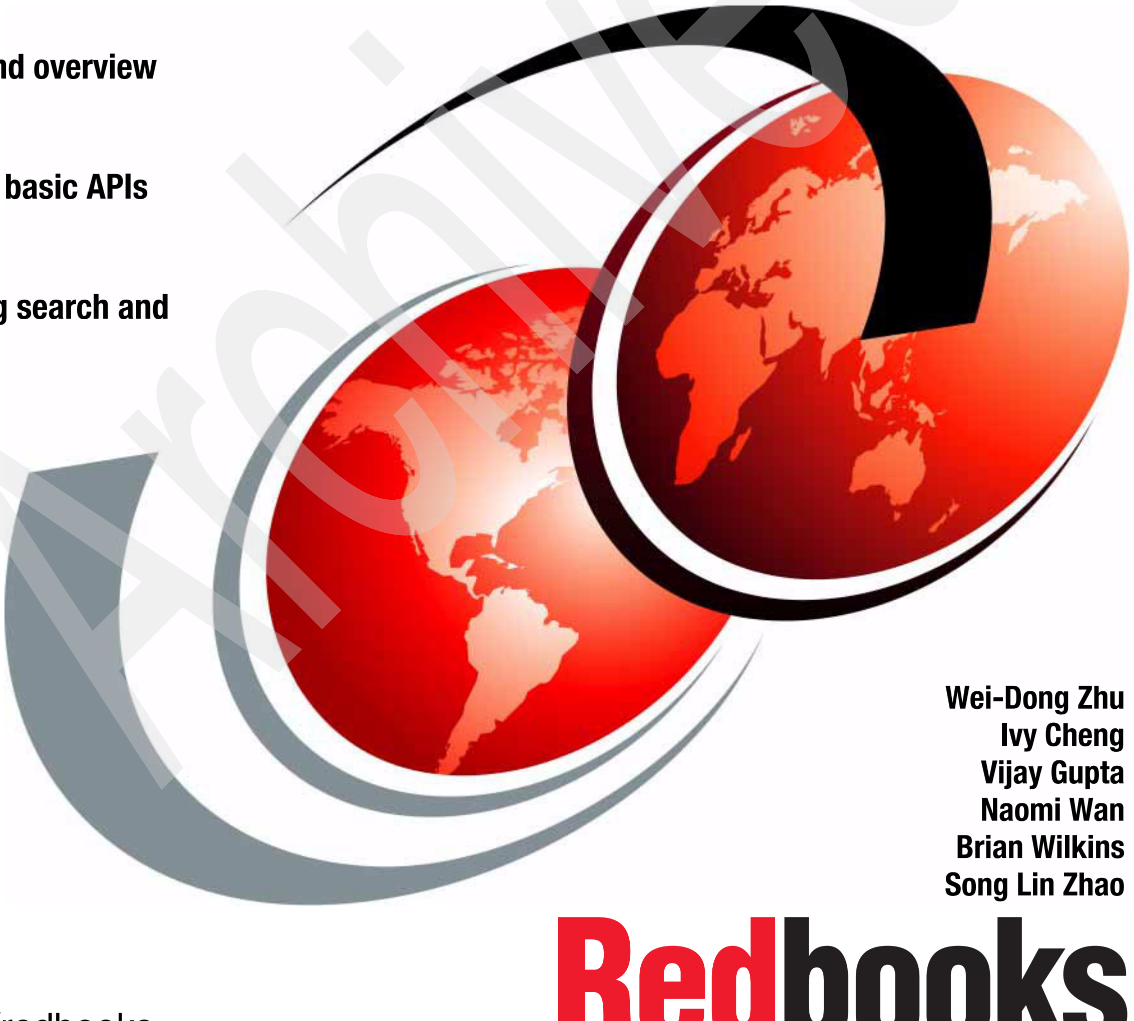
Getting Started with Commerce Module for OmniFind Discovery Edition Customization 101

Concepts and overview

Introducing basic APIs

Customizing search and navigation



Wei-Dong Zhu
Ivy Cheng
Vijay Gupta
Naomi Wan
Brian Wilkins
Song Lin Zhao

ibm.com/redbooks

International Technical Support Organization

Getting Started with Commerce Module for OmniFind Discovery Edition Customization 101

May 2007

SG24-7358-00

Note: Before using this information and the product it supports, read the information in "Notices" on page xix.

First Edition (May 2007)

This edition applies to Version 8 Release 4 of IBM OmniFind Discovery Edition (product number 5724-N53).

Contents

Figures

Tables

Examples

Notices

Trademarks

Preface

IBM® OmniFind™ Discovery Edition enables efficient and accurate ways to search content from various content sources so that users can find the right information whenever and however they want it. *IBM WebSphere® Commerce* is a powerful and flexible solution for running business-to-business (B2B) and advanced business-to-consumer (B2C) e-commerce sites for businesses.

IBM Commerce Module for OmniFind Discovery Edition integrates IBM OmniFind Discovery Edition into IBM WebSphere Commerce. The integrated module makes it easier for people to find the products and services that match their specific requirements, thus helping online retail and catalog companies to convert shoppers into buyers.

This book provides an introduction of customizing the Commerce Module for OmniFind Discovery Edition.

We describe what OmniFind Discovery Edition and WebSphere Commerce do and explain how the out-of-the-box Commerce Module for OmniFind Discovery Edition works. We include simple instructions on installing both products and setting up the environment for running the Commerce Module for OmniFind Discovery Edition. We also provide the development environment setup that you might want to use for customization.

OmniFind Discovery Edition Application Programming Interfaces (APIs) provide the means to customize the appearance and behavior as well as the search functions of e-commerce stores powered by Commerce Module for OmniFind Discovery Edition.

The goal of this book is to help you customize your online store using these APIs. We focus on the Commerce Module for OmniFind Discovery Edition with:

- The online store presentation
- Database as the content source
- Tabbed navigation JSP™ layout

We introduce some of the basic APIs and show examples of using them to run simple searches, refine search results, and navigate within the search results.

No explanation would be complete without actual examples and case studies. In this book, we select a sample starter store that comes with WebSphere Commerce and use it throughout the book for simple explanation and demonstration. We customize the store and describe how things are done.

Specifically, we show you how to use the APIs to add or modify existing menus and navigation options and change some of the appearance and behavior of the existing sample store Web site. We show you how to work with OmniFind Discovery Edition search engine and pass multiple constraints to narrow searches for the online store.

In addition, we show you how you can work with WebSphere Commerce and OmniFind Discovery Edition data model and structure to get or create the necessary product information for customizing online stores. From our examples, you learn how to add new product attributes from the Commerce side and the OmniFind Discovery Edition side as required.

This book is intended to be used by system integrators and solution developers with limited knowledge of WebSphere Commerce and OmniFind Discovery Edition. Use this book as an introduction to the world of OmniFind Discovery Edition and WebSphere Commerce integration. Once you work through the examples, you are on your way to do customization for your business requirements.

Refer to the existing product manuals for reference or details that this book does not cover.

Note: The name of the OmniFind Discovery Edition product has gone through several changes in the past few years:

- Before December 2005, the product was called iPhrase.
- By January 2006, the product was renamed to WebSphere Content Discovery Server, with a version number of 8.3.
- By December 2006, the product has the current new name, OmniFind Discovery Edition, with a version number of 8.4.

In this book, there are mixed references of the wording, WebSphere Content Discovery Server and OmniFind Discovery Edition. They refer to the same product, but different versions. While working on this book, we worked with the version 8.3.1 of the product. At the time, it was still referenced as WebSphere Content Discovery Server. Within the product itself, in many places, it still has reference of iPhrase. Although some of the instructions and descriptions provided in this book might conflict with the latest version, the basic idea, concept, and process are similar and should serve as a useful learning tool for you to get familiar with the product and its APIs.

The team that wrote this book

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, San Jose Center.

Wei-Dong Zhu (Jackie) is a Content Management Project Leader with the ITSO in San Jose, California. She has more than 10 years of software development experience in accounting, image workflow processing, and digital media distribution (DMD). Her development work with one of the DMD solutions contributed to a first time ever win for IBM of a technical Emmy award in 2005. Jackie holds a Master of Science degree in Computer Science from the University of Southern California. Jackie joined IBM in 1996. She is a Certified Solution Designer for IBM DB2® Content Manager.

Ivy Cheng is a Certified Consulting I/T Specialist in Information Management Technical Sales for Sales & Distribution in IBM Canada. She is the Americas Competency Team OLTP Sub-Team lead. Ivy has a long involvement with IBM Data Management products. She worked as a DB2 developer and System Test team lead in the Toronto Lab from 1993 to 1997. She also worked as Senior I/T Specialist in the Software Business Unit on client Information Management projects involving in both proposal and implementation. She has been giving presentations on OLTP sizing, Data Warehouse sizing, and DB2 wireless solutions at various technical conferences. Ivy holds a Master Degree in Computer Science from the University of Western Ontario, Canada.

Vijay Gupta is a Senior Technical Consultant I/T Specialist in Content Management and Discovery for Software Group in IBM USA. He is in the Professional Services and Enablement team and works directly with clients on providing consulting services for Content Discovery Solutions. Vijay recently started his career with IBM as a consultant. Vijay has over ten years of experience in software and network design, development, validation, and verification, and client services. He was a Principal Engineer with Alcatel before joining IBM. His area of expertise include application development, client engineering, technical support, and quality assurance analysis. Vijay holds a Master Degree in Computer Science from University of Pune, India.

Naomi Wan is a Lead Software Developer for the WebSphere Commerce Engagement Services team, based in the IBM Canada Toronto Laboratory. She has five years of experience in WebSphere Commerce, ranging from WebSphere Commerce 5.1 to 6. Naomi holds a degree in Computer Science from University of Waterloo, Canada. Her areas of expertise include solution design and implementation for e-Commerce solutions. She has been certified on IBM DB2 UDB and WebSphere Commerce. She has written extensively on WebSphere Commerce catalog synchronization methodologies and tutorials for WebSphere Commerce.

Brian Wilkins is a Senior Technical Support Consultant in IBM Data Management in the United Kingdom. He has more than 8 years experience in supporting Informix® products and DB2 on Linux®, UNIX®, and Windows®.

Song Lin Zhao is a Software Engineer of WebSphere Commerce department in IBM China Development Lab. He has three years of experience in WebSphere Commerce field. He holds a Master degree from Beijing Jiao Tong University, China. His areas of expertise include EIS/ERP, WebSphere Commerce Deployment, Software Test Automation Framework design and implementation. He has been certified on IBM DB2 UDB and WebSphere Commerce.

Thanks to the following people for their contributions to this project:

Raymond Lau
Elizabeth Ashley
Shirley Braley
R Paul Johnson
Rick Keilty
Brendan McCormick
Kenny Ng
Joshua Payne
Michael Pelts
Yoni Rosen
Wendy Taylor
Yariv Tzaban
IBM Software Group, Content Management and Discovery, Lexington, MA

Sarah Berger
IBM Sales & Distribution, Software Sales, San Diego, CA

Xiaomei Wang
IBM Software Group, Content Management and Discovery Partner Enablement, Toronto, Canada

Emma Jacobs, Deanna Polm, Sangam Racherla, Yvonne Lyon
International Technical Support Organization, San Jose

Become a published author

Join us for a two- to six-week residency program! Help write one of our IBM Redbooks® dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You will have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you will develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

`ibm.com/redbooks/residencies.html`

Comments welcome

Your comments are important to us!

We want our Redbooks to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- Use the online **Contact us** form to review Redbooks, at:

  `ibm.com/redbooks`

- Send your comments in an e-mail to:

  `redbooks@us.ibm.com`

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Part 1

Concepts and overview

In this part of the book, we provide an overview of WebSphere Commerce, OmniFind Discovery Edition, and Commerce Module for OmniFind Discovery. We cover some of the simple concepts. In addition, we introduce OmniFind Discovery Edition APIs.

1

Introduction

In this chapter we provide an overview of OmniFind Discovery Edition and WebSphere Commerce. We introduce the Commerce Module for OmniFind Discovery Edition, which is a packaged integration of OmniFind Discovery Edition with WebSphere Commerce.

We cover the following topics:

- Overview of IBM OmniFind Discovery Edition
- Overview of IBM WebSphere Commerce
- Commerce Module for OmniFind Discovery Edition

1.1 Overview of IBM OmniFind Discovery Edition

IBM OmniFind Discovery Edition offers a powerful and flexible search engine with a user friendly and effective Web-ready front end. It comes with a rich set of functions, tools, and search capabilities.

IBM OmniFind Discovery Edition provides:

- Customizable search and navigation.
- The ability to acquire, index, and search Web content from different content sources with different content format.
- Management Console that enables configuring, tuning, testing and debugging of the system.
- OmniFind Discovery Edition APIs that enable search and navigation customization in addition to what Management Console offers.

Note: The name of the product has gone through several changes in the past few years:

- Before December 2005, the product was called iPhrase.
- By January 2006, the product was renamed to WebSphere Content Discovery Server, with a version number of 8.3.

  The integration between WebSphere Commerce and WebSphere Content Discovery was called: WebSphere Content Discovery for Commerce.
- By December 2006, the product has the current new name, OmniFind Discovery Edition, with a version number of 8.4.

  The integration between WebSphere Commerce and OmniFind Discovery Edition is now called: Commerce Module for OmniFind Discovery Edition.

In this book, there are mixed references of the wordings for WebSphere Content Discovery Server and OmniFind Discovery Edition. They refer to the same product, but different versions. While working on this book, we worked with the version 8.3.1 of the product. At the time, it was still referenced as WebSphere Content Discovery Server. Within the product itself, in many places, it still has references to iPhrase.

OmniFind Discovery Edition acquires (pulls) content from different content sources (for example the existing database and file-based content), process them, index them, and generate the content index. The OmniFind Discovery Edition server uses this content index for search and navigation.

Figure 1-1 shows the build process of OmniFind Discovery Edition.

The build process turns existing content into a content index to be used at runtime by OmniFind Discovery server.

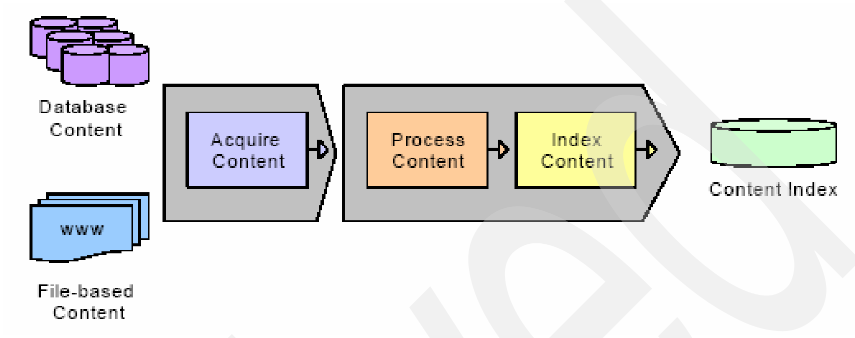


Figure 1-1 OmniFind Discovery Edition build process

OmniFind Discovery Edition supports a wide range of content sources, from structured database content to unstructured Web and file system content. It is built on a foundation of content integration, enterprise searching, text analytics, adaptive presentation and contextual understanding. Ad Hoc Reporting and Repository Connector modules are also available.

Contextual understanding includes functions and capabilities such as:

- Respelling (in case user enters a misspelled word)
- Understanding traits (traits are attributes or characters of a product)
- Parsing out stopwords (words that are not used in searches)
- Stemming (process of reducing words to simplest form)
- Using store location to influence presentation of results
- Automatically identifying things such as brand names in the search query and filtering down on them

Adaptive presentation includes functions and capabilities such as:

- Faceted navigation
- Dynamic changing of faceted navigation
- Breadcrumbing
- Voiceover
- Drilling sideways
- Business rules sidebar presentation based on different triggers
- Different layouts based on different queries and results

OmniFind Discovery Edition supports the following features:

- Full natural language understanding and searches
- Search results categorization
- Keyword search
- Guided search, allowing you to refine searches by drilling down into results by features such as manufacturer, designer, color, and price range
- Attribute relaxation, in which, if users over-specify, OmniFind Discovery Edition broadens the search and offers further assistance
- Spelling correction

The OmniFind Discovery Edition search engine has a unique ability to understand Web site user intent and the application context of user requests. It helps Web site users move beyond ordinary searching to find the products or information that they want.

By understanding what users mean as opposed to what they say, OmniFind Discovery Edition respond to user requests with more relevant information. It understands user intent in a variety of manners — for example, by its ability to automatically correct for poor spelling, and by incorporating concepts such as synonyms and broader or narrower relationships. You can choose to let the system automatically correct the misspelled word and then perform the search for the user, or you can choose a *Did you mean* mode and let the user choose to render or not to render the interrupted meaning. You can also add your own dictionary.

OmniFind Discovery Edition leverage numeric data associated with products to augment this understanding of user intent. One such method is by understanding product traits. For example, you can associate a price trait with the price information from your product catalog. OmniFind Discovery Edition can automatically associate higher prices with a user-entered search word, *expensive*, and associate lower prices with *cheap*. If a user enters a search word, *inexpensive*, OmniFind Discovery Edition understands the concept of inexpensive as a constraint on the price range and filters the results to display cheap products.

OmniFind Discovery Edition's ability to understand natural language offers users intuitive search experiences. For example, a user can enter the following sentence in the search box: "Do you have black boots in size 7?" The user can ask a question regarding a service issue, "How much is the shipping and handling?"

Users are free to express themselves as they normally would with other people, without the requirement for conforming to technical conventions or understanding Boolean operators. Users do not have to include AND or OR in their searches. If they choose to do so, it does not affect their results. For example, to search for handbags of a particular brand, a shopper can enter “IBM handbag” instead of “IBM and handbag”. Users can also enter punctuation and a mixture of upper case and lower case.

In addition to understanding user intent, OmniFind Discovery Edition uses adaptive presentation to guide the content discovery process by presenting answers, navigational refinement options, and proactive guidance in a format that helps users take actions. For example, users can further refine their search results by choosing an option from the left navigation panel. Or, you can use business rules to proactively guide the online shoppers, based on their search context, to special promotions, display cross-sell, or up-sell advertisement in retail environments. If the shopper puts a certain necklace in the shopping cart, you can use a business rule to display the matching earnings, or offer a discount if the shopper buys both. You can choose to use the rich set of OmniFind Discovery Edition’s features such as voiceover, stopwords, and breadcrumbs to tailor dynamic and interactive search experience.

Definitions:

A *voiceover* is a confirmation of a user’s most recent search request. For example, if you search for “lounge chairs”, a voiceover as a confirmation can be displayed on the search result page as follows:

```
You asked: “lounge chairs”.
```

Stopwords are words that do not provide useful meaning in search even though they are entered by users. OmniFind ignores these words and does not use them for content search. For example, if you search for “the lounge chairs in white”, the search engine does not use the following common words (stopwords) in the search:

```
the, in
```

You can add or remove the stopwords in OmniFind Discovery Edition.

A *breadcrumb* displays the user’s current search constraints. It states what the user has requested for the system to produce the search result. For example, if you search for “lounge chairs”, the breadcrumb shows the following text:

```
Your search: lounge chairs
```

1.1.1 Administration

OmniFind Discovery Edition comes with a Management Console (MC) user interface for both system administration and business related tasks. You can also use command line commands for administration.

Using Management Console, you can perform business tasks such as creating content source collections, spidering the Web content source, or configuring and linking to the database content source. You can create business rules for special promotion, merchandising, cross sell, and upsell.You can configure system settings, change layouts, and test your setting and changes. You can also perform user administration and server administration such as project build and deployment.

1.1.2 User interface

OmniFind Discovery Edition enables users to narrow searches on Web sites. Figure 1-2 shows a portion of a search result page when a shopper searches for "chairs" from the online store powered by OmniFind Discovery Edition.

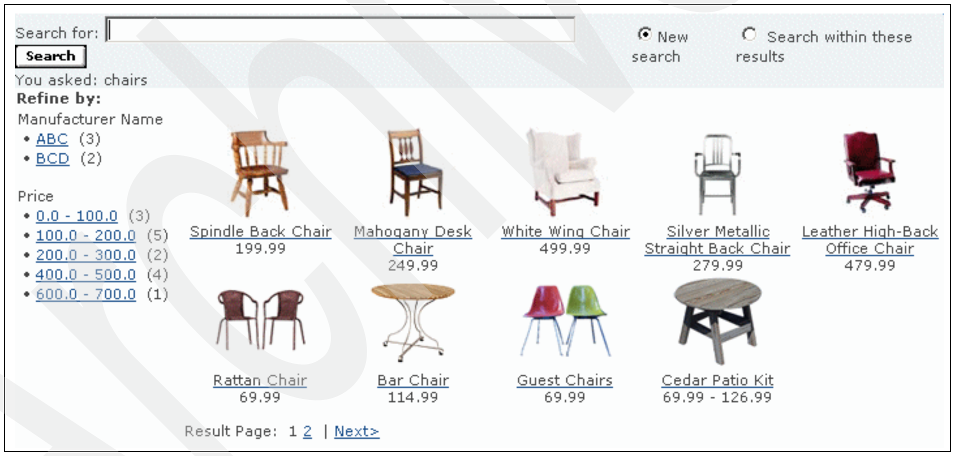


Figure 1-2 Sample page of a Web site powered by OmniFind Discovery Edition

Notice that, on the left navigation panel, the shopper can further narrow searches by manufacturer name or price. The left navigation panel can be configured to narrow searches by other categories.

OmniFind Discovery Edition comes with the following additional components:

- Commerce Module
- Self Service Module
- Contact Centers Module
- Case Resolution Module
- Classification Module

In this book, we focus on Commerce Module for OmniFind Discovery Edition. This business solution helps online shoppers to find the products and services that best match their searches. It helps companies convert shoppers into buyers.

For overview of Commerce Module for OmniFind Discovery Edition, refer to 1.3, “Commerce Module for OmniFind Discovery Edition” on page 13. But first, let us take a brief look at IBM WebSphere Commerce in case you are not familiar with the product.

1.2 Overview of IBM WebSphere Commerce

IBM WebSphere Commerce provides a powerful and flexible infrastructure built on top of IBM industry leading WebSphere platform. It is designed for creating and running complex or high-volume business to business (B2B), and advanced business to consumer (B2C) e-commerce Web sites. WebSphere Commerce is not just providing Web sites. It provides a complete solution for growing companies to improve operational efficiencies and do business on the Web from a simple online store to a fully integrated, multi-channel sales network. It provides a framework for cross-channel business integration. It leverages the great capability of the underlying application server, portal server and business integration capability wherever possible.

WebSphere Commerce can be used for any or all of a company's business models and business access points. It provides a single platform to do business directly with consumers and businesses, indirectly through channel partners, or with all of the above simultaneously. e-commerce is more than just an online shopping cart. It is an integral part of a company’s overall business strategy. It is about doing business in all channels, at the same time supporting consumers and business customers that have become more empowered and demand to do business on their terms — whenever, wherever, and however they want. WebSphere Commerce accelerates the automation and integration of cross-channel marketing and sales processes, helping companies of all sizes become on demand businesses.

Figure 1-3 shows the WebSphere Commerce solution architecture. The two major components of the architecture are the Configurable Business Processes and the Business Context Engine.

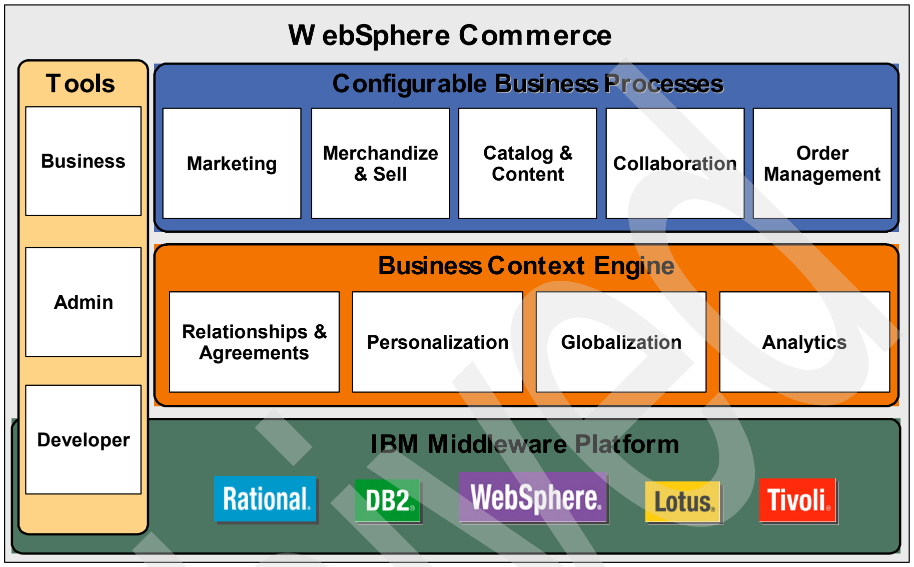


Figure 1-3 WebSphere Commerce solution architecture

The *Business Context Engine* is a key piece of WebSphere Commerce solution. This component is responsible for important elements of the user experience such as personalization, globalization. It is the key element that makes WebSphere Commerce responsive to individual customer requirements. This component also controls the business analytics, customer relationships, and contract agreements.

The *Configurable Business Processes* component provides out-of-the-box business processes such as marketing, merchandising, catalog, collaboration and order management. These processes enable best business practices.

These two major components are built on top of WebSphere platform and leverage the power of the underlying application server and business Integration capability.

Another important component of WebSphere Commerce is the Tools. WebSphere Commerce provides role-based tools for business users, administrators, and developers. The *business tools* and *administration tools* are built as extensions of the WebSphere platform tooling. The *developer tools set* is an extension of the Rational® development environment.

1.2.1 Administration

WebSphere Commerce has three administration tools: the Administration Console that performs the IT related tasks, the Commerce Accelerator that handles the business related tasks, and the Organization Administration Console that allows users to set up organization structure and user roles to perform different business operations.

The *Administration Console* is generally used by the IT personnel. It allows a site administrator to control a Web site or a store. At the site level, a site administrator can publish new stores and configure site-level security policies. At the store level, the site administrator can configure store access management policies.

The *Commerce Accelerator* allows business users and store administrators to maintain online stores and catalogs. For example, a store administrator can use it to manage the appearance and behavior of an online store; a business user can use it to create and maintain orders and track store activities. Users can see and access different menus and options from the home page according to the user roles.

The *Organization Administration Console* allows users to control the organizations that access a site or a store. The tool contains functions that allow users to define access control policies and assign user roles of the site.

Access to all the tools (such as Administration Console, Organization Administration Console and Commerce Accelerator) is based on user roles.

1.2.2 User interface

WebSphere Commerce provides sample stores for each of the supported business models. These sample stores allow you to use the sample organizational structure as the starting point for your company's Web site, or as part of a sample business. The supported organizational structures are Consumer Direct, B2B Direct, Demand Chain, Supply Chain and Extended Sites.

To provide a simple and clear explanation, we use a case study throughout this book to demonstrate WebSphere Commerce functions and capabilities, OmniFind Discovery Edition functions and capabilities, and their integration. We start with the sample Consumer-Direct online store from WebSphere Commerce. This is one of the easiest samples to work with.

Figure 1-4 shows the main page of a sample Consumer-Direct online store powered by WebSphere Commerce.

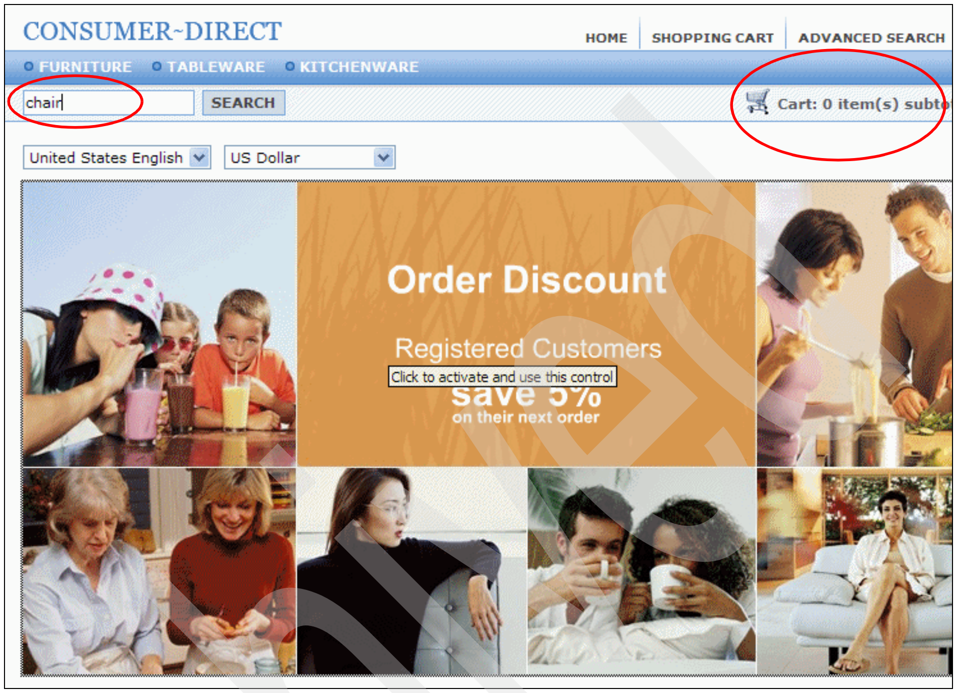
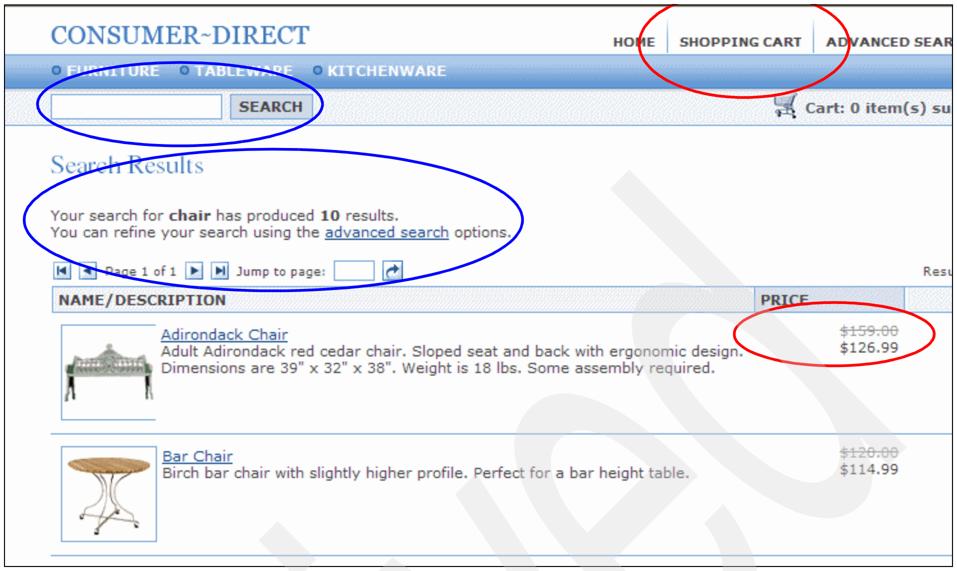


Figure 1-4 Main Page of Consumer-Direct

Consumers can choose from three product categories across the top of the page: Furniture, Tableware, and Kitchenware. Notice the shopping cart status on the top right hand side of the page.

Assuming that a shopper is browsing the online store and wants to search for some chairs, the shopper enters `chair` on the upper left corner in the search box and clicks **Search** or presses **Enter**. The search result page is shown in Figure 1-5.



Search Results

Your search for **chair** has produced **10** results.
You can refine your search using the advanced search options.

NAME/DESCRIPTION	PRICE
Adirondack Chair Adult Adirondack red cedar chair. Sloped seat and back with ergonomic design. Dimensions are 39" x 32" x 38". Weight is 18 lbs. Some assembly required.	~~$159.00~~ $126.99
Bar Chair Birch bar chair with slightly higher profile. Perfect for a bar height table.	~~$120.00~~ $114.99

Figure 1-5 The search result page for "chair"

Notice that, in the search result page, the shopping cart icon is displayed on the top right corner of the page where the shopper can have direct access to the shopping cart. For the returned two products, each product has two prices. One is the retail price, marked with a strike-through, and the other one is the sales price or online price. The page also contains a search box and explains the shopper's search result.

1.3 Commerce Module for OmniFind Discovery Edition

Commerce Module for OmniFind Discovery Edition is a packaged integration of WebSphere Commerce and OmniFind Discovery Edition that helps to speed up the deployment of the integrated solution. Commerce Module integrates OmniFind Discovery Edition search functionality into the WebSphere Commerce deployment. This brings the rich OmniFind Discovery Edition search, navigation and merchandising capabilities into the WebSphere Commerce e-business framework with minimal implementation effort.

The integration has a built-in understanding of the WebSphere Commerce data model. It can automatically acquire the product data that exists in the WebSphere Commerce data model to OmniFind Discovery Edition and it enables quick deployment.

Commerce Module has the following functions and capabilities:

- Uses the same WebSphere Commerce user interface.
- Supports WebSphere Commerce's standard store headers and footers, e-marketing spots, and shopping cart displays within the search results user interface.
- Supports WebSphere Commerce's extended sites and multiple stores hosted in the WebSphere Commerce instance in a single OmniFind Discovery Edition index.
- Supports WebSphere Commerce's contract entitlements in a single OmniFind Discovery Edition index. The integration provides advanced search with a single index for all e-sites or contracts, without special configuration.
- Provides OmniFind Discovery Edition's rich search capabilities, including spelling correction and natural language support.
- Provides OmniFind Discovery Edition's navigational refinement capabilities for seamless browse and search.
- Provides OmniFind Discovery Edition's business rules merchandising capabilities to prioritize or highlight products that can be combined with store location (the last page the user is on) from WebSphere Commerce. For example, when a user searches for pants while on the women's clothing page, the OmniFind Discovery Edition search engine can take advantage of this information to prioritize women's pants for the search result.
- Provides full OmniFind Discovery Edition analytics.
- Provides OmniFind Discovery Edition Layout Editor for rapid user interface prototyping.

The integration is done automatically when you install Commerce Module for OmniFind Discovery Edition. After you install WebSphere Commerce and OmniFind Discovery Edition, you install Commerce Module.

Commerce Module installation does the following operations:

- Inserts a set of JSPs and JARs into WebSphere Commerce store
- Installs the Layout Editor skins into WebSphere Commerce store
- Updates the Struts file.

During the Commerce Module installation, several OmniFind Discovery Edition commands are called to generate features and build the content index. We cover the installation of Commerce Module in Chapter 2, "Environment setup" on page 19.

Advantages of the integration

The strength of WebSphere Commerce is in its e-business infrastructure and tight integration with WebSphere platform and the Rational development platform. It has many desirable online B2B and B2C features such as e-marketing spots, shopping cart display, and contract entitlement. In the area of search, navigation, and display, OmniFind Discovery Edition has a very rich set of functions and features. WebSphere Commerce and OmniFind Discovery Edition complement each other. Commerce Module for OmniFind Discovery Edition comes with tight integration and provides a solid e-business infrastructure to run enterprise business models.

With Commerce Module for OmniFind Discovery Edition, the native access to the WebSphere Commerce product catalog automatically detects the WebSphere Commerce attributes and the other important descriptive content such as names, descriptions, prices, and category, and handles the modeling of the product-item model. Using this data, the integration provides a rich navigational experience immediately with no configuration necessary. These navigation options can be used to refine search results or power navigation across the entire site.

In addition, the integration automatically calculates the sales information stored in WebSphere Commerce and makes it available to the business users, allowing them to incorporate sales data into the search relevancy calculation or create merchandising offers and cross-promotions in such ways as highlighting the best selling products or creating up-sell spots.

User interface layouts incorporate WebSphere Commerce e-spots, shopping carts, and the header and footer of the starter stores. This out-of-the-box capability speeds creation of a Web site.

For B2B customers, OmniFind Discovery Edition automatically understands and respects the WebSphere Commerce contract model. Similarly, for customers using the WebSphere Commerce e-site functionality, OmniFind Discovery Edition also automatically respects the entitlements model of e-sites and provides the shopper only those products from the master catalog that the shopper can access.

The integration empowers business users to handle the business side of the administration work without the involvement of IT personnel. The merchandising manager no longer depends on IT personnel to make changes regarding to special promotions, merchandising, price mark-down, up-sell, and cross-sell spots.

Architecture

Figure 1-6 shows the integration put into the WebSphere Commerce architecture.

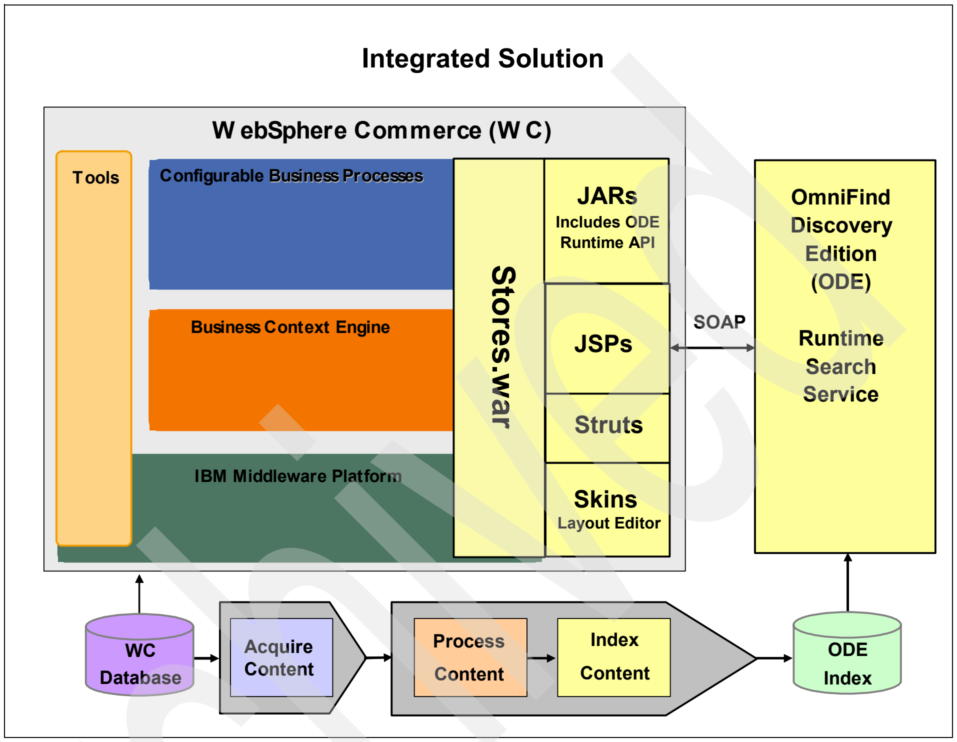


Figure 1-6 Commerce Module for OmniFind Discovery Edition

Terminology:

- JSP:

  Java™ Server Pages (JSP) technology is an extension of servlet (Java classes) technology. A JSP contains servlet and markup (usually HTML) suitable for displaying dynamic information inside a browser.
- WAR:

  A WAR file is a compressed Web application.
- JAR:

  A JAR file is a Java Archive file. It bundles multiple files into a single archive file. Typically a JAR file contains the class files and auxiliary resources associated with applets and applications such as APIs.
- Struts:

  Struts are the framework for building front ends to Java applications that encourage the use of the View and Model approach to presentation layer architecture. The struts configuration file is a link between the View and Model component in the Web Client.

Administration

There is no special administration tool for Commerce Module for OmniFind Discovery Edition. The integration uses the existing administration tools from both products as introduced earlier:

- WebSphere Commerce Administration Console
- WebSphere Commerce Accelerator
- OmniFind Discovery Edition Management Console

You can use different ones for different tasks as described earlier.

2

Environment setup

In this chapter we provide end-to-end installation instructions on how to set up an environment for running Commerce Module for OmniFind Discovery Edition solution.

We cover the following topics:

- Setup overview
- Preparing for installation
- Installing OmniFind Discovery Edition
- Installing WebSphere Commerce Server and the Commerce Module for OmniFind Discovery Edition
- Installing WebSphere Commerce Developer and the Commerce Module for OmniFind Discovery Edition

Note: The description provided in this chapter applies to WebSphere Content Discovery Server V8.3. For OmniFind Discovery Edition V8.4 installation instructions, refer to the publication manual. They are not same.

2.1 Setup overview

WebSphere Commerce provides a set of sample online stores that you can install and set up quickly. OmniFind Discovery Edition provides a powerful search engine that you can integrate into any online stores or Web sites. As mentioned in the previous chapter, the packaged integration, Commerce Module for OmniFind Discovery Edition, allows you to quickly integrate the powerful search engine of the OmniFind Discovery Edition into the WebSphere Commerce stores.

To set up the integrated environment, follow these steps:

1. Prepare for installation (set up a clean system and download software).
2. Install OmniFind Discovery Edition.
3. Install WebSphere Commerce Server and the Commerce Module for OmniFind Discovery Edition

   or

   Install WebSphere Commerce Developer and the Commerce Module for OmniFind Discovery Edition

Two types of environments can be set up for WebSphere Commerce:

- WebSphere Commerce Server on a production environment
- WebSphere Commerce Developer on a development environment

A *production* environment (also called the “live” environment) is installed on Web servers, and is accessible to customers for online shopping activities.

A *development* environment (also sometimes referred to as the “toolkit”) is installed on the developer's machines and is used to customize parts of WebSphere Commerce.

You can choose to install either one of the environments to suit your specific requirements. If WebSphere Commerce Server is installed, you can get a feeling of how the Commerce Module for OmniFind Discovery Edition works, and you can also make use of the tooling available to configure business processes for your store. To customize any store logic, WebSphere Commerce Developer is required. It takes longer time to install WebSphere Commerce Developer but you get the flexibility to customize any store logic of your online store.

This chapter provides instructions to set up OmniFind Discovery Server, as well as instructions to set up WebSphere Commerce. Two sets of instructions are provided for WebSphere Commerce setup, one for the production environment, and one for the development environment.

Note: In this chapter, we show you the *quickest way* to set up your environment from the beginning to the end so you can have a working system.

In addition, to reduce your setup time, we provide the instructions *as simply and minimally as possible*. Our goal is to help you get a working environment up and running quickly. This is not meant to be a set of comprehensive instructions on how to install the system for production usage. Refer to the product manuals for details.

Caution: This set of instructions works well with the product software we obtained at the time of writing. There might be changes in these products. Always check the Web site for the latest software release, fix packs, and installation instructions.

Names: For simplification, we use the following variable names in this book:

- <ODE_HOME>: This stands for:
  C:\IBM\IBM WebSphere Content Discovery Server 8.3

  This represents the home directory where OmniFind Discovery Edition or WebSphere Content Discovery Server is installed.
- <WAS_HOME>: This stands for:
  C:\Program Files\IBM\WebSphere\AppServer

  This represents the application server directory where IBM WebSphere is installed.
- <IPHRASE_HOME>: This stands for:
  C:\IBM\IBM WebSphere Content Discovery Server 8.3
- <WCS_HOME>: This stands for:
  C:\Program Files\IBM\WebSphere\AppServer\profiles\demo\installedApps\WC_demo_cell\WC_demo.ear

2.2 Preparing for installation

Before installing WebSphere Commerce, OmniFind Discovery Edition, and Commerce Module for OmniFind Discovery Edition, perform the following tasks:

- Set up a clean system environment.
- Download the necessary software.

2.2.1 Setting up a clean system environment

It is extremely important for you to start with a clean system in order to set up a trouble-free environment.

Important: Do not use a machine that already has DB2, WebSphere Application Server, or WebSphere Commerce installed. From our experience, simply removing the software *does not* give you a clean system.

We highly recommend that you re-image the system if possible to ensure that you have a clean environment. This saves you many hours of frustration.

If your production system already has certain software and you do not want to re-image the system, remember that the goal of the instructions provided here is to get you set up with a working environment from which you can start customization. The goal is *not* to help you set up a production environment.

Set up a clean system with *only* the following software:

- Windows 2003 Server (Enterprise Edition with Service Pack 1)
- Working Internet Explore browser (6.0 or higher)

Ensure that the following requirements are met:

- Set up a user with all the administrator rights. In our scenario, we set up a user ID, db2admin.
- If you have IIS enabled, disable it. WebSphere Commerce's Quick Installation installs IBM HTTP server as the default server.
- The IP address must be valid in the network, with associated host names in the DSN Server. For details, refer to:

  `http://www.ibm.com/support/docview.wss?uid=swg27007430`
- Do not set up domain for the machine.
- Remove pop-up blockers.
- Disable Internet Explorer® Enhanced Security Configuration:
  a. Go to **Control Panel** → **Add or Remove Programs**.
  b. Click **Add/Remove Windows Components**.
  c. **Internet Explorer Enhanced Security Configuration**.
  d. Click **Next** and then **Finish**.

2.2.2 Downloading the necessary software

Check the Web site for the latest software and fix packs for the required products. IBM Business Partners can contact IBM representatives to obtain the latest software and fix packs or download the software from Partner World Web site if possible:

`https://www.ibm.com/partnerworld/pwhome.nsf/weblook/index.html`

IBMers can download the software from Extreme Leverage at the following site (select **Software downloads** from the Key Tools and Resources drop-down field on the right navigation panel):

`http://w3.ibm.com/software/xl/portal`

Important: The software we have referenced here and used for our book is WebSphere Content Discovery Version 8.3. The latest code level is OmniFind Discovery Edition Version 8.4. If you choose to use the latest version, certain areas and steps are not the same. However, the general concepts and steps should be similar.

For our installation, we download the software to the following directory:

```
C:\InstallMedia
```

We use this directory for the remaining steps of this chapter.

Obtain the following software before you start installation:

- WebSphere Content Discovery Server V8.3 software

  To follow the exercise in this book, use Version 8.3, with Fix Pack 1.
- Commerce Module for OmniFind Discovery Edition Integration package.
- WebSphere Commerce related software

  Assuming that you have access to the Extreme Leverage Web site, use part number CR3MCML (for WebSphere Commerce Developer Enterprise V6.0 eAssembly) and then search and download the following software packages from there:

  – For WebSphere Commerce Server:

    • C900LML - DB2
    • C900MML - WebSphere Application Server
    • C900KML - WebSphere Commerce Enterprise Edition V6

  – For WebSphere Commerce Developer:

    • C81CIML - Rational Application Developer v6.0 part 1
    • C81CJML - Rational Application Developer v6.0 part 2

- C81CKML - Rational Application Developer v6.0 part 3
- C9006ML - WebSphere Commerce Developer Enterprise v6.0
- C900LML - DB2

Note: For your convenience, we provide the instructions to download the packages from Extreme Leverage (Partner World should follow a similar path):

1. Go to the Extreme Leverage Web site:

   `http://w3.ibm.com/software/xl/portal`

2. Click **Software Download** from the left navigation panel.
3. Enter your user ID and password.
4. Select **Find by part number** and enter `CR3MCML` as the part number.
5. Expand **WebSphere Software** → **WebSphere Commerce Developer Enterprise V6.0 eAssembly (CR3MCML)**.
6. Select the following three packages for WebSphere Commerce Server:
   - WebSphere Commerce - WebSphere Application Server Network Deployment V6.0.2.5 for Windows Multilingual (C900MML)
   - WebSphere Commerce Enterprise V6.0 Disk 1 and Disk 2 for Windows, AIX®, Solaris™, iSeries®, Linux on Intel®, Linux on zSeries®, Linux on iSeries and pSeries® Multilingual (C900KML)
   - WebSphere Commerce - DB2 Universal Database™ Enterprise Server Edition and Administrative Client V8.2.3 for Windows Multilingual (C900LML).

   Or select the following packages for WebSphere Commerce Developer:
   - Rational Application Developer V6.0 Windows Part 1 - REQUIRED. Contains core installation files - ESD extractor. Multilingual. (C81CIML)
   - Rational Application Developer V6.0 Windows Part 2 - REQUIRED. Contains core installation files. Multilingual. (C81CJML)
   - Rational Application Developer V6.0 Windows Part 3 - OPTIONAL. Contains WebSphere Application Server V6.0 Integrated Test Environment. Multilingual. (C81CKML)
   - WebSphere Commerce Developer Enterprise V6.0 for Windows Multilingual (C9006ML)
   - WebSphere Commerce - DB2 Universal Database Enterprise Server Edition and Administrative Client V8.2.3 for Windows Multilingual (C900LML).

(See continuation of this note on the following page.)

Note: (continued)

For OmniFind Discovery Edition V8.4 software, download the following two packages:

- Omnifind Discovery Edition V8.4 Windows 2000 Windows Server® 2003 English (C96HXEN)
- Commerce Module for Omnifind Discovery V8.4 AIX 5LV5 Solaris (Sun Microsystems) Windows 2000 Windows Server 2003 Red Hat Linux Suse Linux English (C96I2EN)

For OmniFind Discovery Edition V8.4 Fix Pack 1, you can download it from the following URL:

`http://www.ibm.com/support/docview.wss?rs=3035&uid=swg24015039`

For the latest OmniFind Discovery Edition product support information, go to the following URL:

`http://www.ibm.com/software/data/enterprise-search/omnifind-discovery/support.html`

Important: The software we referenced and used for our book is WebSphere Content Discovery Version 8.3. If you choose to use the latest version (V8.4 with Fix Pack 1), certain areas and steps might not be the same.

Check point

At this point, you should have downloaded all the software:

- WCDS83M

```
WCDS8.3_setup_win32.exe
WCDS_8.3_patch_1_layouts.zip
WCDS_8.3_patch_1_win32.zip
```

- Commerce Module for OmniFind Discovery Edition

```
WCSIntegrationAsset-20060809b.doc
wcsIntegration-v1.1.5-0809b.zip
```

- WCS6EE

```
WC_DB2_ESE_and_Client_Win_V823\DB2 (both client and ese)
WC_WAS_ND_Windows_V6025\WAS6 (disk1 and disk2)
WC_Enterprise_V60\WC60 (disk1 and disk2)
```

Verify that you have all the necessary software before you continue. The integration package number you downloaded is not the same as the ones we show above. Always download the latest packages if they are available.

2.3 Installing OmniFind Discovery Edition

Note: These steps are subject to change with version 8.4 of OmniFind Discovery Edition. At the time of writing, we used version 8.3.1 and the product name then was WebSphere Content Discovery Server.

Follow these steps to install OmniFind Discovery Edition and the patches:

1. Run **`WCDS8.3_setup_win32.exe`** from the following directory:

   C:\InstallMedia\WCDS83M

2. Use default for the rest of the installation *except* the following:

   a. When prompted for the installation type, choose **Custom**.

   b. When prompted for the components to install, select **Turbo Packs** and select **JSP**.

   c. When prompted for the JSP Presentation View installation type, select **Other**. Click **Install**.

3. Install WCDS_8.3_patch_1_win32.zip.

   To install the patch, unzip the file to the home directory where OmniFind Discovery Edition is installed.

   By default, WebSphere Content Discovery Edition Version 8.3.1 is installed in the following directory:

   C:\IBM\IBM WebSphere Content Discovery Server 8.3.

4. Install WCDS_8.3_patch_layout.zip.

   To install the patch, unzip the file to the same directory as above.

5. Change system PATH to include jvm.dll from WebSphere Application Server:

   C:\Program Files\IBM\WebSphere\AppServer\java\jre\bin\classic

2.4 Installing WebSphere Commerce Server and the Commerce Module for OmniFind Discovery Edition

When installing WebSphere Commerce, use *Quick Installation* and default path.

We assume that you download all the files under C:\InstallMedia. To install, follow these steps:

1. Run **setup.exe** from the following directory:

C:\InstallMedia\WCS6EE\WC_Enterprise_V60\WC60\disk1

2. Use default for the rest of the installation and *watch out* for the following input entries:

   a. When prompted for setup type, select **Quick Installation**.

   b. When prompted for the destination path, use default for all of them.

   c. When prompted for the following fields, enter the following values:

      - User ID: `db2admin` (must have administrator authority)
      - User password: `password` (or your choice)
      - Site Administrator ID: `wcsadmin` (or your choice)
      - Site Admin password: `password1` (or your choice)
      - Merchant Key: `0123456789abcdef` (must be 16-digit hex number)
      - Configuration Manager user password: `password1` (or your choice)

      The User ID must have the administrator authority on your Windows system. Windows does not have to be aware of the Site Administrator ID. We show you the user IDs and passwords we use as examples.

      Important: Double check when typing the ID and password. There is no verification here. If you make a typo, you would not be able to log in at the end of the installation.

      We also recommend that you write down your user ID and password information for future reference.

3. When prompted to insert the **IBM DB2 Universal Database Enterprise Server Edition Version 8.2.3 (8.1 FP10) for Windows 2003**, browse to the directory where you downloaded the DB2 software and click **Open**.

   For our environment, we browse to the following directory:

   `C:\InstallMedia\WCS6EE\WC_DB2_ESE_and_Client_Win_V823\DB2`

4. When prompted to insert the **IBM WebSphere Application Server Network Deployment Version 6.0.2.5 for Windows 2003**, browse to the directory where you downloaded WebSphere Application Server software and click **Open**.

   For our environment, we browse to the following directory:

   `C:\InstallMedia\WCS6EE\WC_WAS_ND_Windows_V6025\WAS6`

2.4.1 Publishing a starter store

Once you install WebSphere Commerce successfully, you have to publish at least one starter store in order to develop and test your customization. For this project, we published the following store:

- Consumer-Direct with Housewares as sample data.

Follow these steps to publish the starter store:

1. Start WebSphere Commerce Server by selecting **Start** → **Programs** → **IBM WebSphere** → **Application Server Network Deployment v6** → **Profiles** → **demo** → **Start the server**.
2. Start WebSphere Commerce Administration Console by selecting **Start** → **Programs** → **IBM WebSphere Commerce Server v6.0** → **Administration Console**.
3. Log in with the site administrator ID and password.

   For our example, we use wcsadmin and `password1`.
4. Select Site (Use default).
5. From the top menu, select **Store Archives** → **Publish**.
6. Select **ConsumerDirect.sar**. Click **Next**.
7. For sample data, select **Housewares**. Click **Next** and then click **Finish**.
8. From the top menu, select **Store Archives** → **Publish status** to check whether the store has been published. Wait until the status indicates the starter store is published.
9. Click **Details** → **Launch Store** → **OK**. The store should be launched.
10. Save the URL to favorites in the Web browser for easy launches later.

