatalogRequest request, CategoryKey categoryKey, int viewSize)</pre> <p>Returns a ViewIterator over all product items of the specified category.</p>
int	<pre>getOrphanedItemCount(CatalogRequest request)</pre> <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>

Table 7-3 Method Summary of the CategoryManager Interface (Continued)

Return Type	Method Signature
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>
ViewIterator	<p><code>getSubCategories(CatalogRequest request, CategoryKey categoryKey, int viewSize)</code></p> <p>Returns a <code>ViewIterator</code> over all subcategories of the specified category.</p>

Table 7-3 Method Summary of the CategoryManager Interface (Continued)

Return Type	Method Signature
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 subcategory 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 7-4 shows the method summary for the `CustomDataManager` interface.

Caution: The method signatures shown in Table 7-4 often contain a namespace argument. Note that this namespace does not refer to a Webflow namespace. For more information about Webflow namespaces, see the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline* documentation.

Table 7-4 Method Summary for the CustomDataManager Interface

Return Type	Method Signature
<code>java.util.Map</code>	<pre>getProperties(CatalogRequest request, CatalogItemKey itemKey)</pre> <p>Retrieves all the property values.</p>
<code>java.util.Map</code>	<pre>getProperties(CatalogRequest request, CatalogItemKey itemKey, java.lang.String namespace)</pre> <p>Retrieves all the property values within a namespace.</p>
<code>java.lang.Object</code>	<pre>getProperty(CatalogRequest request, CatalogItemKey itemKey, java.lang.String namespace, java.lang.String key, java.lang.Object defaultValue)</pre> <p>Retrieves the value associated with the named key.</p>
<code>void</code>	<pre>removeProperties(CatalogRequest request, CatalogItemKey itemKey)</pre> <p>Removes all the properties for an item.</p>

Table 7-4 Method Summary for the CustomDataManager Interface (Continued)

Return Type	Method Signature
<code>java.lang.Object</code>	<code>removeProperty(CatalogRequest request, CatalogItemKey itemKey, java.lang.String namespace, java.lang.String key)</code> Removes the property associated with the named key.
<code>void</code>	<code>setProperty(CatalogRequest request, CatalogItemKey itemKey, java.lang.String namespace, java.lang.String key, java.lang.Object value)</code> Associates 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. A query-based search allows a complex Boolean expression on any of the item attributes to be evaluated.

Table 7-5 shows the method summary for the `CatalogQueryManager` interface.

Table 7-5 Method Summary for the CatalogQueryManager Interface

Return Type	Method Signature
<code>ProductItemKey[]</code>	<code>search(CatalogRequest request, CatalogQuery query)</code> Returns the keys of product items that met the criteria of the supplied catalog query object.

Table 7-5 Method Summary for the CatalogQueryManager Interface

Return Type	Method Signature
ViewIterator	<code>search(CatalogRequest request, CatalogQuery query, int viewSize)</code> Returns a <code>ViewIterator</code> over all product items that met the criteria of the supplied catalog query object.

For related information, see “Query-Based Search Syntax” on page 5-76.

Integrating Services with 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 topic in Figure 7-1.

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 7-2, 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 7-2 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>
  <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 `PORTAL_HOME\applications\wlcsApp` directory, where `PORTAL_HOME` is the directory in which you installed WebLogic Portal.)

Note: For related information, see the “Writing Web Application Deployment Descriptors” and “Deploying Web Applications” topics in the *Assembling and Configuring Web Applications* documentation.

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 descriptor 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 7-1), the new services can take advantage of the powerful caching features of the Product 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 topic. For information about steps 2 and 4, please refer to the “Deploying Web Applications” topic in the *Assembling and Configuring Web Applications* documentation.

The following topics are covered in this section:

- Create New Services
- Changes to `ejb-jar.xml`
- Changes to `weblogic-ejb-jar.xml`

The `ejb-jar.xml` and `weblogic-ejb-jar.xml` files are packaged in the `ebusiness.jar` file, which can be found in the `PORTAL_HOME\applications\wlcsApp` directory, where `PORTAL_HOME` is the directory in which you installed WebLogic Portal.