To publish more stores, repeat the foregoing steps.

Tip: To see the flash on the starter storefront page, install the Flash software. You can download the Flash Player from:

`http://www.adobe.com`

To display all languages on the starter store Web site, install Internet Explorer Multilingual.

2.4.2 Installing Commerce Module for OmniFind Discovery Edition

Commerce Module for OmniFind Discovery Edition does the following operations when you install it:

1. Insert customized JSPs and JARs into existing WebSphere Commerce starter stores.

2. Install the OmniFind Discovery Edition Layout Editor skin into the starter stores.
3. Update the Struts file for the starter stores.

In addition, as part of the installation, you have to perform the following steps manually:

1. Generate features (from WebSphere Commerce data model).
2. Build the content index.

Note: These steps are subject to change with version 8.4 of OmniFind Discovery Edition. At the time of writing, we used version 8.3.1 and the product name then was WebSphere Content Discovery Server.

All installation actions take place under the home directory where OmniFind Discovery Edition is installed, unless we stated otherwise. In our scenario, it is the <ODE_HOME> which represents the following directory (see previous note):

- C:\IBM\IBM WebSphere Content Discovery Server 8.3

Replace the <ODE_HOME> directory in the installation instructions with the directory where you installed your OmniFind Discovery Edition software.

Install Commerce Module for OmniFind Discovery Edition with these steps:

1. Copy Commerce Module:
   a. Create the following directory.

      deployment\wcs

   b. Extract Commerce Module, wcsIntegration-v1.1.5-0809b. Extract the files under a wcs directory.
   c. Copy the entire wcs directory (with all the extracted files) to the deployment\wcs directory, such that you have two wcs directories (deployment\wcs\wcs).
2. Configure WCS_CONNECT and WCS_HOME path in the default.prp file.

   The default.prp file is under the deployment\wcs\wcs directory.

   – For WCS_HOME, set it as follows:

   ```
   WCS_HOME C:\Program
   Files\IBM\WebSphere\AppServer\profiles\demo\installedApps\WC_demo
   _cell\WC_demo.ear
   ```

– For WCS_CONNECT, under #DB2, set it as follows:

```
WCS_CONNECT
jdbc:db2:mall&user=db2admin&password=password&driver=COM.ibm.db2.
jdbc.app.DB2Driver
```

3. Test to ensure that your database connection strings are configured properly:
   a. From a command prompt, go to your <ODE_HOME> directory.
   b. Enter the following command:

```
python\python –O deployment\wcs\wcs\lib\wcs\tools\TestDB.py –p
deployment\wcs\wcs\default.prp
```

Note: The command above has a minus (letter) O, not minus zero.

4. Generate the features:
   a. From a command prompt, go to your <ODE_HOME> directory.
   b. Enter the following command:

```
bin\iphrase build –p deployment\wcs\wcs\default.prp -setProperty
"BUILD_MODE generateFeatures"
```

5. Build the index:
   a. From a command prompt, go to your <ODE_HOME> directory.
   b. Enter the following command:

```
bin\iphrase build –p deployment\wcs\wcs\default.prp -clean
```

6. Shut down WebSphere Commerce in WebSphere Application Server and make sure the OmniFind Discovery Edition server is stopped:
   a. To shut down WebSphere Commerce, at the command prompt, enter:

```
C:\Program
Files\IBM\WebSphere\AppServer\profiles\demo\bin\StopServer.bat
server1
```

   b. To verify that the OmniFind Discovery Edition server is not running, go to the following URL and you should get a connection failure message:

```
http://localhost:8777/query?render=1
```

7. Install the Layout Editor skin into <WCS_HOME>\Stores.war:
   a. From a command prompt, go to your <ODE_HOME> directory.
   b. Enter the following command -- use a minus (letter) O, not minus zero:

```
python\python –O
deployment\wcs\wcs\lib\wcs\tools\InstallSearchUI.py –p
deployment\wcs\wcs\default.prp
```

8. Start WebSphere Commerce as a Windows service.
9. Create two OmniFind Discovery Edition server services, one in production mode and one in edit mode:
   a. Create a service, IBMODEwcs, in *production mode* with two workers, using the following command:

      ```
      bin\iphrase windowsService -p deployment\wcs\wcs\default.prp
      -serviceName IBMODEwcs -install -tool server -args “-port 8777
      -numWorker 2”
      ```

      Important: Make sure you use 8777 for the port, because WebSphere Commerce expects OmniFind Discovery Edition to run on port 8777.

   b. Create another service, IBMODEwcs_mc, to run in *edit mode*.

      ```
      bin\iphrase windowsService -p deployment\wcs\wcs\default.prp
      -serviceName IBMODEwcs_mc -install -tool server -args "-port 8777
      -edit"
      ```

      Note: If you want to use the Management Console, you must bring up OmniFind Discovery Edition server in edit mode. This step creates a service in edit mode.

      Caution: Use the same port number 8777. You can run only one of the services at any time.

Now, you can start the service (IBMODEwcs or IBMODEwcs_mc) manually via Windows service.

2.5 Installing WebSphere Commerce Developer and the Commerce Module for OmniFind Discovery Edition

WebSphere Commerce Developer Edition allows you to create customized code and perform Web development tasks. It builds on top of Rational Application Developer, which is the core development environment from IBM.

This integrated development environment (IDE) includes integrated support for Java components, enterprise beans, servlets, JSPs, HTML, XML, and Web services, all in one development environment. If you plan to use WebSphere Commerce Developer Edition to customize your commerce store, follow the steps outlined in this section to set up the development environment.

Attention: If you do select the option to install WebSphere Commerce Server, then you cannot install WebSphere Commerce Developer on the same machine. SKIP this section if you have followed the steps in section 2.4, “Installing WebSphere Commerce Server and the Commerce Module for OmniFind Discovery Edition” on page 26.

2.5.1 Installing RAD v6.0.1

Install RAD 6.0.1 according to the product manual. We do not repeat the same information here in this book.

2.5.2 Upgrading WebSphere Application Server to v6.0.2

Apply the following patches to upgrade the WebSphere Application Server to 6.0.2:

- C81CJML
- C81CKML
- C81CNML
- C81CSML
- C81CUML

2.5.3 Installing RAD and WebSphere Application Server patches

The recommended fix pack for RAD 6.0.1.1 and WebSphere Application Server v6.0.2.5 are the RAD interim fixes 001 and 002 and the Java SDK Update for IBM WebSphere Application Server test environment.

These updates can be applied through the Rational Product Updater:

- From RAD, select **Help** → **Software Updates** → **Rational Product Updater**.
- From Windows, select **Start** → **Programs** → **IBM Rational** → **Rational Product Updater**.

Or, you can download them and install them locally from:

```
http://www.ibm.com/products/finder/us/finders?sid=56122615116075656674l
&tmpl=%2Fproducts%2Ffinder%2Fus%2Fen%2Ffinders&oldC2=5080624&lc=en&oldC
1=5000583&pg=ddfinder&Ne=5000000&rcss=rtlrad&finderN=1000100&cc=us&coll
ectionN=0&C1=5000583&C2=5348729#dropdowns
```

2.5.4 Set up and update your toolkit build

After installing RAD v6.0.1.1 and the WebSphere Application Server v6.0.2.5 test environment, get the WebSphere Commerce Developer (toolkit) driver and run setup.exe. Follow the wizard to complete the installation.

WebSphere Commerce Developer is configured by default to use Cloudscape™ as a database and has all starter stores published in the Cloudscape database. If you are using the Cloudscape database, skip the sections “Creating bootstrap database and configuring data source” and “Publishing Consumer-Direct store”.

If you are using DB2 as your development database, follow all sections below:

1. Creating bootstrap database and configuring data source.
2. Starting the server.
3. Publishing Consumer-Direct store.
4. Testing.
5. Debugging.

Creating bootstrap database and configuring data source

Attention: SKIP this section if you are using Cloudscape. By default, the toolkit is configured for Cloudscape. You can change the data source at any time by following these instructions.

To create the bootstrap database and configure the data source:

1. Back up the current database. You can use the DB2 backup command to back up the database.

   This is an optional step. Performing this step allows the current database to be fully restored later.

2. Launch a DB2 command line window (db2cmd), go to the <toolkit>\bin directory, and run the following batch file:

   ```
   setdbtype.bat
   ```

   If the createdb parameter is included, this step might take up to one hour to complete! *Do not kill the process*. Also, it is normal for this batch file to end with a FileNotFoundException. Ignore this error for now.

Starting the server

To start the server:

3. Start RAD by entering the following command in the <toolkit>/bin directory:

   ```
   startWCToolkit
   ```

4. In RAD, click **WebSphere Commerce Test Server** in the Servers view tab.
5. Click the green triangle or right-click **Test Server** and select **start** to start the server.
6. This should automatically start the WebSphere Application Server and publish your application. It is fully loaded when you see the following message in the console:

   ```
   Server server1 open for e-business.
   ```

Publishing Consumer-Direct store

You only have to do this if you use DB2. If you use Cloudscape, you have the store already.

Testing

After the server is running, try testing the following links (the login ID and the password are wcsadmin and password1):

- IBM WebSphere Commerce Accelerator:

  https://localhost:8000/webapp/wcs/tools/servlet/ToolsLogon?XMLFile=common.mcLogon

- IBM WebSphere Commerce Administration Console:

  https://localhost:8002/webapp/wcs/admin/servlet/ToolsLogon?XMLFile=adminconsole.AdminConsoleLogon

- IBM WebSphere Commerce Organization Administration Console:

  https://localhost:8004/webapp/wcs/orgadmin/servlet/ToolsLogon?XMLFile=buyerconsole.BuyAdminConsoleLogon

- Consumer-Direct:

  http://localhost/webapp/wcs/stores/servlet/ConsumerDirect/index.jsp

Note: You can get these links by opening <toolkit>/hintsandtips.html.

Debugging

Commerce source has been made available in each of the project JARs. This enables you to debug the code that you do not have in the workspace. If you have to make changes to this code, import the project containing the file.

2.5.5 Using the Commerce Module for OmniFind Discovery Edition

For use with WCToolkitEE60 with embedded Cloudscape, WebSphere Content Discovery Server 8.3M (or OmniFind Discovery Edition v8.4), Windows, using the storefront user interface that comes with WCToolkitEE (for example, ConsumerDirect, AdvancedB2BDirect, ConsumerDirectStorefrontAssetStore, B2BStorefrontAssetStore, ConsumerDirectResellerStorefrontAssetStore, or B2BResellerStorefrontAssetStore), perform the following tasks:

1. Install WCToolkitEE60.
2. Create a store you want to use with OmniFind Discovery Edition in WCToolkitEE60.
3. Test your store by starting WCToolkit and load your store's entry page in a browser.
4. Download WebSphere Content Discovery Edition v8.3 or OmniFind Discovery Edition v8.4 and install it.
   a. Install it by running setup_win32.exe.
   b. Choose **Custom** and install everything except the ASP skins.
   c. Choose **not to setup integration for JSP**.
5. Download and install WebSphere Content Discovery Edition v8.3 fix pack 1 (also known as 8.3M) or OmniFind Discovery Edition v8.4, both for the base software and for layouts:
   a. Install by unzipping both ZIP files to <ODE_HOME> directory.
   b. Verify successful installation by going into the software's home, running `bin\iphrase -version` and verifying that the version number is correct one. For WebSphere Content Discovery Edition v8.3 fix pack 1, it is 8.3.7909.
6. Have a JRE™ or JDK™ 1.4 or later installed (tested with 1.4 and 1.5) or leverage your installed RAD environment, which has this already.

   Also make sure your PATH environment variable includes the jre\bin\client\ directory. For example, if JDK 1.5 is installed, its default location is C:\Program Files\Java\jre1.5.0_06\bin\client. If leveraging JDK from RAD, the path should point to <rad install dir>\runtimes\base_v6\java\jre\bin\classic.
7. Download and unzip the Commerce Module for OmniFind Discovery Edition.
   a. In the OmniFind Discovery Edition's home directory <ODE_HOME>, create a deployment directory.
   b. Under deployment, create a wcs directory.

   c. Unzip wcsIntegration-<date>.zip and place the resulting wcs directory such that it appears in <ODE_HOME>\deployment\wcs\wcs. For example, you should have a file <ODE_HOME>\deployment\wcs\wcs\default.prp.

8. Run QuickStart:

   a. Start a command window.

   b. cd to <ODE_HOME>.

   c. Enter the following command:

   ```
   python\python deployment\wcs\wcs\lib\wcs\tools\QuickStart.py
   ```

   d. When prompted, enter your WebSphere Commerce toolkit directory <WCS_HOME>. In our case, it is C:\WCToolkitEE60. This automatically:

      i. Stops WCToolkit server and configures WCS_HOME in <ODE_HOME>\deployment\wcs\wcs\default.prp. Test the Cloudscape connection.

      ii. Generates OmniFind Discovery Edition features from WebSphere Commerce attributes for faceted navigation and generates OmniFind Discovery Edition indexes.

      iii. Integrates the OmniFind Discovery Edition user interface into the basic WebSphere Commerce storefront and changes their quick search and advanced search JSPs/STRUTs registrations.

      iv. Starts WCToolkit server and starts an OmniFind Discovery Edition server in command line mode.

9. If all succeeds, you should be able to go to your store in WebSphere Commerce, enter a search term via quick search or advanced search, and see the result pages generated by OmniFind Discovery Edition.

To stop the OmniFind Discovery Edition server, use **Ctrl-C** in the command window.

To start the OmniFind Discovery Edition server in command line mode:

1. Go to the <ODE_HOME> directory.
2. Run the following command:

   ```
   bin\iphrase server -p deployment\wcs\wcs\default.prp -nocache
   ```

If you have to update OmniFind Discovery Edition indexes:

1. Stop OmniFind Discovery Edition server (use **Ctrl-C** in command window).
2. Stop WebSphere Commerce.
3. Go to the <ODE_HOME> directory.
4. If you have changed attributes, rerun the following steps, otherwise skip them:

a. Run the following commands:

```
bin\iphrase build -p deployment\wcs\wcs\default.prp -setProperty
"BUILD_MODE generateFeatures"
bin\iphrase build -p deployment\wcs\wcs\default.prp
```

b. Start OmniFind Discovery Edition server.

c. Start WebSphere Commerce.

For more details on the integration, obtain the latest documentation. There are also step-by-step instructions of QuickStart should you have to install into a non-WCToolkitEE60 environment.

OmniFind Discovery Edition and WebSphere Commerce integration information can be found at the following URL:

http://btvgsa.ibm.com/home/j/p/jpayne/web/public/index.html

There are charts and flash demos there. You can also download OmniFind Discovery Edition from this page instead of downloading it from Extreme Leverage Web site.

3

Working with the Commerce Module for OmniFind Discovery Edition

In this chapter we describe how the Commerce Module for OmniFind Discovery Edition works out-of-the-box, and we introduce the customized application we developed for this book. This is not intended to be a comprehensive walk-through of all the capabilities and features that the integrated system can offer. Rather, its purpose is to quickly get you familiar with navigation through the existing environment.

We cover the following topics:

- ► Out-of-the-box Commerce Module
- ► Customization

3.1 Out-of-the-box Commerce Module

In Chapter 2, “Environment setup” on page 19, we provide steps to set up the Commerce Module for OmniFind Discovery Edition environment. If you follow these steps, you would have a working environment with:

- WebSphere Commerce installed
- A sample Consumer-Direct starter store published
- OmniFind Discovery Edition installed
- Commerce Module for OmniFind Discovery Edition installed.

In this section, we show you how to work with Commerce Module for OmniFind Discovery Edition. We want you to become familiar with the environment quickly so you can start planning and customizing your own application. We also show you how our customized application, which we developed for this book, looks. We describe the differences between the out-of-the-box version and the customized version. In later chapters of this book, we show you how we customize the appearance and behavior of our sample store.

The Consumer-Direct store is a starter store that is shipped with WebSphere Commerce. When you access the starter store, you see the appearance and behavior of a typical, out-of-the-box WebSphere Commerce store.

With OmniFind Discovery Edition integrated in the WebSphere Commerce store, the solution retains the WebSphere Commerce appearance and behavior because the usage of OmniFind Discovery Edition should be transparent to shoppers. When a shopper submits a text search or clicks a link that refines catalog navigation, a search request is submitted to OmniFind Discovery Edition server and the results are displayed in a search result page. The search result page in the integrated solution has a list of search features and capabilities provided by OmniFind Discovery Edition, including: refine-by list, voiceover, breadcrumbs, stopwords, and spell check. The search result page adopts the WebSphere Commerce appearance and behavior; shoppers get the same presentation experience.

While the integration enhances shoppers’ search experience on WebSphere Commerce, it retains the basic and advanced functions provided by WebSphere Commerce. For example, with OmniFind Discovery Edition integrated, the store can still define content spots to display marketing campaigns and promotions; the checkout process also remains the same.

3.1.1 Navigation through the integrated system

Figure 3-1 shows the main page of the out-of-the box, Consumer-Direct starter store that is created from Chapter 2, “Environment setup” on page 19.

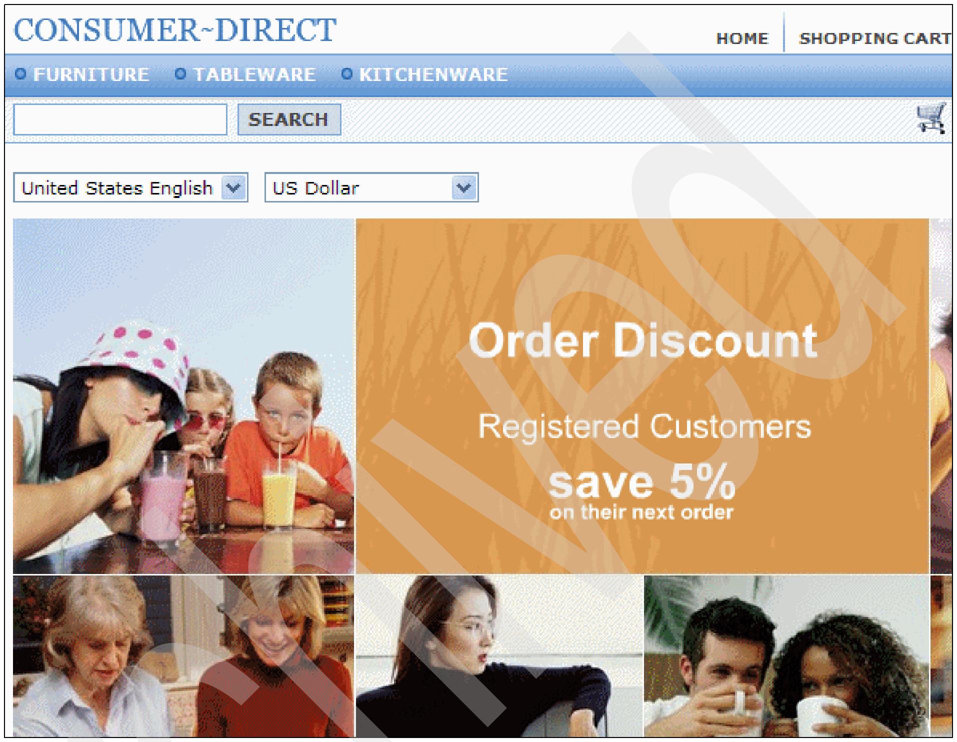


Figure 3-1 Consumer-Direct store: Main page (the same as the default Commerce page)

There are three categories on the top menu:

- Furniture
- Tableware
- Kitchenware

This page looks exactly like a normal WebSphere Commerce starter store. Shoppers can click any of the top menu choices to drill down the categories.

For demonstration purposes, we want to see what furniture the store sells. To see the furniture, click **Furniture**. The Furniture page appears as shown in Figure 3-2.

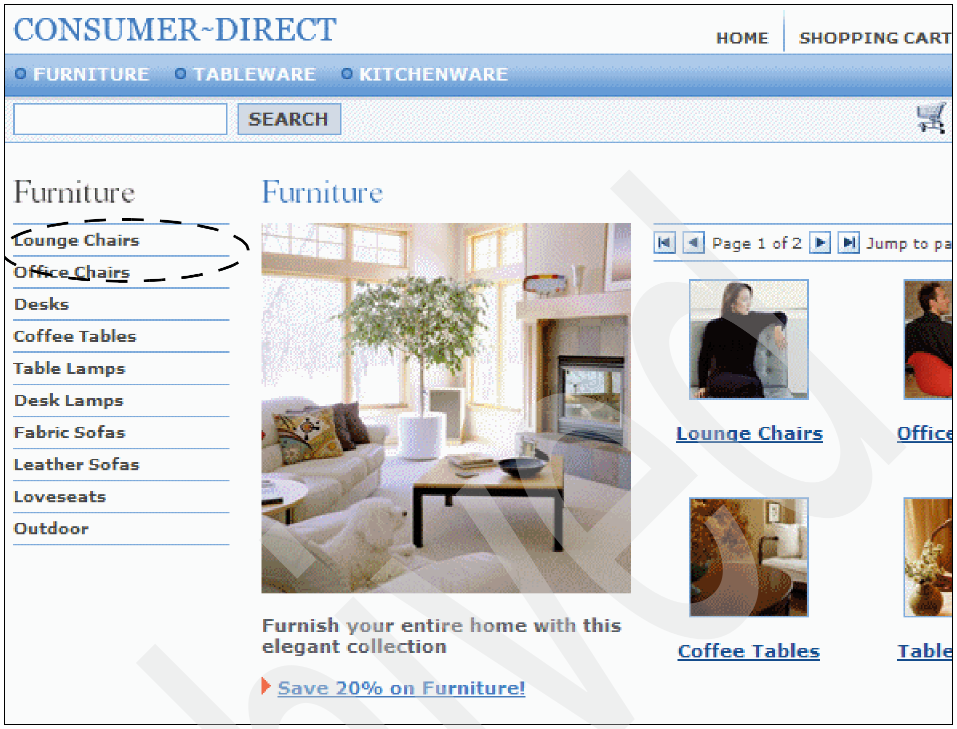


Figure 3-2 Consumer-Direct store: Furniture page (the same as the default Commerce page)

At the Furniture page, there are different types of furniture shown at the left navigation panel:

- Lounge Chairs
- Office Chairs
- Desks
- Coffee Tables
- Table Lamps
- ...

Different types of furniture are also displayed as icons on the right panel of the page. In the middle of the page, there is a special advertising space to attract the shopper's attention.

This page retains the same default appearance and behavior of a WebSphere Commerce starter store page. Shoppers can navigate to the subcategories of the products (in this case, different types of furniture) by using the left side navigation panel or right side icons or associated links.

For demonstration purposes, we want to see what lounge chairs are available on sale at this store. To navigate to view the available lounge chairs, click **Lounge Chairs**. The Lounge Chairs page appears as shown in Figure 3-3.

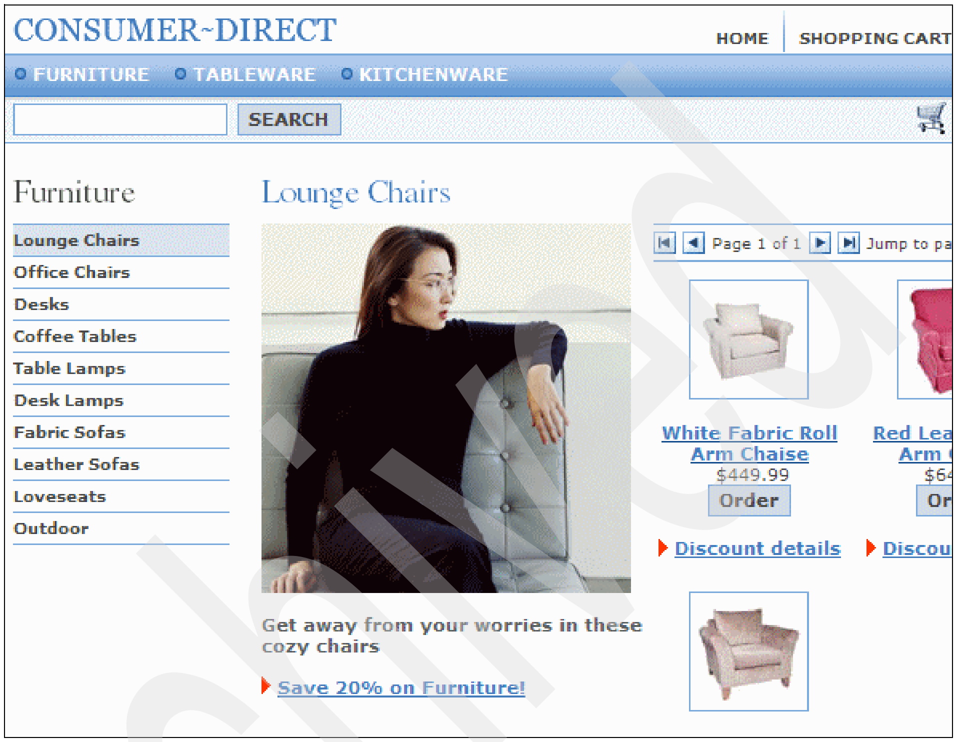


Figure 3-3 Consumer-Direct store: Lounge Chairs page (the same as the default Commerce page)

In the Lounge Chairs page, the left navigation panel shows the same navigation options as the earlier page. It displays different types of furniture the store sells.

On the right panel, it shows the products available under the Lounge Chairs category. For each product type, there is an icon for the product, a description of the product under the icon, the price for the product under the description, and an **Order** button that you can click to start the ordering process.

At this time, the page still retains the same default appearance and behavior of a WebSphere Commerce starter store page. Shoppers can navigate to other types of furniture using the left navigation panel options, or click the icon or description of the product to see more details of the product from the right panel, or click **Order** to order the specific product, or click the icon or link in the middle of the page that shows special promotion items.

When shoppers navigate a WebSphere Commerce store using the navigation menus and links, it looks and feels like a typical WebSphere Commerce store. The underlying code behind the navigations and links are powered by WebSphere Commerce code. When a shopper performs a search, OmniFind Discovery Edition takes over the action; it is responsible for performing the search and returning the search result page to the online store Web site.

For demonstration purposes, we want to see how search works in the integrated system. We perform a search on the main page of the starter store instead of navigating to the page using its menus and links. Navigate to the main page by clicking **Furniture** from the top menu. Enter the following term in the search box of the main page and then click **Search** or the **Enter** key:

```
lounge chiars
```

Notice that we deliberately misspelled the word *chiars*. See Figure 3-4.

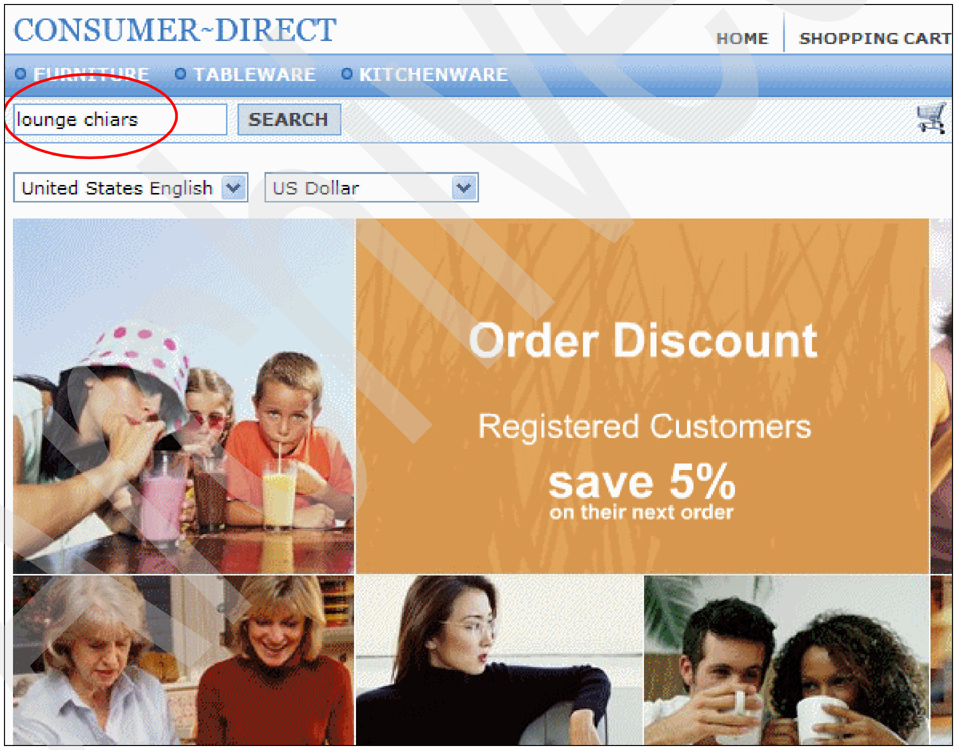
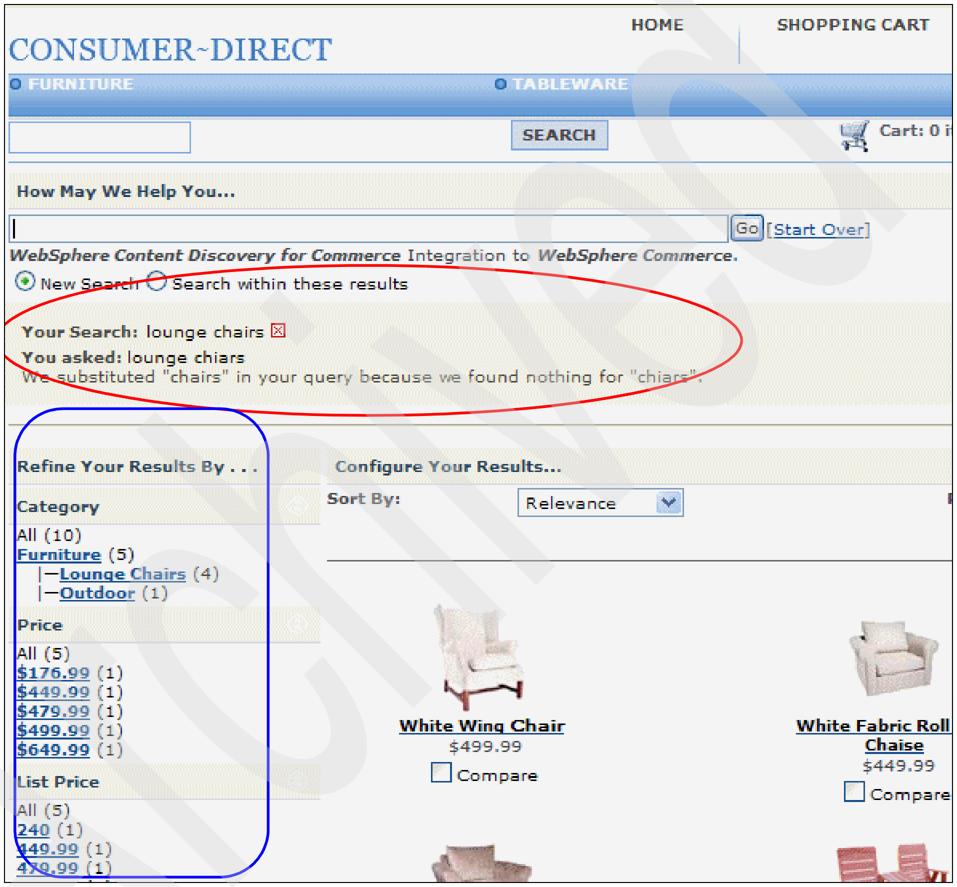


Figure 3-4 Consumer-Direct store: Search for lounge chairs (misspelled "chiars")

As mentioned earlier, when one performs a search from the starter store, OmniFind Discovery Edition takes over. The OmniFind Discovery Edition search performs the search based on the search input.

In this scenario, OmniFind Discovery Edition re-spelled the word *chiars* to *chairs* before the search engine searches for products related to *chairs*. Figure 3-5 shows the result page from this search.



How May We Help You...

WebSphere Content Discovery for Commerce Integration to *WebSphere Commerce*.

Your Search: lounge chairs

You asked: lounge chiars

We substituted "chairs" in your query because we found nothing for "chiars".

Refine Your Results By . . .

Category

All (10)
Furniture (5)
|—Lounge Chairs (4)
|—Outdoor (1)

Price

All (5)
$176.99 (1)
$449.99 (1)
$479.99 (1)
$499.99 (1)
$649.99 (1)

List Price

All (5)
240 (1)
449.99 (1)
479.99 (1)

Configure Your Results...

Sort By: Relevance

White Wing Chair
$499.99
Compare

White Fabric Roll Chaise
$449.99
Compare

Figure 3-5 Consumer-Direct store: Result page from a search on “lounge chairs”

The search result page, as shown in Figure 3-5 on page 45, showcases some of the powerful features of OmniFind Discovery Edition:

- Voiceover - A confirmation of a user's most recent search request:

  ```
  You asked: lounge chiars
  ```

- Breadcrumb - Displays user's current search constraints. States what the user has requested for the system to produce the search result:

  ```
  Your search: lounge chairs
  ```

- Respelling - When a search word is not recognized, respell the word to a recognizable one:

  ```
  We substituted "chairs" in your query because we found nothing for
  "chiars"
  ```

Not only can OmniFind Discovery Edition automatically respell the unrecognized words, it also can display what is actually searched and explain why the words are respelled. The voiceover, breadcrumb, and respelling features are optional. You can configure the system to turn them on or off. You can also configure the system to add new acceptable words in the dictionary.

In the upper left corner, notice that the system asked the following question:

```
How May We Help You ...
```

One of the great strengths of OmniFind Discovery Edition is its ability to process natural language. Shoppers can type in a sentence to search for the products. Shoppers can also search within the results.

On the left navigation panel, shoppers can refine searches by product category or product price. This is also another powerful feature of Commerce Module for OmniFind Discovery Edition. With OmniFind Discovery Edition, normal searches can be narrowed down by product features and feature values or value ranges.

3.2 Customization

In the previous section, we showed you the appearance and behavior of the out-of-the box default integrated environment of WebSphere Commerce and OmniFind Discovery Edition. In this section, we show you the appearance and behavior of the customized application. For the remaining chapters of the book, we show you how to develop the necessary codes for the customization.

3.2.1 Navigation through the customized application

Figure 3-6 to Figure 3-8 show the main page of the Consumer-Direct starter store. Figure 3-6 shows some of the new top menu options we added, for example, By Brand. Figure 3-7 and Figure 3-8 show the drop-down menus of two new top menus, Shop by Occasion and Shop by Recipient.

From navigation perspectives, for the customized application:

- We added the following top menu selections:
  - By Brand
  - Shop by Occasion (with the drop-down menu selections)
  - Shop by Recipient (with the drop-down menu selections)
  - New Arrival
- We modified the following existing top menu selections:
  - Living&Entertaining (renamed it from the Furniture menu selection)
  - Kitchen&Dining (merged Tableware and Kitchenware menu selections)

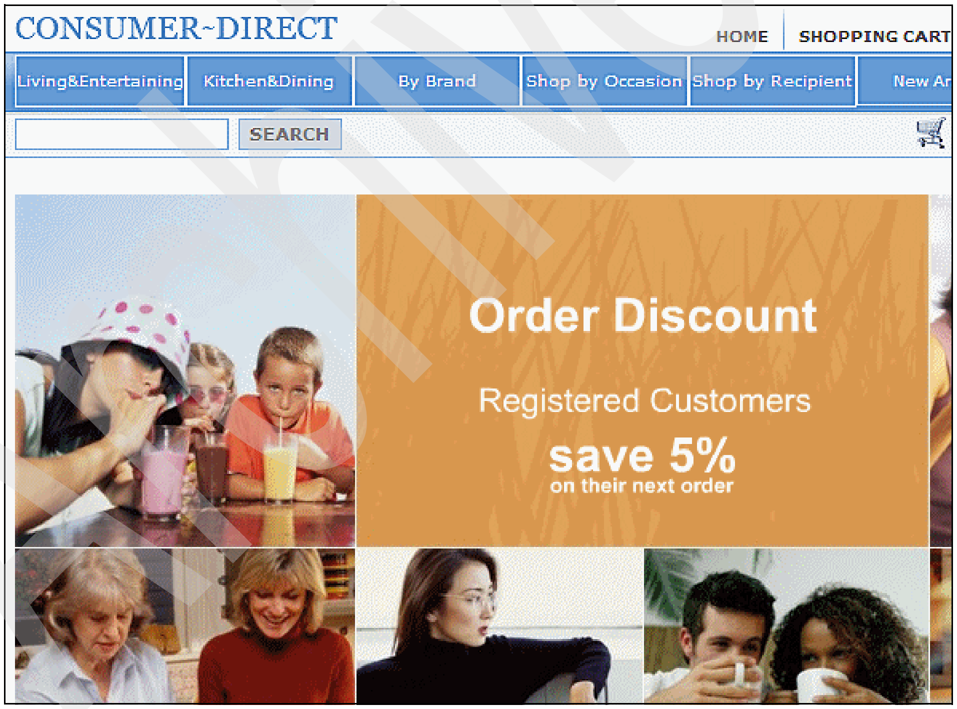


Figure 3-6 Customized store: Main page (new drop-down menu and new top menu)

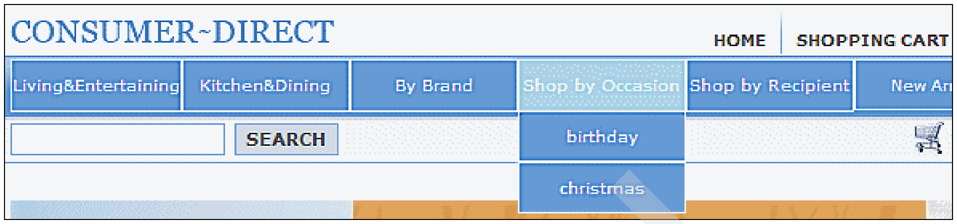


Figure 3-7 Customized store: New top menu, Shop by Occasion

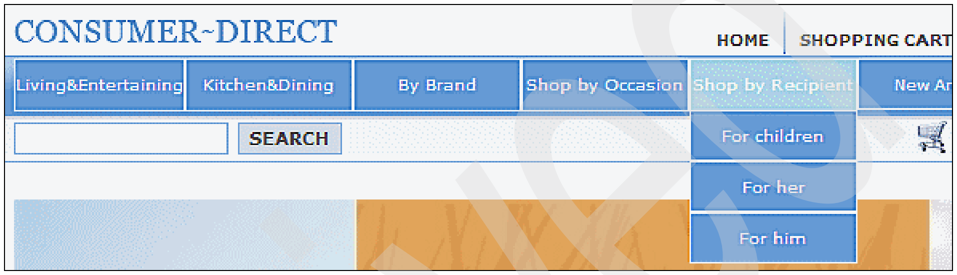


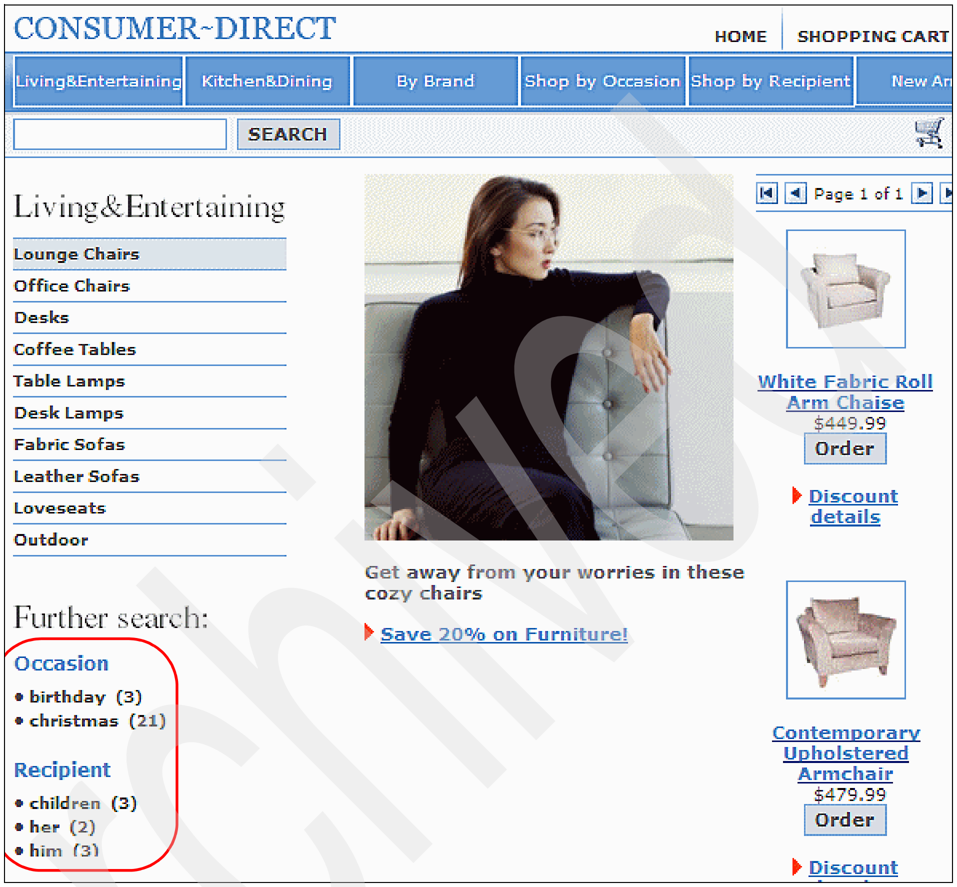
Figure 3-8 Customized store: New top menu, Shop by Recipient

Using the menu selection, navigate to the Lounge Chairs category by moving the mouse over **Living&Entertaining** and then clicking **Lounge Chairs**. Figure 3-9 shows the Lounge Chairs page. Notice that the lower left navigation panel shows the Refine By navigation options.

The default integrated system (Commerce Module for OmniFind Discovery Edition) only displays the Refine By option when a shopper performs a search. In the customized application, with normal menu navigation, the system displays the OmniFind Discovery Edition's search features whenever applicable.

In this particular scenario, we navigate to the page using the menu selection. Although shoppers have not performed a search task, the page shows the Refine By navigation option, which allows shoppers to narrow down the products. We show you how this is done in later chapters of this book using OmniFind Discovery Edition APIs.

Other Refine By navigation options that are not shown in Figure 3-9 include the Refine By manufacturer name and category.



Living&Entertaining

- Lounge Chairs
- Office Chairs
- Desks
- Coffee Tables
- Table Lamps
- Desk Lamps
- Fabric Sofas
- Leather Sofas
- Loveseats
- Outdoor

Get away from your worries in these cozy chairs

Save 20% on Furniture!

White Fabric Roll Arm Chaise
$449.99
Order
Discount details

Contemporary Upholstered Armchair
$479.99
Order
Discount

Further search:

Occasion
- birthday (3)
- christmas (21)

Recipient
- children (3)
- her (2)
- him (3)

Figure 3-9 Customized store: Lounge Chairs page, new Refine By navigation

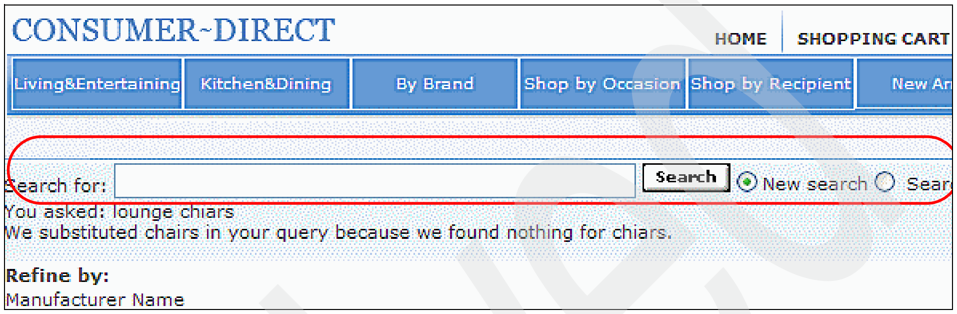
In addition to navigation modification, we also customize the search result page of the application.

To perform a search for the lounge chairs, we repeat the same steps as in the previous section by entering the following words in the search box and clicking **Search** or pressing the Enter key:

```
lounge chiars
```

Notice again that we type a misspelled word “chiars”.

Figure 3-10 shows the search result page. The voiceover and respelling are the same as Figure 3-5 on page 45 in the out-of-the box integrated solution. The difference is that, in the out-of-the box integrated solution, there are two search boxes. One is a generic search box provided by WebSphere Commerce, and the other is an additional one provided by OmniFind Discovery Edition's default search page (after you perform a search). In the customized application, we remove the duplicated search boxes. What you see is only one search box.



You asked: lounge chiars
We substituted chairs in your query because we found nothing for chiars.

Figure 3-10 Customized store: Result page (one search box)

In the default application, there is also an IBM logo on the top right hand corner (not shown in Figure 3-5 on page 45). In the customized application, we remove the IBM logo. This provides a cleaner presentation.

There are other customizations that we have done for this application. What we describe here are just some of them. In Chapter 4, "Customization overview" on page 51, we show you what exactly we customized, and in later chapters of this book, we show you how and where we modified the code and how we do each customization, one at a time.

4

Customization overview

In this chapter we provide an overview of the customization we have done for the sample starter store we set up in Chapter 2, “Environment setup” on page 19. In Chapter 3, “Working with the Commerce Module for OmniFind Discovery Edition” on page 39, we show you some of the screen captures as we navigate through the customized application.

In this chapter, we concentrate on the technical details behind the sample store and the customization. In later chapters of this book, we show you how and where in the source code we make the modifications to get customization.

We cover the following topics:

- Customization summary
- Catalog structure
- Site flow
- Page outline

4.1 Customization summary

Using OmniFind Discovery Edition APIs, you can customize search and navigation of your Commerce stores. For this book, we perform the following customization for our starter store to show you how each can be done:

- Add By Brand view feature.
- Add displaying new products in your site feature.
- Add gift idea pages to your web site.
- Customize category display page.
- Customize search result page.

For information about OmniFind Discovery Edition APIs, refer to Chapter 5, “Introducing OmniFind Discovery Edition APIs” on page 67. For information about our customization, refer to the subsequent chapters afterwards.

4.2 Catalog structure

In order to show different customizations that you can do with your store, we modify the existing catalog structure of the starter store that comes with WebSphere Commerce to create multiple levels of categories.

The default, out-of-the-box data in the sample store comes with two levels of categories. The new data structure used for our customized store contains three levels of categories. We restructure the categories as follows:

- Living & Entertainment
  - Lounge Chairs
  - Office Chairs
  - ...
- Kitchen & Dining
  - Tableware
    - Plates
    - Silverware
    - ...
  - Kitchenware
    - Scented Oils
    - Cooking Oils
    - ...

Figure 4-1 outlines this catalog structure used for our customized store in this book.

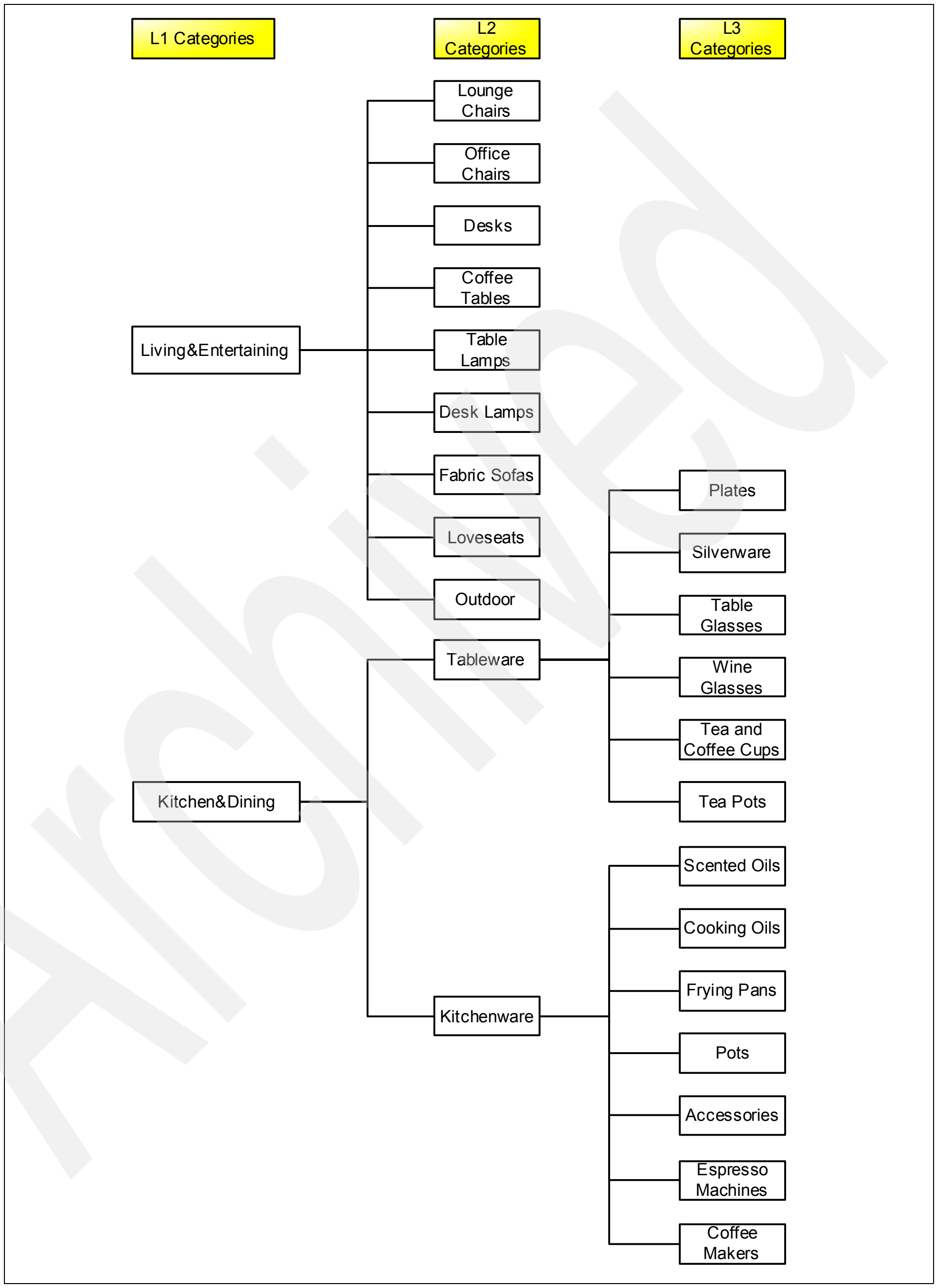


Figure 4-1 Customized store: Catalog structure

Notice that the products of the store are at the lowest level of the catalog structure, and they are not shown in the diagram.

The data preparation for the revised catalog structure is covered in 6.2.4, “Create three-level catalog structure” on page 99.

4.2.1 New attributes

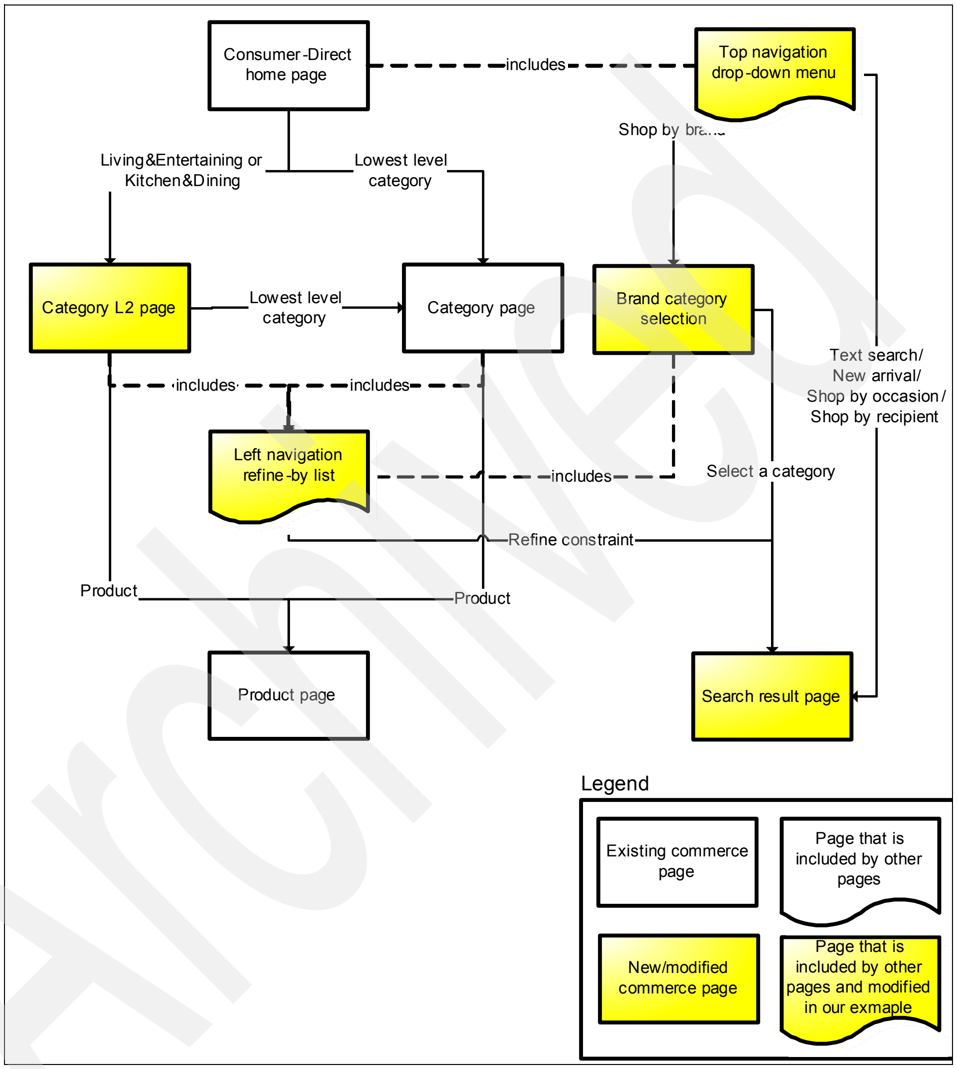
For our customization, we add three additional attributes to the existing Commerce Module for OmniFind Discovery Edition:

- *New* attribute, used to distinguish new arrival products
- *Recipient* attribute, used to support Shop by Recipient enhancement
- *Occasion* attribute, used to support Shop by Occasion enhancement

The data preparation steps for these attributes are covered in the later chapters of the book.