You can find all the source code shown in this section in the `PORTAL_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 some of the Commerce services deployment JAR files; therefore, it is important that you first back up all files and JAR libraries that you intend to modify.

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:

`PORTAL_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 7-3 shows sample implementation source code for:

- `FileProductItemManagerImpl.java`
- `FileCatalogQueryManagerImpl.java`

After you install WebLogic Portal, these files can be found in the `PORTAL_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 7-3 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) 1997-2001 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
    {$Import$_Begin ----- CUSTOM CODE -----
        // USER CHANGES: Add interfaces that are implemented here
    }$Import$_End  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```


7 Using the API to Extend the Product Catalog

```
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   ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    }
```


7 Using the API to Extend the Product Catalog

```
*
* 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
```



```

        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++ )
                                    {
                                        for( int x = 0; x < keywords.length && found
== false; x++ )
                                            {
                                                if( keywords[x] != null && keywords[x].equals(
itemKeywords[i] ) != false )
                                                    {
                                                        found = true;
                                                        resultList.add( itemKeys[n] );
                                                    }
                                                }
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}

```


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 7-5, lines that should be removed are shown in *italics*. Lines that should be added are shown in **bold**.

Listing 7-5 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>
  <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

```

7 Using the API to Extend the Product Catalog

```

    </remote>
  </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>

<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>
  <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 7-6 shows the deletions and additions needed in the `weblogic-ejb-jar.xml` file. The `weblogic-ejb-jar.xml` files is packaged in the `ebusiness.jar` file, which can be found in the `PORTAL_HOME\applications\wlcsApp` directory, where `PORTAL_HOME` is the directory in which you installed WebLogic Portal.

Warning: Create a backup copy of the file before you modify its contents.

Note: In Listing 7-6, lines that should be removed are shown in *italics>*. Lines that should be added are shown in **bold>**.

Listing 7-6 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>
  </reference-descriptor>
</weblogic-enterprise-bean>

<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>
<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>
com.beasys.commerce.ebusiness.catalog.service.examples.file.FileProductItemManager
</ejb-name>
</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>
```

7 Using the API to Extend the Product Catalog