4.3 Site flow

Figure 4-2 outlines the site flow of our customized store.



Consumer-Direct home page
includes
Top navigation drop-down menu
Shop by brand
Living&Entertaining or Kitchen&Dining
Lowest level category
Category L2 page
Lowest level category
Category page
Brand category selection
Text search/ New arrival/ Shop by occasion/ Shop by recipient
includes
includes
Left navigation refine-by list
includes
Select a category
Refine constraint
Product
Product
Product page
Search result page
Legend
Existing commerce page
Page that is included by other pages
New/modified commerce page
Page that is included by other pages and modified in our exmaple

Figure 4-2 Customized store: Site flow

The legends in Figure 4-2 show what pages are the existing WebSphere Commerce pages, what pages are new or modified pages due to our customization, and what pages are included by other pages.

The *Consumer-Direct home page* is the home page of the online store. It is the entry point to the customized store.

The top navigation drop-down menu and left navigation refine-by list page are universal JSPs that are included by other pages of the store.

The *top navigation drop-down menu* displays level 1 (L1) categories from the catalog. In our scenario, they are Living&Entertaining and Kitchen&Dining. When a shopper moves the mouse over a level 1 category, the subcategories (level 2 categories) are displayed in the drop-down list. The top navigation drop-down menu also contains links for shoppers to shop by brand, shop by occasion, shop by recipient, and search for new arrival products. We customize this page for our starter store.

The *left navigation refine-by list* displays features that are relevant to the current selection and allows shoppers to refine the current search. We customize this page and add it to all the category pages.

After a shopper selects a category, either the category level 2 (L2) page is displayed, or the category page is displayed. If a lowest level category is selected, the category page is displayed; otherwise the category L2 page is displayed. In our customized sample store, category L2 page is displayed if either Living&Entertaining or Kitchen&Dining category is selected.

The *category L2 page* shows all the subcategories (level 3 categories) and one product from each subcategory. Shoppers can choose to see all products under one of the subcategories or select the highlighted product at this page. We customize this page for the starter store.

The *category page* shows all products under the selected category. Clicking one of the products takes you to the product detail page. The *product detail page* is the page that WebSphere Commerce provides by default. It is not customized for our starter store.

If a shopper selects to shop by brand, the brand category selection page is displayed. The *brand category page* shows a list of categories that are available for the selected brand. Shoppers can further refine their navigation by selecting a category under the brand category selection page, and the search result page is displayed showing all matching products. We create this page for our starter store.

If a shopper selects *shop by occasion*, a list of available occasions is shown in the top navigation drop-down menu. After the shopper selects an occasion, the search result page shows a list of all matching products.

If a shopper selects *shop by recipient*, a list of available recipients is shown in the top navigation drop-down menu. After the shopper selects a recipient type, the search result page shows a list of all matching products.

If a shopper selects the new arrival link from the top navigation drop-down menu, the search result page is displayed showing all new products.

The *search result page* shows a list of products that matches a search. We customize this page for the starter store.

4.4 Page outline

In this section, we describe what each page contains from a high level point of view.

4.4.1 Consumer-Direct home page

The Consumer-Direct home page is the entry page for our customized starter store. For our customization, we do not directly change this page, but we modify the top navigation page, which is included by the home page. Refer to the top navigation drop-down menu section for more details.

Figure 4-3 outlines the Consumer-Direct home page.

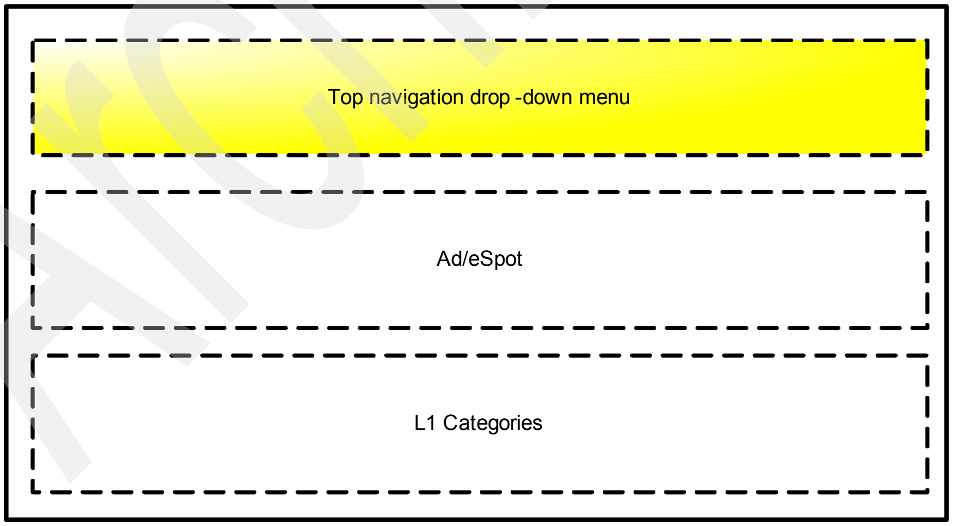


Figure 4-3 Customized store: Consumer-Direct home page

4.4.2 Top navigation drop-down menu

Figure 4-4 outlines the top navigation drop-down menu. This page is included in the Consumer-Direct home page.

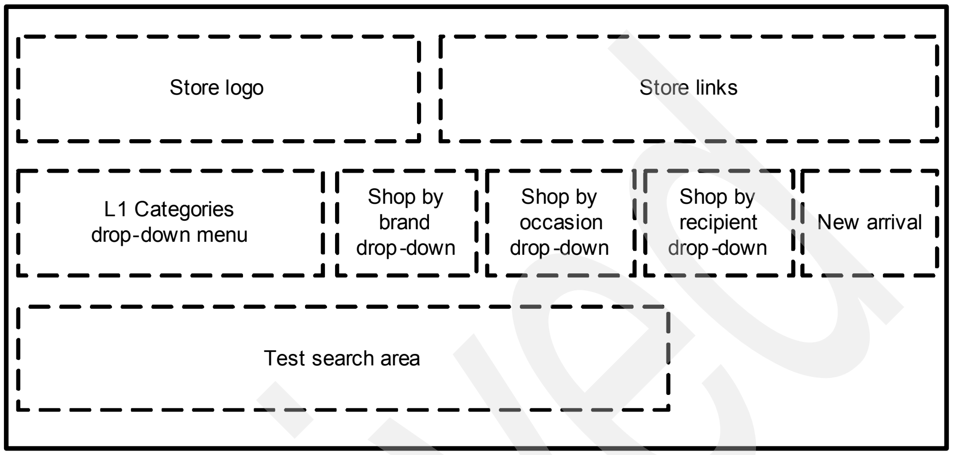

Store logo
Store links
L1 Categories drop-down menu
Shop by brand drop-down
Shop by occasion drop-down
Shop by recipient drop-down
New arrival
Test search area

Figure 4-4 Customized store: Top navigation drop-down menu

4.4.3 Left navigation refine-by list

Figure 4-5 outlines the left navigation refine-by list page.

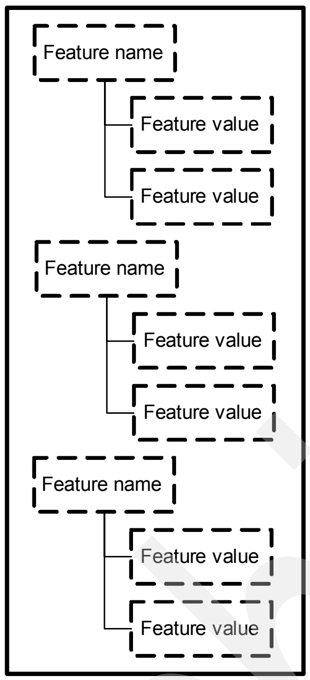


Figure 4-5 Customized store: Left navigation refine-by list

The left navigation refine-by list page is included in the category L2 page, category page, and the brand category page.

4.4.4 Category L2 page

We modify the page (CachedCategoriesDisplay.jsp) that shoppers see after they enter the home page of the store (CachedTopCategoriesDisplay.jsp) and then use Living&Entertaining or Kitchen&Dining as the top category.

On the left side of the window, underneath the menu panel that allows the user to select a subcategory of the top category, we add the left navigation refine-by list where shoppers can narrow down the products they want to select.

On the right side of the window, the original WebSphere Commerce has a panel showing an icon for each subcategory of the top category. We replace each subcategory icon with an image of a particular product in that subcategory. Under the image of the product we put the name and price of the product.

If a shopper clicks the image, name, or price of the product, it takes the shopper to the product page that displays that particular product. This allows the shopper to quickly access the most popular product or specially advertised product for that category before having to go down to the lower category page. Above and below these product items, we put the name of the subcategory. If the shopper clicks the name of the subcategory, it shows the category page that displays all product items in that subcategory.

Figure 4-6 outlines the Category L2 page.

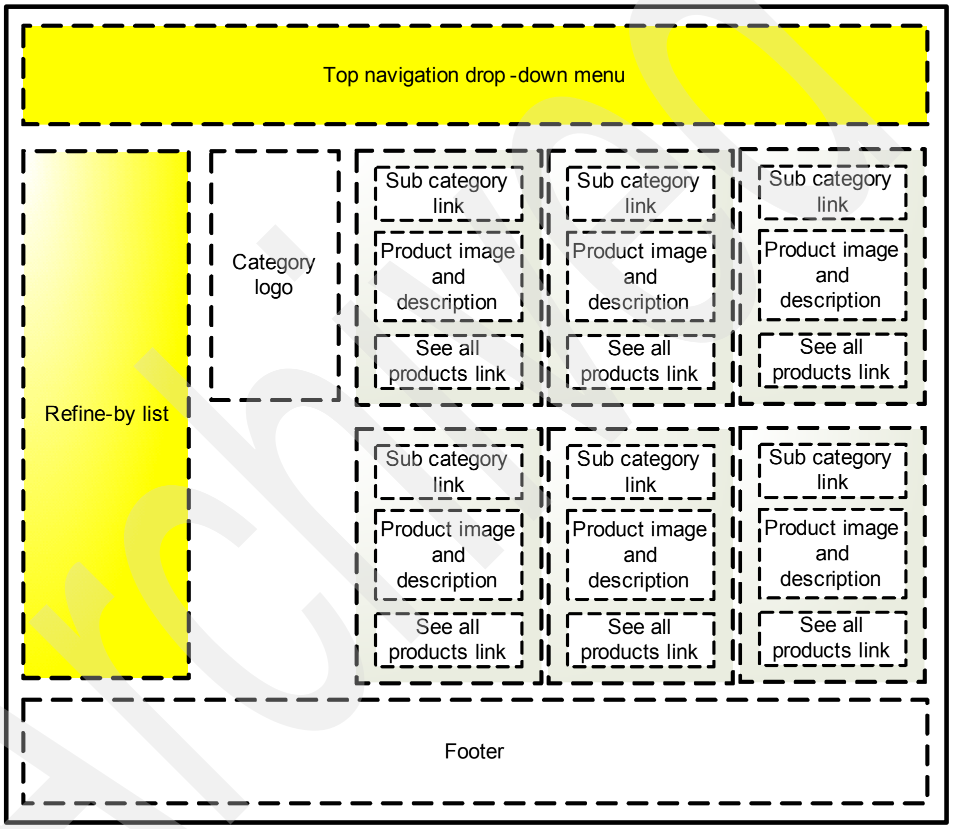


Figure 4-6 Customized store: Category L2 page

4.4.5 Category page

In our customized store, we add an include statement to the category page to include the left navigation refine-by list component. Other elements of the page are left untouched.

Figure 4-7 outlines the category page.

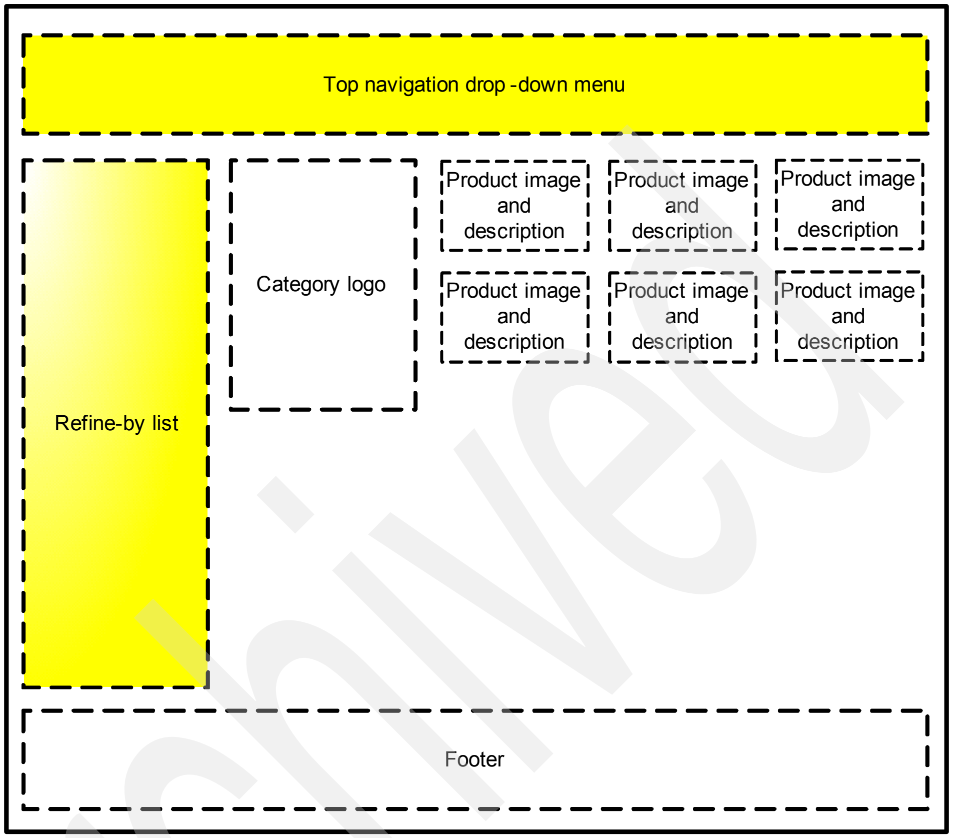

Top navigation drop -down menu
Refine-by list
Category logo
Product image and description
Product image and description
Product image and description
Product image and description
Product image and description
Product image and description
Footer

Figure 4-7 Customized store: Category page

4.4.6 Brand category selection page

For the customized starter store, we add a new JSP page for view by brands. Figure 4-8 outlines the brand category selection page.

There is a drop-down menu that lists all existing manufacturers in the OmniFind Discovery Edition content index. We include the left navigation refine-by list component where shoppers can narrow down their product searches. On the right side of the window, we display the main content about categories under the shopper-selected brand.

With this piece of customization example, you learn how to retrieve an existing feature from OmniFind Discovery Edition index, and how to pass multiple constraints to the search result page.

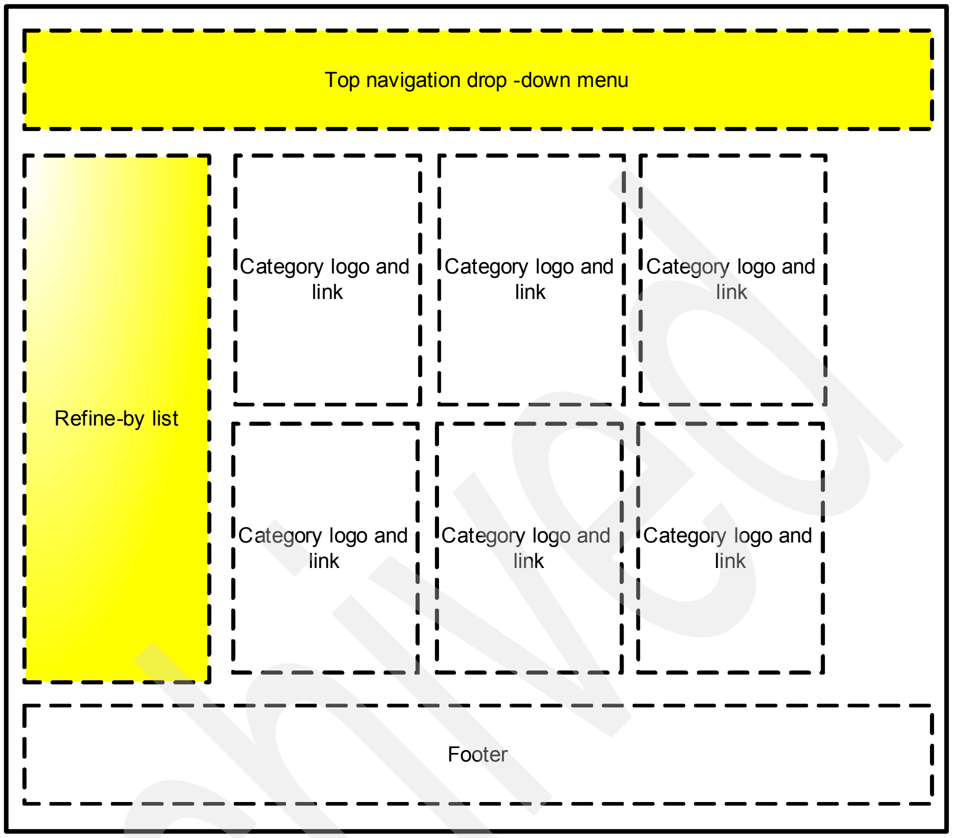


Figure 4-8 Customized store: Brand category selection page

4.4.7 Search result page

The search result page provides a rich set of search and navigation features of OmniFind Discovery Edition within a WebSphere Commerce store. It displays the result items from a shopper's search. It also functions as a glue between the WebSphere Commerce and OmniFind Discovery Edition servers.

Figure 4-9 outlines the search result page, which includes these features:

- *WCSHeader* and *WCSFooter* come with the out-of-the-box Commerce Module for OmniFind Discovery Edition.
- *globalForm* renders the search text area along with radio buttons.
- *voiceOver* renders text based on subsequent actions with the search page.
- *refineByList* provides you with a list of features with which shoppers can narrow down their search results.

- *drillSideWays* provides breadcrumbs to dynamically add and remove features which have range values.
- *pageContext* provides navigation capabilities across multiple pages of features to be refined if they are over one page listing.
- *mainResults* renders the product images, price, and name that match the search criteria.
- *pagination* provides page navigation controls within the mainResults.
- *viewByOptions* provides sort-by, group-by, and view-by capabilities.

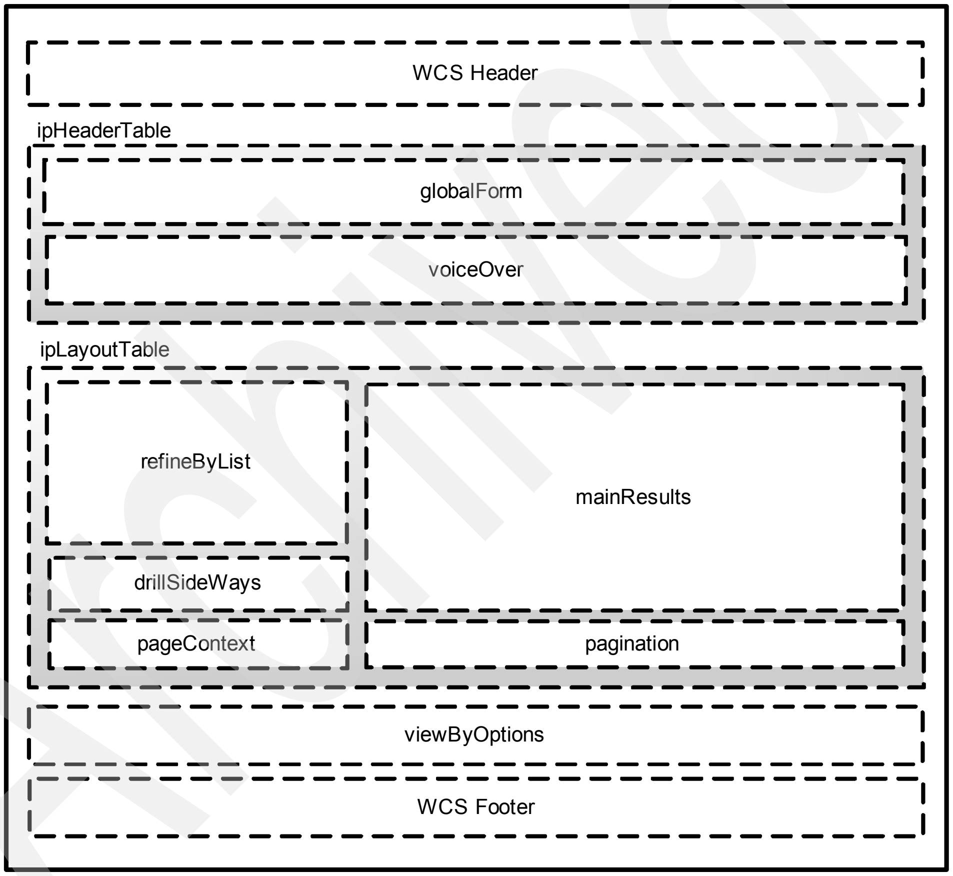


Figure 4-9 Search results page

In later chapters of this book, we show you how we do the customization of the starter store.

Part 2

Implementation

In this part of the book, we introduce OmniFind Discovery Edition APIs and show you how to implement our sample store.

Names: For simplification, we use the following variable names in this book:

- <ODE_HOME>: This stands for:
  C:\IBM\IBM WebSphere Content Discovery Server 8.3

  This represents the home directory where OmniFind Discovery Edition or WebSphere Content Discovery Server is installed.
- <WAS_HOME>: This stands for:
  C:\Program Files\IBM\WebSphere\AppServer

  This represents the application server directory where IBM WebSphere is installed.
- <IPHRASE_HOME>: This stands for:
  C:\IBM\IBM WebSphere Content Discovery Server 8.3
- <WCS_HOME>: This stands for:
  C:\Program Files\IBM\WebSphere\AppServer\profiles\demo\installedApps\WC_demo_cell\WC_demo.ear

5

Introducing OmniFind Discovery Edition APIs

In this chapter we provide an overview of the OmniFind Discovery Edition Application Programming Interfaces (APIs). We describe what they are and how to find more information about them. In addition, we provide examples of how to use the APIs in sample Java programs.

We cover the following topics:

- OmniFind Discovery Edition APIs overview
- Getting started with the API examples
- Running a simple search for a word or phrase
- Refining the results of a search
- Running a complex search
- Navigating search results

5.1 OmniFind Discovery Edition APIs overview

OmniFind Discovery Edition provides a powerful search engine that can be incorporated into the existing online stores and other Web sites (such as self services and contact centers) to enhance search capabilities of the existing system. To leverage the search capabilities and enable you to customize the searches for your application, OmniFind Discovery Edition provides a set of APIs to help you accomplish this.

The OmniFind Discovery Edition APIs provide many functions and capabilities. In this chapter, we focus on the APIs that provide the search capabilities.

OmniFind Discovery Edition has two sets of APIs for running searches:

- Runtime APIs (OneStepRuntimeAPI.jar)
- Layouts APIs (OneStepRuntimeJspLib.jar)

The location of these files depends on the option with which you install OmniFind Discovery Edition. If you install OmniFind Discovery Edition with option one (Integrate with Tomcat) or option three (Other), then both files are located in webapps\jspapps\WEB-INF\lib under the directory in which you install OmniFind Discovery Edition. If you install the software with option two (Integrate with WebSphere), then both files are located under the deployment directory under the WebSphere Application Server.

The *Layout APIs* use the Java Servlets APIs and are intended to be used in JavaServer™ Pages™ (JSPs) or servlets.

The *Runtime APIs* can be used with Java programs that are not necessarily JSPs or servlets.

In this chapter, we describe mainly the Runtime APIs. Unless otherwise noted, the classes described in this chapter belong to the Runtime APIs and not the Layouts APIs.

The Runtime APIs comprise the following packages:

- com.iphrase.runtime - Classes for client connections and utilities
- com.iphrase.runtime.exception - Exception classes used throughout the APIs
- com.iphrase.runtime.query - Classes for building and running search queries
- com.iphrase.runtime.query.constraint - Classes for building search criteria
- com.iphrase.runtime.query.result - Classes that represent search results

Table 5-1 lists some of the important classes in the Runtime APIs. We recommend that you get acquainted with them, because they are essential in manipulating and customizing the searches for your application.

Table 5-1 Important classes in the Runtime APIs

Class	Description
Connector	Manages client connections to the search engine.
Query	Manages searches.
QueryResult	Manages search result class. It can contain one or more Tables.
Table	Manages data returned from a search. It is a two-dimensional grid of rows and columns.
FeatureRow	Manages a row of a table.
FeatureMetaData	Manages data about a table column.
Constraint	Manages search constraints.
Voiceover	Contains descriptions about how a search is executed.
PageContext	Contains information about how a search result is divided into pages.
DrillDown	Manages narrowing down a search by users.
DrillSideways	Manages search criterion changes by users when users narrow down a search.

We discuss these classes in more detail in later sections of the chapter. The APIs also include other classes, which do not appear in Table 5-1,

Note: In order to work with OmniFind Discovery Edition APIs, you must have JDK 1.3.1 or above. Always check with the online documentation for the latest JDK version that the software supports.

Accessing API documentation

Version 8.3 of this software product is called WebSphere Content Discovery Server. You can access the API documentation from the Windows machine where you installed WebSphere Content Discovery 8.3:

1. Select **Start** → **Programs** → **IBM WebSphere Content Discovery Server 8.3** → **Documentation**.
2. Click **Online Reference**.

For OmniFind Discovery Edition Version 8.4 and later versions, you can access the documentation in IBM InfoCenter from the following URL:

```
http://publib.boulder.ibm.com/infocenter/discover/v8r4/topic/com.ibm.di
scovery.ds.nav.doc/cdsnav_welcome.html
```

The documentation for the Runtime APIs and Layouts APIs appears under Reference in the navigation tree.

This book demonstrates only part of the OmniFind Discovery Edition APIs. For more information about the APIs, refer to the product documentation.

5.2 Getting started with the API examples

In the sections following this section, we describe how you can program search tasks using OmniFind Discovery Edition APIs. We cover the implementation from simple to complex:

- In 5.3, “Running a simple search for a word or phrase” on page 71, we show you how to use the APIs to do a simple search for a word or phrase.
- In 5.4, “Refining the results of a search” on page 82, we build on the previous search example to show you how to take the result of the search and refine it by adding more search criteria. That is, we show you how to make a search from within the results of a previous search.
- In 5.5, “Running a complex search” on page 83, we present different types of search criteria and show you how they can be combined to make a complex search.
- In 5.6, “Navigating search results” on page 89, we discuss how to present search results and let users navigate through them.

In the remaining sections of this chapter, we include example programs that use the OmniFind Discovery Edition APIs. Although this book focuses on using the APIs in JSP, we use standalone Java programs in this chapter to illustrate how to use the APIs.

Running standalone example programs

To run the example programs as standalone Java programs, there are some special steps you have to take:

- Set the environment variable, CLASSPATH, to include all the JAR files in the directory which contains OneStepRuntimeAPI.jar and OneStepRuntimeJspLib.jar.
- Include a method, formatValue(), to display the feature values in your Java programs.

In writing these examples, we take the code from formatValue.jsp in the JSP layouts, and prefixed all the method declarations with *static*. The formatValue() method formats data in a way that is intended for Web pages.

The primary purpose of these examples is to illustrate how you might use the APIs in JSP and not to be useful in themselves as standalone Java programs.

When running these examples, the following warning message is displayed:

```
log4j:WARN No appenders could be found for logger
(com.iphrase.runtime.Connector).
log4j:WARN Please initialize the log4j system properly.
```

This is because OmniFind Discovery Edition uses the log4j package internally for logging. You can ignore the warning message, or eliminate this message.

To eliminate the warning message:

- Create a log4j configuration file called log4j.properties with the appropriate settings as follows:

  ```
  log4j.logger.com.iphrase=ERROR, R1
  log4j.logger.org.apache=ERROR, R1
  log4j.appender.R1=org.apache.log4j.RollingFileAppender
  log4j.appender.R1.File=ode.log
  log4j.appender.R1.layout=org.apache.log4j.PatternLayout
  ```

- Include in the CLASSPATH, the directory where the log4j.properties file is located.

For information about log4j, refer to:

`http://logging.apache.org/log4j/docs`

5.3 Running a simple search for a word or phrase

The advantage that OmniFind Discovery Edition offers is the ability to do a managed search. In this section, we show you how to use the APIs to run a simple search for a word or phrase.

We cover the following topics:

- Basic concepts: items and features
- Classes used in the simple search example
- Simple search example and explanation

The complete source code is shown in Example 5-9, “Running a search” on page 79.

5.3.1 Basic concepts: items and features

OmniFind Discovery Edition enables you to do searches in a data collection. A data collection consists of a set of items and a set of features.

An *item* refers to a piece of content that can potentially be one result in a query response. For example, in the data collection presented in this book, the items are the products for sale in the store. The OmniFind Discovery Edition build process indexes each item with all its properties that define it. Additional knowledge and understanding of the content associated with the item can also be added in the system. When performing a search, the indexes are used at runtime.

A *feature* refers to a piece of information about an item. If an item refers to a product for sale in our book example, then the item can have multiple features such as name, category, and price. If an item is a Web page, then the item might have features such as the title of the page, its URL, and a few sentences of sample content.

Typically, if the data collection is created using a relational database, each item corresponds to a row in a database table, and each feature corresponds to a column in the database table. Features are also called *columns*.

There are values associated with features of an item. For example, an item, two-drawer coffee table, has the following three features and the associated values:

Feature	Value
Name	Two-Drawer Coffee Table
Category	Coffee Table
Value	199.99

5.3.2 Classes used for running a simple search

The OmniFind Discovery Edition APIs contain the following Java classes you can use to write a simple search example:

- Connector
- Query
- QueryResult
- ResultSet
- Table

Use the *Connector* to configure the connection to one or more OmniFind Discovery Edition servers. Use the *Query* to do a simple search, and get the results of the search in a *QueryResult*. The QueryResult can include one *ResultSet* from each server. Each *ResultSet* contains a *Table,* which is a two-dimensional grid of rows and columns. Each row of the table represents one item from the collection, and each column represents one feature, so that each cell holds the value of one feature for one item.

For each item, the table has one *FeatureRow*, which contains one value for each feature. There is a utility class called *ValueHandler* whose purpose is to make it easier for programmers to access these values. ValueHandler is part of the Layouts APIs. In addition to one FeatureRow per item, the table also contains a *FeatureMetaData object* for each feature, which contains data about that feature such as its *label*. A label is the text that is typically displayed next to the value of a feature to say what it is.

We show you how to use these classes and methods in a sample search program.

5.3.3 Simple search example and explanation

In this section, we show you how to write a program to perform a simple search for a word, *spoon*.

The program has the following logic:

1. Import the necessary packages.
2. Establish and configure the connection to the server.
3. Set and execute query: Search for the word, *spoon*.
4. Get the search result.
5. Display the search result.

At the end of the section, we provide the source code for the simple search example.

Note: We show code snippets of what you have to program along the way. For the complete source code of the program, refer to Example 5-9 on page 79.

Import the necessary packages

At the beginning of the program, you must import the packages that contain classes of the OmniFind Discovery Edition Runtime APIs and Layouts APIs. In addition, you have to import other Java packages for IO operations and utilities.

Example 5-1 shows the source code for the imports.

Example 5-1 Import necessary packages

```
import com.iphrase.runtime.*;
import com.iphrase.runtime.query.*;
import com.iphrase.runtime.query.result.*;
import com.iphrase.runtime.exception.*;
import com.iphrase.onestep.beans.*;

import java.io.*;
import java.util.*;
```

Establish and configure the connection to the server

A Connector is required to handle a client connection to the OmniFind Discovery Edition server. In this program, the Connector is created by a *ConnectorFactory*.

By setting the properties of a Connector, you can configure:

- Which port numbers on which hosts to use to connect to the OmniFind Discovery Edition server
- The number of seconds to wait before a query is timed out
- The number of seconds to wait before retrying that server after a query has been timed out

In our example, the program configures the Connector to connect to the OmniFind Discovery Edition server using port 8777 of host redsrv01.lexma.ibm.com. It sets the query to time out after waiting for 20 seconds. It sets the system to retry the connection to the server after one second has elapsed when the query is timed out.

Example 5-2 shows the source code to establish and configure the connection to the server.

Example 5-2 Create connector

```
int timeout = 20;
int retry = 1;
String odeService = "redsrv01.lexma.ibm.com:8777";
ArrayList odeAddresses = new ArrayList();
odeAddresses.add(odeService);

ConnectorFactory connectorFactory = ConnectorFactory.newInstance();
Connector connector =
      connectorFactory.getConnector(odeAddresses, timeout, retry);
```

Set and execute query: Search for the word spoon

To set and execute a query, you must:

1. Create a Query object using the createQuery() method of the connector object.
2. Specify what you want to search for by setting the properties of the Query object. In our example, we use the setText() method to set the properties of the Query object to specify that we want to search for the word *spoon*.
3. Execute the query by calling the execute() method of the query object Example 5-3 shows the source code to run a query to search for the word *spoon*.

Example 5-3 Set and execute query

```
Query query = connector.createQuery();
query.setText("spoon");
QueryResult results = query.execute();
```

Get the search result

After executing the query, a QueryResult object is returned from the search. There can be one ResultSet object for each of the OmniFind Discovery Edition servers to which the client is connected.

For our example, we connected to only one OmniFind Discovery Edition server; so the QueryResult contains only one ResultSet.

If there is only one ResultSet, it is always stored as the main ResultSet. You can get it by using the getMainResultSet() method of the ResultSet object.

To get the data collection from the ResultSet, use the getTable() method.

If no items are found from the search, then the ResultSet contains no table. If so, for our example, we report that no items are found and quit the program.

Example 5-4 shows the source code to get the search result data collection.

Example 5-4 Get the search result data collection

```
ResultSet mainResultSet = results.getMainResultSet();
Table mainTable = mainResultSet.getTable();

if (mainTable == null){
  System.out.println("No items matched the query constraint.");
  return;
}
```

Get the search result: Get feature labels

The search result is a data collection stored by a Table object. In our example, the Table object is called mainTable.

The returned data collection contains items and their feature information. As mentioned earlier, features represent attributes or properties of the item. In a relational database term, items represent rows in the data table and features represent columns in the data table. The information about the features are stored by the FeatureMetaData objects. You can get this information by calling the getFeatureMetaDatas() method from the Table (mainTable) object.

Each feature has label information. A *label* is the text that is typically displayed next to the value of a feature to describe what it is. You can get the label information by calling getLabel() from the FeatureMetaData object. In our example, the FeatureMetaData object is called featureMetaDatas.

A label is just one of the data items that a FeatureMetaData object contains. A FeatureMetaData object also contains other data about the feature it describes.

Example 5-5 shows the source code for getting the labels for the features in the result data collection.

Example 5-5 Parse the result data collection: Get feature labels

```
FeatureMetaData[] featureMetaDatas = mainTable.getFeatureMetaDatas();
String[] labels = new String[featureMetaDatas.length];
for (int colIndex = 0; colIndex < featureMetaDatas.length; colIndex++){
  labels[colIndex] = featureMetaDatas[colIndex].getLabel();
}
```

Get the search result: Get feature values for each item

The Table object contains the result items, represented by the *FeatureRow objects*. There is one FeatureRow object for each item. To get the FeatureRow objects, use getFeatureRows() method from the Table object.

The FeatureRow object contains values for features of each result item. You can get the feature values by calling getFeatureValues() method from the FeatureRow objects.

Example 5-6 is a code snippet that gets the items (featureRows) from the result data collection (mainTable). In addition, for each item, the code gets the values for the associated features (featureValues).

Example 5-6 Get items (featureRows) and their associated values (featureValues)

```
FeatureRow[] featureRows = mainTable.getFeatureRows();
for (int rowIndex = 0; rowIndex < featureRows.length; rowIndex++){
  Object[] featureValues = featureRows[rowIndex].getFeatureValues();
}
```

Display the search result

To display the feature values of the items, you can pass these values to a ValueHandler object and use the formatValue() method to display the values in string. *ValueHandler* is a utility class that makes it easy for you to access the various properties of the OmniFind Discovery Edition data types. The class is part of the Layouts APIs. The formatValue() method returns a string to display a feature value. For our program, we use the formatValue() method based on the JSP Tabbed Navigation layout. At this point, do not be concerned about how the method works, just that you use it for simplifying the output of your program. For more information on formatValue(), refer to the Layout APIs documentation.

The program displays a running count of each item, then for each feature, it displays the feature label and feature value.

Example 5-7 shows the source code that goes through the result data collection, runs a count for each item, and displays the values along with its features for each item in the result. This source code builds on top of the code snippet shown in Example 5-6.

Notice that in our example:

- The result data collection is stored in the *mainTable* object (obtained earlier).
- The result data collection contains a collection of the result items. These items are stored in the *featureRows* objects.
- The information (metadata) about the result items' features are stored in the featureMetaDatas objects (obtained earlier). You can get the label information for each feature.
- The values of the features for each item are stored in the featureValues objects.
- There is a one-to-one correspondence between the items' features (featureMetaDatas) and their values (featureValues).
- You pass both the items' features and feature values to the ValueHandler class and use its formatValue() method to format the results.
- Labels contain the label descriptions of the features (obtained earlier). They are used to display the results.

Example 5-7 Display result information

```
FeatureRow[] featureRows = mainTable.getFeatureRows();
for (int rowIndex = 0; rowIndex < featureRows.length; rowIndex++){
  Object[] featureValues = featureRows[rowIndex].getFeatureValues();

  // ...write which number this result item is...
  System.out.println("Item " + Integer.toString(rowIndex + 1) + ":");

  // ...then for each feature ...
  for (int colIndex = 0; colIndex < featureValues.length; colIndex++){
    // ... write the feature label and value use ValueHandler
    ValueHandler valueHandler =
      ValueHandler.CreateInstance(featureMetaDatas[colIndex],
                                  featureValues[colIndex]);
    String currentValueStr=formatValue(valueHandler, "", colIndex, "");

    // Write the feature label (obtained earlier).
    System.out.print(labels[colIndex] + ":");
    // Write the feature value.
    System.out.println(currentValueStr);
  }
}
```

If an exception is thrown during the program, use *try* and *catch* to catch the exceptions. See Example 5-8.

Example 5-8 Handle exceptions

```
try{ .....
}
catch (Exception e){
      System.out.println(e.getMessage());
      e.printStackTrace();
    }
```

Source code for the simple search example

Source code for the simple search example is shown in Example 5-9. This source code combines the code snippets we show earlier in the section, but does not include the formatValue() method.

Example 5-9 Running a search

```
import com.iphrase.runtime.*;
import com.iphrase.runtime.query.*;
import com.iphrase.runtime.query.result.*;
import com.iphrase.runtime.exception.*;
import com.iphrase.onestep.beans.*;

import java.io.*;
import java.util.*;

class ODEExample1{
  public static void main(String[] args) {
    try {
      /*
      Connect to ODE server.
      */

       // Queries submitted using this ODE connection will be timed out
       // after 20 seconds if no response is received from the server.
      int timeout = 20;
       // Retry the connector after 1 sec has elapsed since the query
       // was timed out.
      int retry = 1;
       // Specify the host and port number the ODE service is
listening.
       // This is in the format "<host name>:<port number>"
       // In this example the ODE service is listening on
       // port 8777 of host redsrv01.lexma.ibm.com.
      String odeService = "redsrv01.lexma.ibm.com:8777";
      ArrayList odeAddresses = new ArrayList();
      odeAddresses.add(odeService);

       // Create the ConnectorFactory ...
      ConnectorFactory connectorFactory =
ConnectorFactory.newInstance();
       // ... then use it to create a connector.
      Connector connector =
      connectorFactory.getConnector(odeAddresses, timeout, retry);

      /*
      Submit a query
      */

       // Create a new query.
      Query query = connector.createQuery();

       // Specify the constraint for the query.
       // In here, the constraint is the items in the query result
       // must mention the word "spoon".
      query.setText("spoon");

       // Submit the query and get the query result.
      QueryResult results = query.execute();

       // Take the main result set from the query result.
      ResultSet mainResultSet = results.getMainResultSet();

       // Take the table from the main result set.
      Table mainTable = mainResultSet.getTable();

      if (mainTable == null){
         // If no items matched the constraint specified in the query
         // there is no table in the result set.
        System.out.println("No items matched the query constraint.");
        return;
      }

       // If some item matched the constraint specified in the query
       // the result set contains a table.

      /*
      Write the contents of the main table in the query result.
      */

       // The table contains a FeatureMetaData object for each column
of the table.
       // Take the label for each column from its FeatureMetaData.
      FeatureMetaData[] featureMetaDatas =
mainTable.getFeatureMetaDatas();
      String[] labels = new String[featureMetaDatas.length];
      for (int colIndex = 0; colIndex < featureMetaDatas.length;
colIndex++){
        labels[colIndex] = featureMetaDatas[colIndex].getLabel();
      }

       // The table also contains an array of FeatureRow objects for
       // each item. One FeatureRow per item. Each FeatureRow contains
       // an array of feature values, one value per column.
      FeatureRow[] featureRows = mainTable.getFeatureRows();
```

```
       // For each item ...
      for (int rowIndex = 0; rowIndex < featureRows.length;
rowIndex++){
         // ...get the values of all the features for that item,...
        Object[] featureValues =
featureRows[rowIndex].getFeatureValues();
         // ...write which number item this is...
        System.out.println("Item " + Integer.toString(rowIndex + 1) +
":");
         // ...then for each feature ...
        for (int colIndex = 0; colIndex < featureValues.length;
colIndex++){
           // ... write the feature label and value.
           // ValueHandler is a utility class to make it easy to
           // access the various properties of ODE data types.
           // For each "cell" of the table we create a ValueHandler ...
          ValueHandler valueHandler =
           ValueHandler.CreateInstance(featureMetaDatas[colIndex],
                                       featureValues[colIndex]);
           // ... and pass ValueHandler to the formatValue() function,
           // which returns a string to display the value.
           // The formatValue() function used here is based on the
           // formatValue() function in the JSP Tabbed Navigation
           // layout and is not explained in detail here.
          String currentValueStr = formatValue(valueHandler, "",
colIndex, "");
           // Write the feature label.
          System.out.print(labels[colIndex] + ":");
           // Write the feature value.
          System.out.println(currentValueStr);
        }
        System.out.println();
      }
    }
    catch (Exception e){
      System.out.println(e.getMessage());
      e.printStackTrace();
    }
  }
//A method formatValue needs to be defined here to display a feature
value.
}
```

5.4 Refining the results of a search

In the previous section, we show you how to write a simple program to search for a word or a phrase. In this section, we show you how to run a follow-up search to refine the results of a previous search. That is, we show you how to search within the results of a previous search. We explain how to do this with the OmniFind Discovery Edition APIs.

The example in this section builds on Example 5-9 on page 79. It takes the result of a search for the term, *spoon*, and then it does a follow-up search within the results of that search to find the term, *stainless steel*. The result of the follow-up search now includes the items that contain both the words *spoon* and *stainless steel*. Notice that the returned results also depend on how you configure the search setting. It is possible to return items that contain only one of the terms or contain something similar to the terms. Refer to the OmniFind Discovery Edition documentation for more information on tuning search results.

5.4.1 Example and explanation

From the previous example, we get a result set, mainResultSet. Example 5-10 extracts the code we have shown earlier. This code sets up a query to search for a word *spoon*, executes the query, and obtains the result of the query.

Example 5-10 Get result from the previous example

```
Query query = connector.createQuery();
query.setText("spoon");
QueryResult results = query.execute();
ResultSet mainResultSet = results.getMainResultSet();
```

After running the search, you get a query state of the results. The *query state* is a string that encodes the search that has been done so far. To get the query state, use the getResolvedQueryState() method from the ResultSet object.

After you get the query state, you can create a follow-up query by calling the createQuery() method from the connector object and passing the query state information as the parameter.

You control whether this new query object will search within the results of the previous search or not by setting the setEnableSearchWithin() method.

To refine the search, use the setText() method of the query object as before.

Example 5-11 shows the source code that uses the result from the previous search, creates a new query object, sets up the refined search, and executes the search.

Example 5-11 Refine search result by executing a search within the previous result

```
String queryState = mainResultSet.getResolvedQueryState();

Query followupQuery = connector.createQuery(queryState);
followupQuery.setEnableSearchWithin(true);
followupQuery.setText("stainless steel");

results = followupQuery.execute();
```

After you execute the search, you can get the search result and display it as shown in Example 5-9 on page 79.

The output of this new program includes only items that contain both the terms *spoon* and *stainless steel*.

We include the complete source code of this program as the additional materials for this book. Refer to Appendix A, “Additional material” on page 271 for download instructions.

5.5 Running a complex search

In the previous examples of this chapter, we search for a particular word or phrase. Using OmniFind Discovery Edition APIs, you can also use other types of search. In this section, we explain different types of search constraints and show how you can combine the search conditions to do complex searches.

5.5.1 Constraints

A *constraint* is a condition that items must satisfy in order to be in the result of a query. Constraint is an abstract class, which means that a Constraint object can only be instantiated as a subclass of Constraint. Each subclass of Constraint is for a specific type of constraint. Different types of constraint are applicable to different types of features.

Table 5-2 shows a list of constraints that you can use to compare features with certain values.

Table 5-2 Constraint types and descriptions

Constraint type	Description
Bool	Constrains a feature to be true, or constrains it to be false.
DateTime.Eq	Constrains a feature to be a particular date.
DateTime.Gt	Constrains a feature to be later than a particular date.
DateTime.Gte	Constrains a feature to be no earlier than a particular date.
DateTime.Lt	Constrains a feature to be earlier than a particular date.
DateTime.Lte	Constrains a feature to be no later than a particular date.
DateTime.Neq	Constrains a feature not to be a particular date.
DateTime.Range	Constrains a feature to be between two particular dates.
Enumerated.Equals	Constrains a feature to be a particular string value.
Enumerated.InList	Constrains a feature to be one of a particular list of string values.
Enumerated.NotEquals	Constrains a feature not to be a particular string value.
Enumerated.NotInList	Constrains a feature not to be one of a particular list of string values.
Null	Constrains a feature to be null, or constrains it to be non-null.
Numeric.Eq	Constrains a feature to be a particular numeric value.
Numeric.Gt	Constrains a feature to be greater than a particular numeric value.
Numeric.Gte	Constrains a feature to be no less than a particular numeric value.
Numeric.Lt	Constrains a feature to be less than a particular numeric value.
Numeric.Lte	Constrains a feature to be no greater than a particular numeric value.
Numeric.Neq	Constrains a feature not to be a particular numeric value.
Numeric.Range	Constrains a feature to be between two particular numeric values.

There are other subclasses of Constraint that you can use with taxonomy features. Taxonomy features are organized in a hierarchy of categories. For example, in the case study used in this book, the Category feature is a taxonomy feature.

All items in our sample collection are categorized as follows:

- Living & Entertaining:
  - Tableware:
    - Plates
    - Silverware
    - Table Glasses
  - Kitchenware
- Kitchen & Dining

Living & Entertaining is a category at the first level of the taxonomy. Under this category are the second level categories *Tableware* and *Kitchenware*. Under the second level category *Tableware* are the third level categories *Plates*, *Silverware*, and *Table Glasses*.

All items that belong to a certain category at a lower level of a taxonomy also belong to the corresponding categories at the higher levels of the taxonomy. For example, all items in this collection that belong to the *Plates* category at the third level of the taxonomy also belong to the *Tableware* category at the second level of the taxonomy, and all items that belong to the *Tableware* category also belong to the *Kitchen & Dining* category at the first level of the taxonomy.

Table 5-3 shows a list of constraint types you can have for taxonomy features. You can use them to search for items that have a particular value for its taxonomy feature.

Table 5-3 Taxonomy feature constraint types and descriptions

Constraint type	Description
Taxonomy.Contains	Constrains a taxonomy feature to have one of a particular set of values.
Taxonomy.NotContains	Constrains a taxonomy feature not to have one of a particular set of values.
Taxonomy.StartsWith	Constrains a taxonomy feature to have one of a particular set of values at the top level of the taxonomy.
Taxonomy.NotStartsWith	Constrains a taxonomy feature not to have one of a particular set of values at the top level of the taxonomy.

Table 5-4 shows some other types of constraint that you can use to do searches that exploit other capabilities of OmniFind Discovery Edition.

Table 5-4 Other constraint types

Constraint type	Description
Example	Constrains a feature to have a value similar to that which some particular other item has for the same feature. This type of constraint is used to find Web pages that are similar to some particular other Web page.
Within	Constrains items to be in a certain range of the result of a previous query. For example, it can constrain items to be among the top five most expensive or the bottom 25% least expensive of a previous search result.

OmniFind Discovery Edition also has additional constraints that we do not mention here because they are beyond the scope of this book. We recommend that you work with the set of constraints as presented in Table 5-2 and Table 5-3 first. After you are familiar with them, refer to OmniFind publications to find out more about other constraints that you can use to customize your applications.

5.5.2 Complex searches

You can combine constraints with each other to make a complex search.

A set of constraints that are ORed together is a *disjunction*. For a disjunction to be true, at least one of the constraints in the disjunction must be true.

A set of constraints that are ANDed together is a *conjunction.* For a conjunction to be true, all the constraints in the conjunction must be true.

In order to associate a constraint with a query, you must construct the constraint as a conjunction of disjunctions. Any combination of constraints that are ANDed and ORed together can be framed as a conjunction of disjunctions.

5.5.3 Example and explanation

In this section, we explain how you can do a search for items that match two different conditions of different types.

Import the constraint package

To work with the constraint classes, import the constraint package in addition to the ones we mentioned earlier in Example 5-1 on page 74. See the code snippet in Example 5-12.

Example 5-12 Import the constraint package

```
import com.iphrase.runtime.query.constraint.*;
```

Set up the first constraint

The first constraint of the sample program is that the items must belong to the Kitchenware category. The feature ID of the Category taxonomy in our sample is sitemap taxonomy.

We have to create a constraint object and set the feature (sitemap taxonomy) to include the word *Kitchenware*. Example 5-13 shows the source code that does this using the Contains() method.

Example 5-13 Set sitemap taxonomy constraint to include Kitchenware

```
Constraint constraint1 =
       new com.iphrase.runtime.query.constraint.Taxonomy.Contains(
       "sitemap taxonomy", new String[]{"Kitchenware"});
```

There are two packages that contain a class called Taxonomy:

- com.iphrase.runtime.query.constraint
- com.iphrase.runtime.query.result

Therefore, in our program, we use the full package name.

Set up the second constraint

We want to set up the second constraint such that only the items with prices less than 50 currency units should be returned.

To set up the second constraint, we use Numeric.Lte() method as shown in Example 5-14.

Example 5-14 Set the second constraint to limit items with price less than 50

```
Constraint constraint2 = new Numeric.Lte("Price", 50);
```

The Price feature can be in any currency unit you set up.

Set up the query with the constraints

After the constraints are set, you can create a query using the createQuery() method and pass the constraint objects to the query.

Example 5-15 shows the source code that creates the query and sets the constraints.

Example 5-15 Creates a query and sets constraints

```
Query query = connector.createQuery();

Conjunction conjunction = query.getConstraints();
conjunction.merge(new Disjunction(constraint1));
conjunction.merge(new Disjunction(constraint2));
```

Both constraint1 and constraint2 are created earlier in Example 5-13 and Example 5-14. When you set these constraints for a query, they must be set as conjunctions of disjunctions. In our code, we create a Disjunction object for each constraint, and then merge the Disjunction objects into a Conjunction object.

Execute the query and obtain the result set

After you create the query and set its constraints, execute the query and obtain the results as we have done for the previous program. Example 5-16 shows the source code.

Example 5-16 Execute the query and obtain results

```
QueryResult results = query.execute();
ResultSet mainResultSet = results.getMainResultSet();
Table mainTable = mainResultSet.getTable();
```

Notice that we call the getMainResultSet() method to get the results as we have done earlier. You can also use the getOtherResultSets() method in addition to this method to get other results.

The rest of the program is same as the previous program as shown in Example 5-9. The output of this new program includes only Kitchenware items that are less than 50 unit price (for example, dollars).

We include the complete source code of this program as the additional materials for this book. Refer to Appendix A, “Additional material” on page 271 for download instruction.

5.6 Navigating search results

OmniFind Discovery Edition enables users to navigate their search results to do follow-up searches. In this section, we discuss parts of the OmniFind Discovery Edition APIs that you can use to allow a user to navigate through the search results. We cover the following topics:

- Voiceover
- Page context
- Drill down
- Drill sideways
- Tally features

It is important for you to understand the concepts and what is available when using the OmniFind Discovery Edition APIs. We do not show you the code examples of how to use the APIs to provide these navigation capabilities. We recommend that you look through the source code of our sample project to see how we use them in the program.

With the OmniFind Discovery Edition APIs, there are many other tasks you can do and features you can incorporate in your applications. Some of them include:

- Truncation
- Business rules
- Sorting
- Summarization

We recommend that you check them out when you become familiar with the general sets of the OmniFind Discovery Edition APIs.

5.6.1 Voiceover

Voiceover is the feedback that OmniFind Discovery Edition gives in a query result about the search the user made and how the search engine handled it.

Voiceover can include these things:

- Constraints that the user added.
- Constraints that the user removed.
- If no items matched any of the constraints the user specified, OmniFind Discovery Edition might still return results that match some of the constraints. In this case, the constraints that OmniFind Discovery Edition ignores are considered to be *backed off*. The voiceover can specify what constraints are backed off.

- OmniFind Discovery Edition ignores certain words in a search term because they are too common to be useful for a search. These are called *stopwords*. For example, *a*, *the*, and *it* are stopwords. The voiceover says which stopwords are ignored.
- OmniFind Discovery Edition can determine that a word in a search term is misspelled, and respell it to make it a recognized word. The voiceover indicates what respelling is used.
- OmniFind Discovery Edition can use synonyms, broader concepts, narrower concepts, or other concepts related to the search terms the user specified. The voiceover indicates what concepts are used.

With the OmniFind Discovery Edition APIs you can use the *Voiceover* class to access the voiceover information.

5.6.2 Page context

The OmniFind Discovery Edition server sends the result sets divided into pages for display. *Page context* is the information included with a result set about how the query result is divided into pages.

Page context includes these things:

- The number of the currently displayed page.
- The number of pages in the entire query result.
- The number of items in the entire query result.
- The number of rows in the entire query result. (The total number of items can exceed the total number of rows when the data is summarized, because it can cause more than one item to be summarized into a single row.)
- Whether the data is summarized, and if so, then by what feature.

With the OmniFind Discovery Edition APIs, you can use the *PageContext* class to access page context.

5.6.3 Drill down

Using OmniFind Discovery Edition, you can present a list of feature values for the user to select from in order to narrow down their search. When the user selects feature values, you can do a follow-up search for items that have the selected feature values. This is called *drill down*.

With the OmniFind Discovery Edition APIs, you can use the *DrillDown* class to allow drill down.

5.6.4 Drill sideways

After users have done drill down to narrow down their search results to only those items that have certain feature values, you can allow the users to select alternative values for the same feature. You can then do a follow-up search to replace the previously selected feature values with the newly selected values. This is called *drill sideways*. Using drill sideways, users can change their selection of value for a feature with a single click. Users do not have to first remove the old feature value constraint and then, as a separate operation, add the new constraint.

With the OmniFind Discovery Edition APIs, you can use the *DrillSideways* class to allow drill sideways.

5.6.5 Tally features

For some features, such as numbers or dates, a range of values are more appropriate as a search criterion than a single value. For those features, OmniFind Discovery Edition can allow the user to select a range of values as a search criterion rather than a single value. A feature for which the user can drill down or drill sideways using a range of values rather than a single value is called a *tally feature*.

To allow for tally features, you can use the *TallyFeature* class.

6

Adding the By Brand View feature

In this chapter we use OmniFind Discovery Edition to discover products from different brands (by different manufacturers) for more target searches by shoppers.

We cover the following topics:

- Designing the By Brand View page flow
- Customizing data structure
- Narrow search scope with OmniFind Discovery Edition constraints APIs
- Retrieving an existing feature defined in OmniFind Discovery Edition index
- Passing multiple constraints to the Search Result View page
- Creating new JSP files in WebSphere Commerce starter store for displaying brands, and registering a Struts View in Commerce Server

6.1 Designing the By Brand View page flow

A brand is a symbolic embodiment of all the information connected to a company, product, or services. There can be thousands of products in an online store catalog. Customers might want to search for their favorite products by brand. The By Brand View provides an effective way for customers to find their interested products. With OmniFind Discovery Edition's search capabilities, we show you how to customize the Consumer Direct store to provide the by-brand navigation function.

Figure 6-1 shows the top menu option and the drop-down menu designed for the by-brand option. In our example, it is the first filter for customers to narrow down their search scope.

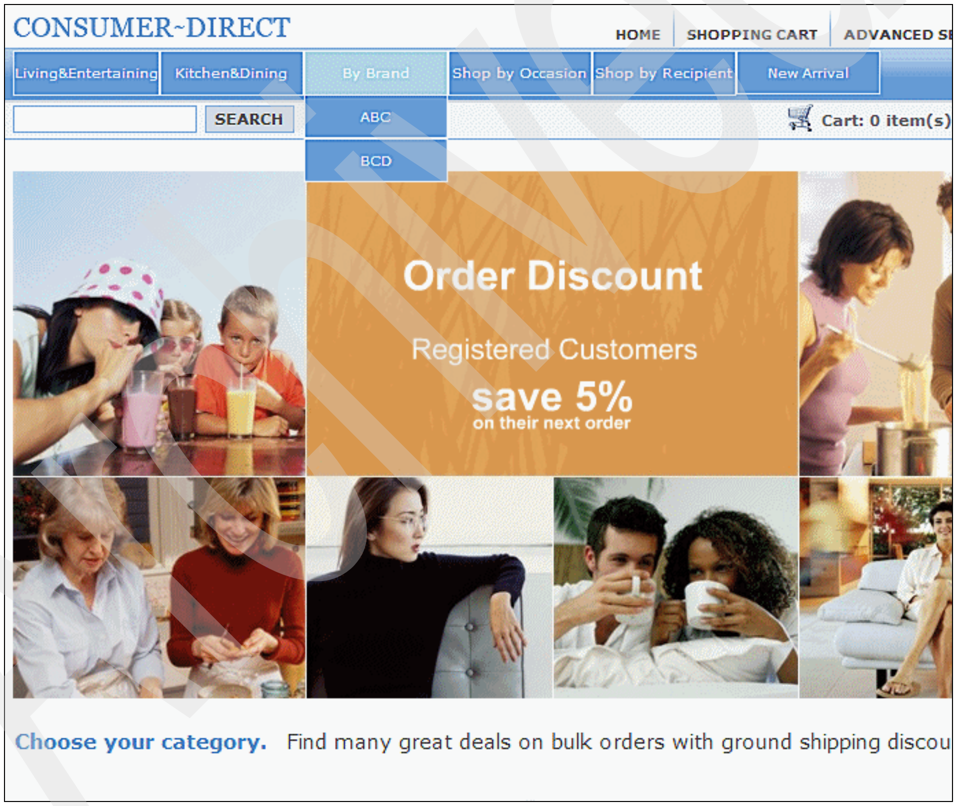


Figure 6-1 By Brand View drop-down menu

6.2 Customizing data structure

To incorporate brand information as one of the top menu selections in the sample store, we require the brand information in both the WebSphere Commerce database table and the OmniFind Discovery Edition features.

6.2.1 Brand related Commerce database tables

In WebSphere Commerce database, the table CATENTRY holds the information related to a catalog entry. Examples of catalog entries include products, items, packages, and bundles.

We use the table field MFName as the identifier for brands of the WebSphere Commerce product items.

Table 6-1 lists CATENTRY table columns information.

Table 6-1 WebSphere Commerce database table CATENTRY

Column Name	Column Type	Description
CATENTRY_ID	BIGINT NOT NULL	The internal reference number of the catalog entry.
PARTNUMBER	VARCHAR(64) NOT NULL	The reference number that identifies the part number of the catalog entry. Along with the MEMBER_ID, these columns are a unique index.
MFPARTNUMBER	VARCHAR(64)	The part number used by the manufacturer to identify this catalog entry.
MFNAME	VARCHAR(64)	VARCHAR(64)
...	...	...

6.2.2 Brand related OmniFind Discovery Edition feature settings

Commerce Module for OmniFind Discovery Edition already includes the MFName in its index out-of-the-box integrated solution. Open the file, <ODE_HOME>\deployment\wcs\wcs\lib\wcs\feature.txt. There are attribute settings for OmniFind Discovery Edition feature MFName, as shown in Example 6-1.

Example 6-1 MFName feature attributes settings

```
"MFName"
  db: sitemap.MFName
  type: text
  build_grammar: no
  enable_feature_for_rendering: yes
  enum: yes
  tally: yes
  build_ir: yes
  render_label: "Manufacturer Name"
  render_summarized_keep_duplicate_values: no
```

Because MFName already exists, we do not have to create a new feature for manufacturer in OmniFind Discovery Edition server. In Chapter 7, “Displaying new products in your site” on page 127, we explain how to create a new feature.

You might want to add more attribute settings to satisfy your business requirements. For example, if you have a single manufacturer for all product items, OmniFind Discovery Edition does not present the manufacturer name in the refinement list for display by default. In this chapter, we ensure that OmniFind Discovery Edition lists all of the manufacturer names, even when there is only a single manufacturer in index.

To force OmniFind Discovery Edition to display refinement options that have only a single value, add the attribute *render_refine_show_single_value* as shown in Example 6-2. The bold text is the line we added for the new setting.

Example 6-2 render_refine_show_single_value setting for MFName

```
"MFName"
  db: sitemap.MFName
  type: text
  build_grammar: no
  enable_feature_for_rendering: yes
  render_refine_show_single_value: yes
  enum: yes
  tally: yes
  build_ir: yes
  render_label: "Manufacturer Name"
  render_summarized_keep_duplicate_values: no
```

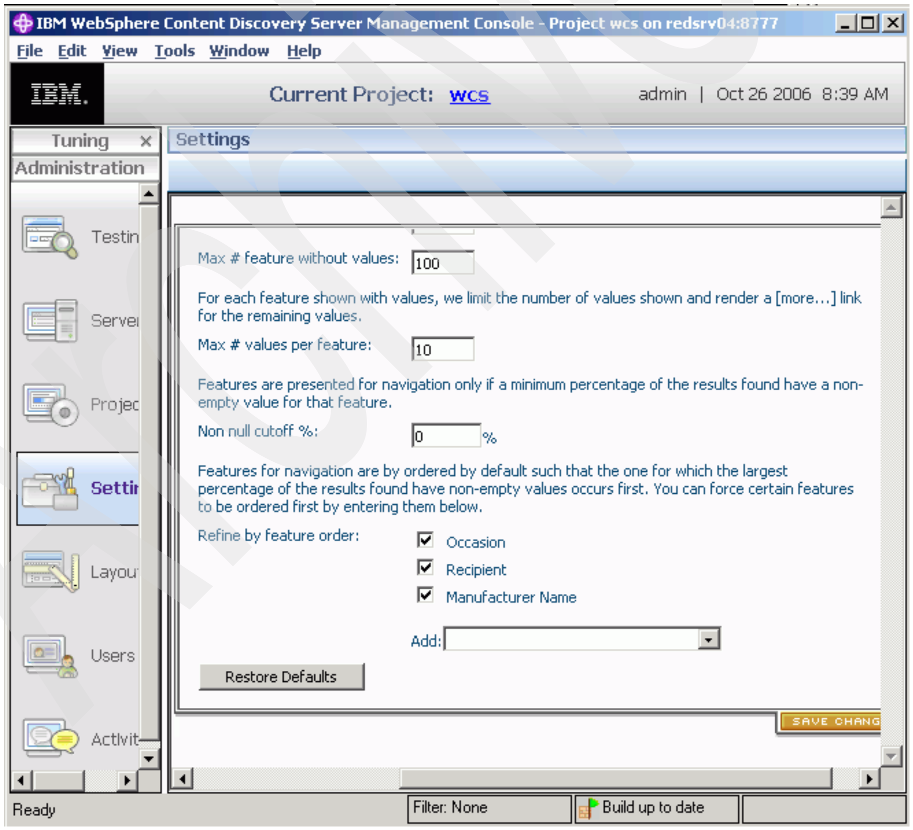
Commerce Module for OmniFind Discovery Edition has many features defined. It is possible that the manufacture feature and some other features that you want do not show up on the refine-by list.

To force the system to always show specific features, use the following steps:

1. Start Management Console and open your project. In our sample, it is the wcs project. Refer to 7.3.1, “Starting the Management Console” on page 136 for the detail steps.
2. After your project is loaded, go to **Administration** → **Settings**.
3. Under **Presentation**, select **Navigation**.
4. Click the **Show Advanced >>** button
5. Under the following section, select **Manufacturer Name** from the Add drop-down box and click **Save Changes**.

   ```
   Features for navigation are by ordered by default such that the one
   for which the largest percentage of the results found have non-empty
   values occurs first. You can force certain features to be ordered
   first by entering them below.
   ```

Your settings should be similar to Figure 6-2.


IBM WebSphere Content Discovery Server Management Console - Project wcs on redsrv04:8777
File Edit View Tools Window Help
IBM.
Current Project: wcs
admin | Oct 26 2006 8:39 AM
Tuning
Administration
Testin
Serve
Projec
Settir
Layou
Users
Activit
Settings

Max # feature without values: 100

For each feature shown with values, we limit the number of values shown and render a [more...] link for the remaining values.

Max # values per feature: 10

Features are presented for navigation only if a minimum percentage of the results found have a non-empty value for that feature.

Non null cutoff %: 0 %

Features for navigation are by ordered by default such that the one for which the largest percentage of the results found have non-empty values occurs first. You can force certain features to be ordered first by entering them below.

Refine by feature order:
- ☑ Occasion
- ☑ Recipient
- ☑ Manufacturer Name

Add:

Figure 6-2 Settings to force Manufacturer feature to always display on refine-by list

6.2.3 Populate sample data

A copy of the WebSphere Commerce database, which contains all sample data used for this chapter, can be obtained from the Redbooks download page. Refer to Appendix A, “Additional material” on page 271 for download information.

Alternatively, you can run a script file to issue update statements for the data we require in this chapter. Get a copy of the script file from the downloaded material, chapter_6_By_Brand\ByBand_data.sql.

After you get the script file, run the update statements as follows:

1. Open a DB2 Command Window.
2. Go to the directory where the script file, ByBrand_data.sql, is downloaded.
3. Connect to the database by issuing the following statement:

   ```
   db2 connect to <DB name> user <DB schema owner ID> using <password>
   ```

   For our case study, <db name> is "mall", <user id> is "db2admin", and <password> is "password".

   So, we use the following command:

   ```
   db2 connect to mall user db2admin using password
   ```

4. Run the script file by issuing the statement:

   ```
   db2 -tvf ByBrand_data.sql
   ```

5. To check if the data is populated, issue the following command:

   ```
   db2 -tvf ByBrand_listdata.sql
   ```

   You should get a list some records, with information filled out in the MFName field.

After you load the new data, you have to rebuild OmniFind Discovery Edition index to pick up the new data.

To rebuild OmniFind Discovery Edition content index:

1. Open a command prompt and switch to the <*ODE_HOME*> directory.
2. Run the following command:

   ```
   bin\iphrase build -p deployment\wcs\wcs\default.prp
   ```

   Note: If you are using Cloudscape as your database, WebSphere Commerce *must not* be running when you run the build.

3. Wait until the build finishes. Close the command prompt window.

6.2.4 Create three-level catalog structure

For our case study, we restructure the out-of-box Commerce store catalog to have three levels of categories.

You can create the three-level categories by using either one of the following two methods:

- Use Commerce Accelerator to restructure the category levels:

  a. Change category name from *Furniture* to *Living&Entertaining*.

  b. Create a new category and name it *Kitchen&Dining*.

  c. Move categories *Tableware* and *Kitchenware* to be under the new category *Kitchen&Dining*.

  To start Commerce Accelerator, select **Start** → **Programs** → **IBM WebSphere** → **WebSphere Commerce v6.0** → **WebSphere Commerce Accelerator**.

  To modify the existing categories, select **Merchandise** → **Catalog Management**.

  For detail information on how to change a category, refer to:

  http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.user.doc/tasks/tpnchcat.htm?resultof=%22%43%61%74%61%6c%6f%67%22%20%22%63%61%74%61%6c%6f%67%22%20%22%4d%61%6e%61%67%65%6d%65%6e%74%22%20%22%6d%61%6e%61%67%22%20

- Use our database backup provided as the additional material associated with this book:

  d. Open a DB2 Command Window.

  e. Go to the directory where the database backup file is downloaded.

  f. Enter the following DB2 command:

  ```
  db2 restore db mall from ConsumerDirect_3levelCatalog_BAK
  ```

After you modify the catalog structure, you need to rebuild OmniFind Discovery Edition index to make it effective.

6.2.5 Back up data and system environment

We recommend performing a WebSphere Commerce and OmniFind Discovery Server environment backup before and after any major changes to the system.

To back up the system environment:

1. Create a new directory to hold the backup files, for example c:\backup.
2. Copy <WCS_HOME>\Stores.war directory to c:\backup.
3. Copy <ODE_HOME>\deployment\wcs directory to c:\backup.
4. Shut down WebSphere Commerce server and OmniFind Discovery Edition server.
5. Open DB2 command prompt.
6. Perform the following command:

   ```
   db2 backup db mall to c:\backup
   ```

   This effectively backs up the entire WebSphere Commerce database directory.
7. Restart both WebSphere Commerce server and OmniFind Discovery Edition server.

To restore the system environment:

1. Replace <WCS_HOME>\Stores.war with the backup copy under c:\backup.
2. Replace <ODE_HOME>\deployment\wcs with the backup under c:\backup.
3. Shut down WebSphere Commerce server and OmniFind Discovery Edition server.
4. Open DB2 command prompt.
5. Perform the following command:

   ```
   db2 restore db mall to c:\backup
   ```

   This restore the entire WebSphere Commerce database.
6. Restart both WebSphere Commerce server and OmniFind Discovery Edition server.
7. Rebuild OmniFind Discovery Edition index:
   a. Go to <ODE_HOME> directory.
   b. Enter the following command:

      ```
      bin\iphrase build -p deployment\wcs\wcs\default.prp -clean
      ```

6.3 JSP page flows

We implement the sample store pages as follows (see Figure 6-3 for graphical representation of the page flows):

1. When the store home page is brought up, the By Brand menu in the top navigation menu is filled with the manufacturer list retrieved from OmniFind Discovery Edition index.
2. When a user selects a manufacturer name from the top navigation drop-down menu, the By Brand View page appears.
3. The refine list and the main content area in the By Brand View page are updated to show the corresponding information related to the selected manufacturer. The main content area shows a list of categories containing the products of the selected manufacturer.
4. When clicking the links in the refine-by list, the request is redirected to the Search Result View page with two constraints: one is the manufacturer constraint, and the other is the selected refinement entry constraint.
5. When clicking the category links in the main content area, the request is also redirected to the Search Result View page with two constraints: one is the manufacturer constraint, and the other is category constraint.

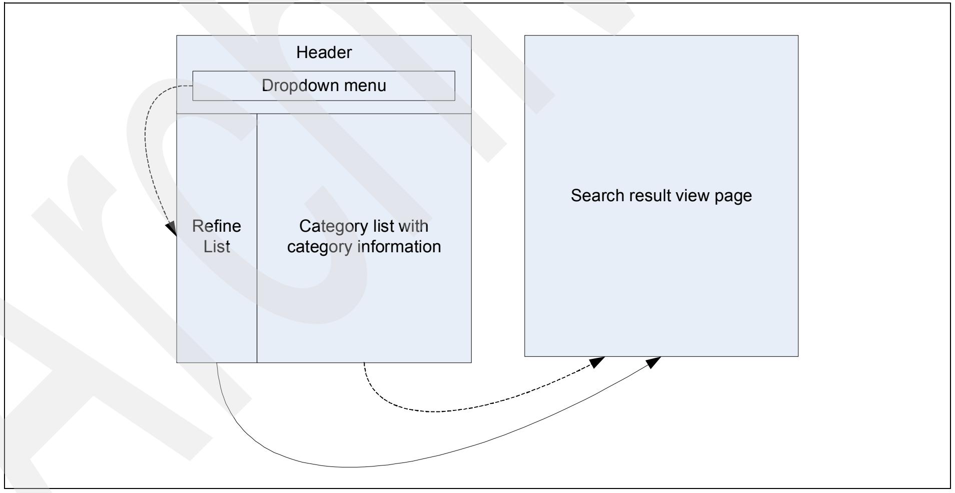


Figure 6-3 By Brand View page flow

OmniFind Discovery Edition provides extensive APIs to ease the search application programming. For basics about the APIs, refer to Chapter 5, “Introducing OmniFind Discovery Edition APIs” on page 67.

For the By Brand implementation, we have to do the following tasks:

- Implement brand list drop-down menu.
- Implement refinement list JSP fragment.
- Implement By Brand View display page.

6.4 Implement brand list drop-down menu

We create a JSP, getHeaderFeatures.jsp, to retrieve manufacturer names from OmniFind Discovery Edition index. Then we display them in Consumer-Direct storefront.

6.4.1 Implement getHeaderFeatures JSP

Create a new JSP file getHeaderFeatures.jsp in the following directory:

<WCS_HOME>\Stores.war\ConsumerDirect\include\styles\style1

Use this JSP to get manufacturer name list from OmniFind Discovery Edition index and pass the information to our sample store header display JSP to display manufacture list in the drop-down menu. The core code to get the manufacturer listing is shown in Example 6-3.

Example 6-3 Snippet for retrieving manufacturer name list

```
<%@ page import="javax.servlet.*"%>
<%@ page import="javax.servlet.jsp.*"%>
<%@ page import="javax.servlet.jsp.PageContext"%>
<%@ page import="java.io.*"%>
<%@ page import="java.util.*"%>
<%@ page import="com.iphrase.runtime.exception.*"%>
<%@ page import="com.iphrase.runtime.query.*"%>
<%@ page import="com.iphrase.runtime.query.result.*"%>
<%@ page import="com.iphrase.onestep.beans.*"%>
<%@ page import="com.iphrase.runtime.query.constraint.*"%>
<%@ page import="com.iphrase.runtime.*"%>
<%@ page import="com.iphrase.runtime.layout.config.*"%>
<%@ page import="com.iphrase.runtime.layout.param.*" %>
<%@ page import="com.iphrase.runtime.layout.util.*" %>


<%@ include file="formatValue.jsp"%>
......
```

```
<%
//Variable for saving retrieved manufacturer names
Vector manufacturerVector = new Vector();
%>
<%
com.iphrase.runtime.query.result.ResultSet resultSet =
queryResultRqst.getMainResultSet();
DrillDown drillDown = resultSet.getDrillDownPlus();
com.iphrase.runtime.query.result.TallyFeature[] tallyFeatures =
drillDown.getTallyFeatures();
%>


<%
if(drillDown != null){

   for (int j = 0; j < tallyFeatures.length; j++){
      com.iphrase.runtime.query.result.TallyFeature currentFeature =
tallyFeatures[j];
      DrillDownList ddList = new DrillDownList(query, currentFeature,
true);

      String currentFeatureLabel = currentFeature.getLabel();

      if (currentFeatureLabel.equals("Manufacturer Name")){
      //Got the MFname
         for(int listIndex = 0; listIndex < ddList.getCount();
listIndex++){
            com.iphrase.onestep.beans.TallyValueHandler valueHandler =
ddList.getValues()[listIndex];

            String formatValueStr = formatValue(valueHandler, "", -1,
"");
            formatValueStr = truncateText(formatValueStr, 100);

            manufacturerVector.addElement(formatValueStr);
      }
      }
%>
```

The first part of the code imports the necessary packages for the JSP file.

The second part of the code includes a formatValue.jsp, which is a utility JSP used for rendering values returned by the OmniFind Discovery Edition server.

The third part of the code gets results and tally features from the sample store.

The last piece of the code goes through two loops. The outer loop goes through each feature to see if it is Manufacturer Name. If it is, it goes through an inner loop to get feature values (manufacturer names) and store them in the manufacturerVector vector.

6.4.2 Modify CachedHeaderDisplay JSP page

The CachedHeaderDisplay.jsp in the directory, <WCS_HOME>\Stores.war\ConsumerDirect\include\styles\style1, is used to display Consumer-Direct starter storefront header, such as top categories list and mini shopping cart.

We modify the CachedHeaderDisplay.jsp file to display the manufacturer list in storefront page header. See Example 6-4.

Example 6-4 Snippet for displaying manufacturer list in storefront page header

```
......
<%@ include file="getHeaderFeatures.jsp"%>
......

<li><a class="hide" >By Brand  </a>
   <a href="#">By Brand
      <table><tr><td>
      <ul>
         <%for(int i=0;i<manufacturerVector.size();i++){

pageContext.setAttribute("brandName",manufacturerVector.elementAt(i));
%>
            <c:url var="ByBrandViewURL" value="WCDSByBrandView">
               <c:param name="MFName" value="${brandName}" />
               <c:param name="catalogId" value="${WCParam.catalogId}"
/>
               <c:param name="storeId" value="${WCParam.storeId}" />
            </c:url>
         <li> <a HREF="<c:out value="${ByBrandViewURL}" />" > <c:out
value="${brandName}" /> </a></li>
         <%}%>
      </ul>
      </td></tr></table>
      </a>
</li>
```

Later in this chapter, we explain the URL format of *WCDSByBrandView* link.

Note: Always use *WCParam* to get request parameter values (in this example, catalogId) within WebSphere Commerce development. *WCParam* and *WCParamValues* are WebSphere Commerce specific versions of the standard implicit JSP objects *param* and *paramValues* to facilitate access to decrypted HTTP request parameters. Although, in many cases, the standard implicit objects provide equivalent functionality to their Commerce-specific counterparts, they are not guaranteed to work properly in the case of encrypted parameters.

To view the manufacturer list drop-down menu example, refresh the Consumer-Direct home page. See the brand list drop-down menus as shown in Figure 6-4.

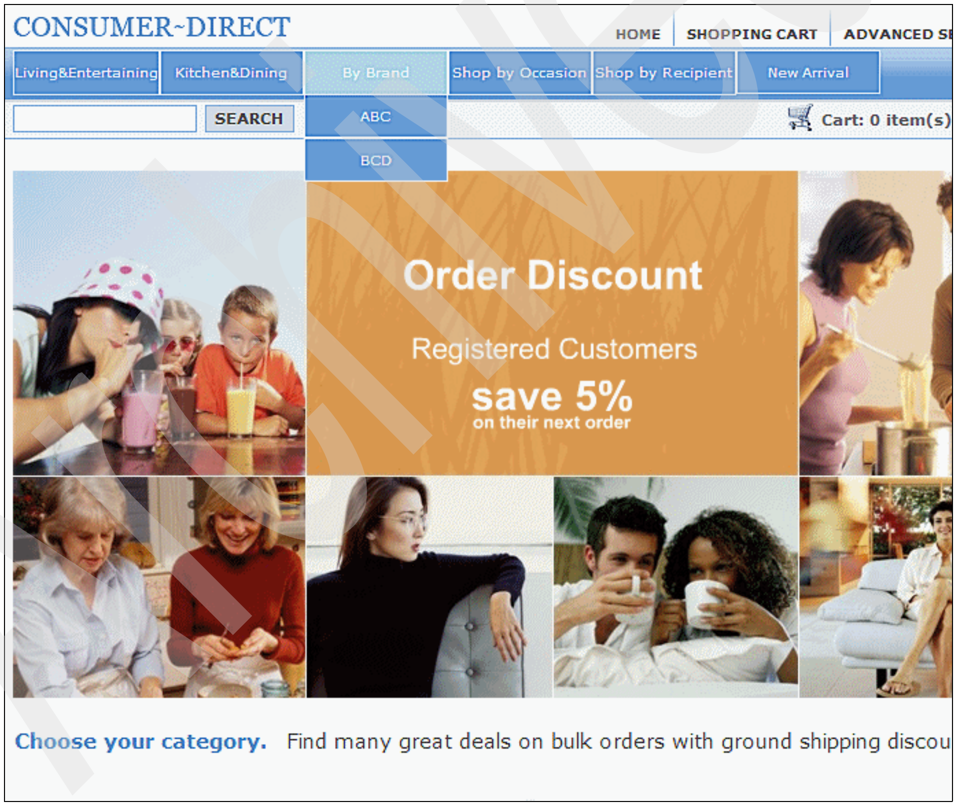


Figure 6-4 By Brand View drop-down menu

6.4.3 Dynamic cache in WebSphere Commerce

You might be curious about the naming of CachedHeaderDisplay.jsp, which has a prefix of Cached. This is a JSP designed for caching in WebSphere Commerce.

In general, caching improves response time and reduces system load. Caching techniques have long been used to improve the performance of Internet applications. Most techniques cache static content (content that rarely changes) such as graphic and text files. However, many Web sites serve dynamic content, containing personalized information or data that changes more frequently. Caching dynamic content requires more sophisticated caching techniques, such as those provided by the WebSphere Application Server 6.0 dynamic cache, a built-in service for caching and serving dynamic content.

WebSphere Commerce uses the WebSphere Application Server dynamic cache service for caching servlets or JSP files and commands that extend from the WebSphere Application Server CacheableCommand interface. The dynamic cache service, servlet caching, and disk off load are enabled by default, during the creation of a WebSphere Commerce instance.

The caching behavior of the WebSphere Application Server dynamic cache service is specified by cache policies defined by <cache-entry> elements in cache specification configuration XML (cachespec.xml) files.

Note: For additional information about WebSphere Commerce dynamic caching, refer to WebSphere Commerce 6.0 Info Center:

`http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/index.jsp`

Select **Administering** → **Dynamic Caching** from the left navigation tree.

6.5 Implement refinement list JSP fragment

A refinement list is useful to guide users to narrow down their search scope. We create a compact refinement list JSP fragment in this section. Compact, in this context, means that this fragment is a reusable asset that you can include anywhere in your customized JSP pages, for example, category display page. You do not have to worry about implementing a refinement list, one by one, for each JSP view page.

We use the refinement list JSP fragment in the By Brand View page.

6.5.1 Refinement list JSP page flow

Figure 6-5 outlines the page flow of the refinement list.

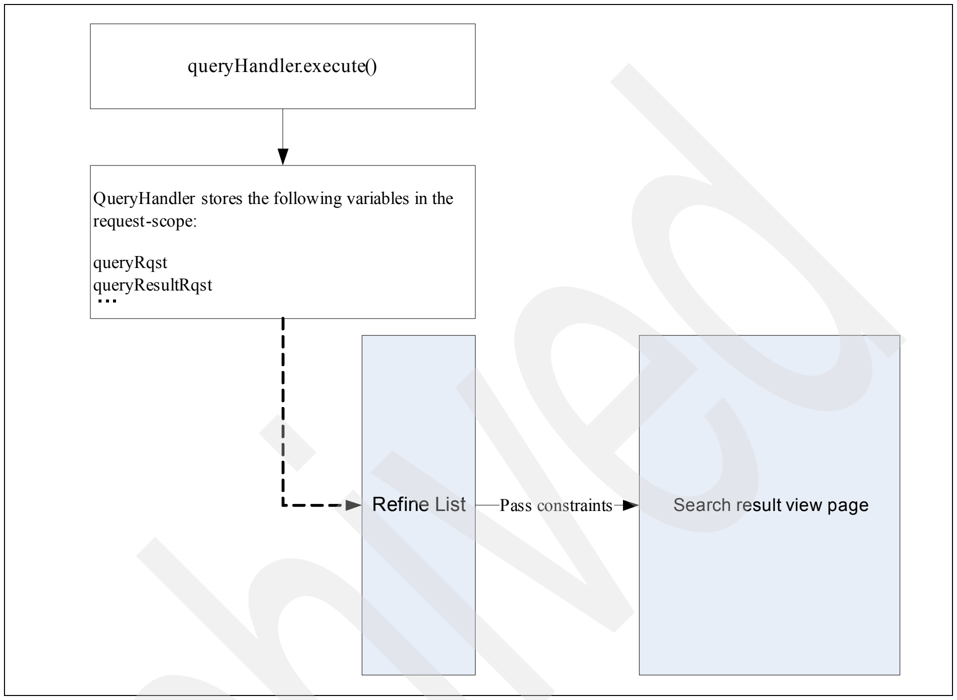


Figure 6-5 Refinement list JSP page flow

Before including the refinement list JSP fragment, you should invoke QueryHandler.execute(). You can set your own constraints in advance, so that the refinement list JSP fragment can display the corresponding content.

After invoking QueryHandler.execute(), QueryHandler builds and stores the following variables in the request-scope so that they are easily accessible from the JSP code:

- queryRqst — an instance of the Query class
- queryResultRqst — an instance of the QueryResult class
- voiceOverRqst — an instance of the VoiceOver bean
- drillDownTabsRqst — an instance of the DrillDownTabs bean
- breadcrumbsRqst — an instance of the Bread crumb bean

We use the first two variables (see Figure 6-5) in our example. The refinement list JSP fragment can retrieve the variables above stored by QueryHandler and use them for displaying the refinement list. When you click the links on the refinement list page, your request is redirected to the Search Result View page.

6.5.2 Sample queryHandler for refinement list JSP fragment

You can create a QueryHandler as usual, and set your constraints, if any. In Example 6-5, we set a manufacturer constraint, so that the refinement list only displays the valid list for the selected manufacturers.

Example 6-5 Sample queryHandler for refinement list JSP fragment

```
<%--*****************************************************************
   CREATE QueryHandler Object prior to using refineList.jspf
*****************************************************************--%>
<%
   //QueryHandler provides high-level access to the Query object.
   QueryHandler queryHandler = new QueryHandler();

   String timeout = (String)request.getParameter("ip_timeout");
   queryHandler.setTimeout(timeout != null ? new
Float(timeout).floatValue() : (float)60);
   queryHandler.addServerAddress("localhost:8777");
   queryHandler.initializeWithPageContext(pageContext);

   String Brand_values[] = {request.getParameter("MFName")};
   Constraint constrt_Brand = new Enumerated.InList("MFName",
Brand_values);

   Conjunction conjunction =
queryHandler.getRuntimeQuery().getConstraints();

   conjunction.merge(new Disjunction(constrt_Brand));

   queryHandler.execute();

%>
```

6.5.3 Display refinement list labels

Create a file refineList.jspf, which can be included in any other page, so you can easily get a refinement list portlet in the future.

Note: Using the JSPF extension for JSP segments is one of the best practices for JSP programming.

A JSP page can consist of one or more files. For example, a JSP page can contain:

- A top-level JSP page
- Several files containing dynamically included JSP pages
- Several files containing statically included JSP segments

Unlike the top-level or dynamically included pages, JSP segments do not have to be legal JSP pages. JSP segments might not compile properly.

To enable code development and support tools, such as the WebSphere Commerce JSP batch compiler, to differentiate between the two types of files, use the .jsp extension for the source files of complete JSP pages and use the .jspf extension for the source files of JSP segments.

To display the refinement list labels, there are several major steps:

1. Retrieve variables stored by QueryHandler.
2. Display refinement list feature labels.
3. Display refinement list feature drill down value labels.

Step 1: Retrieve variables stored by QueryHandler

Use getAttribute() from the pageContext to retrieve variables (queryResultRqst and queryRqst) stored by QueryHandler. The bold text in Example 6-6 contains the source code for the variable retrieval.

From the queryResultRqst variable, you can get the main result set (resultSet) using the getMainResultSet() method.

From the resultSet object, you can get the drill-down object (drillDown) using the getDrillDownPlus() method.

From the drill-down object, you can get the tally features (tallyFeatures) using the getTallyFeatures() method.

You can also get drillDownTabsRqst variable from the page context.

See Example 6-6 for the code snippet.

Example 6-6 refineList.jspf: Retrieve variables stored by QueryHandler

```
<%--*****************************************************************
Retrieve variables stored by QueryHandler in the request-scope
*****************************************************************--%>
<%
   //QueryHandler builds and stores the following variables in the
request-scope
   QueryResult queryResultRqst =
(QueryResult)pageContext.getAttribute("queryResultRqst",
PageContext.REQUEST_SCOPE);
   Query queryRqst = (Query)pageContext.getAttribute("queryRqst",
PageContext.REQUEST_SCOPE);

   com.iphrase.runtime.query.result.ResultSet resultSet =
queryResultRqst.getMainResultSet();
   DrillDown drillDown = resultSet.getDrillDownPlus();
   com.iphrase.runtime.query.result.TallyFeature[] tallyFeatures =
drillDown.getTallyFeatures();
   DrillDownTabs drillDownTabsRqst =
(DrillDownTabs)pageContext.getAttribute("drillDownTabsRqst",
PageContext.REQUEST_SCOPE);
%>
```

Step 2: Display refinement list feature labels

We can display the feature labels first. Figure 6-6 shows some of the feature labels in our sample implementation.

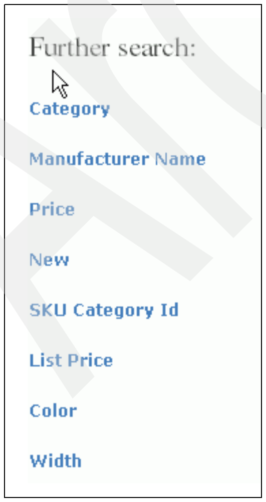


Figure 6-6 refineList.jspf: Display refinement list feature labels

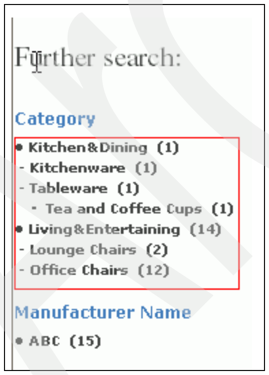
To display the feature labels, go through the tallyFeatures object in a for loop. For each tallyFeature, use the getIsViewByFeature() to make sure that this feature is viewable and then get the feature label by using the getLabel() method. We also get the DrillDownList object (ddList) using the queryRqst variable and the current tallyFeature. See the code snippet in Example 6-7.

Example 6-7   refineList.jspf - Display refinement list feature labels

```
<%
for (int j = 0; j < tallyFeatures.length; j++){
   com.iphrase.runtime.query.result.TallyFeature currentFeature =
tallyFeatures[j];
   DrillDownList ddList = new DrillDownList(queryRqst, currentFeature,
true);

   if(!currentFeature.getIsTabFeature() &&
!currentFeature.getIsViewByFeature()){
      String label = currentFeature.getLabel();%>

      <br>
      <h2><%=label%></h2>
%>
```

Step 3: Display refinement list feature drill-down value labels

After we get the feature labels in Step 2, enhance the page to display sublevel value labels as shown in Figure 6-7.



Further search:

Category
- Kitchen&Dining (1)
- Kitchenware (1)
- Tableware (1)
  - Tea and Coffee Cups (1)
- Living&Entertaining (14)
- Lounge Chairs (2)
- Office Chairs (12)

Manufacturer Name
- ABC (15)

Figure 6-7   refineList.jspf: Display refinement list feature drill-down value labels

We obtained the DrilldownList object (ddList) in Step 2. For each feature, go through its ddList, get their values, and use valueHandler and the formatValue() method to format the output displayed in Figure 6-7 on page 111. Example 6-8 implements the appearance and behavior of the page.

Example 6-8   refineList.jspf: Display refinement list feature drill-down value labels

```
<h2><%=label%></h2>
<%
for(int listIndex = 0; listIndex < ddList.getCount(); listIndex++){
   com.iphrase.onestep.beans.TallyValueHandler valueHandler =
ddList.getValues()[listIndex];
   String formatValueStr = formatValue(valueHandler, "", -1, "");
   formatValueStr = truncateText(formatValueStr, 100);

   %>

<%--*****************************************************************
*  Decide which symbol to display to indicate the recursion level, e.g.
*
*       O Kitchen&Dining
*        - Kitchenware
*        - Tableware
*          . Tea and Coffee Cups
*****************************************************************--%>

            <%if(valueHandler.getRecursionLevel() == 0){%>
               &bull;
            <%}else if(valueHandler.getRecursionLevel() == 1){%>
               &nbsp;-
            <%}else{%>
               &nbsp;&nbsp;&nbsp;&middot;
            <%}%>

            <%=formatValueStr%>

            <%-- Prints the tally count of the value, this is optional
--%>
            <%if(valueHandler.getTallyCount() > 0 &&
valueHandler.getIsRefineShowCounts()){%>
               &nbsp;(<%=valueHandler.getTallyCount()%>)
            <%}%>

            </a>
<%}%>
```

6.5.4 Pass multiple constraints to Search Result View page

In order to make refineList.jspf work, ensure that it can pass constraints to the Search Result View page so that we can use the Search Result View page to display the search results.

In many scenarios, we have multiple constraints. For example, as Figure 6-8 shows, there are two constraints for the By Brand View page. The first constraint, Constraint 1, is the manufacturer constraint. When a user clicks the link in the refine list, a second constraint is generated for the drill-down. So we have to let refineList.jspf know the accumulated query status.

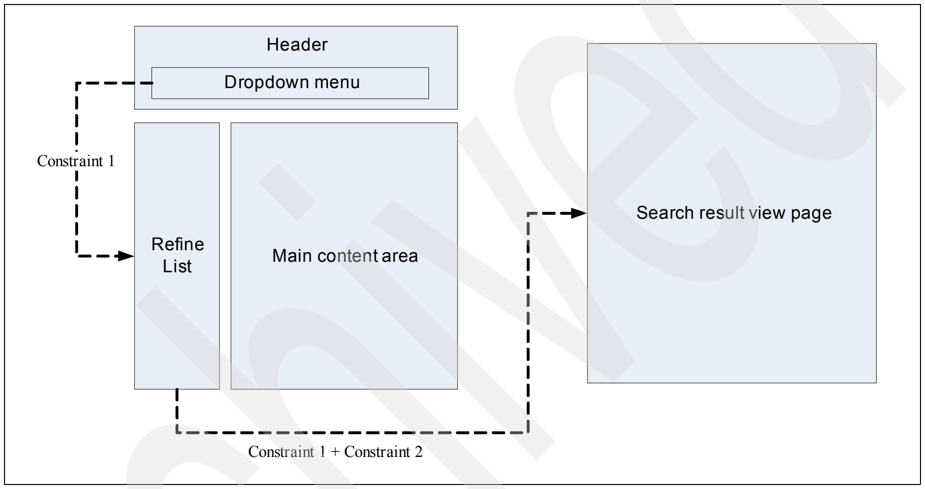


Figure 6-8 Pass multiple constraints to Search Result View page

OmniFind Discovery Edition has mechanisms to pass multiple constraints at one time. It uses a *serialization string* to save the resolved query status, which can be used for a follow up query.

Follow the steps below to pass multiple constraints:

1. Open <WCS_HOME>\Stores.war\ConsumerDirect\include\styles\style1\CachedHeaderDisplay.jsp.

   Locate the form, CatalogSearchForm. Add a new field, *ip_state*, inside the form CatalogSearchForm. See the bold text in Example 6-9. Use this field to save the resolved query status serialization string.

Example 6-9 Add ip_state form field

```
<form name="CatalogSearchForm" action="CatalogSearchResultView"
method="get" id="CatalogSearchForm">
... ...
<input type="hidden" name="storeId" value='<c:out
value="${WCParam.storeId}" />'
id="WC_CachedHeaderDisplay_FormInput_storeId_In_CatalogSearchForm_1"/>
<input type="hidden" name="catalogId" value='<c:out
value="${WCParam.catalogId}"/>'
id="WC_CachedHeaderDisplay_FormInput_catalogId_In_CatalogSearchForm_1"/
>
... ...
<input type="hidden" name="ip_state" value="" />

</form>
```

2. Open <WCS_HOME>\Stores.war\ConsumerDirect\ByBrandView\WCDSByBrandView.jsp.

   Inside the JavaScript™ function updater(), get the resolved query state by calling getResolvedQueryState() method from the resultSet object. See the bold text in Example 6-10.

Example 6-10 Get serialization string of resolved Query

```
function updater(key, value) {
   eval("document.CatalogSearchForm." + key + ".value = '" + value +
"';");
   document.CatalogSearchForm.ip_state.value =
'<%=resultSet.getResolvedQueryState()%>';

   doSubmit();
}
```

   For more information about these APIs, refer to OmniFind Discovery Edition Runtime API Java doc.

6.5.5 Display refinement list feature drill-down value links

From the previous sections, we got the refinement list labels and we have the JavaScript functions to pass constraints. Now it is easy to add the links for the feature drill-down values so that users can click them for further searches. To do so, add a new line (the bold text) as shown in Example 6-11.

Example 6-11 refineList.jspf: Display refinement list feature drill-down value links

```
<h2><%=label%></h2>
<%
for(int listIndex = 0; listIndex < ddList.getCount(); listIndex++){
   com.iphrase.onestep.beans.TallyValueHandler valueHandler =
ddList.getValues()[listIndex];
   String formatValueStr = formatValue(valueHandler, "", -1, "");
   formatValueStr = truncateText(formatValueStr, 100);

   %>


<%--*********************************************************************
*  Decide which symbol to display to indicate the recursion level, e.g.
*
*       O Kitchen&Dining
*        - Kitchenware
*        - Tableware
*          . Tea and Coffee Cups
*******************************************************************--%>
           <a class="sn_link" href="javascript:updater('ip_constrain',
encodeURIComponent('<%=Tools.escapeJS(valueHandler.getConstraint().toSt
ring())%>'))">

           <%if(valueHandler.getRecursionLevel() == 0){%>
              &bull;
           <%}else if(valueHandler.getRecursionLevel() == 1){%>
              &nbsp;-
           <%}else{%>
              &nbsp;&nbsp;&nbsp;&middot;
           <%}%>

           <%=formatValueStr%>

           <%-- Prints the tally count of the value, this is optional
--%>
           <%if(valueHandler.getTallyCount() > 0 &&
valueHandler.getIsRefineShowCounts()){%>
              &nbsp;(<%=valueHandler.getTallyCount()%>)
           <%}%>

           </a>
<%}%>
```

Refresh the WCDSByBrandView JSP page to see the changes.

6.6 Implement By Brand View display page

We create separate JSPs for the By Brand View feature. We show you how you can reuse the existing JSP snippets to compose a new JSP to satisfy a new requirement.

6.6.1 Preparation

To accomplish the preparation work, do the following steps:

1. Locate <WCS_HOME>\Stores.war\ConsumerDirect.
2. Create a new directory, ByBrandView.
3. Create a new JSP WCDSByBrandView.jsp under the ByBrandView directory.

Define three input parameters for WCDSByBrandView.jsp:

- MFName: Used to specify the manufacturer name or brand name. Obtain this value from the manufacturer list drop-down menu.
- catalogId: Used to retrieve category detailed information.
- storeId: Used for page context.

6.6.2 Get the category ID: Server side setting

In order to display a list of categories that contains the products from a specified manufacturer, we retrieve the categoryId first from OmniFind Discovery Edition index. We then use this categoryId to retrieve additional information from the WebSphere Commerce server.

Commerce Module for OmniFind Discovery Edition does not have an explicit category ID feature. OmniFind Discovery Edition has its own data structure for category type data. It views the WebSphere Commerce category as Taxonomy-type feature within its index.

Note: The taxonomy feature uses a hierarchy of categories to organize content. The taxonomy level is used by the system to configure various aspects of the rendering, such as the groups in which the search results are displayed.

Commerce Module for OmniFind Discovery Edition has a taxonomy feature called a *sitemap taxonomy id*, which is defined in its index. This feature corresponds to the category concept in WebSphere Commerce. Open the file, <ODE_HOME>\deployment\wcs\wcs\lib\wcs\feature.txt. You can see that the attributes are defined for sitemap taxonomy id as shown in Example 6-12.

Example 6-12 Default settings of sitemap taxonomy id

```
"sitemap taxonomy id"
  db: taxonomy.taxonomy_id
  multiselect: table
  type: text
  build_grammar: no
  render_label: SKU Category Id
  mc_group: "MetaData"
  build_grammar: no
  build_lexicon: no
  build_ir: no
  multiselect: table
  tally: no
  enable_feature_for_sorting: no
  render_summarized_keep_duplicate_values: no
```

In order to get the values of this feature:

- Set *tally* to yes. This allows you to get the number of results for each distinct value or each range of values.
- Set *render_refine_show_single_value* to yes. This configures OmniFind Discovery Edition to display the result value even if there is only one result.
- Set *enum* to yes. This configures the feature to be an enumerated type.

See the bold text in Example 6-13.

Example 6-13 New settings of sitemap taxonomy id

```
"sitemap taxonomy id"
  db: taxonomy.taxonomy_id
  multiselect: table
  ...
  tally: yes
  enable_feature_for_sorting: no
  render_summarized_keep_duplicate_values: no
  render_refine_show_single_value: yes
  enum: yes
```

To make the settings effective, do the following steps:

1. Rebuild OmniFind Discovery Edition index:
   a. From a command prompt, go to your <ODE_HOME> directory.
   b. Enter the following command:

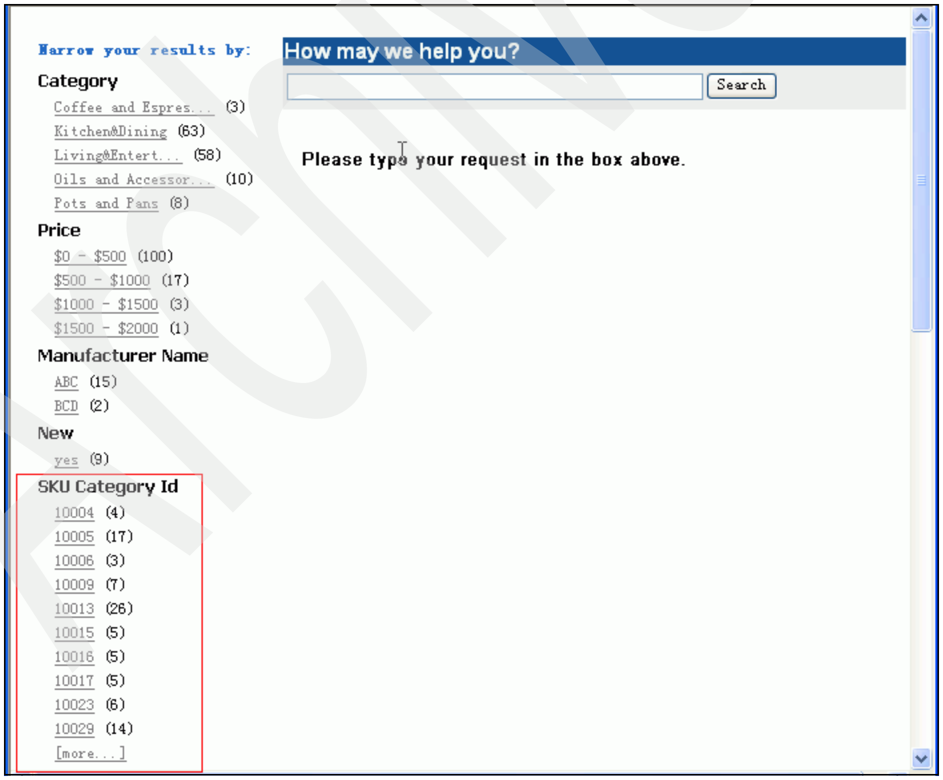
      ```
      bin\iphrase build -p deployment\wcs\wcs\default.prp
      ```

2. Restart OmniFind Discovery Edition Windows service:
   a. Select **Start** → **Settings** → **Control Panel** → **Administrative Tools** → **Services**.
   b. Select **IBMODEwcs** from the service list and restart it.

To view the effect of the settings above, open the following URL in a Web browser:

`http://`*`wcds_server_hostname`*`:8777/iphrase/query?render=1`

You should see a new group of SKU Category Id in the refine list as shown in Figure 6-9 (assuming that you imported the book data earlier).



Narrow your results by:

Category
- Coffee and Espres... (3)
- Kitchen&Dining (63)
- Living&Entert... (58)
- Oils and Accessor... (10)
- Pots and Pans (8)

Price
- $0 - $500 (100)
- $500 - $1000 (17)
- $1000 - $1500 (3)
- $1500 - $2000 (1)

Manufacturer Name
- ABC (15)
- BCD (2)

New
- yes (9)

SKU Category Id
- 10004 (4)
- 10005 (17)
- 10006 (3)
- 10009 (7)
- 10013 (26)
- 10015 (5)
- 10016 (5)
- 10017 (5)
- 10023 (6)
- 10029 (14)
- [more...]

How may we help you?

Search

Please type your request in the box above.

Figure 6-9 SKU Category Id

6.6.3 Get the category ID: JSP implementation

After making category ID settings from the OmniFind Discovery Edition server side, you can now get the category IDs using OmniFind Discovery Edition APIs.

Go through the tallyFeatures in a loop. For each tallyFeature, check if it is SKU Category Id. The *SKU Category Id* is the label of the *sitemap taxonomy id feature*. If it is, get its values. This is similar to the steps of getting a common feature information. Example 6-14 shows how to get it with OmniFind Discovery Edition APIs. Bold text represents the code related to this implementation.

Example 6-14 Get category ID from OmniFind Discovery Edition index

```
<%
   Vector category_IDs = new Vector();   //Save category IDs
   Vector category_Links = new Vector(); //Save category links
%>
<%
   for (int j = 0; j < tallyFeatures.length; j++){
      com.iphrase.runtime.query.result.TallyFeature currentFeature =
tallyFeatures[j];
      DrillDownList ddList = new DrillDownList(query, currentFeature,
true);

      String currentFeatureLabel = currentFeature.getLabel();
      if (!currentFeatureLabel.equals("SKU Category Id")){
      continue;
      }


      for(int listIndex = 0; listIndex < ddList.getCount();
listIndex++){
         com.iphrase.onestep.beans.TallyValueHandler valueHandler =
ddList.getValues()[listIndex];

         String formatValueStr = formatValue(valueHandler, "", -1, "");
         formatValueStr = truncateText(formatValueStr, 100);
         category_IDs.addElement(formatValueStr);

category_Links.addElement(valueHandler.getConstraint().toString());
      }
   }
%>
```

6.6.4 Retrieve category information from WebSphere Commerce

With category ID, we can retrieve more detailed information from WebSphere Commerce, such as category short description.

Example 6-15 shows you how to retrieve the image and short description from WebSphere Commerce for the categories containing products of the specified manufacturer, and set the image hyper links with OmniFind Discovery Edition APIs for further user searches. The bold text in the example highlights the important part of the code related to this implementation.