```
<ejb-name>  
  com.beasys.commerce.ebusiness.catalog.service.query.JdbcCatalogQueryManager  
</ejb-name>  
<ejb-name>  
  com.beasys.commerce.ebusiness.catalog.examples.file.FileCatalogQueryManager  
</ejb-name>
```

8 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 Product Catalog architecture. Although the strings displayed to visitors can easily be translated by editing various property files (such as `wlcs-catalog.properties`), maintaining dynamic multilingual catalog content and allowing visitors 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 4-43.

To meet these objectives, the Commerce services include 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 7-3.

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
 - Important Note About Currencies
- 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 visitors. 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 and provides important information about using alternative currencies.

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 8-2.

The language attribute of the `CatalogRequest` object informs the product catalog system about the language a visitor 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 visitor’s language preference.

An example of setting the language on a `CatalogRequest` object is shown in Listing 8-1.

Listing 8-1 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 8-2.

Listing 8-2 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 visitor'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 8-2.

Note: For more information about the Product Catalog 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 8-3.

Listing 8-3 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 8-3, 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 visitors 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 8-3.

Limiting Search Results by Language

The default `CatalogQueryManager` Tier 1 Service Provider allows you 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 8-2. An example is shown in Listing 8-1.

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 Portal 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 8-4.

Listing 8-4 Default Language for Catalog Requests as Set in the wlbs-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 Product Catalog (if non-null). The other APIs do not use this attribute and will return Product 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)").

Important Note About Currencies

All currencies dealt with by the Price Service for a pricing operation must be identical (case sensitive). This means that: all items in the shopping cart must have the same currency and any discounts applicable during the operation must also have the same currency. Failure to ensure that currencies are identical may result in a `CurrencyMismatchException` at some point during execution .

The three areas where currencies are set are:

- The individual product items in the product catalog, specified using the WebLog Portal Administration Tools.
- The shipping cost in the `CalculateShippingPC` Pipeline Component (obtained via the `ShippingMethod.getPrice()` method).
- Discount definitions in the E-Business Control Center (for fixed price and fixed amount off discounts only).

Using the Catalog Architecture to Maintain Internationalized Product Catalogs

Depending on your business and technical requirements, you can use the Product 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 building and 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 visitor'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 visitor based on their specified language preference. If the language attribute of the category or item does not match that of the visitor's specified

language, the item or category is not displayed. Otherwise, if the language attribute does match that of the visitor'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 visitor.

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 included in the Commerce services, see Chapter 3, "Using the 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 visitor 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 6, “Product Catalog JSP Tag Library Reference.”

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 visitor 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 visitor should be able to reach these items through searches. For more information about orphaned catalog items, see “Removing an Item from One or More Categories” on page 4-35. For more information about the visibility of items, see “Controlling the Visibility of Items in the Product Catalog” on page 4-18.

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 visitor 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

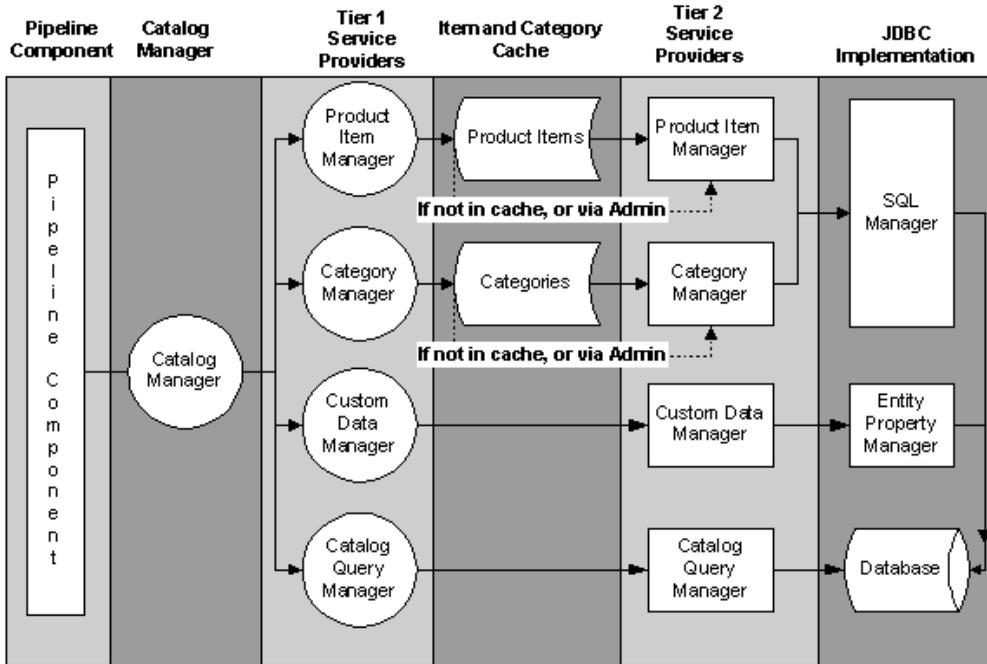
The third method for internationalization of product catalogs:

- 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 8-1 shows the same Product Catalog architecture that is described in “Catalog Architecture and Services” on page 7-3.

Figure 8-1 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 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 included in the Commerce services, see Chapter 3, “Using the 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 8-10, 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)
```

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 Administration Tools will work as expected, because there is still only one `CatalogManager` instance.