Example 6-15 Display category information with further search links

```
<!--///////DISPLAY Category one by one//////////////////////-->
<table cellpadding="0" cellspacing="0" border="0" class="t_table"
valign="top" id="table">

<%
   for (int j = 0; j < category_IDs.size(); j++){

pageContext.setAttribute("brandCategoryId",category_IDs.elementAt(j));

pageContext.setAttribute("brandCategoryLink",category_Links.elementAt(j
));

           CategoryDataBean brandCategoryBean = new
CategoryDataBean();

brandCategoryBean.setCatalogId(pageContext.getAttribute("catalogId").to
String());

brandCategoryBean.setCategoryId(category_IDs.elementAt(j).toString());
           com.ibm.commerce.beans.DataBeanManager.activate
(brandCategoryBean, commandContext);

%>

           <tr>

              <%-- Show category image and short description if
available --%>
              <td class="t_img_view">

              <!--///////DISPLAY THUMBNAIL//////////////////////-->

              <%
```

```
                  String imgFurtherSearchLink =
"javascript:updater('ip_constrain',
encodeURIComponent('"+Tools.escapeJS(category_Links.elementAt(j).toStri
ng())+"'))";
                  if(!
brandCategoryBean.getDescription().getThumbNail().equals("")) {%>
                              <a href="<%=imgFurtherSearchLink%>">
                              <span class="t_img_border">
                              <img

src=<%=brandCategoryBean.getObjectPath()+brandCategoryBean.getDescripti
on().getThumbNail()%>

alt=<%=brandCategoryBean.getDescription().getShortDescription()%>
                                 border="0" />
                              </span>
                              </a>
               <%}else{%>
                              <a href="<%=imgFurtherSearchLink%>">
                              <img
                                 src="<c:out
value="${jspStoreImgDir}"/>images/NoImageIcon.jpg"
                                 alt="<fmt:message key="No_Image"
bundle="${storeText}"/>"
                                 border="0"/>
                              </a>
               <%}%>
               <!--///////END DISPLAY
THUMBNAIL//////////////////////-->

               <br/>

               <!--///////DISPLAY CATEGORY
NAME//////////////////////-->
               <%if(!
brandCategoryBean.getDescription().getShortDescription().equals(""))
{%>
                        <br/>
                        <span class="productName">
                           <a
href="<%=imgFurtherSearchLink%>"><%=brandCategoryBean.getDescription().
getName()%> </a>
                        </span>
                        <br/>
               <%}%>
```

```
              <!--///////END DISPLAY CATEGORY
NAME//////////////////////-->
              </td>
           </tr>

<%
   }
   //End For loop
%>
</table>

<!--///////END DISPLAY Category one by one/////////////////////-->
```

6.7 Create Struts view for WCDSByBrandView.jsp

In this section, we register a Struts view entry for the newly developed WCDSByBrandView.jsp. After that, we load access control policy for the new view to ensure that everyone has the authority to view the JSP page.

Besides the inherent advantage of the Struts framework, you get additional benefits by creating a new view for your presentation JSP files:

- With the created Struts view, you do not have to worry about the WebSphere Commerce command dynamic links generated using the relative path within your JSP.
- You can easily embrace the WebSphere Commerce security features for the access control of your JSP pages.

Generally, creating a new view includes the following steps:

- Name the view and map it to the JSP page in the struts-config-ext.xml file.
- Create a new properties file in which translatable text for JSP pages are stored.

Note: In our example, we do not cover creating of the new properties file.

- Create and load access control policies for the view.

6.7.1 Add Struts mappings entries

We use *WCDSByBrandView* as the view name for WCDSByBrandView.jsp in our example. You can name the view to whatever you want; this is just a mapping.

To add new entries for WCDSByBrandView in the Struts configuration file, do the following steps:

1. Open <WCS_HOME>\Stores.war\WEB-INF\struts-config-ext.xml.
2. Under <global-forwards> section, add the WCDSByBrandView entry as shown in bold text of Example 6-16.

Example 6-16 Add global-forwards setting for new JSP in Struts configuration file

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD
Struts Configuration 1.1//EN"
"http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd">
<struts-config>
<!--version=0-->
   <!-- Data Sources -->
   <data-sources>
   </data-sources>

   <!-- Form Beans -->
   <!-- Global Exceptions -->
   <global-exceptions>
   </global-exceptions>

   <!-- Global Forwards -->
   <global-forwards>

<forward className="com.ibm.commerce.struts.ECActionForward"
name="WCDSByBrandView/10001" path="/ByBrandView/WCDSByBrandView.jsp"/>
......
```

The String WCDSByBrandView/10001 means it is for a store with a store ID of 10001. You can change it to be another store ID if this is not applicable. To look up your store ID, use the following SQL command:

```
select STOREENT_ID from STOREENT where IDENTIFIER =
'<ConsumerDirect_name>';
```

Where: *<ConsumerDirect_name>* is the name of your store.

3. Add action mapping for /WCDSByBrandView under in struts-config-ext.xml as shown in Example 6-17.

Example 6-17 Add action mapping for new View in Struts configuration file

```
   <!-- Action Mappings -->
   <action-mappings type="com.ibm.commerce.struts.ECActionMapping">

<action path="/WCDSByBrandView"
type="com.ibm.commerce.struts.BaseAction">
<set-property property="credentialsAccepted" value="0:P"/>
</action>
......
```

Where, in this coding example:

- credentialsAccepted:

  The value of P indicates that partially authenticated users are entitled to access this resource.

6.7.2 Load access control policy for WCDSByBrandView

We have created the mappings for WCDSByBrandView in Struts configuration file. To enable users to see the new view, we must enable access control policy for the new view.

Steps to enable user access to the new view are as follows:

1. Create a new access control policy XML file for WCDSByBrandView. For our example, we name this file, createWCDSByBrandView.xml, with the content as shown in Example 6-18. Place it into the directory:

   C:\Program Files\IBM\WebSphere\CommerceServer60\xml\policies\xml

Example 6-18 Sample XML for new Struts View access control

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>

<Policies>
<!--
 ::::::::::::::::::::::::::::::::::::::::::::::::::

 This file does not contain any translatable data.

 ::::::::::::::::::::::::::::::::::::::::::::::::::
-->
<!--
 This segment used instead of DTD during non-build validation

<Policies xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation='../xsd/AccessControl.xsd'
  version="1.0">

-->
 <Action Name="WCDSByBrandView" CommandName="WCDSByBrandView"/>

 <ActionGroup Name="ConsumerDirectAllUsersViews"
OwnerID="RootOrganization">
  <ActionGroupAction Name="WCDSByBrandView"/>
   </ActionGroup>
</Policies>
```

2. At a command prompt, navigate to the following directory: C:\Program Files\IBM\WebSphere\CommerceServer60\bin.
3. Run the acpload command:

   ```
   acpload <db_host_name> <db_name> <db_user> <db_password>
   createWCDSByBrandView.xml <schema_name>
   ```

   Where:

   – *<db_host_name>* is the host name of the machine on which your database runs. This parameter is required for remote databases only.
   – *<db_name>* is the name of your database.
   – *<db_user>* is the name of the database user.
   – *<db_password>* is the password for your database user.
   – *<schema_name>* is the database schema name. For DB2, it is the database user who created the database and owns the schema.

   For our case study, we issue this command:

   ```
   acpload mall db2admin password createWCDSByBrandView.xml db2admin
   ```

4. Restart WebSphere Commerce server to ensure that the Struts configuration changes and the new access control policies become effective. To restart Commerce Server:
   a. Navigate to directory:

      <WAS_HOME>\profiles\demo\bin

   b. Run the following command:

      ```
      startServer.bat server1
      ```

7

Displaying new products in your site

Having a designated page to show new products in your site allows you to present the latest products to shoppers, as well as enhancing their navigation experience. In addition, it encourages shoppers to visit your site regularly to check for new products.

In this chapter we discuss how to build the new arrival feature on a WebSphere Commerce sample store using OmniFind Discovery Edition. We cover the following topics:

- Designing the page flow
- Customizing data structure in WebSphere Commerce
- Creating a corresponding feature
- Rebuilding project in OmniFind Discovery Edition
- Creating metadata rule to map WCS data into feature
- Testing the intermediate result
- Presenting the results in your Web site
- Testing the results in your Web site

7.1 Designing the page flow

To allow shoppers to find what they want more easily, New Arrival is displayed on the top navigation menu (see Figure 7-1). By doing this, we can narrow the search by searching for new products as a constraint.

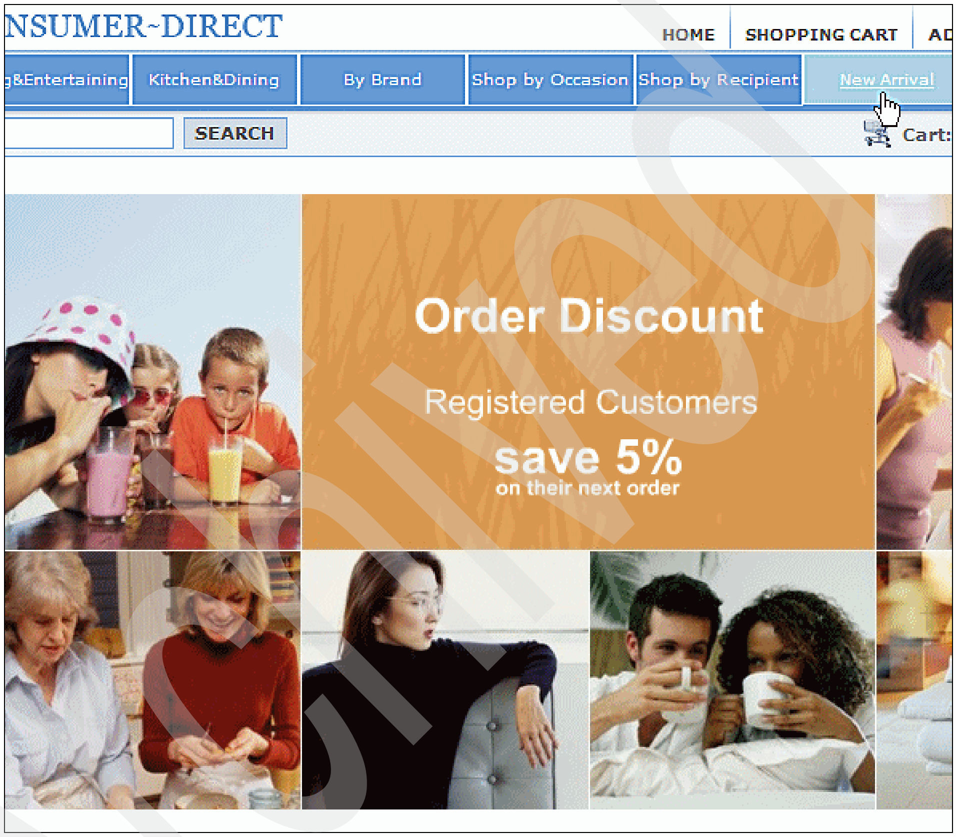


Figure 7-1 Home page having new arrival link from selection

After the search is submitted, the search results page displays all products that match the search criteria.

7.2 Customizing data structure in WebSphere Commerce

In this section we discuss the rationale of designing the data structure and describe different ways to prepare the data.

7.2.1 Data design

New arrival is a product specific attribute. Each product should have a flag to tell whether it is a new product or not. In WebSphere Commerce, the new arrival should be set at item level, because item is the actual object to be purchased. A product is a group of items that exhibit the same attributes, and it might not accurately reflect the new arrival data. For example, a product might consist of a type of a shirt. Each item might be a particular color and size the shirt. If the shirt is not a new product, but the red colored shirt is new, this can only be set at the item level.

The CATENTRY table in WebSphere Commerce contains a few customizable columns. CATENTRY.FIELD1 is one of the customizable columns, and we use this field to store our new arrival data. CATENTRY.FIELD1 has an integer type, and we define the meaning as in Table 7-1.

Table 7-1 CATENTRY.FIELD1 values

CATENTRY.FIELD1 value	Meaning
1	Item is new.
Other integer value	Item is not new.
Null	Item is not new.

As mentioned before, the data has to be set at the item level, so make sure that CATENTRY.CATENTTYPE_ID='ItemBean' and not 'ProductBean'.

7.2.2 Data used in our sample application

For demonstration purposes, a few items in the WebSphere Commerce catalog are marked as new, as shown in Table 7-2.

Table 7-2 New products in our sample catalog

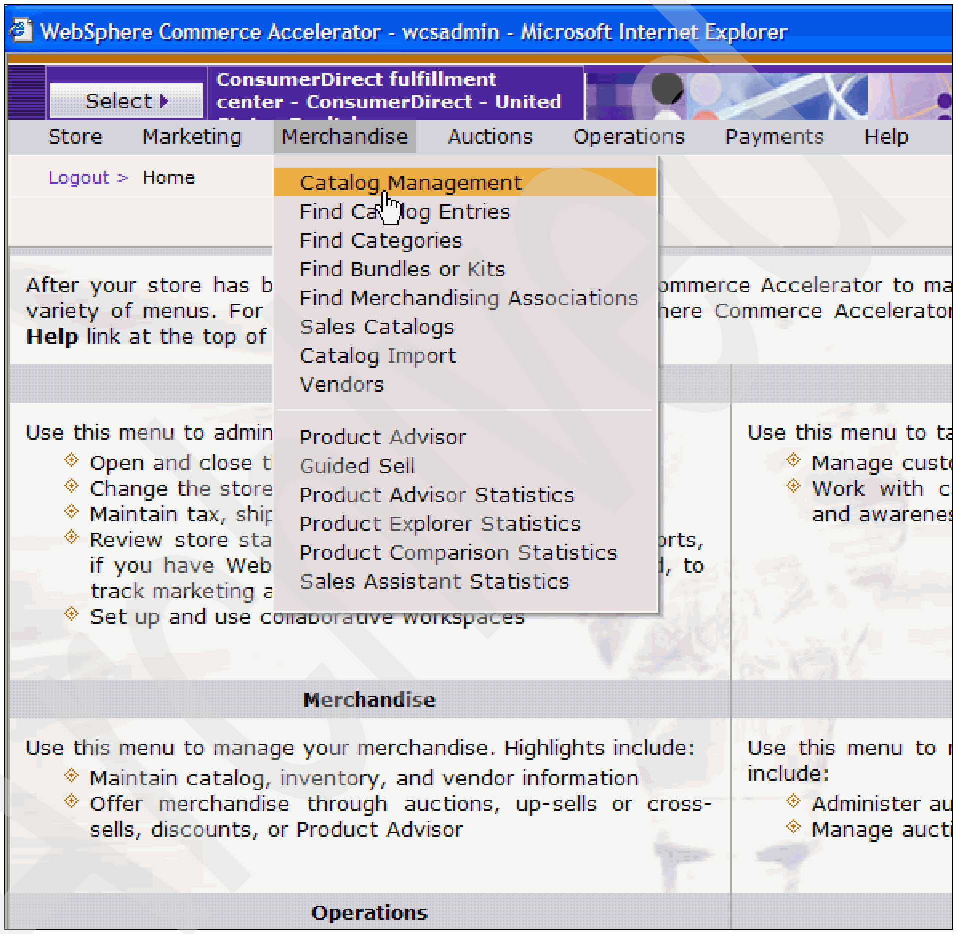
Item Name	Item SKU	Parent Category Tree
White Fabric Roll Arm Chaise	FULO-0101	Living&Entertaining - Lounge Chairs
Red Leather Roll Arm Chaise	FULO-0201	Living&Entertaining - Lounge Chairs
Steamer Pot	KIPO-0101	Kitchen&Dining - Kitchenware - Pots
“Havenwood” Dinnerware Set, No Trim, serving size 1, Salad Plate	TAPL-0102	Kitchen&Dining - Tableware - Plates
“Havenwood” Dinnerware Set, No Trim, serving size 4, Salad Plate	TAPL-0105	Kitchen&Dining - Tableware - Plates
“Havenwood” Dinnerware Set, No Trim, serving size 8, Salad Plate	TAPL-0108	Kitchen&Dining - Tableware - Plates
“Havenwood” Dinnerware Set, Red Trim, serving size 1, Salad Plate	TAPL-0111	Kitchen&Dining - Tableware - Plates
“Havenwood” Dinnerware Set, Red Trim, serving size 4, Salad Plate	TAPL-0114	Kitchen&Dining - Tableware - Plates
“Havenwood” Dinnerware Set, Red Trim, serving size 8, Salad Plate	TAPL-0117	Kitchen&Dining - Tableware - Plates

The data can be set up in two ways, using the WebSphere Commerce Accelerator to update the catalog data, or to update the database directly.

Using Commerce Accelerator to populate sample data

We can use the Catalog Management tool inside WebSphere Commerce Accelerator to define the data we require. Follow the steps below to manually set up the data using WebSphere Commerce Accelerator:

1. After logging on to the WebSphere Commerce Accelerator, select **Merchandise** → **Catalog Management**. See Figure 7-2.


WebSphere Commerce Accelerator - wcsadmin - Microsoft Internet Explorer
Select
ConsumerDirect fulfillment center - ConsumerDirect - United
Store
Marketing
Merchandise
Auctions
Operations
Payments
Help
Logout > Home
Catalog Management
Find Catalog Entries
Find Categories
Find Bundles or Kits
Find Merchandising Associations
Sales Catalogs
Catalog Import
Vendors
Product Advisor
Guided Sell
Product Advisor Statistics
Product Explorer Statistics
Product Comparison Statistics
Sales Assistant Statistics
Merchandise
Operations

After your store has b ommerce Accelerator to ma variety of menus. For here Commerce Accelerator **Help** link at the top of

Use this menu to admin
- Open and close t
- Change the store
- Maintain tax, ship
- Review store sta orts, if you have Web d, to track marketing a
- Set up and use collaborative workspaces

Use this menu to ta
- Manage cust
- Work with c and awarene

Use this menu to manage your merchandise. Highlights include:
- Maintain catalog, inventory, and vendor information
- Offer merchandise through auctions, up-sells or cross-sells, discounts, or Product Advisor

Use this menu to include:
- Administer au
- Manage aucti

Figure 7-2 Commerce Accelerator: Launching the Catalog Management tool

2. Follow the parent category tree information provided in Table 7-2 to find the first item (White Fabric Roll Arm Chaise). Click the button **List Catalog Entries** to see a list of products under the selected category. See Figure 7-3.

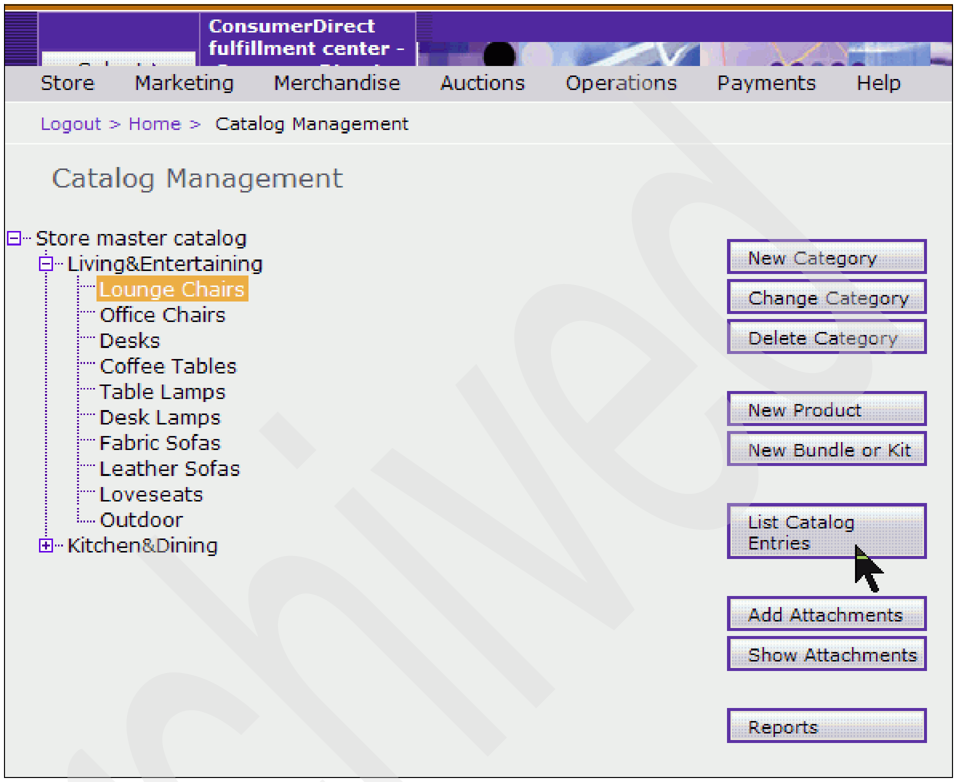
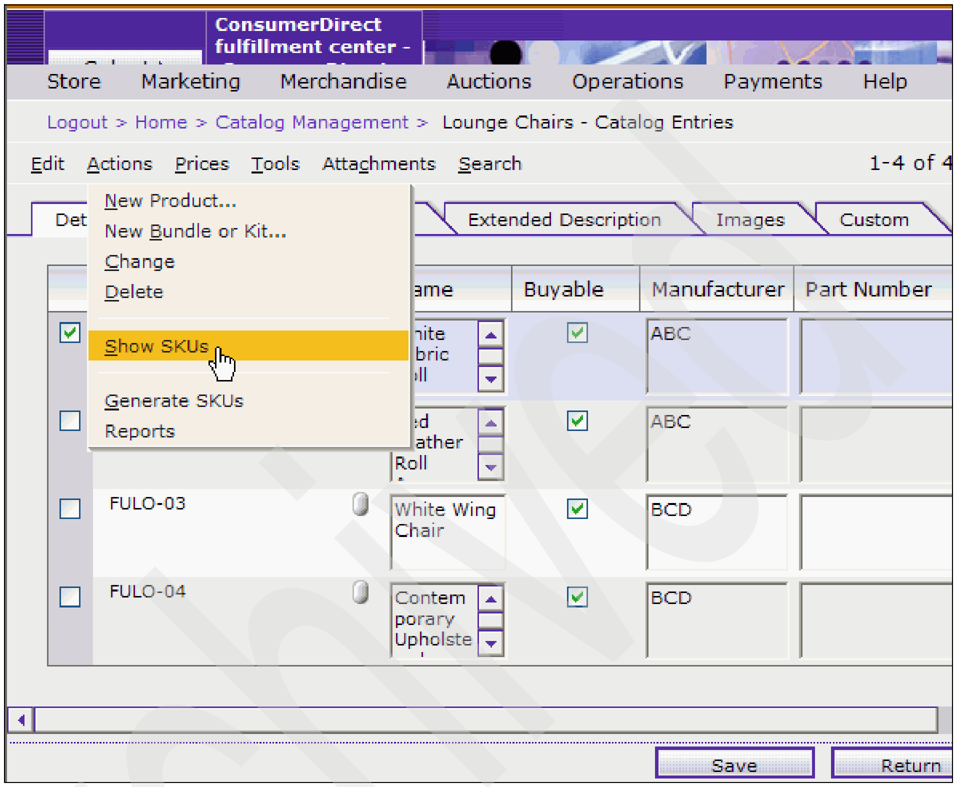


Figure 7-3 Commerce Accelerator: Listing products under selected category

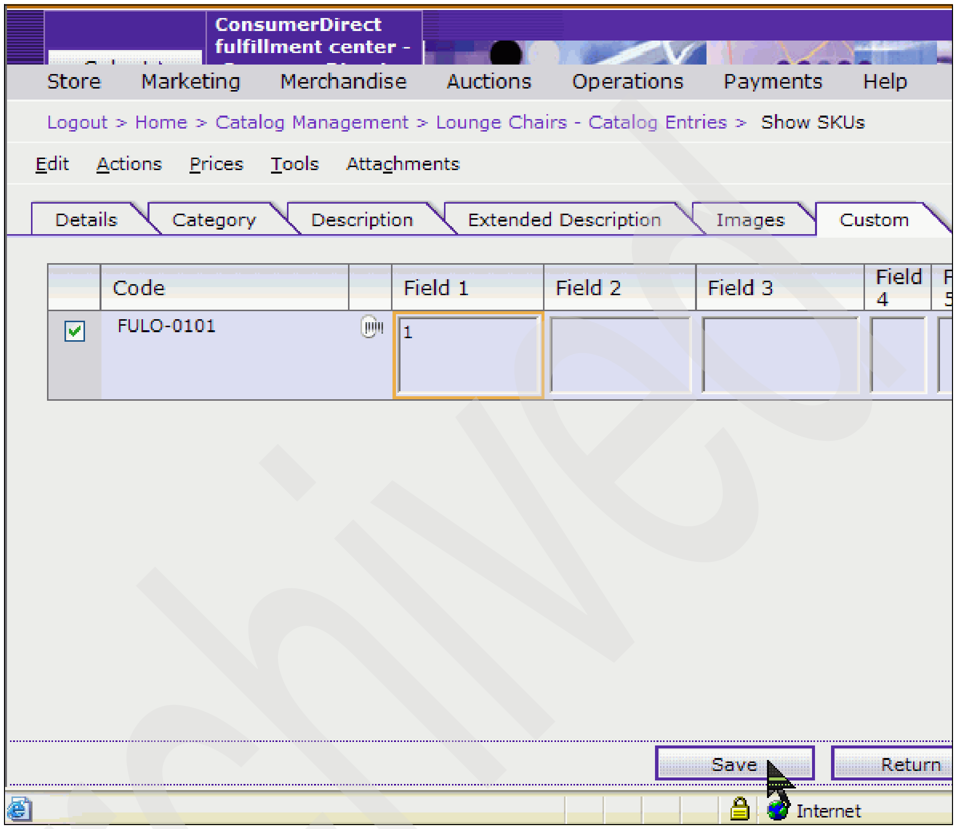
3. Select the product (White Fabric Roll Arm Chaise for the first item) and select **Actions** → **Show SKUs** from the drop-down menu. See Figure 7-4.



		ame	Buyable	Manufacturer	Part Number
✓		hite bric ll	✓	ABC	
		ed ather Roll	✓	ABC	
	FULO-03	White Wing Chair	✓	BCD	
	FULO-04	Contem porary Upholste	✓	BCD	

Figure 7-4 Commerce Accelerator: Accessing items of a product

4. Switch to the custom tag, locate the item (FULO-0101), and specify 1 (one) under field 1. See Figure 7-5. Click **Save** to save the changes.



	Code		Field 1	Field 2	Field 3	Field 4
✓	FULO-0101		1			

Figure 7-5 Commerce Accelerator: Setting field1 to 1 (new) and save the changes

5. Repeat steps 1 to 4 for all items defined in Table 7-2 on page 130.

Tip: For a production environment, we recommend mass update via scripts. Refer to the next section for loading the data using SQL statements.

Updating Commerce database to populate sample data

A copy of the Commerce database, which contains all sample data used for this book, can be obtained from the Redbooks download page. Refer to Appendix A, “Additional material” on page 271 for download instruction.

Alternatively, you can run a script file to issue update statements for the data we require in this chapter. Get a copy of the script file from the downloaded material, chapter_7_New_Arrival\NewArrival_data.sql.

When you get the script file, run the update statements as follows:

1. Open a DB2 Command Window.
2. Go to the directory where the script file, NewArrival_data.sql, is downloaded.
3. Connect to the database by issuing the statement:

   ```
   db2 connect to <DB name> user <DB schema owner ID> using <password>
   ```

   For our case study, <db name> is "mall", <user id> is "db2admin", and <password> is "password".

   So, we use the following command:

   ```
   db2 connect to mall user db2admin using password
   ```

4. Run the script file by issuing the statement:

   ```
   db2 -tvf NewArrival_data.sql
   ```

5. To check if the data is populated, issue the following command:

   ```
   db2 -tvf NewArrival_listdata.sql
   ```

You should get a list some records, with Field1 field set to 1.

The update statements are shown in Example 7-2.

Example 7-1 SQL statements to populate sample data

```
update catentry set field1=1 where partnumber=’FULO-0101’;
update catentry set field1=1 where partnumber=’FULO-0201';
update catentry set field1=1 where partnumber=’KIPO-0101';
update catentry set field1=1 where partnumber=’TAPL-0102';
update catentry set field1=1 where partnumber='TAPL-0105';
update catentry set field1=1 where partnumber='TAPL-0108';
update catentry set field1=1 where partnumber='TAPL-0111';
update catentry set field1=1 where partnumber='TAPL-0114';
update catentry set field1=1 where partnumber='TAPL-0117';
```

7.3 Creating a corresponding feature

In our example, when the user clicks **New Arrival** from the store home page, a search is submitted to OmniFind Discovery Edition for products that are marked as new. Then a list is returned showing products that match the search criteria.

To accomplish this, a feature should be created in OmniFind Discovery Edition so that the search can interpret a query about new arrival products. We call the new feature **New**.

To create this feature in OmniFind Discovery Edition, we use the Management Console from OmniFind Discovery Edition as described in the following section.

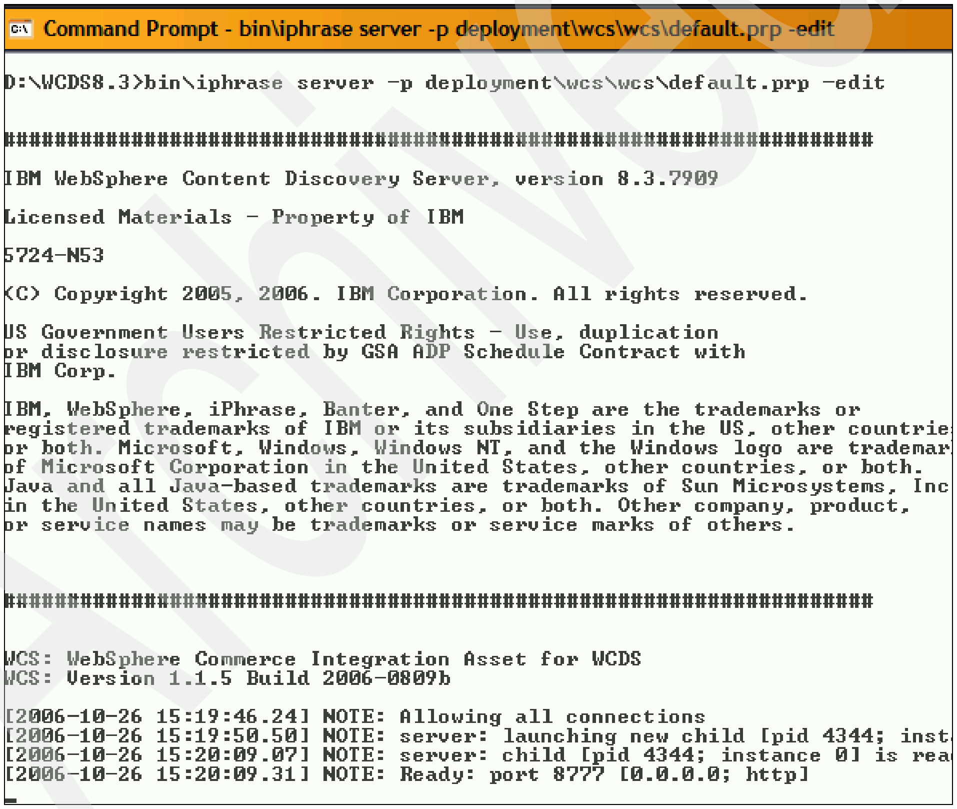
7.3.1 Starting the Management Console

To start the Management Console, make sure no OmniFind Discovery Edition service is running. Then follow these steps:

1. Start the OmniFind Discovery Edition server in edit mode. To do that, open a command prompt window and **cd** to the <ODE_HOME> directory. Then issue the command:

   ```
   bin\iphrase server -p deployment\wcs\wcs\default.prp -edit
   ```

   Wait until the server is ready. See Figure 7-6.



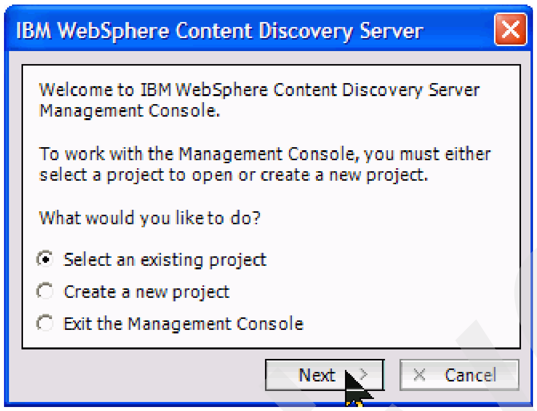
Command Prompt - bin\iphrase server -p deployment\wcs\wcs\default.prp -edit

```
D:\WCDS8.3>bin\iphrase server -p deployment\wcs\wcs\default.prp -edit

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

IBM WebSphere Content Discovery Server, version 8.3.7909

Licensed Materials - Property of IBM

5724-N53

(C) Copyright 2005, 2006. IBM Corporation. All rights reserved.

US Government Users Restricted Rights - Use, duplication
or disclosure restricted by GSA ADP Schedule Contract with
IBM Corp.

IBM, WebSphere, iPhrase, Banter, and One Step are the trademarks or
registered trademarks of IBM or its subsidiaries in the US, other countrie
or both. Microsoft, Windows, Windows NT, and the Windows logo are trademar
of Microsoft Corporation in the United States, other countries, or both.
Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc
in the United States, other countries, or both. Other company, product,
or service names may be trademarks or service marks of others.


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

WCS: WebSphere Commerce Integration Asset for WCDS
WCS: Version 1.1.5 Build 2006-0809b

[2006-10-26 15:19:46.24] NOTE: Allowing all connections
[2006-10-26 15:19:50.50] NOTE: server: launching new child [pid 4344; inst
[2006-10-26 15:20:09.07] NOTE: server: child [pid 4344; instance 0] is rea
[2006-10-26 15:20:09.31] NOTE: Ready: port 8777 [0.0.0.0; http]
_
```

Figure 7-6 Starting OmniFind Discovery Edition server in edit mode

2. From Windows, select **Start** → **All Programs** → **IBM WebSphere Content Discovery Server 8.3** → **Management Console**

3. From the Management Console Logon window, type the password to log on to the Management Console (default password is admin). Make sure port 8777 is used for the Management Console.
4. Choose **Select an existing project** and click **Next**. See Figure 7-7.


IBM WebSphere Content Discovery Server

Welcome to IBM WebSphere Content Discovery Server Management Console.

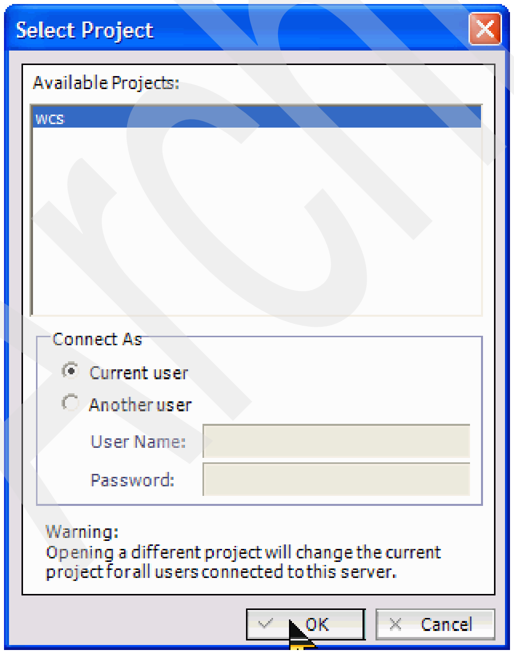
To work with the Management Console, you must either select a project to open or create a new project.

What would you like to do?

[✓ Select an existing project]
Create a new project
Exit the Management Console

Figure 7-7 Management Console: Choose Select an existing project at prompt

5. Select **wcs** under Available Projects and click **OK**. See Figure 7-8.


Select Project

Available Projects:

wcs

Connect As

[✓ Current user]
Another user

User Name:
Password:

Warning:
Opening a different project will change the current project for all users connected to this server.

Figure 7-8 Management Console: Select wcs project

7.3.2 Create a new feature using Management Console

After the wcs project is loaded, follow these steps to create a new feature:

1. Go to **Administration** → **Project**. Expand the **wcs** project. Right click **sitemap id** from the right panel and select **New Feature...** See Figure 7-9.

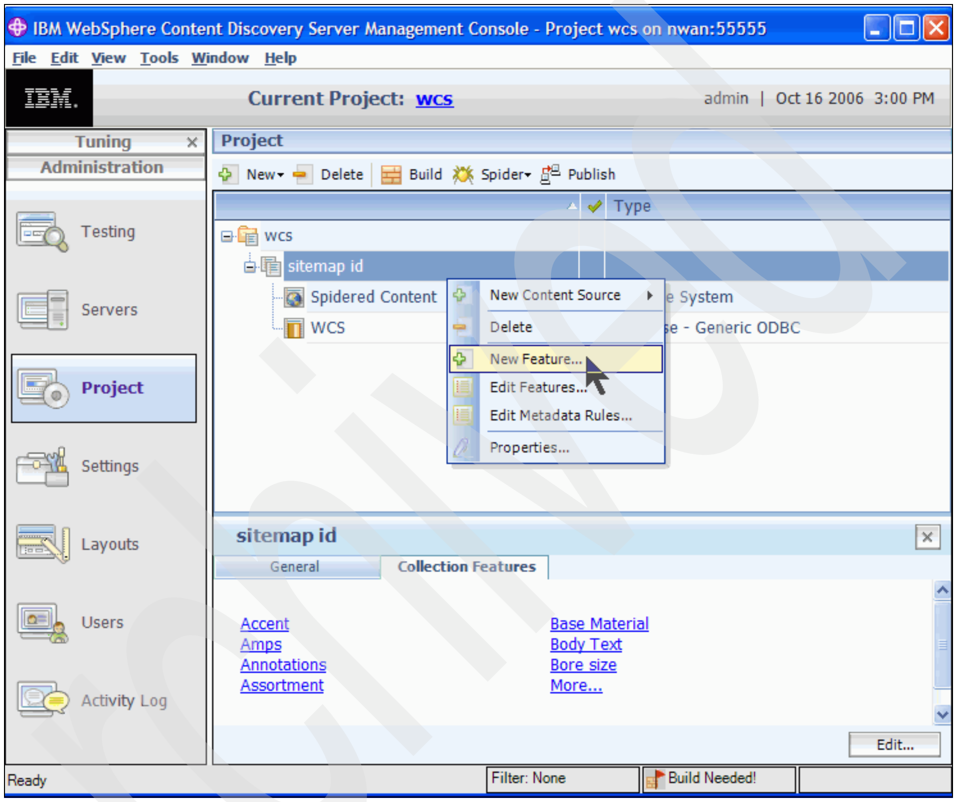
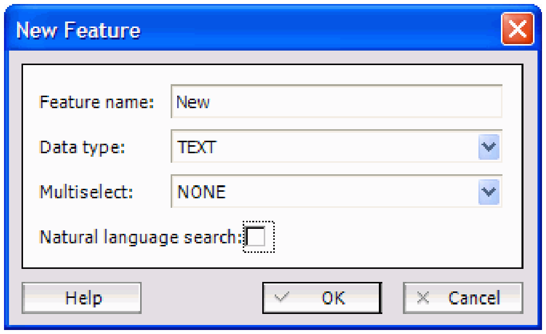


Figure 7-9 Management Console: Create a new feature for the wcs project

2. We name this new feature **New**. For data type, select **TEXT**, because we set the New feature to have a value of yes if an item is new. Also, disable **Natural language search** because we do not require the natural language processor to process the search for new items. See Figure 7-10.


Feature name: New

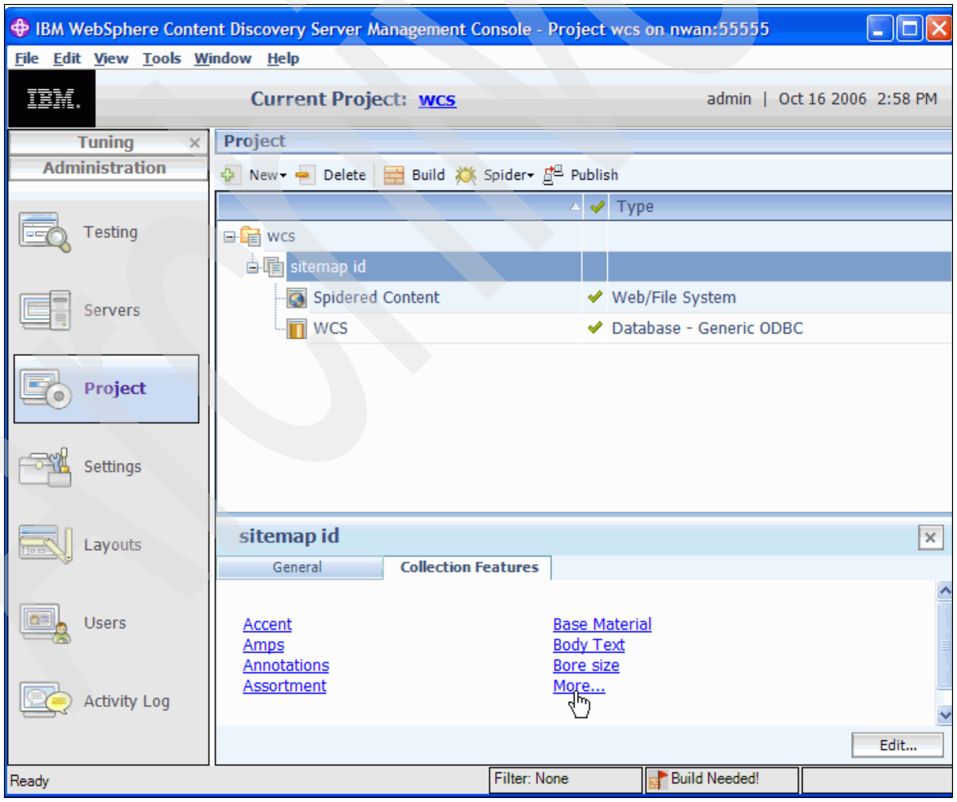
Data type: TEXT

Multiselect: NONE

Natural language search: ☐

Figure 7-10 Management Console: New feature window

3. After the feature is created, from **Administration** → **Project**, expand the **wcs** project and click **sitemap id** from the right panel. On the lower panel of the window, select the **Collection Features** tab and click **More...** to open the edit feature window. See Figure 7-11.


IBM WebSphere Content Discovery Server Management Console - Project wcs on nwan:55555

Current Project: wcs

admin | Oct 16 2006 2:58 PM

	Type
wcs	
sitemap id	
Spidered Content	✓ Web/File System
WCS	✓ Database - Generic ODBC

sitemap id

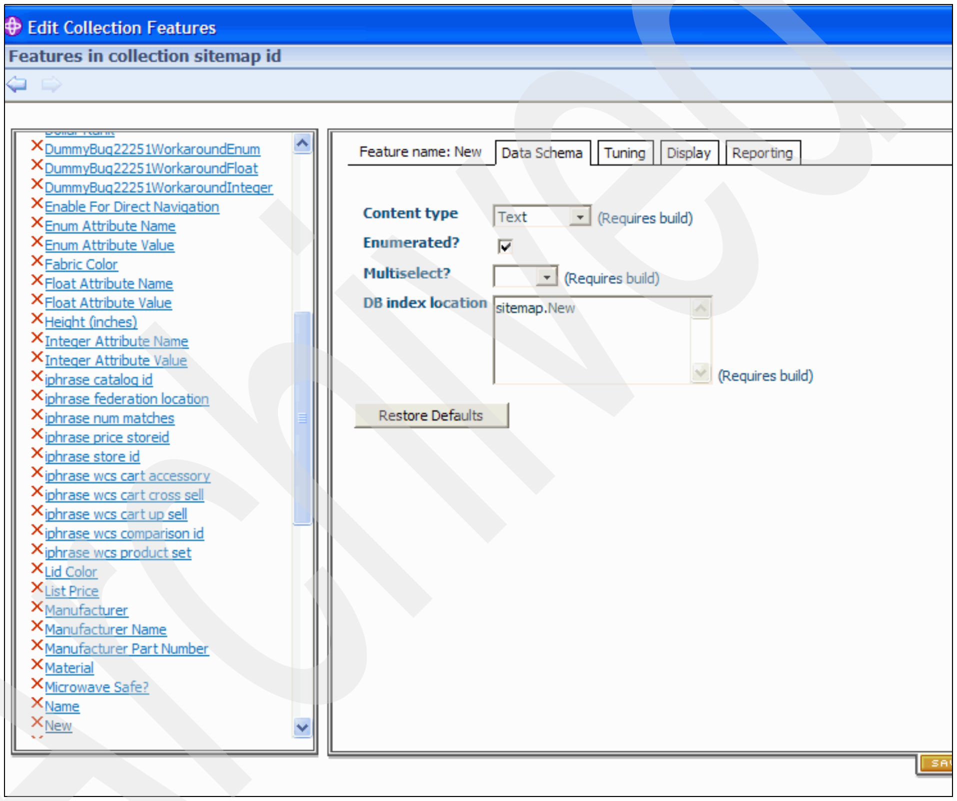
Accent, Amps, Annotations, Assortment, Base Material, Body Text, Bore size, More...

Ready | Filter: None | Build Needed!

Figure 7-11 Management Console: Edit an existing feature

4. Select **New** from the list of features on the left panel. Click the **Data Schema** tab from the right panel to modify the data schema settings for the New feature:
   - Set Content type to **Text**.
   - Make sure that Enumerated? is checked, because the values of this feature are known in advance.
   - Leave the other settings as they are.

   Your Data Schema settings should look as shown inFigure 7-12.



Edit Collection Features

Features in collection sitemap id

- DummyBug22251WorkaroundEnum
- DummyBug22251WorkaroundFloat
- DummyBug22251WorkaroundInteger
- Enable For Direct Navigation
- Enum Attribute Name
- Enum Attribute Value
- Fabric Color
- Float Attribute Name
- Float Attribute Value
- Height (inches)
- Integer Attribute Name
- Integer Attribute Value
- iphrase catalog id
- iphrase federation location
- iphrase num matches
- iphrase price storeid
- iphrase store id
- iphrase wcs cart accessory
- iphrase wcs cart cross sell
- iphrase wcs cart up sell
- iphrase wcs comparison id
- iphrase wcs product set
- Lid Color
- List Price
- Manufacturer
- Manufacturer Name
- Manufacturer Part Number
- Material
- Microwave Safe?
- Name
- New

Feature name: New | Data Schema | Tuning | Display | Reporting

Content type: Text (Requires build)
Enumerated? ✓
Multiselect? (Requires build)
DB index location: sitemap.New (Requires build)

Restore Defaults

Figure 7-12 Management Console: New feature, Data Schema settings

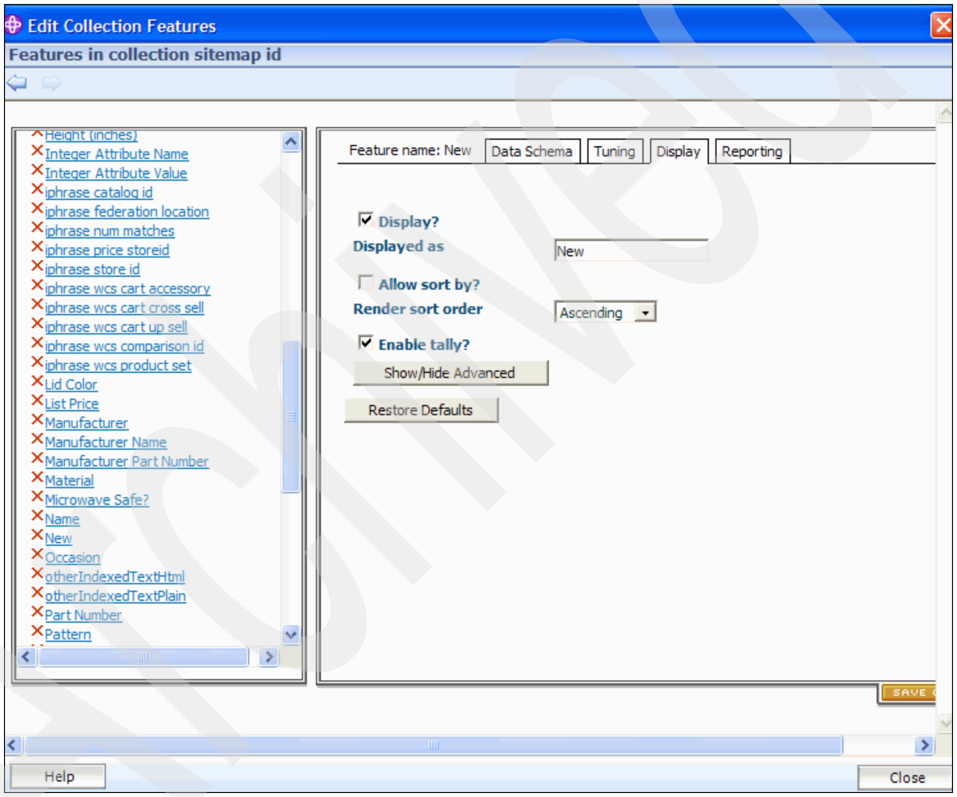
5. Click the **Display** tab.

   – Make sure that the Display? check box is checked. Set Displayed as to **New**.

   – We do not have to sort the search result for the new feature, because the values are yes for all new items. Make sure that the Allow sort by? check box is unchecked.

   – Make sure that the Enable tally? check box is checked so we can see a counter at the left navigation.

   Your Display settings should look as shown in Figure 7-13.


Edit Collection Features

Features in collection sitemap id

Height (inches)
Integer Attribute Name
Integer Attribute Value
iphrase catalog id
iphrase federation location
iphrase num matches
iphrase price storeid
iphrase store id
iphrase wcs cart accessory
iphrase wcs cart cross sell
iphrase wcs cart up sell
iphrase wcs comparison id
iphrase wcs product set
Lid Color
List Price
Manufacturer
Manufacturer Name
Manufacturer Part Number
Material
Microwave Safe?
Name
New
Occasion
otherIndexedTextHtml
otherIndexedTextPlain
Part Number
Pattern

Feature name: New | Data Schema | Tuning | Display | Reporting

☑ Display?
Displayed as: New
☐ Allow sort by?
Render sort order: Ascending
☑ Enable tally?

Figure 7-13 Management Console: New feature, Display settings

6. Click **Save Changes** to save. Then click **Close** to close the Edit Collection Features window.

7.4 Rebuilding project in OmniFind Discovery Edition

After we create a feature in OmniFind Discovery Edition, we have to rebuild the project to generate the required files for this new feature:

1. Notice that the Management Console indicates that a build is required for OmniFind Discovery Edition to pick up the new feature. See Figure 7-14. Double click the message, **Build Needed**, to open up the Build Options window.

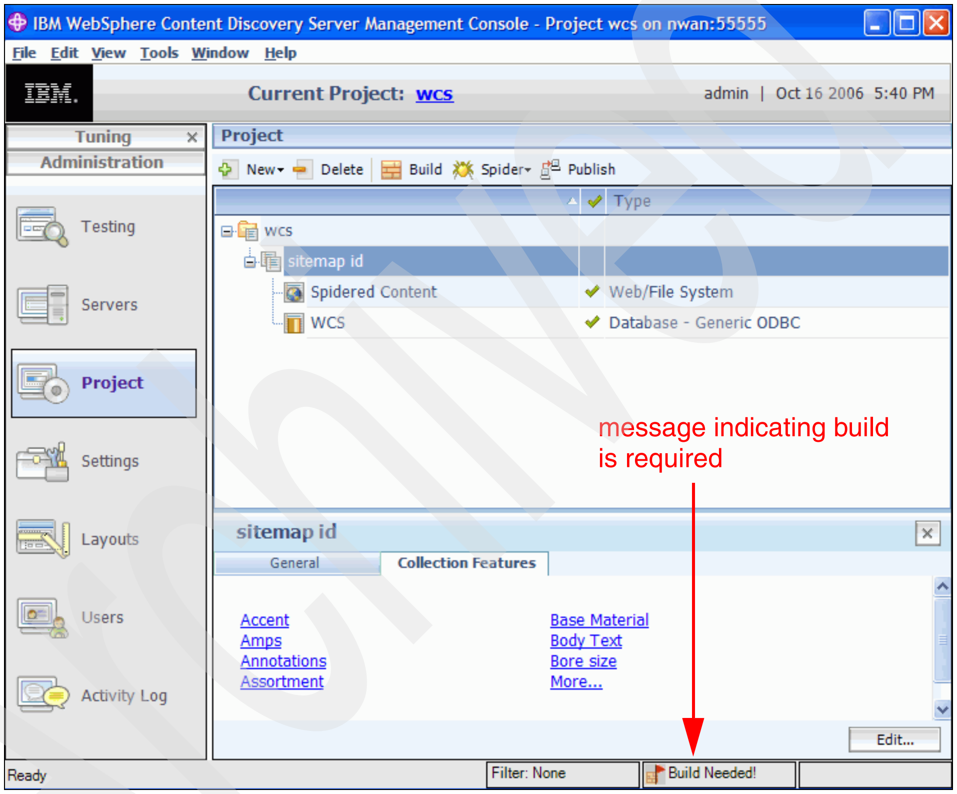
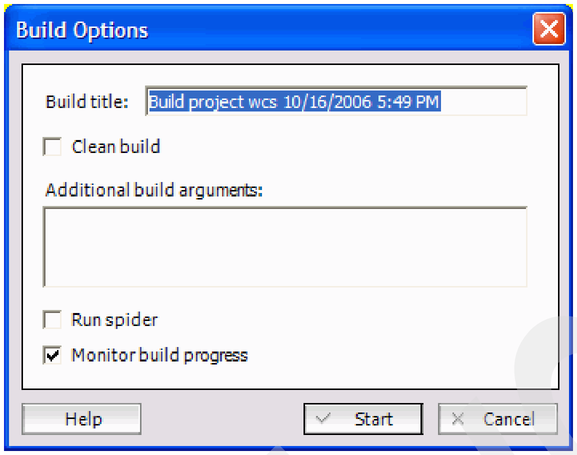


Figure 7-14 Management Console: Message indicating build is required

2. Make sure **Monitor build progress** is selected and click **Start**. See Figure 7-15.



Build title: Build project wcs 10/16/2006 5:49 PM

[ ] Clean build

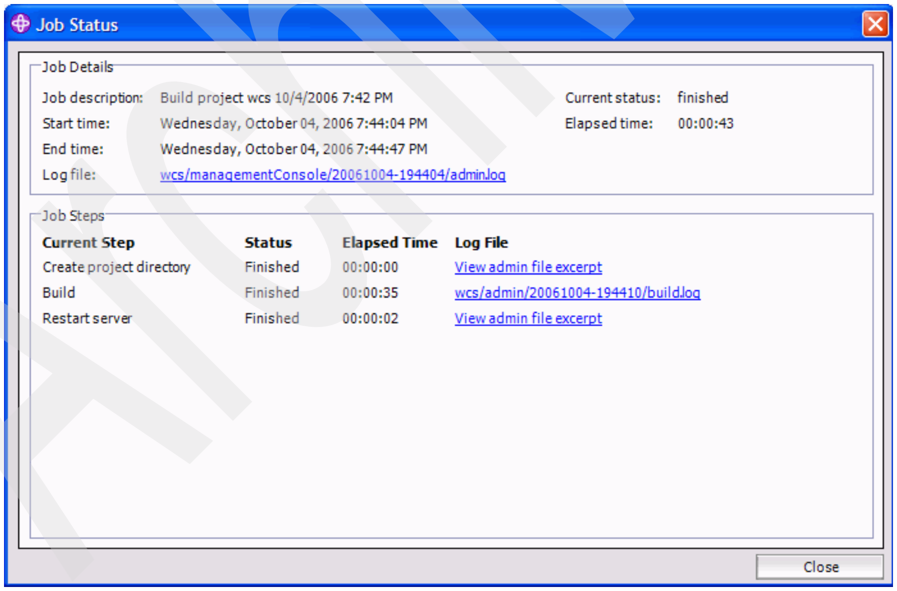
Additional build arguments:

[ ] Run spider

[✓] Monitor build progress

Figure 7-15 Management Console: Start a build

3. Monitor the Job Status window. Close the window after all the create project directory, build, and restart server jobs are finished. See Figure 7-16.



Job description:	Build project wcs 10/4/2006 7:42 PM	Current status:	finished
Start time:	Wednesday, October 04, 2006 7:44:04 PM	Elapsed time:	00:00:43
End time:	Wednesday, October 04, 2006 7:44:47 PM		
Log file:	wcs/managementConsole/20061004-194404/admin.log		

Current Step	Status	Elapsed Time	Log File
Create project directory	Finished	00:00:00	View admin file excerpt
Build	Finished	00:00:35	wcs/admin/20061004-194410/build.log
Restart server	Finished	00:00:02	View admin file excerpt

Figure 7-16 Management Console: Job Status window displaying build status

7.5 Creating metadata rule to map WCS data into feature

We create a new metadata rule in OmniFind Discovery Edition to map the data we defined in 7.2, “Customizing data structure in WebSphere Commerce” on page 129 to the feature we created in 7.3, “Creating a corresponding feature” on page 135.

To create a new metadata rule in OmniFind Discovery Edition:

1. From the Management Console, make sure you are currently working on the wcs project. You can find this information in the top panel of the Management Console window. See Figure 7-17. If your current project is not wcs, go to **File** → **Open Project...** and select the **wcs** project.

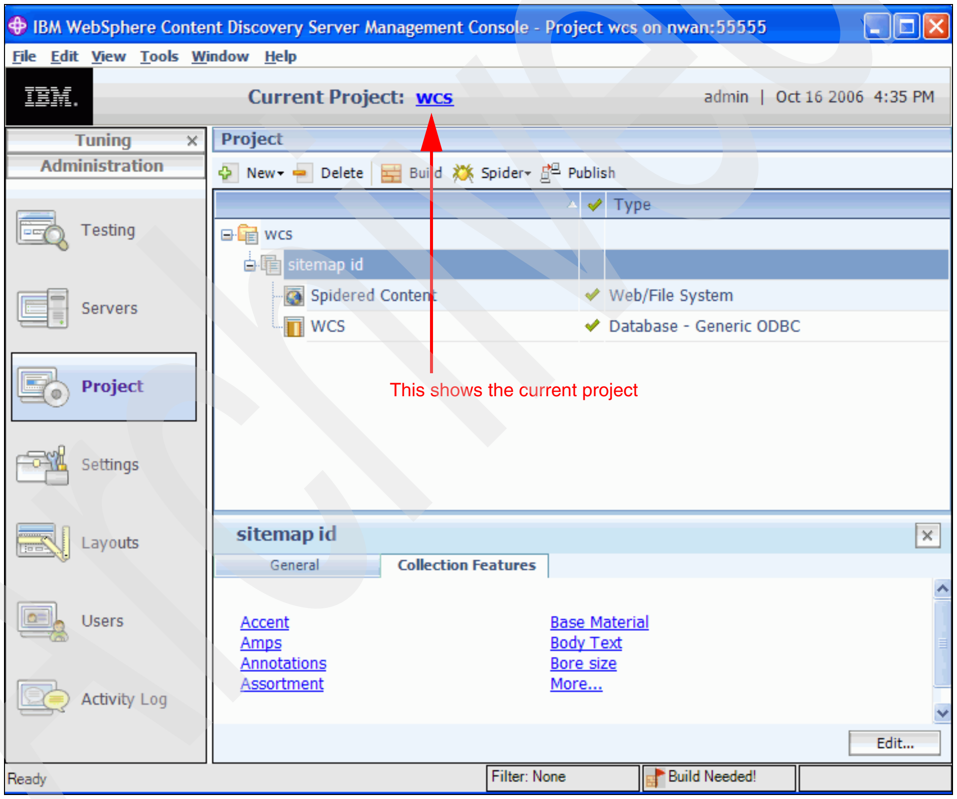


Figure 7-17 Management Console: Check your current project

2. From the Management Console, select **Administration** → **Project**. Expand the **wcs** project. Right click **sitemap id** and select **Edit Metadata Rules...** See Figure 7-18.

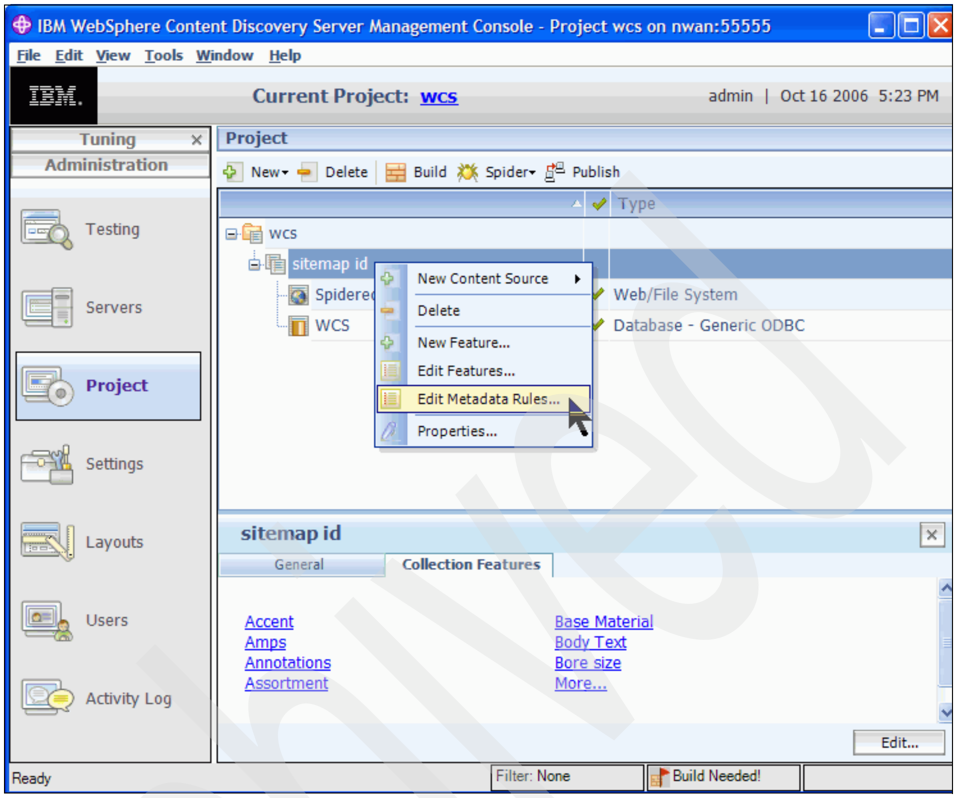
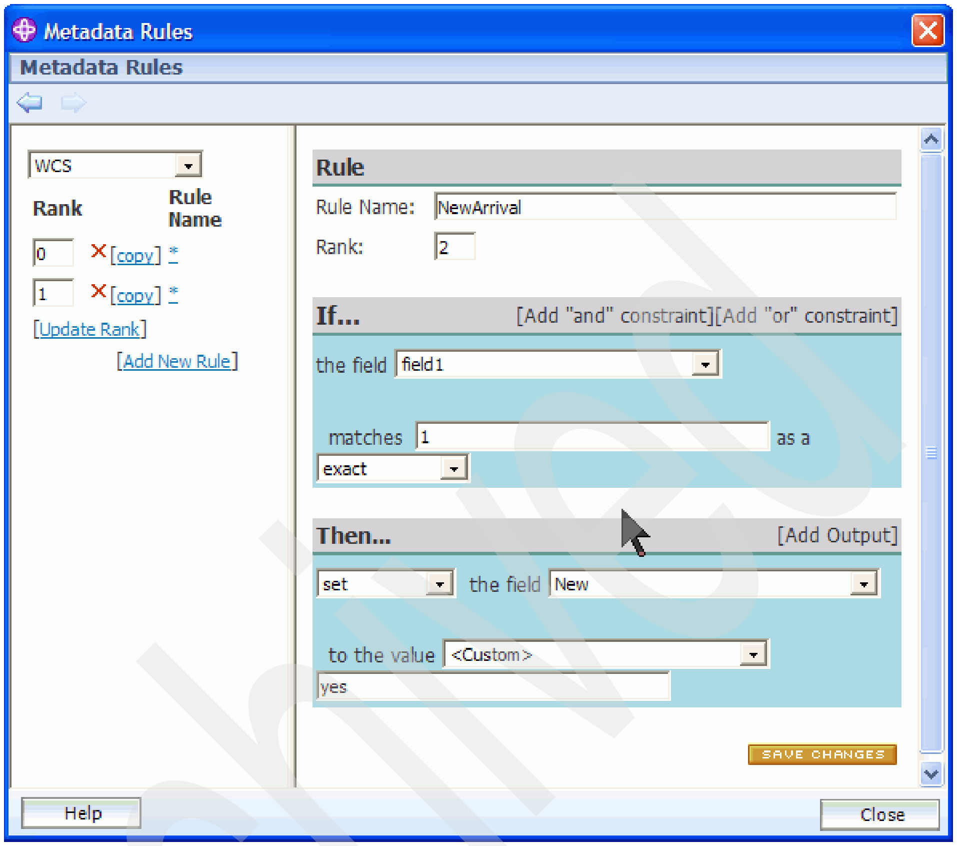


Figure 7-18 Management Console: Edit Metadata Rules of the wcs project

3. From the Metadata Rules window, under the drop-down list on the top left corner, select **wcs**. Click **Add New Rule**.
4. Define the metadata rule. We map the data in the way that if field1 from WebSphere Commerce has a value of 1, then the New feature setting of the item in OmniFind Discovery Edition has a text value of yes. Follow the settings as displayed on Figure 7-19.


Metadata Rules

Rule Name: NewArrival

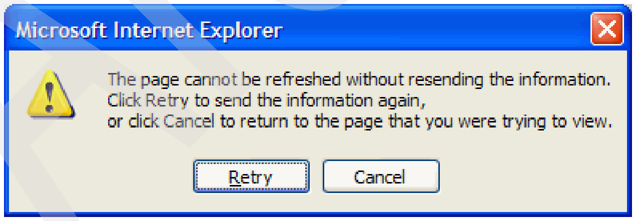
Rank: 2

If... the field field1 matches 1 as a exact

Then... set the field New to the value <Custom> yes

Figure 7-19 Management Console: Setting metadata rule

5. Click **Save Changes** and Close the Metadata Rules window. You might get an error from Microsoft® Internet Explorer that looks similar to Figure 7-20.


Microsoft Internet Explorer

The page cannot be refreshed without resending the information.
Click Retry to send the information again,
or click Cancel to return to the page that you were trying to view.

Figure 7-20 Warning from Microsoft Internet Explorer

Simply click **Retry** and **Close** the Metadata Rules window.

6. After defining a new metadata rule, we have to rebuild and regenerate indexes for the wcs project. Refer to section 7.4, “Rebuilding project in OmniFind Discovery Edition” on page 142 for instructions on how to rebuild the wcs project.

7.6 Testing the intermediate result

After setting up the metadata rule and regenerating the indexes for OmniFind Discovery Edition, OmniFind Discovery Edition should have all the information required to search for new arrival items. We use the Management Console Testing tool to test our work.

7.6.1 Displaying single value results

The New feature has only one value — yes, if the item is a new arrival item. By default, OmniFind Discovery Edition does not display options with only one result in the refine-by list. For the New feature, we would like to see it under the refine-by list on the left navigation panel.

To change the default settings in OmniFind Discovery Edition, open the file, <ODE_HOME>\deployment\wcs\wcs\feature.txt, which contains the features definition. Locate the New feature and add a line to set the value of render_refine_show_single_value to yes.

The feature definition of the New feature should look similar to the bold text in Example 7-2.

Example 7-2 New feature definition in feature.txt file

```
New
  type: text
  version: 1.9
  modified: 2006-10-16 16:19:04.18
  ir_identifier: "sitemap id"
  db: sitemap.New
  build_ir: no
  render_label: New
  enum: yes
  build_lexicon: no
  tally: yes
  resolve_traits: height number quality size
  render_refine_show_single_value: yes
  build_grammar: no
  enable_feature_for_rendering: yes
  enable_feature_for_sorting: no
```

After adding the line to the file, save and close it.

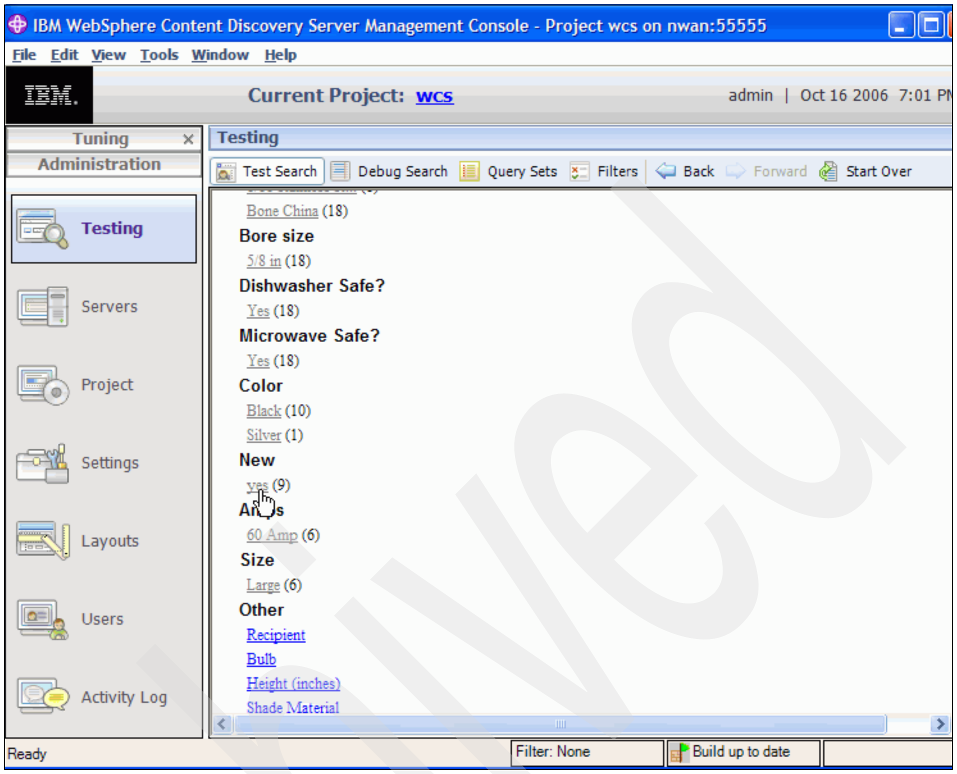
Tip: For the list of feature attribute, refer to Chapter 6 “Feature Attributes” of the OmniFind Discovery Edition Reference Guide.

7.6.2 Viewing the results

The Testing tool of the Management Console lets you run your current configuration against your environment, to ensure that you are getting the search results you want.

To access the Testing tool:

1. On the Tuning or the Administration shortcut menu, click **Testing**.
2. On the toolbar located at the top of the main panel, make sure **Test Search** is selected. This button is selected by default.
3. Find the New feature at the left navigation display of the search panel. You should see it under the refine list with a label **New**. See Figure 7-21.

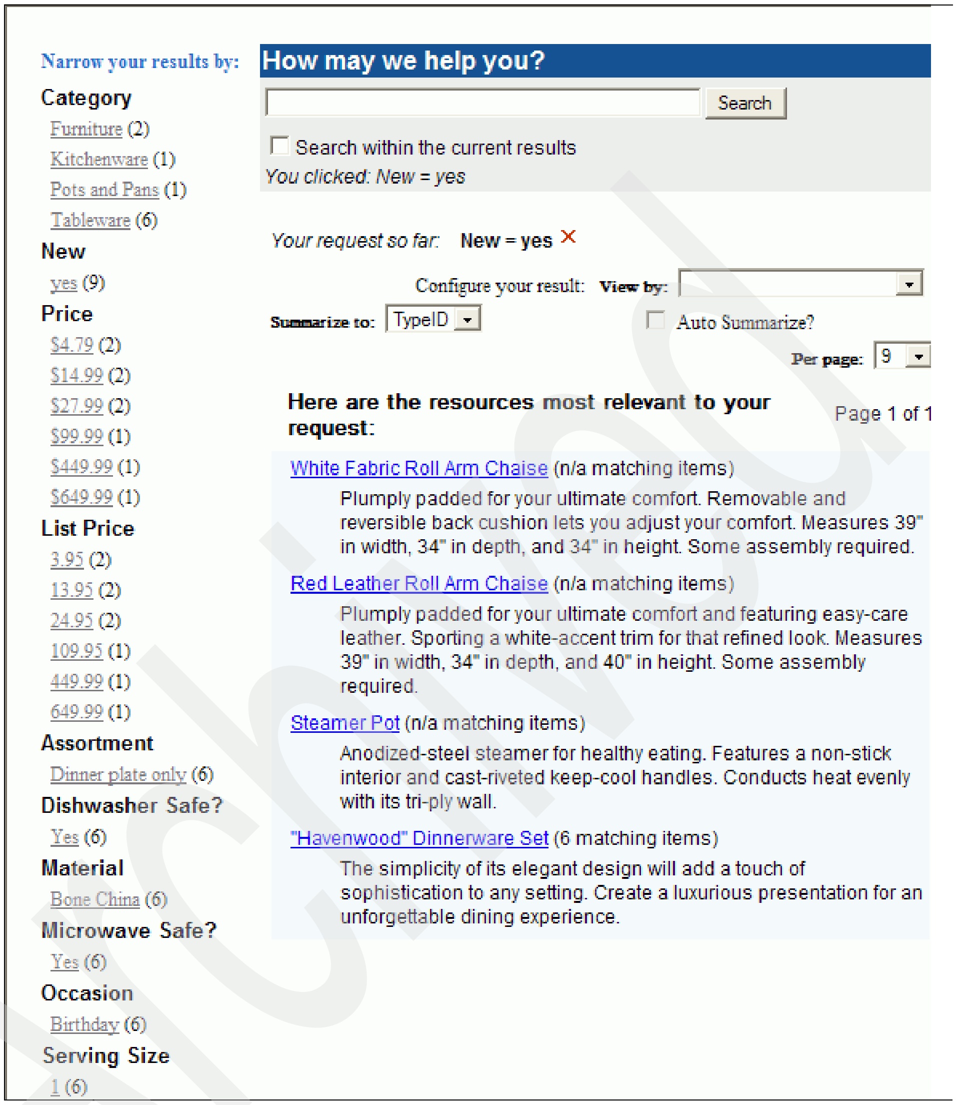


Testing

Bone China (18)
Bore size
5/8 in (18)
Dishwasher Safe?
Yes (18)
Microwave Safe?
Yes (18)
Color
Black (10)
Silver (1)
New
yes (9)
Amps
60 Amp (6)
Size
Large (6)
Other
Recipient
Bulb
Height (inches)
Shade Material

Figure 7-21 Management Console: Testing tool showing the New feature

4. Click **yes** under the New feature.
5. You should see four products from the search results. Refer to Figure 7-22.


Narrow your results by:

Category

Furniture (2)
Kitchenware (1)
Pots and Pans (1)
Tableware (6)

New

yes (9)

Price

$4.79 (2)
$14.99 (2)
$27.99 (2)
$99.99 (1)
$449.99 (1)
$649.99 (1)

List Price

3.95 (2)
13.95 (2)
24.95 (2)
109.95 (1)
449.99 (1)
649.99 (1)

Assortment

Dinner plate only (6)

Dishwasher Safe?

Yes (6)

Material

Bone China (6)

Microwave Safe?

Yes (6)

Occasion

Birthday (6)

Serving Size

1 (6)

How may we help you?

Search

Search within the current results

You clicked: New = yes

Your request so far: New = yes ✕

Configure your result: View by:

Summarize to: TypeID

Auto Summarize?

Per page: 9

Here are the resources most relevant to your request:

Page 1 of 1

White Fabric Roll Arm Chaise (n/a matching items)

Plumply padded for your ultimate comfort. Removable and reversible back cushion lets you adjust your comfort. Measures 39" in width, 34" in depth, and 34" in height. Some assembly required.

Red Leather Roll Arm Chaise (n/a matching items)

Plumply padded for your ultimate comfort and featuring easy-care leather. Sporting a white-accent trim for that refined look. Measures 39" in width, 34" in depth, and 40" in height. Some assembly required.

Steamer Pot (n/a matching items)

Anodized-steel steamer for healthy eating. Features a non-stick interior and cast-riveted keep-cool handles. Conducts heat evenly with its tri-ply wall.

"Havenwood" Dinnerware Set (6 matching items)

The simplicity of its elegant design will add a touch of sophistication to any setting. Create a luxurious presentation for an unforgettable dining experience.

Figure 7-22 Management Console: Search results from the OmniFind Discovery Edition

Note: The tally count displays 9 matches and we see 4 products in the search results page. This is because the search is performed at item level but the display is at the product level. Refer back to Table 7-2 on page 130 for the data we defined. We have 4 different products and 9 different items in our new arrival data.

7.7 Presenting the results in your Web site

After verifying that the data is set up and indexed correctly in OmniFind Discovery Edition, we can customize our WebSphere Commerce JSP to display the new arrival feature.

We add a link to the top navigation drop-down menu. The link invokes an OmniFind Discovery Edition search, and then all matching products are displayed in the search results page.

The top navigation drop-down menu is implemented in the file, CachedHeaderDisplay.jsp. For this book, we have created a file getHeaderFeatures.jsp in 6.4.1, "Implement getHeaderFeatures JSP" on page 102. Change it to a reference when merging the file to implement interactions with OmniFind Discovery Edition code.

We modify both CachedHeaderDisplay.jsp and getHeaderFeature.jsp to implement our runtime changes for the new arrival feature.

7.7.1 Customizing CachedHeaderDisplay.jsp

We have to add a link to the top navigation drop-down menu. To do this:

1. Open CachedHeaderDisplay under the directory, <WCS_HOME>\Stores\WebContent\ConsumerDirect\include\styles\style1.
2. Locate where the drop-down menu is defined. To do this, search for <div class="dropDownMenu"> in the file.
3. Add the code in Example 7-3 under the dropDownMenu list.

Example 7-3 Adding new arrival link to the top navigation drop-down menu

```
<%-- CUSTOMIZATION - New Arrival feature --%>
<li>
   <a HREF="javascript:updaterNew()" class="menuitem" >New Arrival
   </a>
</li>
```

The code calls a JavaScript function updaterNew() when the New Arrival link is clicked. We implement the JavaScript function in the next section.

7.7.2 Customizing getHeaderFeature.jsp

We implement the updaterNew() function in getHeaderFeature.jsp. The updaterNew() function creates a new constraint using OmniFind Discovery Edition API, and set New = yes to the constraint to submit to OmniFind Discovery Edition.

Note: We build our code on top of the getHeaderFeature.jsp created in 6.4.1, "Implement getHeaderFeatures JSP" on page 102. Some of the code from that chapter is reused in this chapter. Make sure you complete that section before doing this chapter.

1. Create a new constraint using the OmniFind Discovery Edition API. Add the code shown in Example 7-4 to getHeaderFeature.jsp.

Example 7-4 Create a constraint for the New feature

```
<%
Constraint cNew = new Enumerated.Equals("New", "yes");
cNew.clearRequired();
%>
```

2. Implement updaterNew() to pass the Constraint to OmniFind Discovery Edition and encode an URI as shown Example 7-5.

Example 7-5 updaterNew() function

```
function updaterNew() {
   headerUpdater('ip_constrain',
encodeURIComponent('<%=cNew.toString()%>'));
}
```

3. As shown in Example 7-6, updaterNew() calls headerUpdater(), a JavaScript function called by other JavaScript functions. The headerUpdater() function sets the encoded constraint URI to ip_constrain and submits the search.

Example 7-6 headerUpdater() function

```
<%-- function called by other javascript functions --%>
<%-- this function is used in By Brand, New Arrival, and Gift Idea
CUSTOMIZATION--%>
function headerUpdater(key, value) {
   eval("document.CatalogSearchForm." + key + ".value = '" + value +
"';");
   doSubmit();
}
```

4. Save and close the files.

7.8 Testing the results in your Web site

To test the results, navigate to the Consumer-Direct store home page. Click the **New Arrival** link from the top navigation drop-down menu; you should be routed to the search results page and see 4 products in the result set. Compare the search results with the data we defined in Table 7-2 on page 130.

Adding a gift idea page to your site

A gift idea page provides content that your visitors would likely want to buy. You can also use this page to display featured products that are recommended by the store. It can improve the customer navigation experience on your site because you provide your visitors with more reasons to visit and you can potentially make extra sales.

In this chapter, we introduce a gift idea page to allow shoppers to find gifts more easily. There are two new browsing options provided for shoppers:

- By recipient type (Gifts for him, her, boys, girls, and babies)
- By occasion (wedding/anniversary, thank you, and holidays)

In this chapter we discuss how to build the gift idea page on a WebSphere Commerce sample store using OmniFind Discovery Edition. We cover the following topics:

- Designing the gift idea page
- Customizing data structure in WebSphere Commerce
- Generating indexes to capture the data structure
- Configure settings to always display the features
- Testing the intermediate result
- Presenting the results in your Web site
- Testing the results in your store

8.1 Designing the gift idea page

The gift idea page helps shoppers find the gift they want more easily. To demonstrate the use of the gift idea page, we present two additional browsing options to shoppers in our sample application:

- By recipient type (gifts for him, gifts for her, gifts for children)
- By occasion (birthday and Christmas)

Each of the browsing options (by recipient type and by occasion) is added as a new feature to the WebSphere Commerce Discovery Server. If you select a refinement criterion under a browsing option; the criterion is passed as a constraint into OmniFind Discovery Edition to get a list of products that matches the search criterion.

For example, if you select **Shop by Recipient** → **For her**, OmniFind Discovery Edition gets all the products with the feature *by recipient* set to *for her*.

Figure 8-1 and Figure 8-2 show the top menu options and their drop-down menus designed for the gift idea options, shop by occasion and shop by recipient.

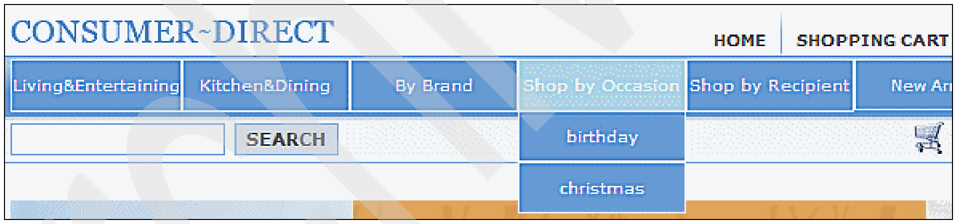


Figure 8-1 Customized store: New top menu, Shop by Occasion

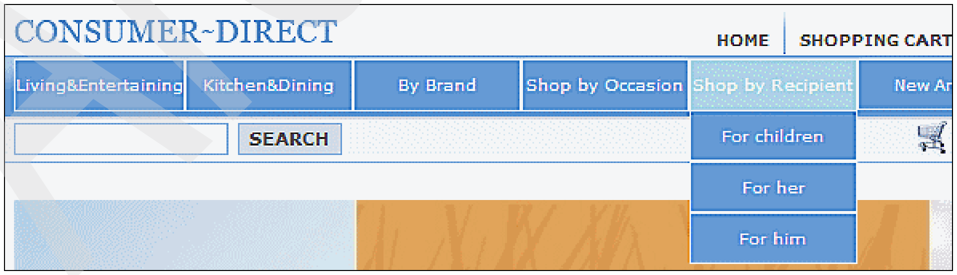


Figure 8-2 Customized store: New top menu, Shop by Recipient

8.2 Customizing data structure in WebSphere Commerce

In this section, we cover customizing the data structure in WebSphere Commerce.

8.2.1 Data design

The data structure must be flexible enough to allow a site administrator to add additional browsing options later without additional coding. To accomplish this, you define features as name-value pairs, and treat them as attributes of products in WebSphere Commerce.

Table 8-1 and Table 8-2 define the data for our sample application.

Table 8-1 Name-value pair for by recipient browsing option

Name	Value
Recipient	him
Recipient	her
Recipient	children

Table 8-2 Name-value pair for by occasion browsing option

Name	Value
Occasion	birthday
Occasion	christmas

Later on, if you want to add a new option to an existing browsing option, you can simply define a new value for it. For example, if you add wedding to the browsing option, by occasion, you add a new name-value pair (Occasion, wedding) to the data.

To add a new browsing option, define a new name for the browsing option and add values to it.

WebSphere Commerce already has an attribute data structure defined for products. In our example, we take advantage of the ATTRIBUTE table and ATTRVALUE tables from WebSphere Commerce.

For more information about the tables, refer to the online reference at the following URLs:

- Attribute table:

  `http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.database.doc/database/attribute.htm`

- Attrvalue table:

  `http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.database.doc/database/attrvalue.htm`

8.2.2 Data used in our sample application

We define a few products for the search, *by recipient*. Refer to Table 8-3 for the list of *shop by occasion* data.

Table 8-3 Data for by recipient search

Attribute Value	Product Name	Product SKU	Parent Category Tree
her	Arouse-Your-Senses Gift Set	KISC-01	Kitchen&Dining - Kitchenware - Scented Oils
her	Arouse-Your-Senses Deluxe Gift Set	KISC-02	Kitchen&Dining - Kitchenware - Scented Oils
him	Leather High-Back Office Chair	FUOF-03	Living&Entertaining - Office Chairs
him	Executive Six-Drawer Desk	FUDE-01	Living&Entertaining - Desks
him	Brewmaster Deluxe Coffeemaker	KICOF-01	Kitchen&Dining - Kitchenware - Coffee Makers
children	Student's Desk	FUDE-03	Living&Entertaining - Desks
children	Craft Table	FUDE-02	Living&Entertaining - Desks
children	Adjustable Desk Lamp	FUDEL-04	Living&Entertaining - Desk Lamps

Similarly, we have to define data for shop *by occasion*. Refer to Table 8-4 for the list of *shop by occasion* data used in our sample store.

Table 8-4 Data for by occasion search

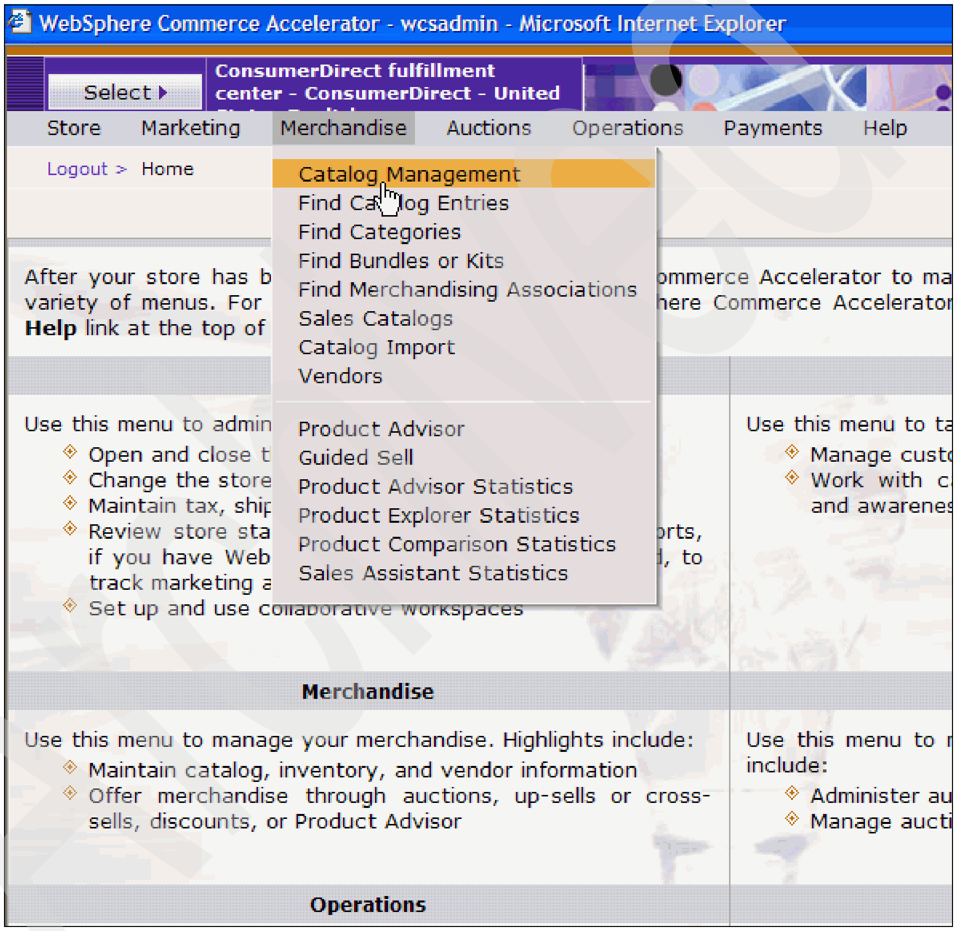
Attribute Value	Product name	Product SKU	Parent Category Tree
birthday	Black Coffeecup	TAWI-01	Kitchen&Dining - Tableware - Tea and Coffee Cups
birthday	"Corrolus" Wineglasses	TAWI-03	Kitchen&Dining - Tableware - Wine Glasses
birthday	Coffee and Espresso Bar	KIES-01	Kitchen&Dining - Kitchenware - Espresso Machines
christmas	"Havenwood" Dinnerware set	TAPL-01	Kitchen&Dining - Tableware - Plates
christmas	Purple Leather Sofa	FULE-02	Living&Entertaining - Leather Sofas
christmas	Classic Blue-Frabic-Roll-Arm Love Seat	FULOV-01	Living&Entertaining - Loveseats
christmas	White Frabic Loveseat	FULOV-02	Living&Entertaining - Loveseats

Using Commerce Accelerator to populate sample data

We can use the Catalog Management tool inside WebSphere Commerce Accelerator to define the data we require.

Follow these steps to set up the data manually using WebSphere Commerce Accelerator:

1. After logging on to the WebSphere Commerce Accelerator, select **Merchandise** → **Catalog Management**. See Figure 8-3.


WebSphere Commerce Accelerator - wcsadmin - Microsoft Internet Explorer
Select
ConsumerDirect fulfillment center - ConsumerDirect - United
Store
Marketing
Merchandise
Auctions
Operations
Payments
Help
Logout > Home
Catalog Management
Find Catalog Entries
Find Categories
Find Bundles or Kits
Find Merchandising Associations
Sales Catalogs
Catalog Import
Vendors
Product Advisor
Guided Sell
Product Advisor Statistics
Product Explorer Statistics
Product Comparison Statistics
Sales Assistant Statistics
Merchandise
Operations

After your store has b ommerce Accelerator to ma
variety of menus. For here Commerce Accelerator
Help link at the top of

Use this menu to admin
- Open and close t
- Change the store
- Maintain tax, ship
- Review store sta orts,
  if you have Web d, to
  track marketing a
- Set up and use collaborative workspaces

Use this menu to ta
- Manage custo
- Work with c
  and awarenes

Merchandise

Use this menu to manage your merchandise. Highlights include:
- Maintain catalog, inventory, and vendor information
- Offer merchandise through auctions, up-sells or cross-sells, discounts, or Product Advisor

Use this menu to r
include:
- Administer au
- Manage aucti

Operations

Figure 8-3 Commerce Accelerator: Launching the Catalog Management tool

2. Follow the parent category tree information provided in Table 8-3 to find the first product (Arouse-Your-Senses Gift Set). Click **List Catalog Entries** to see a list of products under the selected category. See Figure 8-4.

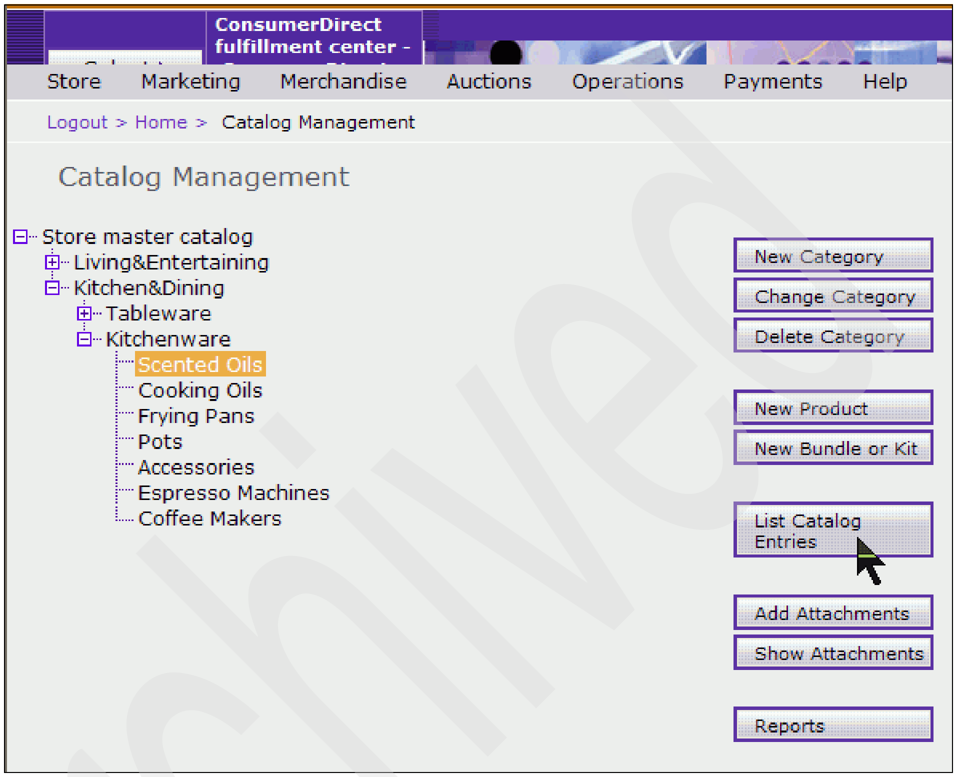


Figure 8-4 Commerce Accelerator: Listing products under selected category

3. Select the product (Arouse-Your-Senses Gift Set for the first item) and select **Tools → Descriptive Attributes...** from the drop-down menu. See Figure 8-5.

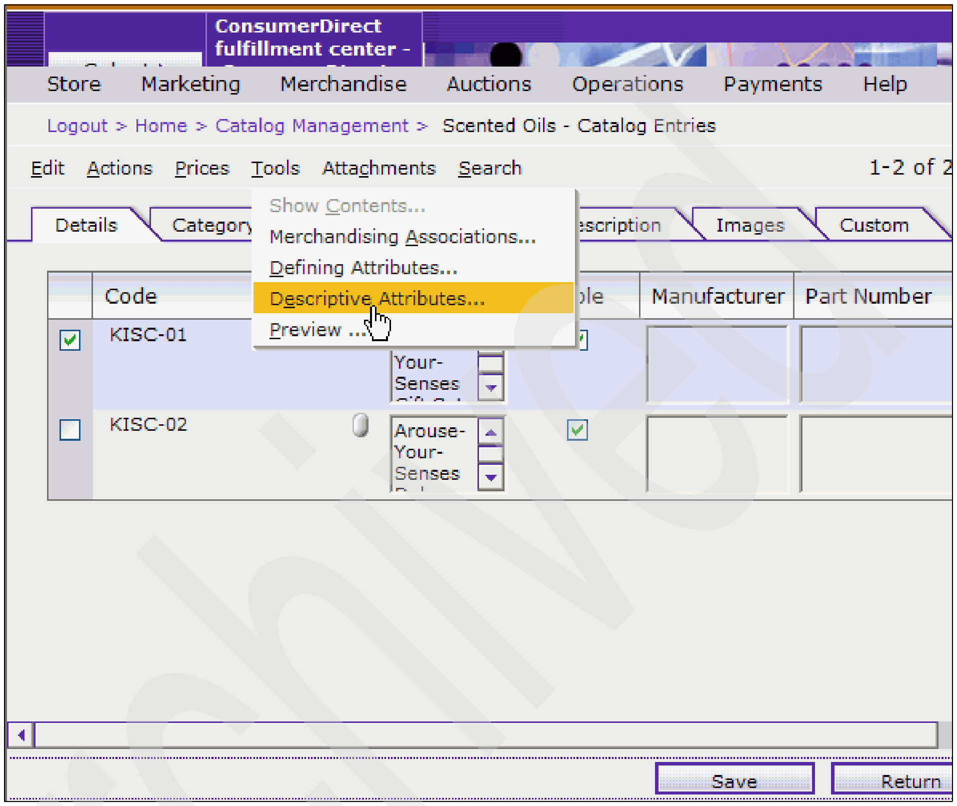
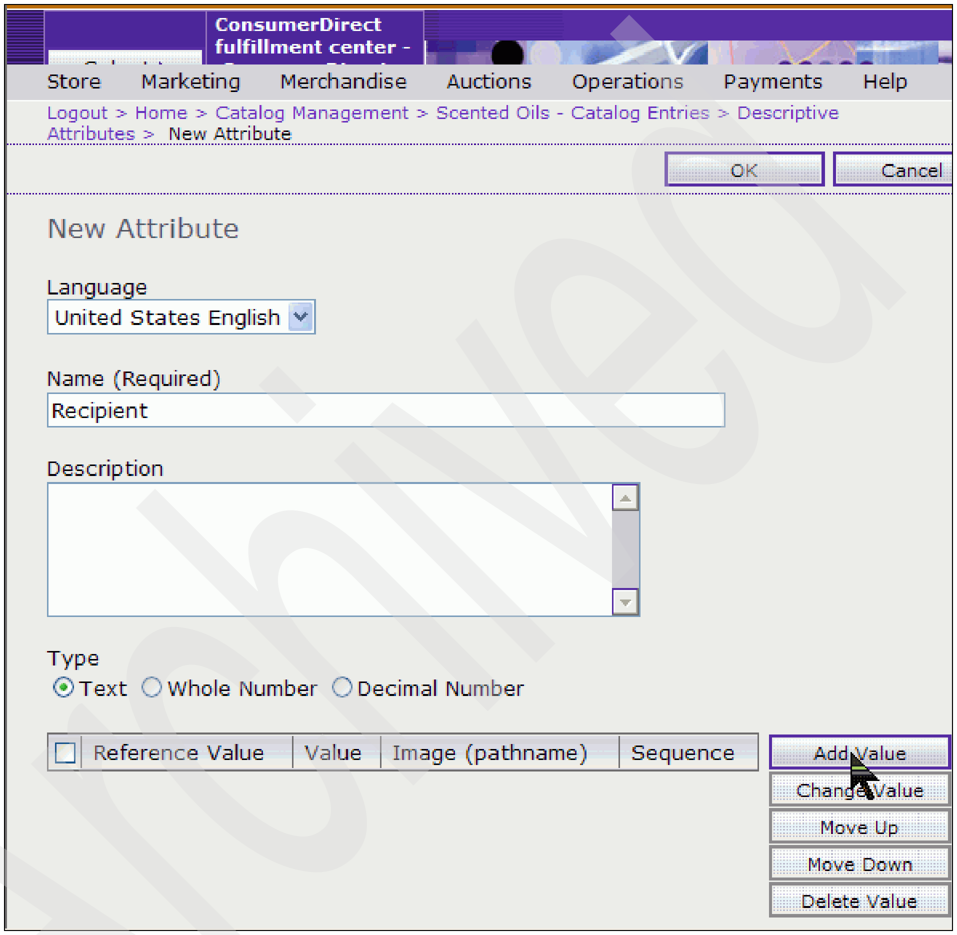


Figure 8-5 Commerce Accelerator: Editing descriptive attributes of a product

4. From the Description Attributes window, click **New**.
5. Under the Name field for the new attribute, enter Recipient. Make sure Type is selected as Text. Click **Add Value** to define a value of this attribute. See Figure 8-6.



ConsumerDirect fulfillment center -
Store
Marketing
Merchandise
Auctions
Operations
Payments
Help
Logout > Home > Catalog Management > Scented Oils - Catalog Entries > Descriptive Attributes > New Attribute
OK
Cancel
Add Value
Change Value
Move Up
Move Down
Delete Value

New Attribute

Language: United States English

Name (Required): Recipient

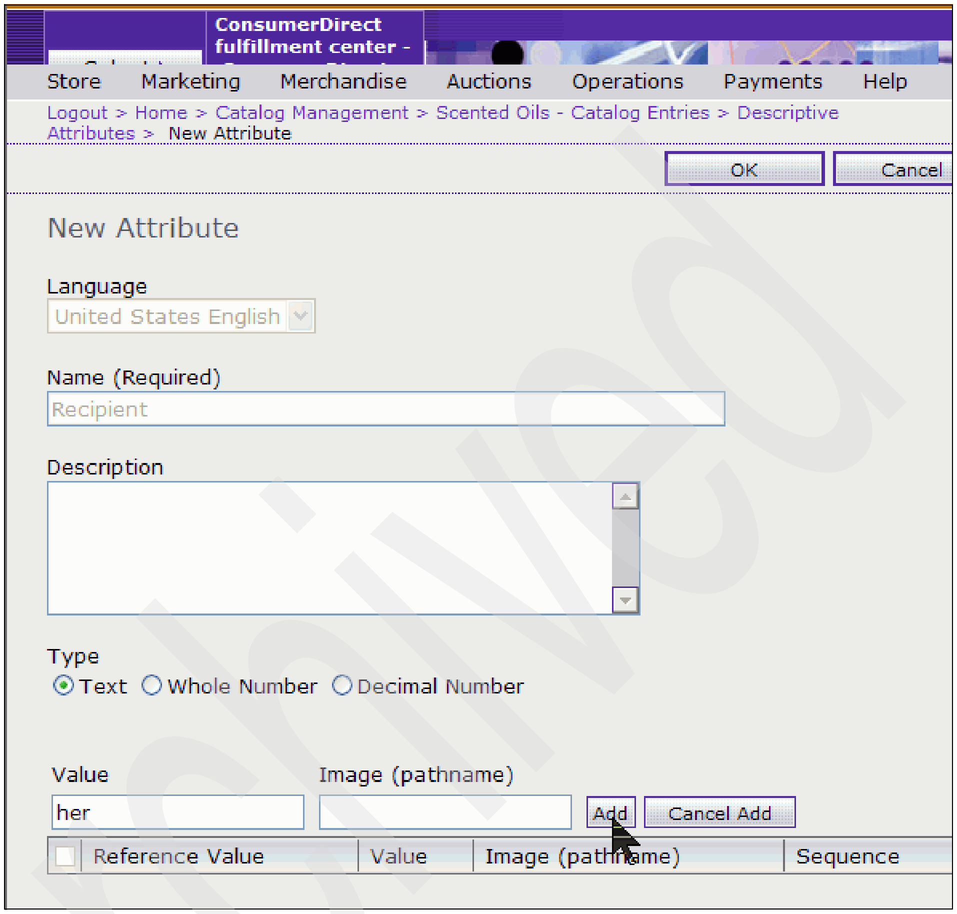
Description:

Type: [✓ Text] Whole Number Decimal Number

☐	Reference Value	Value	Image (pathname)	Sequence

Figure 8-6 Commerce Accelerator: Defining a new attribute

6. Under the Value field, type her. Click **Add** when finished. See Figure 8-7.


ConsumerDirect fulfillment center -
Store
Marketing
Merchandise
Auctions
Operations
Payments
Help
Logout > Home > Catalog Management > Scented Oils - Catalog Entries > Descriptive Attributes > New Attribute
OK
Cancel
New Attribute

Language
United States English

Name (Required)
Recipient

Description

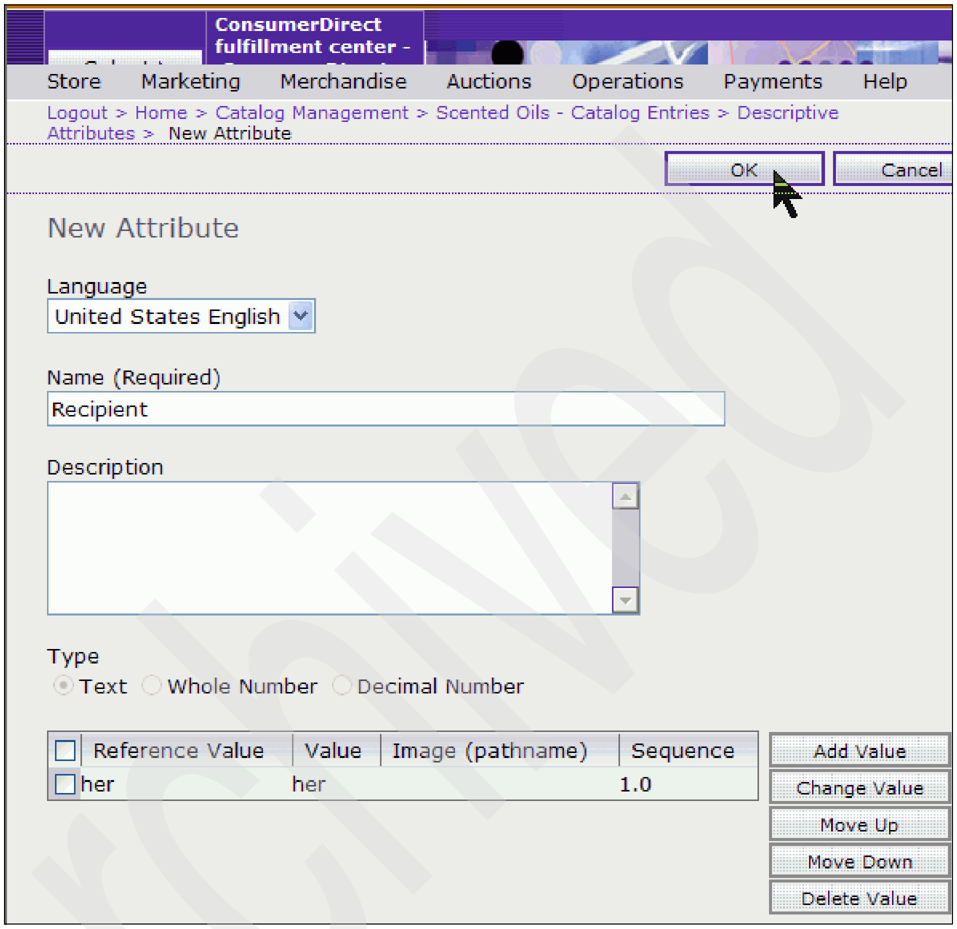
Type
[✓ Text] Whole Number Decimal Number

Value: her
Image (pathname):

	Reference Value	Value	Image (pathname)	Sequence

Figure 8-7 Commerce Accelerator: Adding value to an attribute

7. A value is now defined for the attribute. Click **OK** when finish. See Figure 8-8.


ConsumerDirect fulfillment center -
Store
Marketing
Merchandise
Auctions
Operations
Payments
Help
Logout > Home > Catalog Management > Scented Oils - Catalog Entries > Descriptive Attributes > New Attribute
OK
Cancel
Add Value
Change Value
Move Up
Move Down
Delete Value

New Attribute

Language
United States English

Name (Required)
Recipient

Description

Type
(●) Text ( ) Whole Number ( ) Decimal Number

☐	Reference Value	Value	Image (pathname)	Sequence
☐	her	her		1.0

Figure 8-8 Commerce Accelerator: Completing the add attribute process

8. Repeat steps 2 to 7 for all products defined in Table 8-3 on page 158 and Table 8-4 on page 159.

Tip: For a production environment, we recommend doing a mass update via SQL scripts. See the next section for loading the data using SQL statements.

Updating Commerce database to populate sample data

A copy of the commerce database that contains all sample data used for this book can be obtained from the Redbooks download page. Refer to Appendix A, "Additional material" on page 271 for information.

Alternatively, you can run an SQL script directly to update your database. Get a copy of the script file from the downloaded material, chapter_8_Gift_Idea\GiftIdea_data.sql.

When you get the script file, run the update statements as follows:

1. Open a DB2 Command Window.
2. Go to the directory where the script file, GiftIdea_data.sql, is downloaded.
3. Connect to the database by issuing the statement:

   ```
   db2 connect to <DB name> user <DB schema owner ID> using <password>
   ```

   For our case study, <db name> is "mall", <user id> is "db2admin", and <password> is "password".

   So, we use the following command:

   ```
   db2 connect to mall user db2admin using password
   ```

4. Run the script file by issuing the statement:

   ```
   db2 -tvf GiftIdea_data.sql
   ```

5.) To check if the data is populated, issue the following command:

   ```
   db2 -tvf GiftIdea_listdata.sql
   ```

   You should get a list some records, with Field1 field set to 1.

Example 8-1 shows the update statements used to set up the data for the first product in Table 8-3 on page 158.

Example 8-1 Update data

```
-- *****************
-- Recipient data
-- For her
-- Arouse-Your-Senses Gift Set (KISC-01)
delete from attribute where catentry_id=(select catentry_id from
catentry where partnumber='KISC-01') and name='Recipient';
insert into attribute (attribute_id, language_id, attrtype_id,
optcounter, usage, name, catentry_id) values
((select max(attribute_id)+1 from attribute), -1, 'STRING', 0, '2',
'Recipient', (select catentry_id from catentry where
partnumber='KISC-01'));
```

```
delete from attrvalue where catentry_id=(select catentry_id from
catentry where partnumber='KISC-01') and name='her';
insert into attrvalue (attrvalue_id, language_id, attribute_id,
attrtype_id, optcounter, name, stringvalue, catentry_id) values
((select max(attrvalue_id)+1 from attrvalue), -1, (select
max(attribute_id) from attribute), 'STRING', 0, 'her', 'her', (select
catentry_id from catentry where partnumber='KISC-01'));
```

8.3 Generating indexes to capture the data structure

After populating required data to the WebSphere Commerce database, next you must build the indexes in OmniFind Discovery Edition.