Index

A

- adding categories 4-8
- adding items 4-14
- administration tasks
 - adding categories 4-8
 - adding items 4-14
 - assigning items to categories 4-19
 - catalog cache settings 4-41
 - changing administrator password 4-4
 - controlling visibility of items 4-18
 - deleting categories 4-37
 - deleting items 4-31
 - editing attributes
 - categories 4-23
 - items 4-23
 - editing availability of items 4-29
 - introduction 4-1
 - moving items 4-39
 - starting the server 4-2
 - wlcs-catalog.properties file 4-43
- assigning items to categories 4-19
- attribute-based search syntax 5-76
- attributes
 - custom
 - for items 4-39
 - language 8-3
 - and expression-based queries 8-6
 - CatalogRequest object 8-3, 8-12
 - image 8-5
 - product items and categories 8-4
- availability of items

editing 4-29

B

- BEA, contacting xiv
- browse.jsp template 5-19
- business logic 5-11, 5-52, 5-61, 5-70

C

- cache
 - catalog 4-41
 - cat.tld tag library 6-2
- catalog
 - adding categories 4-8
 - adding items 4-14
 - administration 4-1
 - API overview 7-2
 - architecture and services 7-3, 8-1
 - using for internationalization 8-7
 - assigning items to categories 4-19
 - cache 4-41
 - caching settings 4-41
 - deleting categories 4-37
 - deleting items 4-31
 - development roles 1-8
 - Dublin Core standard 2-4
 - editing attributes
 - categories 4-23
 - items 4-23
 - editing availability of items 4-29
 - editing schema definition 4-46

- Entity-Relation diagram 2-2
- getProperty tag 6-4
- hierarchy 1-6
- input processors 5-82
- internationalization
 - images 8-5
 - products and categories 8-4
- introduction 1-1
- iterateThroughView tag 6-8
- iterateViewIterator tag 6-6
- JSP tag library 6-2
- link with order processing 1-9
- moving items 4-39
- multiple instances 8-11
- overview of schema 2-1
- persisting language information 8-4
- pipeline components 5-89
- using API to extend 7-1
- visibility of items 4-18
- wlcs-catalog.properties file 4-43, 8-1
- writing your own catalog service 7-17
- CATALOG_ENTITY Database Table 2-6
- CATALOG_PROPERTY_KEY Database Table 2-6
- CATALOG_PROPERTY_VALUE Database Table 2-7
- CatalogIP input processor 5-82
- CatalogManager interface 7-5, 8-11
 - extending 8-12
- CatalogPC pipeline component 5-89
- CatalogQueryManager interface 7-14
 - and internationalization 8-5
- CatalogRequest object
 - language attribute 8-3, 8-12
- categories
 - adding 4-8
 - assigning items to 4-19
 - deleting 4-37
 - displayed to users 4-30
 - editing attributes 4-23
- category.jsp

- about 5-29
- CategoryManager interface 7-9
- changing the administrator password 4-4
- comparison operators in query 5-78
- contacting BEA xiv
- controlling number of search results 5-80
- controlling visibility of items 4-18
- currencies, mismatched 8-6
- custom attributes
 - for items 4-39
- CustomDataManager interface 7-13

D

- Data Junction 3-13
- data loaders
 - introduction 3-12
 - third party 3-13
- database tables
 - CATALOG_ENTITY 2-6
 - CATALOG_PROPERTY_KEY 2-6
 - CATALOG_PROPERTY_VALUE 2-7
 - WLCS_CATEGORY 2-8
 - WLCS_PRODUCT 2-12
 - WLCS_PRODUCT_CATEGORY 2-16
 - WLCS_PRODUCT_KEYWORD 2-17
- DataStage by Informix 3-13
- DBLoader
 - and multiple catalog instances 8-12
 - database considerations 3-10
 - dbloader.properties files 3-4
 - input file 3-2
 - introduction 3-1
 - log files 3-9
 - running 3-7
 - validations 3-10
- Defined Constraints 2-22
- deleting categories 4-37
- deleting items 4-31
- development roles
 - for catalog 1-8