Building a set of indexes involves two major steps:

1. Generate or update the set of OmniFind Discovery Edition features from the attributes defined in WebSphere Commerce.
2. Build the indexes.

8.3.1 Generate or update OmniFind Discovery Edition features

To generate or update the features in OmniFind Discovery Edition, follow these steps:

1. Open a command prompt and switch to the *<ODE_HOME>* directory.
2. Run the following command to auto-discover the WebSphere Commerce attributes and generate the corresponding OmniFind Discovery Edition features from them:

   ```
   bin\iphrase build -p deployment\wcs\wcs\default.prp -setProperty
   “BUILD_MODE generateFeatures”
   ```

Tip: When do you have to generate or regenerate features in OmniFind Discovery Edition?

In general, if you have not added, deleted, or changed any attribute names in WebSphere Commerce, you do not have to run the build with generate features option.

3. Wait until the build finishes. You can keep the command prompt window open for the next section.

8.3.2 Building the indexes

To build the indexes in OmniFind Discovery Edition for searches:

1. Open a command prompt and switch to the <*ODE_HOME*> directory, if you close the command prompt from the previous section.
2. Build the indexes by running:

   ```
   bin\iphrase build -p deployment\wcs\wcs\default.prp
   ```

> **Note:** If you are using Cloudscape as your database, WebSphere Commerce *must not* be running when you run the build.

3. Wait until the build finishes. Close the command prompt window.

8.4 Configure settings to always display the features

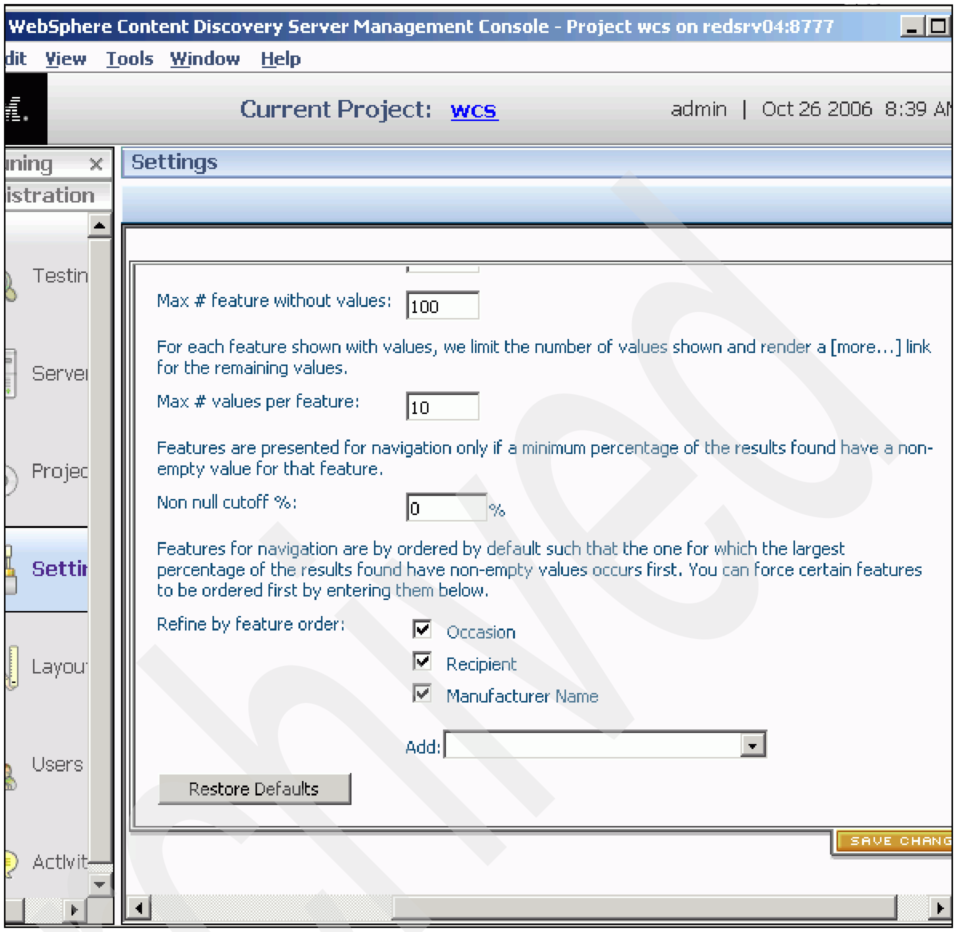
By default, Commerce Module for OmniFind Discovery Edition has many features defined. Therefore, the Recipient and Occasion features might not be shown on the refine-by list.

To force OmniFind Discovery Edition to always show these features, follow these steps:

1. Start Management Console and open the wcs project. Refer to 7.3.1, “Starting the Management Console” on page 136 for the detail steps.
2. After the wcs project is loaded, go to **Administration** → **Settings**.
3. Under **Presentation**, select **Navigation**.
4. Click the **Show Advanced >>** button.
5. Under the section that says:

   ```
   Features for navigation are by ordered by default such that the one
   for which the largest percentage of the results found have non-empty
   values occurs first. You can force certain features to be ordered
   first by entering them below.
   ```

   Add **Recipient** to the *Refine by feature order* field and click **Save Changes**.
6. Repeat step 5 to add **Occasion**, then click **Save Changes.** Your settings should look similar to Figure 8-9.


WebSphere Content Discovery Server Management Console - Project wcs on redsrv04:8777
dit View Tools Window Help
Current Project: wcs
admin | Oct 26 2006 8:39 AI
ning
istration
Testin
Serve
Projec
Settir
Layou
Users
Activit
Settings

Max # feature without values: 100

For each feature shown with values, we limit the number of values shown and render a [more...] link for the remaining values.

Max # values per feature: 10

Features are presented for navigation only if a minimum percentage of the results found have a non-empty value for that feature.

Non null cutoff %: 0 %

Features for navigation are by ordered by default such that the one for which the largest percentage of the results found have non-empty values occurs first. You can force certain features to be ordered first by entering them below.

Refine by feature order:
- [✓] Occasion
- [✓] Recipient
- [✓] Manufacturer Name

Add:

Figure 8-9 Management Console: Settings to always display specific features

8.5 Testing the intermediate result

After setting up the required data and rebuilding the indexes for OmniFind Discovery Edition, OmniFind Discovery Edition should have all the information required to search for shop by recipient and shop by occasion products. We use the Management Console Testing tool to test our work.

You can test the intermediate results using Management Console:

1. Start Management Console and open the wcs project. Refer to 7.3.1, “Starting the Management Console” on page 136 for the detailed steps.
2. After the wcs project is loaded, go to the **Tuning** or the **Administration** shortcut menu, and click **Testing**.
3. On the toolbar located at the top of the main panel, make sure **Test Search** is selected. This button is selected by default.
4. Find the Recipient feature at the left navigation display of the search panel. You should see it under the refine list with the label, **Recipient**. See Figure 8-10.

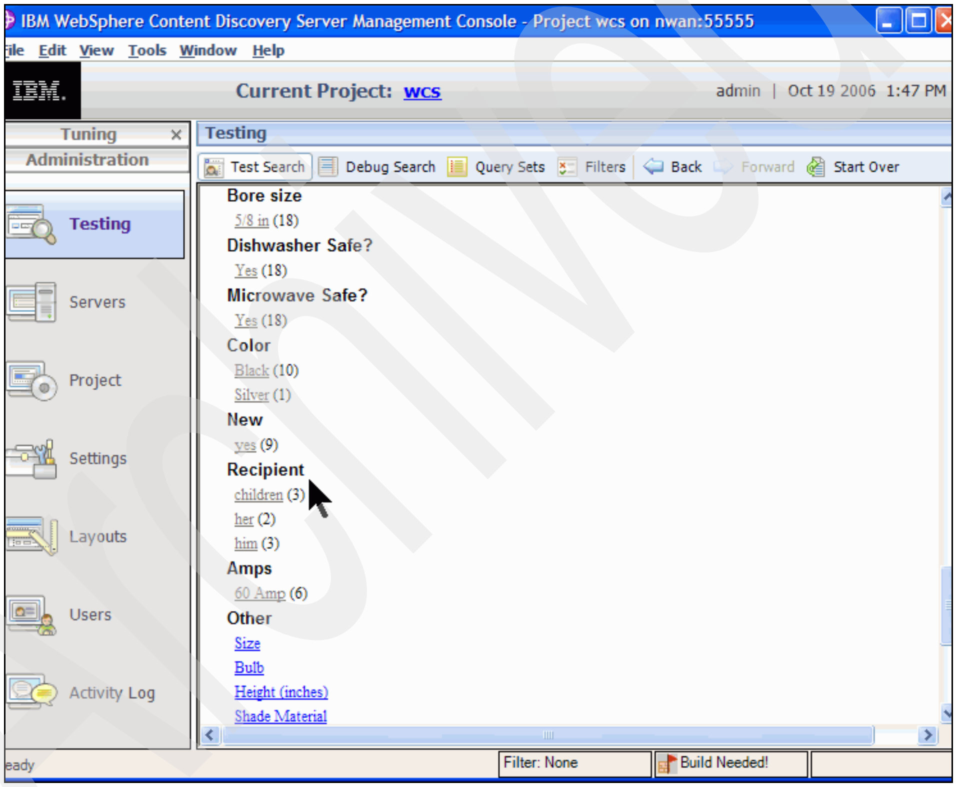
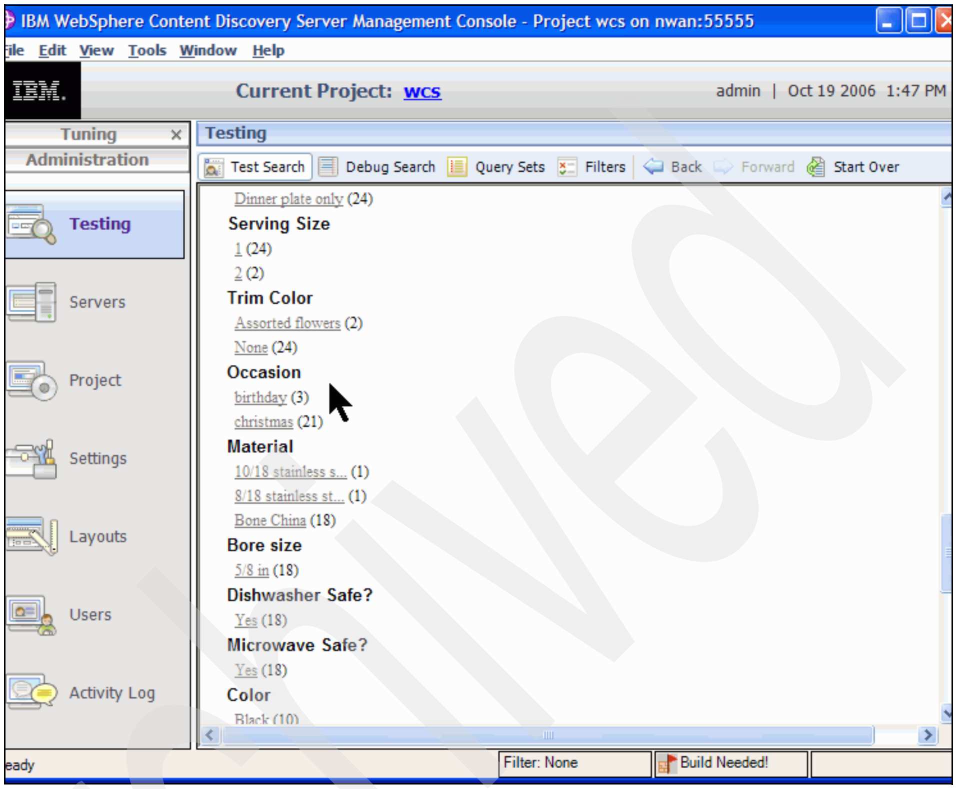


Figure 8-10 Recipient feature

5. Click each of the links, **children**, **her**, and **him**, to refine the search by the selected recipient value. Compare the results with the data defined in Table 8-1 on page 157. To start a new search, click the **Start Over** button at the top.

6. Find the Occasion feature at the left navigation display of the search panel. You should see it under the refine list with a label **Occasion**. See Figure 8-11.



Dinner plate only (24)
Serving Size
1 (24)
2 (2)
Trim Color
Assorted flowers (2)
None (24)
Occasion
birthday (3)
christmas (21)
Material
10/18 stainless s... (1)
8/18 stainless st... (1)
Bone China (18)
Bore size
5/8 in (18)
Dishwasher Safe?
Yes (18)
Microwave Safe?
Yes (18)
Color
Black (10)

Figure 8-11 Occasion feature

7. Click each of the **birthday** and **christmas** links to refine the search by the selected occasion value. Compare the results with the data defined in Table 8-2 on page 157. To start a new search, click the **Start Over** button on the top.

8.6 Presenting the results in your Web site

After verifying that the data is indexed correctly in OmniFind Discovery Edition, we can customize our WebSphere Commerce JSP to display the shop by recipient and shop by occasion features.

For each of the shop by recipient and shop by occasion features, we add a new menu item to the top navigation drop-down menu. If the shopper moves the mouse over the menu item, a list of available feature values is displayed in the drop-down menu.

For example, if a shopper moves the mouse over shop by recipient, the shopper sees three choices: for children, for her, and for him. If the shopper moves the mouse over shop by occasion, the shopper sees two choices: birthday and christmas. After the shopper makes a selection, the action invokes a OmniFind Discovery Edition search and all matching products are displayed in the search results page.

The top navigation drop-down menu is implemented in CachedHeaderDisplay.jsp and, for this book, we have created a file, getHeaderFeatures.jsp in 6.4.1, “Implement getHeaderFeatures JSP” on page 102. Change it to a reference when merging the file to implement interactions with OmniFind Discovery Edition code. In Chapter 7, “Displaying new products in your site” on page 127, we modify these two JSPs for new arrival changes. In this chapter, we modify these two files again for the shop by recipient and shop by occasion features.

8.6.1 Customizing getHeaderFeature.jsp

First we have to define variables to hold the values we require. We use two vectors to hold the values of the two features, and we use two strings to hold the name of the features.

We define the variables we require as shown in Example 8-2. See the bold text.

Example 8-2 Defining variables for Recipient and Occasion features

```
<%
Vector manufacturerVector = new Vector();
// CUSTOMIZATION
// occasionVector holds feature values for the occasion feature
Vector occasionVector = new Vector();
// recipientVector holds feature values for the recipient feature
Vector recipientVector = new Vector();

// occasionId holds the name of the occasion feature
String occasionId = "";
// recipientId holds the name of the recipient feature
String recipientId = "";
%>
```

We have to determine the available feature values when shoppers mouse over the shop by recipient or shop by occasion features. To do this, we add additional logic to find the information we require when we go through the drill-down list we used in 6.6, "Implement By Brand View display page" on page 116.

Inside the *for* loop, when we go through the drill-down list (Example 6-4 on page 104), we check if the feature is Recipient or the feature is Occasion. If so, we add the value of the feature to the recipientVector or occasionVector, depending on which feature it is.

The recipientVector and occasionVector are used in CachedHeaderDisplay.jsp when we display the options in the top navigation drop-down menu. We also store the name of the features in the occasionId and recipientId variables that are used when we define the constraints for the features. See the new codes shown in bold text in Example 8-3.

Example 8-3 populating values and names of Occasion and Recipient features

```
<%
if(drillDown != null){

   for (int j = 0; j < tallyFeatures.length; j++){
      com.iphrase.runtime.query.result.TallyFeature currentFeature = 
tallyFeatures[j];
      DrillDownList ddList = new DrillDownList(query, currentFeature, 
true);

      String currentFeatureLabel = currentFeature.getLabel();

      if (currentFeatureLabel.equals("Manufacturer Name")){
      //Got the MFname
         for(int listIndex = 0; listIndex < ddList.getCount(); 
listIndex++){
            com.iphrase.onestep.beans.TallyValueHandler valueHandler = 
ddList.getValues()[listIndex];

            String formatValueStr = formatValue(valueHandler, "", -1, 
"");
            formatValueStr = truncateText(formatValueStr, 100);

            manufacturerVector.addElement(formatValueStr);
         }  // for
      }  // if

      // CUSTOMIZATION - Get the available feature values for shop by 
occasion
```

```
      if (currentFeatureLabel.equals("Occasion")){
         for(int listIndex = 0; listIndex < ddList.getCount();
listIndex++){
            com.iphrase.onestep.beans.TallyValueHandler valueHandler =
ddList.getValues()[listIndex];

            String formatValueStr = formatValue(valueHandler, "", -1,
"");
            formatValueStr = truncateText(formatValueStr, 100);

            occasionVector.addElement(formatValueStr);
         }  // for
         occasionId = currentFeature.getId();
      }  // if

      // CUSTOMIZATION - Get the available feature values for shop by
recipient
      if (currentFeatureLabel.equals("Recipient")){
         for(int listIndex = 0; listIndex < ddList.getCount();
listIndex++){
            com.iphrase.onestep.beans.TallyValueHandler valueHandler =
ddList.getValues()[listIndex];

            String formatValueStr = formatValue(valueHandler, "", -1,
"");
            formatValueStr = truncateText(formatValueStr, 100);

            recipientVector.addElement(formatValueStr);
         }  // for
         recipientId = currentFeature.getId();
      }  // if
   } // for
}  // if
%>
```

Next, for each of the Recipient and Occasion features, we create a JavaScript function to pass the constraints to OmniFind Discovery Edition and encode a URI for the feature.

Each of the JavaScript functions does the following tasks:

1. It takes the feature value as a parameter.
2. It constructs the Constraint object using OmniFind Discovery Edition API.
3. It passes the feature value to the constraint.

4. It calls headerUpdater(), a JavaScript function called by other JavaScript functions.
5. The headerUpdater() function sets the encoded constraint URI to ip_constrain and submits the search to OmniFind Discovery Edition.

See Example 8-4.

Example 8-4 JavaScript to encode URI and submit search constraint

```
<%-- CUSTOMIZATION Shop by occasion feature --%>
function updaterOccasion(value) {
   <%
      Constraint cOccasion = new Enumerated.Equals(occasionId,
"nullString");
      cOccasion.clearRequired();
   %>
   var orgString = "<%=cOccasion.toString()%>";
   var finalString = orgString.replace("nullString", value);
   headerUpdater('ip_constrain', encodeURIComponent(finalString));
}

<%-- CUSTOMIZATION Shop by recipient feature --%>
function updaterRecipient(value) {
   <%
      Constraint cRecipient = new Enumerated.Equals(recipientId,
"nullString");
      cRecipient.clearRequired();
   %>
   var orgString = "<%=cRecipient.toString()%>";
   var finalString = orgString.replace("nullString", value);
   headerUpdater('ip_constrain', encodeURIComponent(finalString));
}
```

8.6.2 Customizing CachedHeaderDisplay.jsp

We have to add two dropdowns to the top navigation drop-down menu, one for shop by recipient, and one for shop by occasion. To do this, open CachedHeaderDisplay.jsp. Locate where the drop-down menu is defined. To locate the code where the drop-down menu is defined, search for <div class="dropDownMenu">. Under the dropDownMenu and after the by brand drop-down is defined, add the code as shown in Example 8-5 to add the two dropdowns we require.

Example 8-5 Define dropdowns for shop by recipient and shop by occasion

```
<li><a class="hide" >By Brand  </a>
   <a href="#">By Brand
      <table><tr><td>
      <ul>
         <%for(int i=0;i<manufacturerVector.size();i++){

pageContext.setAttribute("brandName",manufacturerVector.elementAt(i));
%>
            <c:url var="ByBrandViewURL" value="WCDSByBrandView">
               <c:param name="MFName" value="${brandName}" />
               <c:param name="catalogId" value="${WCParam.catalogId}"
/>
               <c:param name="storeId" value="${WCParam.storeId}" />
            </c:url>
         <li> <a HREF="<c:out value="${ByBrandViewURL}" />" > <c:out
value="${brandName}" /> </a></li>
         <%}%>
      </ul>
      </td></tr></table>
      </a>
</li>

<%-- CUSTOMIZATION Shop by occasion --%>
<li><a class="hide" >Shop by Occasion  </a>
   <a href="#">Shop by Occasion
      <table><tr><td>
      <ul>
         <%
            for(int i=0;i<occasionVector.size();i++){
               String currentLabel = (String)
occasionVector.elementAt(i);
         %>
         <li> <a
HREF="javascript:updaterOccasion('<%=currentLabel%>')">
<%=currentLabel%> </a></li>
         <%
            }
         %>
      </ul>
      </td></tr></table>
   </a>
</li>
```

```
<%-- CUSTOMIZATION Shop by recipient --%>
<li><a class="hide" >Shop by Recipient  </a>
   <a href="#">Shop by Recipient
      <table><tr><td>
      <ul>
         <%
            for(int i=0;i<recipientVector.size();i++){
               String currentLabel = (String)
recipientVector.elementAt(i);
         %>
         <li> <a
HREF="javascript:updaterRecipient('<%=currentLabel%>')"
class="menuitem"> For <%=currentLabel%> </a></li>
         <%
            }
         %>
      </ul>
      </td></tr></table>
   </a>
</li>
```

8.7 Testing the results in your store

To test the results, navigate to the Consumer-Direct store home page. Move the mouse over the **By Recipient** link from the top navigation drop-down menu. Check and see if the feature values are displayed in the drop-down. Using our example, you should see **For children**, **For her**, and **For him** under the Shop by Recipient drop-down. Click each of the links and compare the search results with the data we defined in Table 8-1 on page 157.

Repeat the same steps for the **Shop by Occasion** drop-down. You should see **Birthday** and **Christmas** under the drop-down menu. Click each of the links and compare the search results with the data we defined in Table 8-2 on page 157.

Customizing the category display page

In this chapter we describe how we used OmniFind Discovery Edition to customize the category page of our sample store.

9.1 Designing the category display page flow

Figure 9-1 shows the catalog structure used for the sample store.

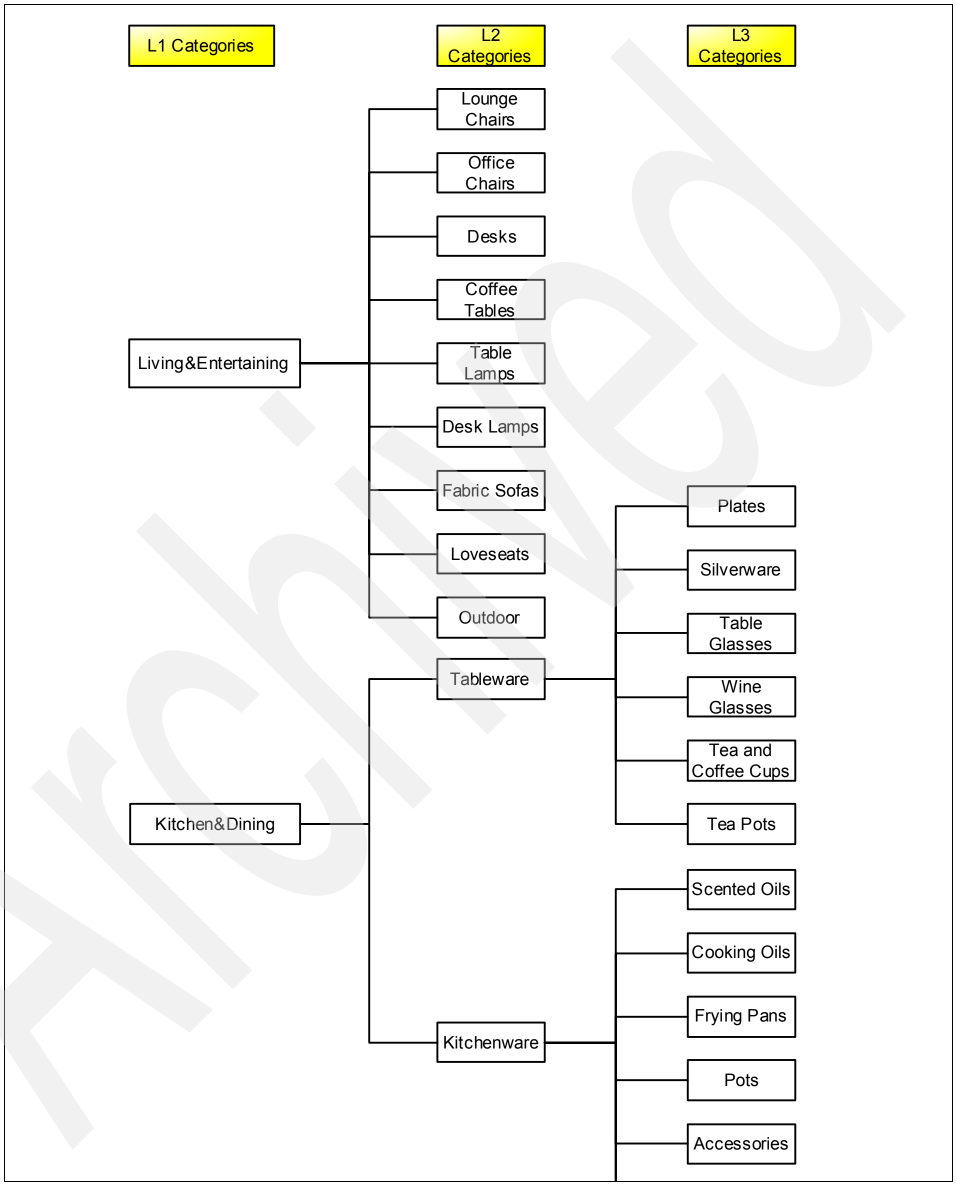


Figure 9-1 Sample store: Catalog structure

From the home page of our sample store, customers can select a top level category from the first page (CachedTopCategoriesDisplay.jsp). The choice of top level category is *Living & Entertaining* or *Kitchen & Dining*. After the customer clicks one of the top menu selections, the category L2 page appears (CachedCategoriesDisplay.jsp). By default, the category L2 page shows the subcategories of the chosen top level category. If the customer clicks on the subcategory link or icon, the product page that shows all products under the subcategory appears.

It would be convenient for customers and beneficial to stores, if at the category L2 page, we show one hot-selling item or promotional item for each subcategory along with the normal subcategory link. This helps the store to promote a particular item. The customer can select the item without further clicking through the subcategory link or icon.

Figure 9-2 shows the modified category L2 page. On the right hand side of the page, we replaced the generic icons of the subcategories with images of items from each subcategory and added description and price of the items. The links to show all products under the subcategory still remain. On the left navigation panel, we add a Refine by list component (Further search) from which the customer can select a feature value or a range of values to narrow down the items that match their choice.

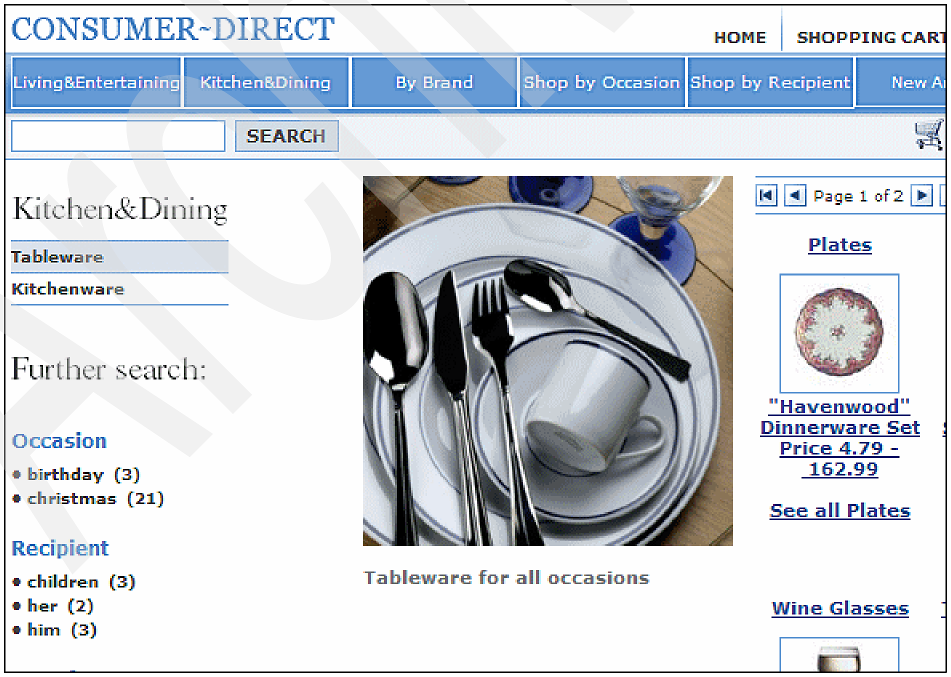


Figure 9-2 Sample store: Category L2 page

For our customization, the individual item that appears along with each subcategories is the item that OmniFind Discovery Edition happens to return at the top of the result set. It is possible that you set up a business rule to make the item of your own choosing be at the start of the result set, so that it is that item which appears on the page. Such a business rule can be set up using Management Console. You can choose to display the most profitable item, the most popular item, or use whatever other criterion that you choose for your business requirement.

Figure 9-3 outlines the components making up the category L2 page.

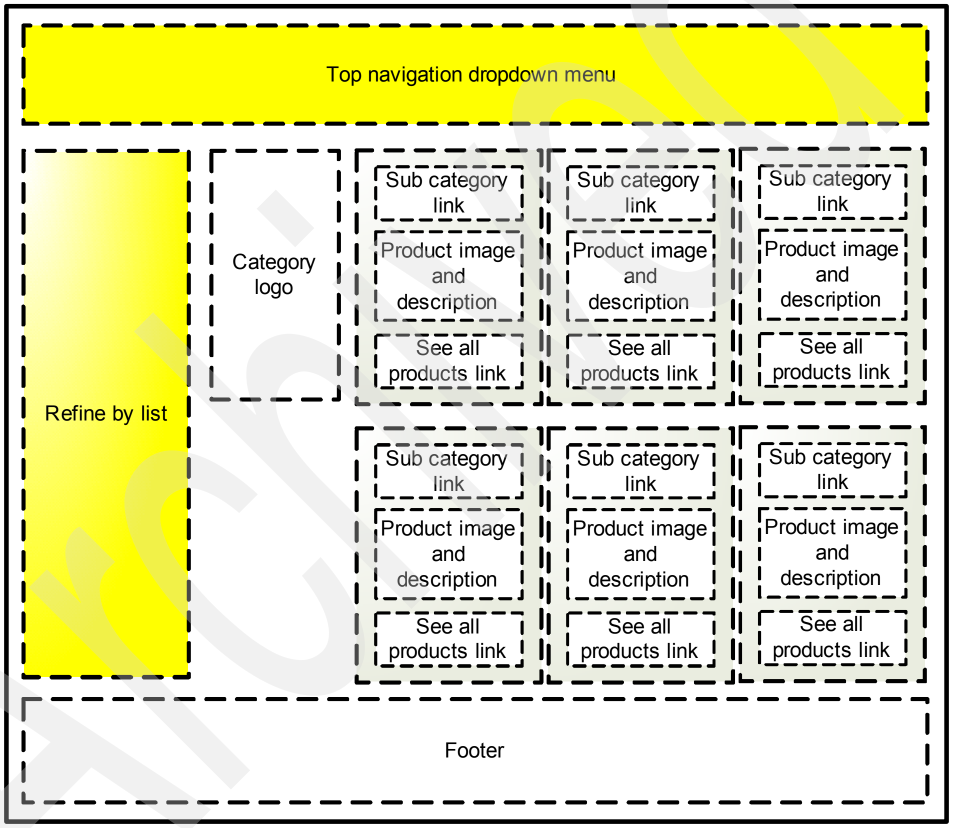


Figure 9-3 Sample store: Category L2 page

The full path name of this page is <WCS_HOME>\Stores.war\ConsumerDirect\Snippets\Catalog\CategoryDisplay\CachedCategoriesDisplay.jsp

9.2 Display of subcategories

On the right of the original WebSphere Commerce page is a panel that shows an icon for each subcategory of the chosen top level category. We want to customize the application such that for each subcategory, there is a product surfaced at that level and users can select the product right away. As mentioned earlier, this particular product can be a promotional product, the most popular product, or the best recommended product. This saves users the time of traversing down deeper to the lower product page and, most importantly, it facilitates the online store in selling the particular product under that subcategory.

To accomplish this, we replace each subcategory icon with the image of one particular product in that subcategory. Under the image of the product, we put the name and price of the product. There is still a link that the user can click to see all the products under this subcategory (Figure 9-2 on page 181).

If the customer clicks the image, name, or price of the product, it takes them to a page (ProductDisplay.jsp) that displays that particular product. Above and below the product icon and information, we put the name of the subcategory. If the customer clicks the name of the subcategory, it displays all items in that subcategory.

Let us now show you how to implement this page.

9.2.1 Importing packages

Near the top of the page, we import the packages that contain the OmniFind Discovery Edition APIs.

These are the packages that contain the OmniFind Discovery Edition Runtime API. See Example 9-1.

Example 9-1 Import packages

```
%><%@ page import="com.iphrase.runtime.exception.*"
%><%@ page import="com.iphrase.runtime.query.*"
%><%@ page import="com.iphrase.runtime.query.result.*"
%><%@ page import="com.iphrase.runtime.query.constraint.*"
%><%@ page import="com.iphrase.runtime.*"
%><%@ page import="com.iphrase.onestep.beans.*"
```

The bold text denotes the package that is required for the OmniFind Discovery Edition Layout APIs.

9.2.2 Identifying where to insert new code

Before you customize the existing file, CachedCategoriesDisplay.jsp, study and understand what it does. Identify the code that renders the icon for each subcategory. Example 9-2 shows the code in a loop that starts the display.

Example 9-2 Code that renders icons for each subcategory

```
<%-- idCounter starts with 1 --%>
<c:forEach var="subCategory" items="${category.subCategories}"
varStatus="idCounter">
   <c:if test="${((idCounter.count > (currentPage * pageSize)) &&
(idCounter.count <= ((currentPage+1) * pageSize)))}">
      <%-- Always assume images exist --%>
      <c:choose>
         <c:when test="${pageView == 'image'}">
<td class="t_img_view">
         </c:when>
         <c:otherwise>
<td class="t_row_img" valign="top">
         </c:otherwise>
      </c:choose>

      <%-- The URL command that will display the sub-category --%>
      <c:url var="categoryDisplayUrl" value="CategoryDisplay">
         <c:param name="catalogId" value="${WCParam.catalogId}" />
         <c:param name="storeId" value="${WCParam.storeId}" />
         <c:param name="categoryId" value="${subCategory.categoryId}"
/>
         <c:param name="langId" value="${langId}" />
         <c:param name="parent_category_rn" value="${categoryId}" />
         <c:param name="top_category" value="${categoryId}" />
         <c:param name="pageView" value="${WCParam.pageView}" />
      </c:url>
```

We insert our code after this point.

9.2.3 Configuring the connection

In order to customize a page to use OmniFind Discovery Edition, you have to insert some code to connect to a OmniFind Discovery Edition server.

We placed our code in a block that catches any exception or error and displays an appropriate message. See Example 9-3.

Example 9-3 Exception handling

```
<%
try{
...
}
catch (Throwable e)
{
  %><b>Cannot display data</b><%
}%>.
```

An exception is thrown, for example, if the application cannot connect to the OmniFind Discovery Edition server.

Our code uses a *QueryHandler* object to handle the connection to OmniFind Discovery Edition. QueryHandler is in the Layouts API. It provides high-level access to a Runtime API *Query* object. At the start of the block, our code creates the QueryHandler:

```
QueryHandler queryHandler = new QueryHandler();
```

It then sets the QueryHandler to do time out if they do not have search responses from the server after 60 seconds:

```
queryHandler.setTimeout(60);
```

It sets the host and the port number that the QueryHandler should use to connect to the server. In this case the host is localhost and the port number is 8777:

```
queryHandler.addServerAddress("localhost:8777");
```

QueryHandler is designed to read variables from and write variables to the JSP page context. When using a QueryHandler, you always have to call initializePageContext() to read variables from the JSP page context:

```
if(!queryHandler.initializeWithPageContext(pageContext))
      return;
```

9.2.4 Running the search

We have to identify each subcategory for which to display an image. Notice that in the original WebSphere Commerce page, the JavaServer Standard Tag Library (JSTL) variable `subCategory.description.name` is set to the name of the subcategory whose icon it displays. In order to be able to access this value in scriptlet code to use the OmniFind Discovery Edition APIs, we assign it to a scriptlet variable as shown in Example 9-4, which implements these steps:

1. Use the JSTL tags to assign it from the JSTL variable to a variable in the JSP page context.
2. Use the scriptlet code to assign it from the variable in the JSP page context to a scriptlet variable.

Example 9-4 Find the current subcategory

```
/*
  Find the current subcategory.
  */

   // In the original WebSphere Commerce page the JSTL variable
   // subCategory.description.name has already been set to a
   // subcategory name.
   // We assign this value to a scriptlet variable called
   // selectedSubCategory

   // We do this by first moving it to
   // the variable ip_selectedSubcategory
   // in the JSP page context...
  %><c:set var="ip_selectedSubCategory"
           value="${subCategory.description.name}"
           scope="request"></c:set><%
   // ...then moving the value from the JSP page context variable
   // to the scriptlet variable.
  String selectedSubCategory =
  (String)request.getAttribute("ip_selectedSubCategory");
```

We then used OmniFind Discovery Edition to find a single item from this subcategory.

The code defines the constraint that the Category feature value must be for this subcategory. When defining a constraint, use the feature identifier. For our sample store, the identifier for the Category feature is `sitemap taxonomy`. Constraints are explained in 5.5.1, “Constraints” on page 83. See Example 9-5, which shows how to define the constraints.

Example 9-5 Define the constraints

```
String values[] = {selectedSubCategory};
  Constraint constraint =
   new com.iphrase.runtime.query.constraint.Taxonomy.Contains(
   "sitemap taxonomy", values);
```

This coding sample gets the Query object out of the QueryHandler:

```
Query query = queryHandler.getRuntimeQuery();
```

It gets a reference to the query constraints. The constraints are framed as a *conjunction*. Conjunctions are explained in 5.5.2, “Complex searches” on page 86:

```
Conjunction conjunction = query.getConstraints();
```

Then it sets the query to have the constraint it defined above, namely that the Category feature value must be this subcategory:

```
conjunction.merge(new Disjunction(constraint));
```

Be aware that performance might be impacted if you retrieve all the items in a given subcategory. We really only want one item from each subcategory. To do so, we use the setTruncation() function. It sets the query to retrieve only one item for each value of the Category feature:

```
query.setTruncation("sitemap taxonomy", 1);
```

Execute the QueryHandler to run the search:

```
queryHandler.execute();
```

This statement retrieves the query result. When a QueryHandler is executed, it sets the variable queryResultRqst in the JSP page context to be the QueryResult object that was returned:

```
QueryResult queryResultRqst =
(QueryResult)pageContext.getAttribute("queryResultRqst",
PageContext.REQUEST_SCOPE);
```

A QueryResult can contain more than one ResultSet. Our code takes the first ResultSet from the QueryResult:

```
com.iphrase.runtime.query.result.ResultSet resultSets[] =
      queryResultRqst.getResultSets();
     com.iphrase.runtime.query.result.ResultSet resultSet =
      resultSets[0];
```

9.2.5 Extracting data from the result

Because we set the query to retrieve only one item for each value of the Category feature, the ResultSet should contain no more than one product in the given subcategory. We have to extract from the ResultSet, the name, the price, and the URL of the thumbnail image so that we can display them on the page. We also have to extract the product ID because this forms part of the URL of the WebSphere Commerce page to which we want to take the customer when they click the product name, price, or thumbnail image.

Our code declares and initializes variables to store the product ID, name, price and thumbnail image (Example 9-6).

Example 9-6 initialize variables to store product ID, name, price, thumbnail image

```
String ip_productId;
ip_productId = "Product ID not set";
String ip_name;
ip_name = "Name not set";
String ip_price;
ip_price = "Price not set";
String ip_thumbnail;
ip_thumbnail = "Thumbnail not set";
```

The following code gets the Table from the ResultSet:

```
com.iphrase.runtime.query.result.Table mainTable =
resultSet.getTable();
```

The following code proceeds to extract the feature values from the Table only if the search returned some items. If table does not return any item, the Table is equal to null:

```
if (mainTable != null) {
```

If table is not null, meaning that some values were returned, we take the array of FeatureMetaData from the ResultSet. There is one FeatureMetaData object for each feature, and it contains data about that feature, such as its *label*. A label is the text that is typically displayed next to the value of a feature to say what it is:

```
FeatureMetaData[] metas = mainTable.getFeatureMetaDatas();
```

For each item returned from a search, the Table has one *FeatureRow*. The FeatureRow contains one value for each feature. Our search should return no more than one item, so we take only the first FeatureRow from the Table:

```
FeatureRow[] rows = mainTable.getFeatureRows();
FeatureRow row = rows[0];
```

For each feature value in the FeatureRow, it creates a ValueHandler object. ValueHandler is a utility class whose purpose it is to make it easier to access the values of many different types of features. ValueHandler is in the Layouts API (Example 9-7).

Example 9-7 Createing a ValueHandler object

```
for (int featureStatus = 0;
         featureStatus < row.getFeatureValues().length;
         featureStatus++){
```

```
      Object val = row.getFeatureValues()[featureStatus];
      FeatureMetaData currentFeature = metas[featureStatus];
      ValueHandler valueHelperRqst =
       ValueHandler.CreateInstance(currentFeature, val);
```

To convert the value to a string for display, use the formatValue() method. The formatValue() method is in the file formatValue.jsp in the OmniFind Discovery Edition Tabbed Navigation layout. To use this method, we copy formatValue.jsp from the Tabbed Navigation layout into the same directory as the page we are customizing, and include this directive near the top:

```
<%@ include file="formatValue.jsp"%>
```

For the Price feature display, our code called formatValue() to convert the value to a string for display (Example 9-8).

Example 9-8 Converting the value to a string for display

```
if (currentFeature.getLabel().equals("Price")){
        ip_price = formatValue(valueHelperRqst,
                               "&nbsp;-&nbsp;",
                               rowStatus,
                               resultSet.getUserData().toString());
   }
```

For the Thumbnail feature display, we use formatValue() also. Because the Thumbnail feature is of type ImageURL, the formatValue() returns the URL of the thumbnail image for display (Example 9-9).

Example 9-9 Using the thumbnail feature

```
else if (currentFeature.getLabel().equals("Thumbnail")){
        ip_thumbnail = formatValue(valueHelperRqst,
                                   "&nbsp;-&nbsp;",
                                   rowStatus,

   resultSet.getUserData().toString());
   }
```

For the Product ID feature display, we call formatValue() to extract the substring required to form part of the URL of the WebSphere Commerce page to display that product (Example 9-10).

Example 9-10 Extracting the substring required

```
else if (currentFeature.getLabel().equals("TypeID")){
        ip_productId = formatValue(valueHelperRqst,
                       "&nbsp;-&nbsp;",
                        rowStatus,
                        resultSet.getUserData().toString());
            ip_productId = ip_productId.substring(12,17);
          }
```

The Name feature display is of type Link. formatValue() returns a string embedded in an anchor (`<a></a>`) HTML tag. Our code stripped this tag off (Example 9-11).

Example 9-11 Code that strips off the <a> tag from ip_name

```
else if (currentFeature.getLabel().equals("Name")){
        ip_name = formatValue(valueHelperRqst,
                              "&nbsp;-&nbsp;",
                              rowStatus,
                              resultSet.getUserData().toString());
        // If valueHelperRqst is a ValueHandler for which
        // getLink() is true, strip off the <a> tag from ip_name.
        if (ip_name.startsWith("<a")){
          int startPos, endPos;
          startPos = ip_name.indexOf(">");

          if (startPos != -1){
            endPos = ip_name.substring(startPos + 1).indexOf(">");
            if (endPos != -1){
              ip_name = ip_name.substring(startPos + 1,
                                          startPos + endPos - 2);
            }
          }
        }
      }
```

9.2.6 Rendering the page

We used the data retrieved from OmniFind Discovery Edition to change the appearance of the page.

We remove the original code as shown in Example 9-12.

Example 9-12 Original code that is removed

```
      <a href="<c:out value="${categoryDisplayUrl}" />"
id="WC_CachedCategoriesDisplay_Link_ForSubCatImg_<c:out
value="${idCounter.count}"/>">
      <c:choose>
```

```
         <c:when test="${!empty subCategory.description.thumbNail}">
            <span class="t_img_border"><img src="<c:out
value="${subCategory.objectPath}${subCategory.description.thumbNail}"
/>" alt="<c:out value="${subCategory.description.shortDescription}" />"
border="0" /></span>
         </c:when>
         <c:otherwise>
            <span class="t_img_border"><img src="<c:out
value="${jspStoreImgDir}"/>images/NoImageIcon_sm.jpg" alt="<fmt:message
key="No_Image" bundle="${storeText}"/>" border="0"/></span>
         </c:otherwise>
      </c:choose>
      </a><br/>
      <c:if test="${pageView == 'image'}">
         <br/>
      </c:if>
      <c:if test="${pageView == 'detailed'}">
</td><td class="t_row_detail">
      </c:if>
   <span class="productName">
      <a href="<c:out value="${categoryDisplayUrl}" />"
id="WC_CachedCategoriesDisplay_Link_ForSubCat_<c:out
value="${idCounter.count}"/>">
         <c:out value="${subCategory.description.name}"
escapeXml="false"/>
      </a>
```

In place of the removed code, we insert the code as shown in Example 9-13.

Example 9-13 Inserted code

```
      <span class="productname">
      <a href="<c:out value="${categoryDisplayUrl}" />"
id="WC_CachedCategoriesDisplay_Link_ForSubCat_<c:out
value="${idCounter.count}"/>">
         <c:out value="${subCategory.description.name}"
escapeXml="false"/>
      </a>
      </span><br><br>
      <a
href=/webapp/wcs/stores/servlet/ProductDisplay?catalogId=<c:out
value="${catalogId}" escapeXml="false"/>&storeId=<c:out
value="${storeId}"
escapeXml="false"/>&productId=<%=ip_productId%>&langId=<c:out
value="${langId}" escapeXml="false"/>
```

```
id="WC_CachedCategoriesDisplay_Link_ForThumbNail<c:out
value="${idCounter.count}"/>">
            <span class="t_img_border"><%=ip_thumbnail%></span>
      </a><br>
      <span class="productname">

      <a
href=/webapp/wcs/stores/servlet/ProductDisplay?catalogId=<c:out
value="${catalogId}" escapeXml="false"/>&storeId=<c:out
value="${storeId}"
escapeXml="false"/>&productId=<%=ip_productId%>&langId=<c:out
value="${langId}" escapeXml="false"/>
id="WC_CachedCategoriesDisplay_Link_ForName2<c:out
value="${idCounter.count}"/>">
         <%=ip_name%>
      </a>
      </span><br>
      <span class="productname">
      <a
href=/webapp/wcs/stores/servlet/ProductDisplay?catalogId=<c:out
value="${catalogId}" escapeXml="false"/>&storeId=<c:out
value="${storeId}"
escapeXml="false"/>&productId=<%=ip_productId%>&langId=<c:out
value="${langId}" escapeXml="false"/>
id="WC_CachedCategoriesDisplay_Link_ForPrice<c:out
value="${idCounter.count}"/>">
         Price <%=ip_price%>
      </a>
      </span>
      <br/>
      <c:if test="${pageView == 'image'}">
         <br/>
      </c:if>
      <c:if test="${pageView == 'detailed'}">
</td><td class="t_row_detail">
      </c:if>
      <span class="productName">
      <a href="<c:out value="${categoryDisplayUrl}" />"
id="WC_CachedCategoriesDisplay_Link_ForSubCat_<c:out
value="${idCounter.count}"/>">
         <c:out value="See all ${subCategory.description.name}"
escapeXml="false"/>
      </a>
```

By using `<span class=”productname”></span>` tags, we use the Cascading Style Sheets (CSS) of the original page to keep the WebSphere Commerce appearance and behavior.

Example 9-14 shows the code that we insert to display the subcategory name, and links to the WebSphere Commerce page that shows the given subcategory.

Example 9-14 Display subcategory name and links to Commerce page

```
      <span class="productname">
      <a href="<c:out value="${categoryDisplayUrl}" />"
id="WC_CachedCategoriesDisplay_Link_ForSubCat_<c:out
value="${idCounter.count}"/>">
         <c:out value="${subCategory.description.name}"
escapeXml="false"/>
      </a>
      </span><br><br>
```

Example 9-15 shows the code that we insert to display the product image, and links to a page that shows that product. The href in the anchor tag is a URL that includes `catalogId` (a JSTL variable), `storeId` (a JSTL variable), `productId` (a scriptlet variable), and `langId` (a JSTL variable). By using the tags, `<span class=”t_img_border”></span>`, we keep the same WebSphere Commerce appearance and behavior for thumbnail images.

Example 9-15 Display product image and links

```
      <a
href=/webapp/wcs/stores/servlet/ProductDisplay?catalogId=<c:out
value="${catalogId}" escapeXml="false"/>&storeId=<c:out
value="${storeId}"
escapeXml="false"/>&productId=<%=ip_productId%>&langId=<c:out
value="${langId}" escapeXml="false"/>
id="WC_CachedCategoriesDisplay_Link_ForThumbNail<c:out
value="${idCounter.count}"/>">
            <span class="t_img_border"><%=ip_thumbnail%></span>
      </a><br>
```

Example 9-16 shows the code that we insert to display the name of the product, and also links to a page that shows that product.

Example 9-16 Display the name of the product and the links to the product page

```
      <span class="productname">
      <a
href=/webapp/wcs/stores/servlet/ProductDisplay?catalogId=<c:out
```

```
value="${catalogId}" escapeXml="false"/>&storeId=<c:out
value="${storeId}"
escapeXml="false"/>&productId=<%=ip_productId%>&langId=<c:out
value="${langId}" escapeXml="false"/>
id="WC_CachedCategoriesDisplay_Link_ForName2<c:out
value="${idCounter.count}"/>">
         <%=ip_name%>
      </a>
      </span><br>
```

Example 9-17 shows the code that we insert to display the price of the product, and also links to a page that shows that product.

Example 9-17 Display product price and link of the product page

```
      <span class="productname">
      <a
href=/webapp/wcs/stores/servlet/ProductDisplay?catalogId=<c:out
value="${catalogId}" escapeXml="false"/>&storeId=<c:out
value="${storeId}"
escapeXml="false"/>&productId=<%=ip_productId%>&langId=<c:out
value="${langId}" escapeXml="false"/>
id="WC_CachedCategoriesDisplay_Link_ForPrice<c:out
value="${idCounter.count}"/>">
         Price <%=ip_price%>
      </a>
      </span>
```

Example 9-18 shows the code that we insert to display a message, `See all subcategory`, and links to the WebSphere Commerce page that shows the given subcategory.

Example 9-18 Display See all subcategory message and subcategory link

```
      <span class="productName">
      <a href="<c:out value="${categoryDisplayUrl}" />"
id="WC_CachedCategoriesDisplay_Link_ForSubCat_<c:out
value="${idCounter.count}"/>">
         <c:out value="See all ${subCategory.description.name}"
escapeXml="false"/>
      </a>
```

9.3 Drill down menu

We add to the page a menu panel from which the customer can select a value or value ranges for the features defined in OmniFind Discovery Edition for the store. When the customer selects a feature value or a value range from the menu panel, the system returns a search result page, which displays the products that match the customer's selection.

Narrowing a search in this way is called drill down. We add the drill down menu on the left side of the page underneath a menu, which the page already has, showing subcategories of the chosen top level category. We implement this by including a JSP fragment very similar to the one described in detail in 6.5, "Implement refinement list JSP fragment" on page 106.

9.3.1 Importing packages

For this customization, we import OmniFind Discovery Edition packages, just as in 9.2.1, "Importing packages".

9.3.2 Determining where to insert new code

By inspecting the original code of the page, we determine that the existing menu under which we wish to create our drill down menu is rendered by this code:

```
<%@ include 
file="../../../Snippets/ReusableObjects/CategoriesSidebarDisplay.jspf"%
>
```

We insert our code after that point.

9.3.3 Configuring the connection

Add the code to configure the connection to the OmniFind Discovery Edition server, as described in 9.2.3, "Configuring the connection" on page 184.

9.3.4 Running the search

To generate a drill down menu, first run a search to find what feature values there are.

Our code uses the QueryHandler to do a search. We require a search constraint. In this case, we define the constraint to be that the Category feature must not be null. The identifier of the Category feature in our store is `sitemap taxonomy`.

This constraint should match every item in our store. Constraints are explained in 5.5.1, “Constraints” on page 83:

```
Constraint constraint = new Null("sitemap taxonomy", false);
```

We get the Query object out of the QueryHandler:

```
Query query = queryHandler.getRuntimeQuery();
```

We get a reference to the query constrains. The query constraints are framed as a *conjunction*. Conjunctions are explained in 5.5.2, “Complex searches” on page 86:

```
Conjunction conjunction = query.getConstraints();
```

We set the query constraint to be that the Category must not be null:

```
conjunction.merge(new Disjunction(constraint));
```

We execute the QueryHandler to run the search:

```
queryHandler.execute();
```

This QueryHandler retrieves the query result. When a QueryHandler is executed, it sets the variable queryResultRqst in the JSP page context to be the QueryResult object that it returned:

```
QueryResult queryResultRqst =
(QueryResult)pageContext.getAttribute("queryResultRqst",
PageContext.REQUEST_SCOPE);
```

9.3.5 Rendering the drill down menu

We add the following line of code:

```
%><%@include file="refineList.jspf"%><%
```

We copy the file refineList.jspf, which is explained in 6.5, “Implement refinement list JSP fragment” on page 106, to the same directory as the file CachedCategoriesDisplay.jsp, which we are customizing.

We remove the following line from refineList.jspf in order to make this work:

```
<%@include file="formatValue.jsp"%>
```

This is done because we have already inserted the same directive into CachedCategoriesDisplay.jsp for the customization described in 9.2, “Display of subcategories” on page 183.