documentation, where to find it xiii

Dublin Core standard

catalog 2-4

E

editing attributes

categories 4-23

items 4-23

Entity-Relation diagram

for catalog tables 2-2

ETI-EXTRACT by ETI 3-13

event(s)

browse.jsp 5-39

itemdetails.jsp 5-52

main.jsp 5-11

search.jsp 5-61

searchresults.jsp 5-70

expression-based search queries 8-6

ExpressionSearchIP input processor 5-83

extending catalog

using API 7-1

G

GetAncestorsPC pipeline component 5-90

GetCategoryIP input processor 5-84

GetCategoryPC pipeline component 5-91

GetParentPC pipeline component 5-92

GetProductItemIP input processor 5-85

GetProductItemPC pipeline component 5-93

GetProductItemsPC pipeline component 5-94

getProperty tag 6-4

GetSubcategoriesPC pipeline component 5-95

H

hierarchy

catalog 1-6

I

improving performance

catalog cache 4-41

input processors

for catalog JSPs 5-82

Internationalization

and catalog architecture 8-7

currencies 8-6

language and country codes 8-2

methods

filtering content 8-7

multiple catalog instances 8-10

parsing language-specific data 8-8

service routing 8-12

non-ASCII characters 5-77

of images 8-5

of product items and categories 8-4

product catalog support 8-1

itemdetails.jsp

about 5-51

items

adding 4-14

defining custom attributes 4-39

deleting 4-31

displayed to users 4-30

editing attributes 4-23

editing availability 4-29

moving 4-39

visibility of 4-18

itemssummary.jsp

about 5-35

iterateThroughView tag 6-8

iterateViewIterator tag 6-6

J

JSP tag library

cat.tld 6-2

JSP tags

getPipelineProperty 5-16

Utilities

smnav 6-10
example 6-11

K

KeywordSearchIP input processor 5-86

L

language

attribute

and expression-based queries 8-6

CatalogRequest object 8-3, 8-12

product items and categories 8-4

limiting search results by 8-5

M

main.jsp template 5-7

methods

internationalization

filtering content 8-7

multiple catalog instances 8-10

parsing language-specific data 8-8

service routing 8-12

MoveAttributeIP input processor 5-87

MoveAttributePC pipeline component 5-96,
5-97

moving items 4-39

multiple catalog instances 8-11

O

object, CatalogRequest

language attribute 8-3, 8-12

optimizing catalog cache 4-41

P

passwords

changing administrator's 4-4

performance

improving with catalog cache 4-41

pipeline components

for catalog 5-89

PowerConnect by Informatica 3-13

printing product documentation xiii

product catalog

architecture 8-1

using for internationalization 8-7

editing schema definition 4-46

internationalization

images 8-5

product items and categories 8-4

introduction 1-1

multiple instances 8-11

overview of features 1-3

persisting language information 8-4

ProductItemManager interface 7-8

Q

query

comparison operators 5-78

expression-based and language attribute
8-6

query-based search syntax 5-76

R

related information xiv

RemoveAttributeIP input processor 5-88

RemoveAttributePC pipeline component 5-
98

S

schema

catalog tables 2-1

editing product catalog 4-46

file

for internationalization 8-11

search results

- controlling number of 5-80
- limiting by language 8-5
- search syntax 5-76
- search.jsp template 5-56
- SearchPC pipeline component 5-99
- searchresults.jsp template 5-66
- server
 - starting 4-2
- smnav JSP tag 6-10
 - example 6-11
- SQL Scripts 2-18
- Start page
 - for JSPs 5-4
- starting the server 4-2
- support, technical xiv

W

- WLCS_CATEGORY Database Table 2-8
- WLCS_PRODUCT Database Table 2-12
- WLCS_PRODUCT_CATEGORY Database
 - Table 2-16
- WLCS_PRODUCT_KEYWORD Database
 - Table 2-17
- wlcs-catalog.properties file 4-43, 8-1