10

Customizing the search page

In this chapter we discuss the customization and integration of WebSphere Commerce and OmniFind Discovery Edition products, which empower and enable users to quickly customize their online WebSphere Commerce stores by leveraging the rich search and navigational capabilities offered by OmniFind Discovery Edition.

We outline in detail the steps undertaken to customize and integrate capabilities of WebSphere Commerce and OmniFind Discovery Edition for an existing sample starter store called Consumer-Direct. We show you how to use the OmniFind Discovery Edition Layout APIs and Runtime APIs, existing out-of-the-box JSPs in designing and rendering the user search results on the search result Web page. This helps you to gain a full understanding about the appearance and behavior of the starter store that we work on and also helps you to customize your own stores.

We cover the following topics in this chapter:

- Summary of APIs
- Out-of-the-box search result page
- Customization using TabbedNav skin
- Customization using Commerce Layout skin
- Troubleshooting and debugging hints and tips

10.1 Summary of APIs

In this section, we build upon the OmniFind Discovery Edition APIs that are introduced in Chapter 5, “Introducing OmniFind Discovery Edition APIs” on page 67, and cover some of the APIs that are helpful in customizing the OmniFind Discovery Edition search results page.

OmniFind Discovery Edition has two sets of APIs for running searches and rendering search results:

- Runtime APIs
- Layouts APIs

The OmniFind Discovery Edition search results page provides a rich set of search and navigation features within a WebSphere Commerce store. It consists of the following common JSP components:

- main.jsp
- globalForm.jsp
- voiceOver.jsp
- refineByList.jsp
- drillSideWays.jsp
- mainResultsTable.3a.jsp
- pagination.jsp
- viewByOptions.jsp

We highlight some of the OmniFind Discovery Edition Layout APIs and Runtime APIs used in these JSP files. Become familiar with them so that you can use them for your own customization. Do not worry if you do not understand all of them. If you have not read Chapter 5, “Introducing OmniFind Discovery Edition APIs” on page 67, this is a good time to go back and read through the chapter. This helps you to better understand the APIs we cover in this section.

Figure 10-1 shows these components on a search result page.

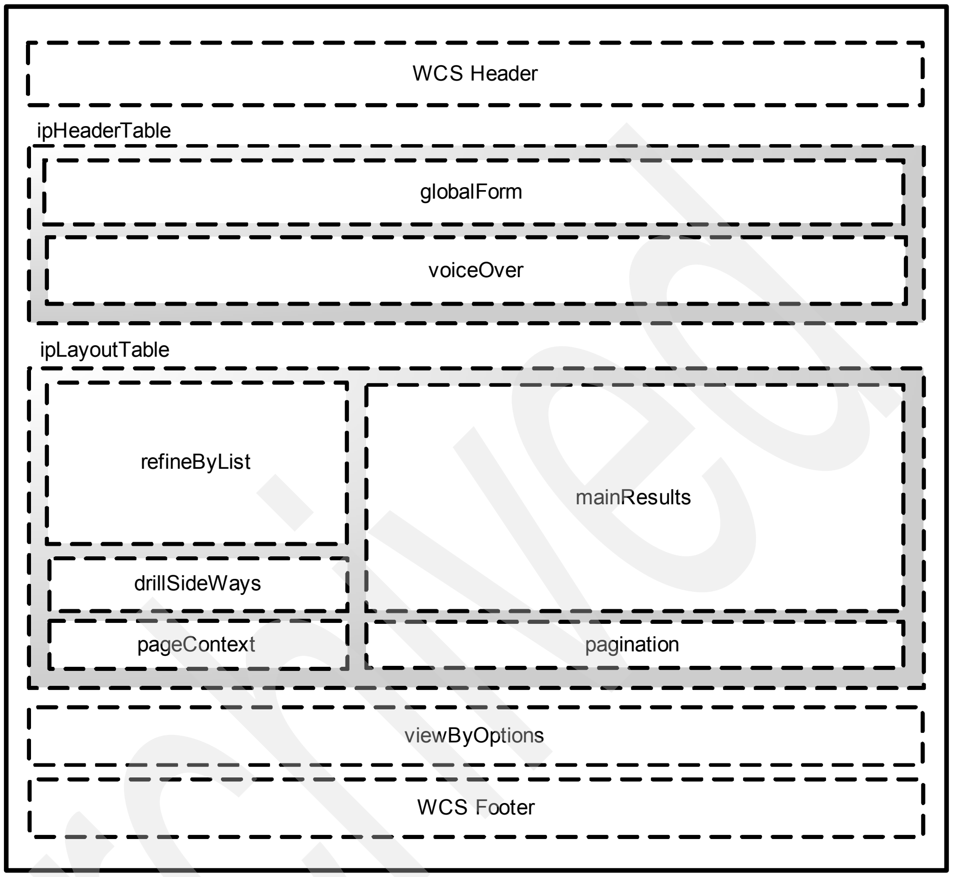
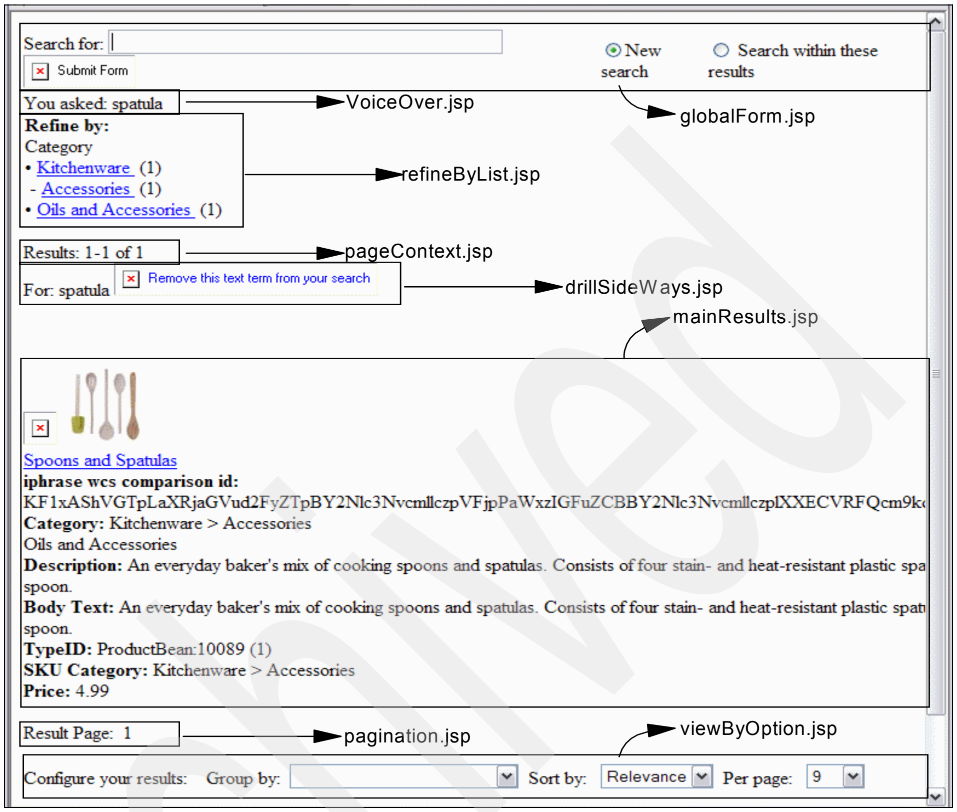


Figure 10-1 Search result page JSP components

Figure 10-2 shows a sample search results page with these components.



You asked: spatula
Refine by:
Category
• Kitchenware (1)
- Accessories (1)
• Oils and Accessories (1)

Results: 1-1 of 1
For: spatula

Spoons and Spatulas
iphrase wcs comparison id:
KF1xAShVGTpLaXRjaGVud2FyZTpBY2Nlc3NvcmllczpVFjpPaWxzIGFuZCBBY2Nlc3NvcmllczplXXECVRFQcm9k
Category: Kitchenware > Accessories
Oils and Accessories
Description: An everyday baker's mix of cooking spoons and spatulas. Consists of four stain- and heat-resistant plastic spa
spoon.
Body Text: An everyday baker's mix of cooking spoons and spatulas. Consists of four stain- and heat-resistant plastic spat
spoon.
TypeID: ProductBean:10089 (1)
SKU Category: Kitchenware > Accessories
Price: 4.99

Result Page: 1

Figure 10-2 Typical JSP components in a search result page

10.1.1 APIs used in main.jsp

main.jsp is the JSP page that is called from with the WebSphere Commerce Store Front Page when we submit a search form. *main.jsp* is responsible for instantiating a *QueryHandler* object and assigning appropriate values to the variables such as IP Address and the Port Number on which the service is running. OmniFind Discovery Edition finds this information based on the Resource Bundle parameter.

QueryHandler

QueryHandler provides high-level access to the OmniFind Discovery Edition Runtime API Query object. There are three important steps that QueryHandler performs when processing each search request:

1. It provides connection parameters required for communication with the OmniFind Discovery Edition server. The OmniFind Discovery Edition server address is obtained in one of the following ways:
   - The web.xml server context parameter
   - The ip_server URL parameter (for example, ip_server=localhost:8777)
   - A call to QueryHandler.addServerAddress (localhost:8777) function
2. It extracts search parameters from the *HttpServletRequest* object and provides them as an input to the OmniFind Discovery Edition server.
3. It receives search results from the OmniFind Discovery Edition server and prepare them to be used by other JSP code. QueryHandler builds and stores the following variables in the request-scope:
   - queryRqst — An instance of the Query class
   - queryResultRqst — An instance of the QueryResult class
   - voiceOverRqst — An instance of the VoiceOver bean
   - drillDownTabsRqst — An instance of the DrillDownTabs bean
   - breadcrumbsRqst — An instance of the Breadcrumb bean

Example 10-1 shows the code snippet from main.jsp that uses the QueryHandler API. See the bold text.

Example 10-1 Code snippet from main.jsp using QueryHandler API

```
<%@ page import="com.iphrase.runtime.query.constraint.*" %>
 <%
 QueryHandler queryHandler = new QueryHandler();
 QueryHandler.initializeWithPageContext(pageContext);

 // add hidden constraint every time
 String[] values = {"foo", "bar"};
 Constraint c = new Enumerated.InList("some_feature", values);
 c.setHidden();
 QueryHandler.getRuntimeQuery().getConstraints().merge(new 
Disjunction(c));

 queryHandler.execute();
 %>
```

10.1.2 APIs used in globalForm.jsp

globalForm.jsp renders the search text box along with radio buttons. On a search result page, it renders the search text box with a **Search** button along with a radio button for **New Search** and **Search Within**. See Figure 10-2 on page 200. Some of the OmniFind Discovery Edition Layout and Runtime APIs used in this JSP are:

- resultSet.getIsRenderSearchWithin()
- encodeURIComponent()

resultSet.getIsRenderSearchWithin()

As mentioned in Chapter 5, "Introducing OmniFind Discovery Edition APIs" on page 67, you use the *Query* class to do a simple search, and get the results of the search in a *QueryResult*. The QueryResult includes *ResultSet*.

resultSet.getIsRenderSearchWithin() returns a boolean variable representing whether the OmniFind Discovery Edition value for the **Render Search Within** flag is enabled or disabled. Based on the this boolean value, globalForm.jsp can render the appropriate radio button. See its usage in bold text in Example 10-2.

Example 10-2 Code snippet from globalForm.jsp showing the use of APIs

```
</script>
<form id="globalForm" name="globalForm" action="<%=actionName%>"
method="GET" onsubmit="JavaScript:doSubmit(); return false;">
   <input type="hidden" name="ip_server" value='<%=ip_server%>'>

   <!-- start ipSearchTable table in globalForm.jsp -->

   <table border="0" cellspacing="0" cellpadding="0"
id="ipSearchTable">
      <tr>
         <td>
            <span
class="ipEmphasis"><%=prose.getString("searchForStr")%></span>
            <input type="text" name="ip_row_text" class="ipSearchInput"
size="50">
            <input type="image" src="<c:out
value="${basedir}"/>/images/<%=prose.getString("searchButton")%>"
alt="Submit Form" class="ipSearchInput">
         </td>
            <%if((queryResultRqst != null) &&
(resultSet.getIsRenderSearchWithin() == true)){%>
               <td>
```

```
                <input type="radio" class="ipRadio"
name="ip_searchWithin" value="Off"
checked="true"><%=prose.getString("newSearchStr")%></input>
              </td>
              <td>
                <input type="radio" class="ipRadio"
name="ip_searchWithin" value="On">
<%=prose.getString("searchWithinStr")%></input>
              </td>
            <%}%>
          <td>
        </tr>
   </table>
   <%// end ipSearchTable %>
```

encodeURIComponent()

The OmniFind Discovery Edition search inquiry form is rendered as part of the code snippet shown in Example 10-2. The doSubmit() function from the code snippet calls *encodeURIComponent()* to encode the ip_row_text value entered in the search text box by users and assign it to the JSP document's globalForm's ip_text variable. Example 10-3 shows an example of the encodeURIComponent() API used in the doSubmit() function.

Example 10-3 doSubmit() function inside globalForm.jsp for Search Text HTML Form

```
function doSubmit(){
   try{
      window.focus();
      if(document.globalForm.ip_row_text.value != "")
      {
         document.globalForm.ip_text.value =
encodeURIComponent(document.globalForm.ip_row_text.value);
         document.globalForm.ip_row_text.value = "";
      }
      document.globalForm.submit();
   }
   catch(e){
      doPostSubmit();
   }
  }
```

10.1.3 APIs used in voiceOver.jsp

voiceOver.jsp is used to repeat the search actions that user does on the main search result page. The VoiceOver object is included in both the Runtime APIs as well as the Layout APIs. Some of the OmniFind Discovery Edition Layout APIs and Runtime APIs used in this JSP include:

- resultSet.getVoiceOver()
- voiceOver.getYouAsked()
- voiceOver.getYouAskedWithin()
- VoiceOver.Item
- voiceOver.getYouChanged()
- voiceOver.getStopWords()

resultSet.getVoiceOver()

Gets the voiceover details for this result set. This API returns a VoiceOver object which can be used to make additional API calls. This is part of the OmniFind Discovery Edition Runtime APIs.

voiceOver.getYouAsked()

Gets the *you asked* string, which contains the text that the user typed in the search box. This is part of the OmniFind Discovery Edition Layout APIs.

voiceOver.getYouAskedWithin()

Gets the *Search Within* strings which contain an array of all strings used within a series of consecutive searches (when the **Render Search Within** option is enabled and the user selected the **Search Within** option). This is part of the OmniFind Discovery Edition Layout APIs.

VoiceOver.Item

Contains data for a single new or removed constraint in the VoiceOver component. Its constructor takes in two parameters: a Constraint Object and a list of features.

voiceOver.getYouChanged()

Gets the *You clicked* entries data and returns an array of VoiceOver.Item objects containing the data for formatting the new constraints.

voiceOver.getStopWords()

Gets the stopwords. These are common words that are generally not reflective of the content on the page. The OmniFind Discovery Edition search engine ignores these words. Some stopwords include: *a*, *the*, and *of*. This OmniFind Discovery Edition Runtime API returns an array of a string containing the stopwords.

Example 10-4 shows how the VoiceOver.getYouAsked() Layout API can be used in your JSPs. Example 10-5 shows how the voiceover.getStopwords() Runtime API can be used in your JSPs.

Example 10-4 Use of getYouAsked() - Layout API

```
            <%// You asked %>
            <%if(!voiceOverRqst.getYouAsked().equals("")){%>
               <span 
class="ipEmphasis"><%=prose.getString("youAskedStr")%>
</span><%=ValueHandler.escapeHTML(voiceOverRqst.getYouAsked())%><br>
            <%}%>
```

Example 10-5 Use of getStopWords() - Runtime API

```
            <%Voiceover voiceover = resultSet.getVoiceover();%>
            <%// Stopwords %>
            <%
            String[] stopwords = voiceover.getStopwords();
            int stopwords_size = 0;
            if(stopwords != null && (stopwords_size = 
stopwords.length) > 0){%>
               <span 
class="ipEmphasis"><%=getStopwordsFormatting(stopwords, prose)%>
</span>
               <br>
            <%}//if getStopwords%>
```

10.1.4 APIs used in refineByList.jsp

refineByList.jsp is used to render features and values that can be used as additional search filtering mechanism. Some of the OmniFind Discovery Edition Layout APIs and Runtime APIs used in this JSP include:

- DrillDown
- TallyFeature
- resultSet.getDrillDownPlus()
- drillDown.getTallyFeatures()

DrillDown

Using OmniFind Discovery Edition, you can present a list of feature values for the user to select in order to narrow down their search. Examples of features include category, price, size, and color. When a user selects the feature values, OmniFind Discovery Edition can do a follow-up search for items that have the selected feature values. This is called *drill down*. The *DrillDown* class is used by the OmniFind Discovery Edition Runtime APIs.

TallyFeature

For some features, such as numbers or dates, a range of values are more appropriate as a search criterion than a single value. For those features, OmniFind Discovery Edition can allow the user to select a range of values as a search criterion rather than a single value. A feature for which the user can drill down or drill sideways using a range of values rather than a single value is called a *tally feature*.

To allow for tally features, you can use the *TallyFeature* class.

For more explanation on drill down and tally features, refer to 5.6, “Navigating search results” on page 89 and 6.5, “Implement refinement list JSP fragment” on page 106.

resultSet.getDrillDownPlus()

Gets the drill-down (also known as refine-by list) information for the resultSet object. This API returns an object of type DrillDown class.

drillDown.getTallyFeatures()

Gets the TallyFeature array for this DrillDown and returns an array of TallyFeatures.

Example 10-6 shows the usage of resultSet.getDrillDownPlus() and drillDown.getTallyFeatures() with bold text in the refineByList.jsp code snippet.

Example 10-6 Code snippet from refineByList.jsp to show use of APIs discussed

```
   // Set a feature that is printed in the tabs. It should ! be printed
in the refine by list
   String tabbedFeature = drillDownTabsRqst.getTabsFeatureId();%>
   <!-- start ipRefineTable table in refineByList.jsp -->
   <table cellspacing="0" cellpadding="1" id="ipRefineTable">
      <!--tr><td><br></td></tr-->
      <tr><td valign="top"
class="ipRefineByTitle"><b><%=prose.getString("refineByStr")%></b></td>
</tr>
      <!--tr><td valign="top"
class="sn_header"><b><%=prose.getString("refineByStr")%></b></td></tr--
>
      <%DrillDown drillDown = resultSet.getDrillDownPlus();

   for (int j = 0; j < drillDown.getTallyFeatures().length; j++) {
      TallyFeature currentFeature = drillDown.getTallyFeatures()[j];
      DrillDownList ddList = new DrillDownList(query, currentFeature,
true);
```

```
      boolean tdOn = false;
                     %>
                     <%// Checks if the user clicked the [more...] link.
Extract the expandDrillDown parameter from the URL %>
                     <%String renderRefineByMore =
request.getParameter("ip_expandDrillDown");
      int renderRefineFullNumColumns = 0;
      int actualValuesToShow = NUM_REFINEBY_PER_FEATURE <
ddList.getCount() ? NUM_REFINEBY_PER_FEATURE : ddList.getCount();
      boolean showLessLink = false;
      boolean showMoreLink = currentFeature.getIsMaxValuesExceeded();
      if((renderRefineByMore != null) && (renderRefineByMore.length() >
0) && currentFeature.getId().equals(renderRefineByMore)){%>
              <%// Sets the number of columns for RefineBy if [more..]
was clicked %>
         <%String renderRefineFullNumColumnsStr =
(String)currentFeature.getAttributes().get(new
String("render_refine_full_num_columns"));
         if (renderRefineFullNumColumnsStr != null)
            renderRefineFullNumColumns =
Integer.parseInt(renderRefineFullNumColumnsStr);%>
            <%// Holds the attributes.render_refine_full_page flag. %>
            <%Object renderRefineFullPageRqstObj =
currentFeature.getAttributes().get(new
String("render_refine_full_page"));
         if(renderRefineFullPageRqstObj != null)
            pageContext.setAttribute("renderRefineFullPageRqst",
renderRefineFullPageRqstObj, pageContext.REQUEST_SCOPE);

         showLessLink = true;
         actualValuesToShow = ddList.getCount();
      }
```

10.1.5 APIs used in drillSideWays.jsp

drillSideWays.jsp allows a user to change the criterion used to narrow down a search result. Some of the OmniFind Discovery Edition Layout APIs and Runtime APIs used in this JSP include:

- Breadcrumb.Item
- breadcrumbsRqst.getItems()

Breadcrumb.Item

Breadcrumbs are a user interface navigation technique, the purpose of which is to give users a way to keep track of their location within the search results. They provide links back to each previous page that the user navigated through in order to get to the current page. Breadcrumbs provide a trail for the user to follow back to the starting point of a Web site.

Breadcrumb.Item contains data for formatting of a single breadcrumb (DrillSideways) entry. The Breadcrumb data structure is optimized for access from JSP. The single instance of Breadcrumb is created by the QueryHandler and stored in the request-scope.

breadcrumbsRqst.getItems()

Gets all the breadcrumb items within the breadcrumbsRqst object. Returns an array of Breadcrumb.Item entries.

Example 10-6 shows the usage of the APIs in drillSideWays.jsp.

Example 10-7 Code snippet from drillSideWays.jsp to show use of APIs

```
      Breadcrumb.Item bcitems[] = breadcrumbsRqst.getItems();
      if(bcitems.length != 0){%>
         <tr valign="baseline">
         <td width="1" class="breadcrumb"><span
class="ipEmphasis"><%=prose.getString("forStr")%>:&nbsp;</span></td>
         <td class="ipleft">
         <%for (int j = 0; j < bcitems.length; j++){
            Breadcrumb.Item bcitem = bcitems[j];%>
            <!-- title -->
            <%if(bcitem.getLabel().length() > 0){
            %>
               <%=bcitem.getLabel()%>
            <%
            }
```

10.1.6 APIs used in mainResultsTable.3a.jsp

mainResultsTable.3a.jsp renders the products or items that match the search criteria in a M X N matrix format. Some of the OmniFind Discovery Edition Layout APIs and Runtime APIs used in this JSP include:

- resultSet.getTable()
- mainTable.getFeatureMetaDatas()
- mainTable.getFeatureRows()

resultSet.getTable()

Each resultSet contains a *table* which is a two-dimensional grid of rows and columns. Each row of the table represents one item from the collection, and each column represents one feature, so that each cell holds the value of one feature for one item.

resultSet.getTable() gets the table for this resultSet object and returns the Table Object containing the search results.

mainTable.getFeatureMetaDatas()

For each item from the resultSet, the table contains a *FeatureMetaData object* for each feature. This object contains metadata about the features.

mainTable.getFeatureMetaDatas() gets the feature metadata array for the mainTable object and returns an array of the FeatureMetaData objects.

mainTable.getFeatureRows()

For each item from the resultSet, the table has one *FeatureRow*, which contains one value for each feature.

mainTable.getFeatureRows() gets the array of feature rows for the mainTable object and returns an array of the FeatureRow objects.

Example 10-8 shows the usage of the APIs in the mainResultsTable.3a.jsp.

Example 10-8 Code snippet from mainResultsTable.3a.jsp to show use of APIs

```
<!-- render MainResults table -->
<!-- start ipResultsList table in mainResultsTable.3a.jsp -->
<table border="0" cellspacing="0" cellpadding="0" id="ipResultsList">
<%
Table mainTable = resultSet.getTable();
if (mainTable != null) {
   // If query returned results in main table
   FeatureMetaData[] metas = mainTable.getFeatureMetaDatas();
   FeatureRow[] rows = mainTable.getFeatureRows();

   //first row of 3
%>
   <tr id="ipResultsRowDelimiter">
<%
   int rowStatus;
   for (rowStatus = 0; rowStatus < rows.length; rowStatus++){
      FeatureRow row = rows[rowStatus];
      //populate the table
      featuresCount=1;
      if (rowStatus > 0 && rowStatus%ROWS_PER_PAGE_3A==0) {
%>
   </tr>
   <tr id="ipResultsRowDelimiter">
<%
      }
%>
```

10.1.7 APIs used in pagination.jsp

pagination.jsp is used in rendering page navigation for the main results table. Some of the OmniFind Discovery Edition Layout APIs and Runtime APIs used in this JSP include:

- resultSet.getPageContext().getTotalPages()
- resultSet.getPageContext().getCurrentPage()

resultSet.getPageContext().getTotalPages()

Gets the total number of pages of search results to be rendered, depending on the number of search results to be rendered per page.

resultSet.getPageContext().getCurrentPage()

Gets the current page with search results based on the page context.

Example 10-9 shows the usage of the APIs in the pagination.jsp code snippet.

Example 10-9 Code snippet from pagination.jsp to show use of APIs

```
if((resultSet.getTable() != null) && (resultSet.getPageContext() !=
null) && (resultSet.getPageContext().getTotalRows() > 0)){%>
   <!-- start ipResultPage table in pagination.jsp -->
   <table border="0" cellspacing="0" cellpadding="0"
class="ipResultPage">
      <tr><td colspan="2" class="ipHalflineDivider">&nbsp;</td></tr>
      <tr>
         <td class="ipleftregular"><span
class="ipEmphasis"><%=prose.getString("resultPageStr")%>&nbsp;

            <%// set startPage %>
               <%if(resultSet.getPageContext().getTotalPages() <= 10){
                  startPage=1;
               }else{
                 int tempStart =
resultSet.getPageContext().getCurrentPage()+1;
                    if(tempStart < 6){
                       startPage=1;
                    }else if((tempStart >= 6) && ((tempStart+4) <=
resultSet.getPageContext().getTotalPages())){
                       startPage=tempStart - 5;
                    }else if((tempStart >= 6) && ((tempStart+4) >
resultSet.getPageContext().getTotalPages())){

startPage=resultSet.getPageContext().getTotalPages()-9;
                    }

              }%>

           <%// set endPage %>
              <%if(resultSet.getPageContext().getTotalPages() <= 10){
                 endPage=resultSet.getPageContext().getTotalPages();
              }else{
                 int tempEnd =
resultSet.getPageContext().getCurrentPage()+1;
                    if((tempEnd+4) >=
resultSet.getPageContext().getTotalPages()){

endPage=resultSet.getPageContext().getTotalPages();
                    }else{
                       endPage=startPage+9;
                    }

              }%>
```

10.1.8 APIs used in viewByOptions.jsp

viewByOptions.jsp is used to render group-by, sort-by and relevance view options on the search result page. Some of the OmniFind Discovery Edition Layout APIs and Runtime APIs used in this JSP include:

- resultSet.getViewByOptions()
- resultSet.getSortByOptions()
- sortByOption.getDirection()

resultSet.getViewByOptions()

Gets the view-by also known as group-by options for this result set and returns an array of the FeatureMetaData objects.

resultSet.getSortByOptions()

Gets the sort-by options for this result set and returns an array of the SortFeature Objects.

sortByOption.getDirection()

Gets the sorting direction for the feature represented in sortByOption object.

See Example 10-10 and Example 10-11 for sample usage of the APIs in viewByOptions.jsp file.

Example 10-10 Code snippet from viewByOptions.jsp to show use of APIs

```
            <%if(resultSet.getViewByOptions().length > 0){%>
               <%// group by %>
               <span
class="DisplayOption"><%=prose.getString("groupByStr")%></span>&nbsp;
               <select id = "viewByOptions" class="DropDownContent"
onChange="javascript:updater('<%=resultSet.getUserData()%>',
'ip_viewBy', encodeURIComponent(this.options[selectedIndex].value))">
                  <%selectedIndex=-1;
                  for (int state = 0; state <
resultSet.getViewByOptions().length; state++){
                     FeatureMetaData i =
resultSet.getViewByOptions()[state];

                        if(resultSet.getViewBySelected() == state){%>
                           <option value="<%=i.getId()%>"
selected><%=i.getLabel()%></option>
                        <% selectedIndex=state;
                        }else{%>
                           <option value="<%=i.getId()%>"
><%=i.getLabel()%></option>
                        <%}

                  }
```

Example 10-11 Code snippet from viewByOptions.jsp to show use of APIs

```
<%// Sort By %>
   <%if(resultSet.getTable() != null){
      if(resultSet.getSortByOptions().length > 0){%>
         <td class="ipright">&nbsp;&nbsp;<span
class="DisplayOption"><%=prose.getString("sortByStr")%> </span>&nbsp;
         <select id="sortByOptions" class="DropDownContent"
onChange="javascript:updater('<%=resultSet.getUserData()%>',
'ip_sortBy', encodeURIComponent(this.options[selectedIndex].value))">
            <%selectedIndex=-1;
            for (int state = 0; state <
resultSet.getSortByOptions().length; state++){
               com.iphrase.runtime.query.result.SortFeature
sortByOption = resultSet.getSortByOptions()[state];
                  if(resultSet.getSortBySelected() == state){%>
                     <option
value="<%=sortByOption.getId()%>//<%=sortByOption.getDirection()%>"
selected><%=sortByOption.getLabel()%></option>
                     <%selectedIndex=state;
                  }else{%>
                     <option
value="<%=sortByOption.getId()%>//<%=sortByOption.getDirection()%>"
><%=sortByOption.getLabel()%></option>
                  <%}

         }
```

10.2 Out-of-the-box search result page

The integration solution, with WebSphere Commerce using OmniFind Discovery Edition, uses *Layout Editor*, an in-house proprietary tool developed for rapid search results rendering along with user interface prototyping returned by OmniFind Discovery Edition server.

One of the goals for this book is to illustrate how a WebSphere Commerce store can be customized. Commerce Module for OmniFind Discovery Edition comes with a set of existing JSPs that we can use to start customizing our sample application. They include the TabbedNav skins and the Commerce skins. The *TabbedNav skin* uses most up-to-date Runtime APIs and Layout APIs. In this book, we show you how you can customize this set of JSPs.

There are multiple reasons why we want to use the TabbedNav skin instead of the existing Layout Editor approach. Some reasons are as follows:

- Layout Editor is a home-grown tool that is mainly used for quick user interface prototyping. Layout Editor is useful for small to medium customers needing a quick deployment. TabbedNav skin is useful for medium to large customers with custom requirements and user experience requirements.
- With Layout Editor, although you have the flexibility to create the basic appearance and behavior for many user interfaces, this might not be exactly what you want for your application.
- To make modification of the codes that work with Layout Editor is not an easy task. It requires your understanding of how Layout Editor works, especially the internal data structure used to store different user interface elements.
- TabbedNav skin is simple to understand and does not rely on additional internal data structure for keeping the user interface elements.
- TabbedNav skin makes use of the latest Runtime APIs and Layout APIs, thus you can leverage the latest technologies and functions available in your customized code.

Open a browser and point it to the following URL:

```
http://<server_name>/webapp/wcs/stores/servlet/TopCategoriesDisplay?lan
gId=-1&storeId=10001&catalogId=10001
```

The Consumer-Direct Store using the current existing out-of-the-box integration solution looks as shown in Figure 10-3.

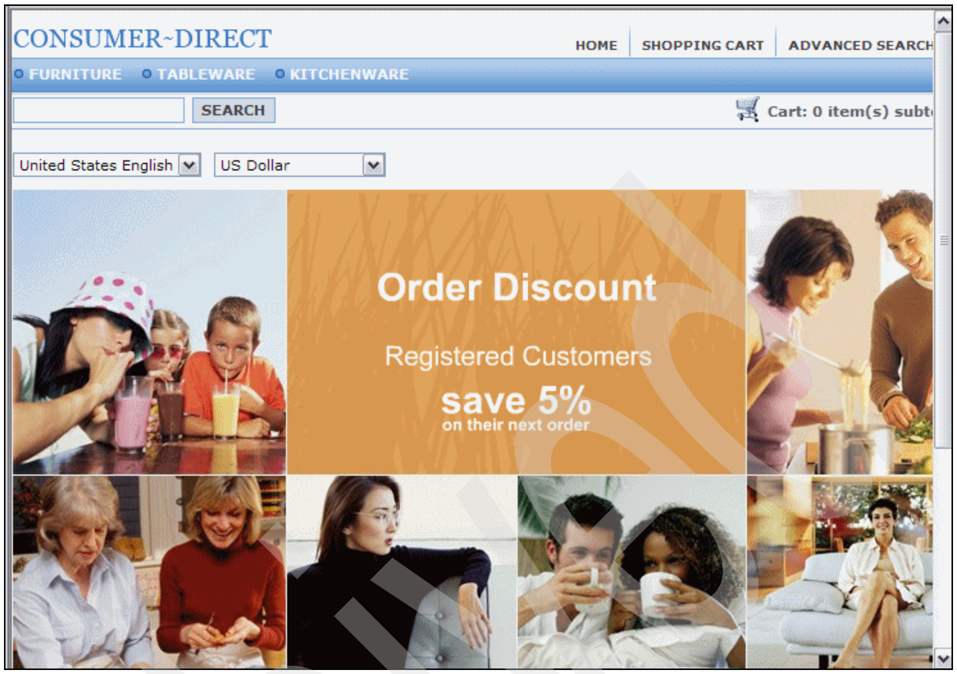
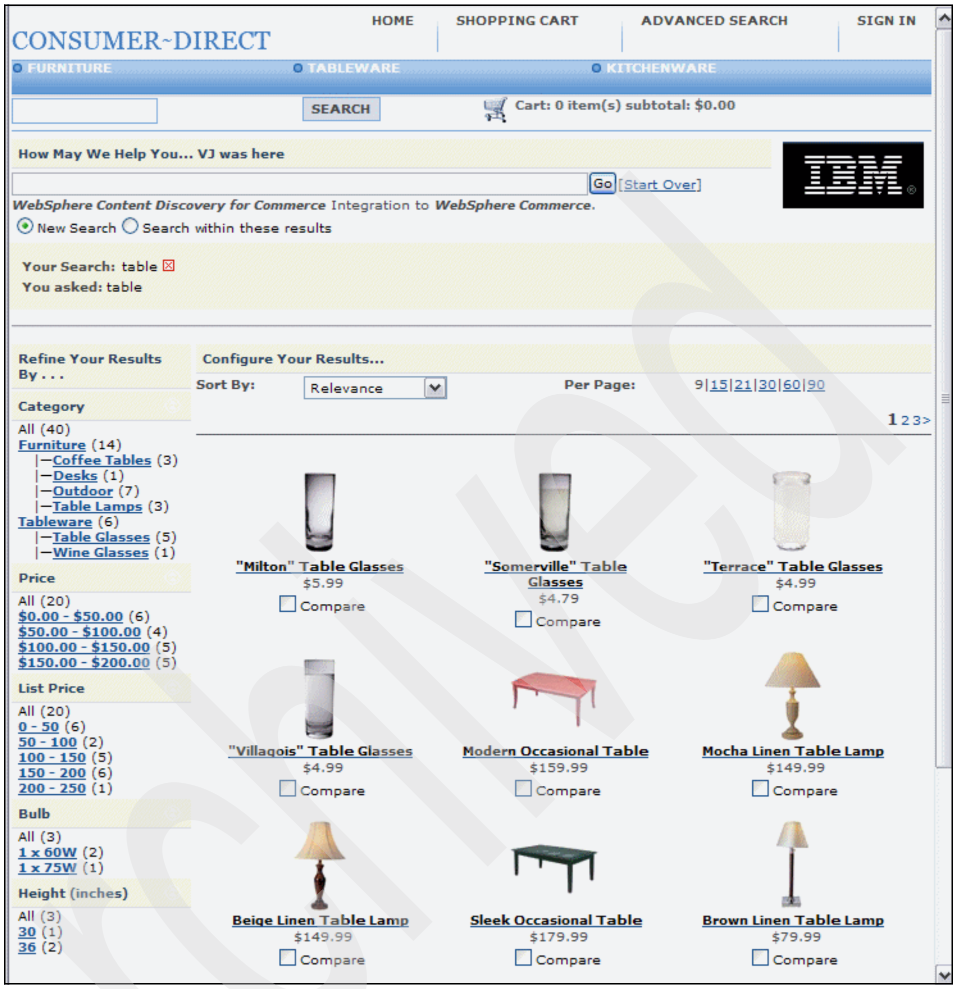


Figure 10-3 Consumer-Direct storefront page

Search for *tables* using the Store Front Page. The result page is rendered using the Layout Editor user interface prototype with the OmniFind Discovery Edition search engine on the back end. The search result page looks similar to the one in Figure 10-4.

Notice that the search result page is functional, and provides the rich search capabilities that OmniFind Discovery Edition offers. To change the appearance and behavior of the page, you go to the OmniFind Discovery Edition Management Console to make the appropriate changes. There will be times when the changes you want to make cannot be done through the Management Console's Layout Editor alone. In this case, we highly recommend starting your customizing with the TabbedNav skin.


HOME | SHOPPING CART | ADVANCED SEARCH | SIGN IN

CONSUMER~DIRECT

FURNITURE TABLEWARE KITCHENWARE

SEARCH

Cart: 0 item(s) subtotal: $0.00

How May We Help You... VJ was here

Go [Start Over]

WebSphere Content Discovery for Commerce Integration to *WebSphere Commerce*.

[✓ New Search] Search within these results

Your Search: table
You asked: table

Refine Your Results By . . .

Category
All (40)
Furniture (14)
|—Coffee Tables (3)
|—Desks (1)
|—Outdoor (7)
|—Table Lamps (3)
Tableware (6)
|—Table Glasses (5)
|—Wine Glasses (1)

Price
All (20)
$0.00 - $50.00 (6)
$50.00 - $100.00 (4)
$100.00 - $150.00 (5)
$150.00 - $200.00 (5)

List Price
All (20)
0 - 50 (6)
50 - 100 (2)
100 - 150 (5)
150 - 200 (6)
200 - 250 (1)

Bulb
All (3)
1 x 60W (2)
1 x 75W (1)

Height (inches)
All (3)
30 (1)
36 (2)

Configure Your Results...

Sort By: Relevance

Per Page: 9|15|21|30|60|90

1 2 3>

"Milton" Table Glasses $5.99 Compare

"Somerville" Table Glasses $4.79 Compare

"Terrace" Table Glasses $4.99 Compare

"Villagois" Table Glasses $4.99 Compare

Modern Occasional Table $159.99 Compare

Mocha Linen Table Lamp $149.99 Compare

Beige Linen Table Lamp $149.99 Compare

Sleek Occasional Table $179.99 Compare

Brown Linen Table Lamp $79.99 Compare

Figure 10-4 Out-of-the-box integrated solution search result page

10.3 Customization using TabbedNav skin

The existing solution integrates the WebSphere Commerce product with OmniFind Discovery Edition product. The integration solution uses the Layout Editor tool for the user interface prototype. As we mentioned earlier, it is more difficult for you to customize the store with the Layout Editor skin. The set of JSPs are hard to understand and hard to customize. In this section, we describe how you can change the Consumer-Direct store to use the TabbedNav skin instead of the Layout Editor skin. Once you switch to the TabbedNav skin, it would be easy for you to do future customization.

This goal of this section is to teach you how to make the WebSphere Commerce store search results work with the TabbedNav skin, covered in four parts:

1. Software packages and system environment requirements:

   We cover the prerequisites for software and system requirements before starting the customization effort.

2. Setting up TabbedNav directories and files in your store:

   We cover how to extract and construct a TabbedNav directory structure from the out-of-the-box deployment of Commerce Module for OmniFind Discovery Edition.

3. Struts configuration:

   We cover how you can dynamically change a *View* to a different *Model* in terms of Model-View-Controller (MVC) architecture. More specifically, we show you how to change CatalogSearchResultView from WCDSSearchApplication (the Layout Editor version) to WCDSTabbedNav (the TabbedNav version).

4. Code customization using WCDSTabbedNav: main.jsp:

   We cover the first set of customizations in building a search result page based on the TabbedNav skin.

At the end of this section, you should be able to see search results very much similar to Figure 10-4, with an important difference of the underlying search rendering technology. That is, the rendering is done by the TabbedNav skin instead of the Layout Editor skin.

Some of the benefits of using the TabbedNav skin are:

- The TabbedNav skin come as standard with any version of OmniFind Discovery Edition product.
- The TabbedNav skin uses most up-to-date Runtime APIs and Layout APIs.
- The TabbedNav skin is simple to understand and give a very good starting point for customizing the search experience.

10.3.1 Software packages and system environment requirement

Before you proceed, make sure that you follow Chapter 2, “Environment setup” on page 19 and successfully set up a Consumer-Direct starter store in the integrated environment that uses WebSphere Commerce and OmniFind Discovery Edition. You should be able to see the Consumer-Direct home page as shown in Figure 10-3 on page 215. If you search for tables (by entering tables at the search box), the search result page should look something similar to Figure 10-4 on page 216.

This step ensures that WebSphere Commerce and the OmniFind Discovery Edition products are installed correctly and the integration among these products is also performed successfully.

Set up two system environment variables as follows if they have not been set:

- `WCS_HOME=C:\Program%20Files\IBM\WebSphere\AppServer\profiles\demo\installedApps\WC_demo_cell\WC_demo.ear`
- `IPHRASE_HOME=C:\IBM\IBM WebSphere Content Discovery Server 8.3`

When the integration is installed, a set of JSP files under WebSphere Commerce directory should be updated to work with the OmniFind Discovery Edition search engine.

Since we create the Consumer-Direct store before the integration, the JSP files associated with the store should be updated. The JSP files associated with this store are located under the <WCS_HOME>\Stores.war\ConsumerDirect directory as follows:

```
ShoppingArea\CatalogSection\SearchSubsection\AdvancedCatalogSearchForm.
jsp
include\styles\style1\CachedHeaderDisplay.jsp
include\styles\style1\CachedSidebarDisplay.jsp
include\styles\style1\ContentContainerTop.jsp
include\styles\style1\SidebarDisplay.jsp
include\styles\style2\CachedHeaderDisplay.jsp
include\styles\style2\ContentContainerTop.jsp
include\styles\style2\SidebarDisplay.jsp
```

In addition to the foregoing JSP files, the following XML file related to Struts configuration is also updated for our Consumer-Direct store or any stores that you created before you perform the integration steps:

```
<WCS_HOME>\Stores.war\WEB-INF\struts-config-ext.xml
```

10.3.2 Setting up TabbedNav directories and files in your store

We have to set up the necessary directories in our sample store to contain the standard TabbedNav files.

Use the following steps to get a working skeleton of TabbedNav skin into the WebSphere Commerce Consumer-Direct Store (if you use another store, substitute our store directory with your store directory):

1. Create a new WCDSTabbedNav directory:

   ```
   <WCS_HOME>\Stores.war\ConsumerDirect\WCDSTabbedNav
   ```

2. Create two sub-directories under the newly created directory:

   ```
   languages
   layout-tabbedNav
   ```

3. Copy all the JSP files from:

   ```
   <IPHRASE_HOME>\webapps\jspapps\layout-tabbednav\jsp
   ```

   to:

   ```
   <WCS_HOME>\Stores.war\ConsumerDirect\WCDSTabbedNav
   ```

4. Copy the following two directories:

   ```
   <IPHRASE_HOME>\webapps\jspapps\layout-tabbednav\CSS
   <IPHRASE_HOME>\webapps\jspapps\layout-tabbednav\images
   ```

   to:

   ```
   <WCS_HOME>\Stores.war\ConsumerDirect\WCDSTabbedNav\layout-tabbednav
   ```

Attention: This directory structure changes when we fix the image files to be displayed on the search result page as explained in “Final modifications: Images and Search box” on page 258.

5. Copy all files from:

   ```
   <IPHRASE_HOME>\webapps\jspapps\layout-tabbednav\jsp\languages
   ```

   to:

   ```
   <WCS_HOME>\Stores.war\ConsumerDirect\WCDSTabbedNav\languages
   ```

10.3.3 Struts configuration

Struts is a framework for creating Java Web applications. Web applications create dynamic response and differ from conventional Web sites, which deliver only static pages.

A typical Web application has three layers:

- Presentation layer
- Business logic layer
- Database content source layer

The presentation layer code interacts with business logic layer’s EJB™ beans or regular Java beans usually over HTTP connection using servlet. The EJB or regular Java beans in the business logic layer then interact with single or multiple database content sources at the back end to dynamically construct responses.

Web applications based on JavaServer Pages or JSP file technology can sometimes be very hard to maintain and understand, especially because the database code, page design code, and control flow code can all reside in the same JSP file.

Model-View-Controller (MVC) architecture separates the software application based on the functionality. The *Model* represents the business or database code. The *View* represents the page design code. The *Controller* represents the navigational code. The Struts framework is designed to help developers create Web applications that utilize the MVC architecture.

The framework provides three key components:

- *Request* handler provided by the application developer that is mapped to a standard URI.
- *Response* handler that transfers control to another resource which completes the response.
- *Tag library* that helps developers to create interactive form-based applications with server pages.

In this section, we show you how to dynamically change a *View* to point to a different *Model*. We change the *CatalogSearchResultView* (that is used by WebSphere Commerce) to request and render search results from *WCDSSearchApplication* to *WCDSTabbedNav*.

The following steps show you how to update the Struts configuration file to use WCDSTabbedNav:

1. Open the struts-config-ext.xml file under the following directory for edit:

   ```
   <WCS_HOME>\Stores.war\WEB-INF
   ```

2. Find the code in the file as shown in Example 10-12.

Note: If you followed our steps in Chapter 2, “Environment setup” on page 19 and created the Consumer-Direct store first, the store ID should be 10001. If you created more than one store for your integrated environment, look for a similar line with a number that identifies your store.

Example 10-12 struts-config-ext.xml file snippet: Original

```
<forward className="com.ibm.commerce.struts.ECActionForward"
name="CatalogSearchResultView/10001"
path="WCDSSearchApplication\main.jsp">
```

3. Replace the line as shown in Example 10-13.

Example 10-13 struts-config-ext.xml file snippet: Changes

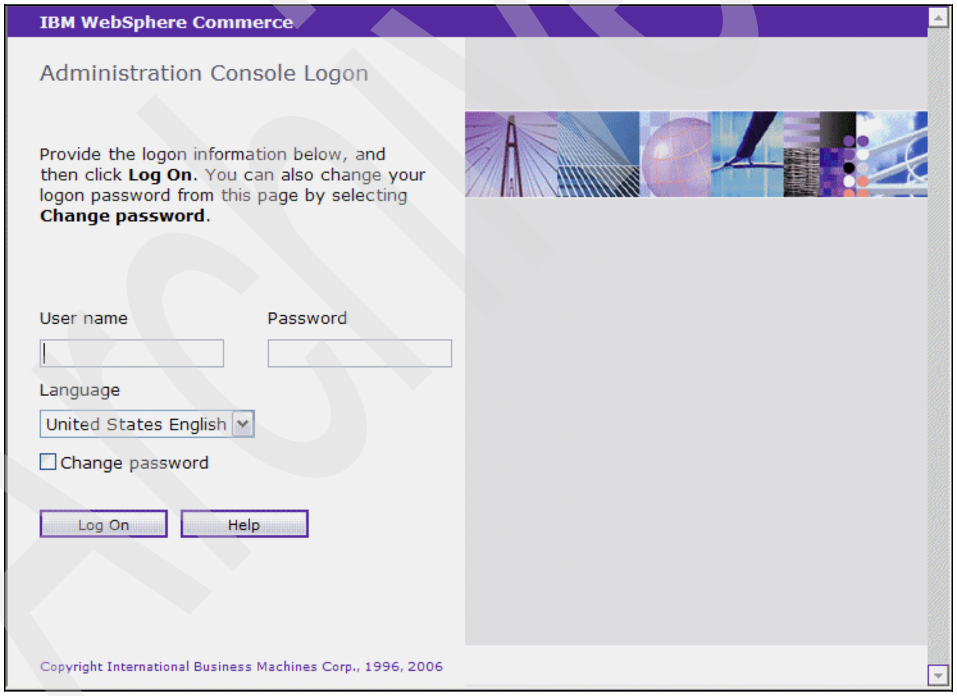
```
<forward className="com.ibm.commerce.struts.ECActionForward"
name="CatalogSearchResultView/10001" path="WCDSTabbedNav\main.jsp">
```

4. Reload the Struts configuration using WebSphere Commerce Administration Console for the foregoing changes to take affect.

Reloading the Struts configuration

If you restart your WebSphere Commerce application, the system automatically reload the latest Struts configuration file. This might take some time. Alternatively, you can use the following steps to restart Struts configuration:

1. Launch the WebSphere Commerce Administration Console by selecting **Start** → **Programs** → **IBM WebSphere** → **Commerce Server v6.0** → **Administration Console**. See Figure 10-5.


IBM WebSphere Commerce

Administration Console Logon

Provide the logon information below, and then click **Log On**. You can also change your logon password from this page by selecting **Change password**.

User name

Password

Language

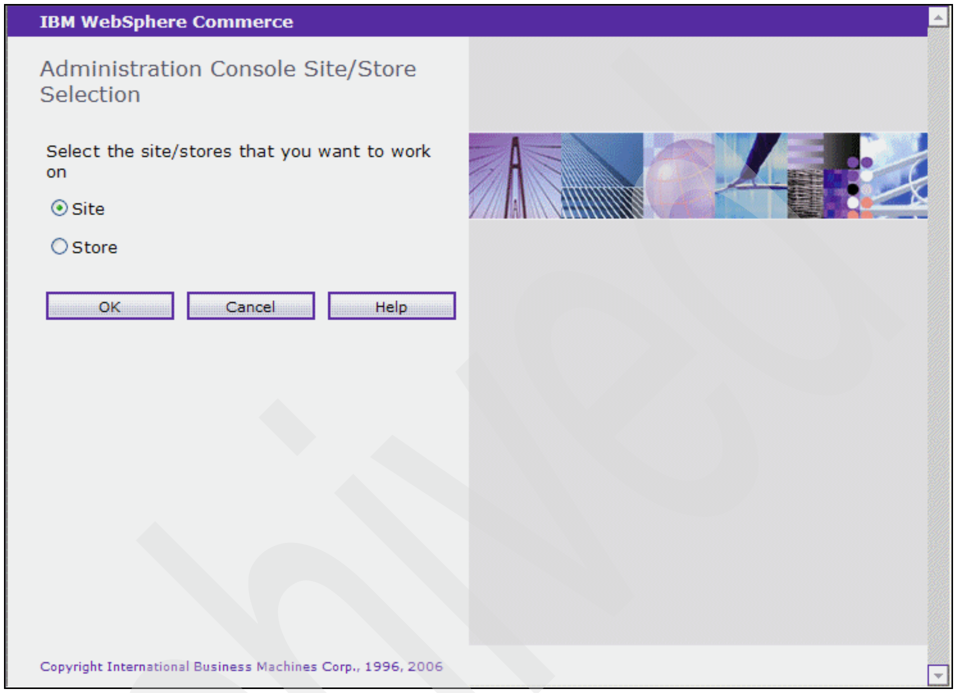
United States English

☐ Change password

Copyright International Business Machines Corp., 1996, 2006

Figure 10-5 WebSphere Commerce: Administration Console login page

2. Enter the user name and password. In our sample store, we enter wcsadmin and password1 to log into the Administration Console.
3. Select **Site** in the Selection Page. See Figure 10-6.


IBM WebSphere Commerce

Administration Console Site/Store Selection

Select the site/stores that you want to work on

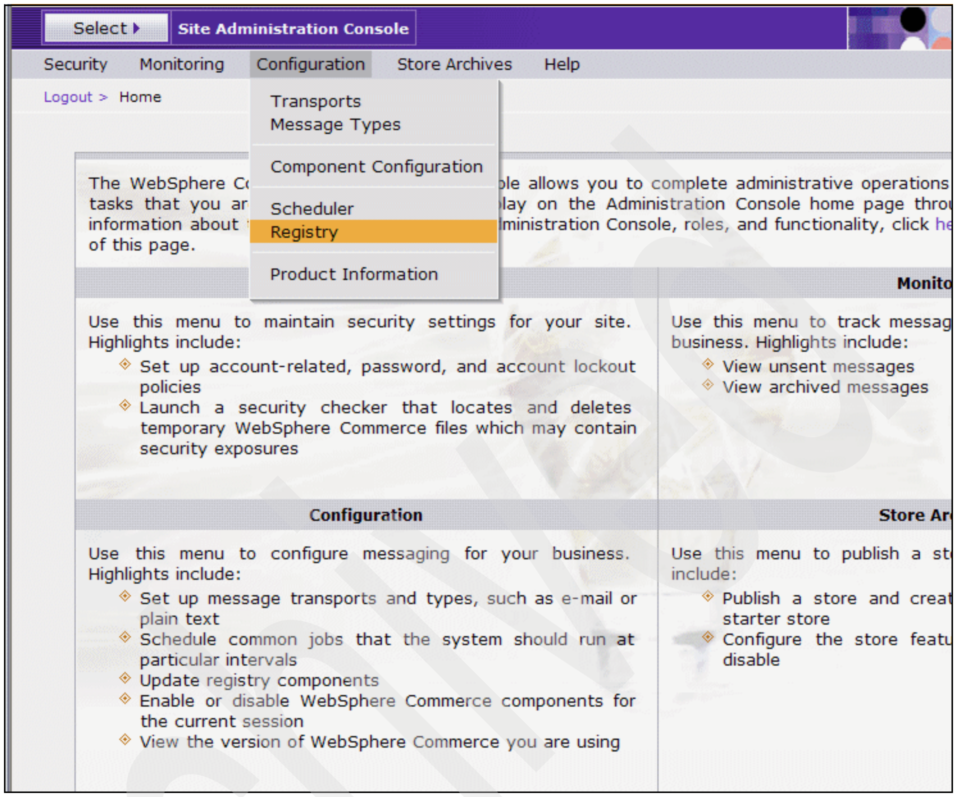
[✓ Site]

Store

Copyright International Business Machines Corp., 1996, 2006

Figure 10-6 WebSphere Commerce: Administration Console SiteStore selection

4. Select **Configuration** → **Registry**. See Figure 10-7.



The WebSphere C ble allows you to complete administrative operations tasks that you ar lay on the Administration Console home page throu information about lministration Console, roles, and functionality, click he of this page.

Use this menu to maintain security settings for your site. Highlights include:

- Set up account-related, password, and account lockout policies
- Launch a security checker that locates and deletes temporary WebSphere Commerce files which may contain security exposures

Use this menu to track messag business. Highlights include:

- View unsent messages
- View archived messages

Use this menu to configure messaging for your business. Highlights include:

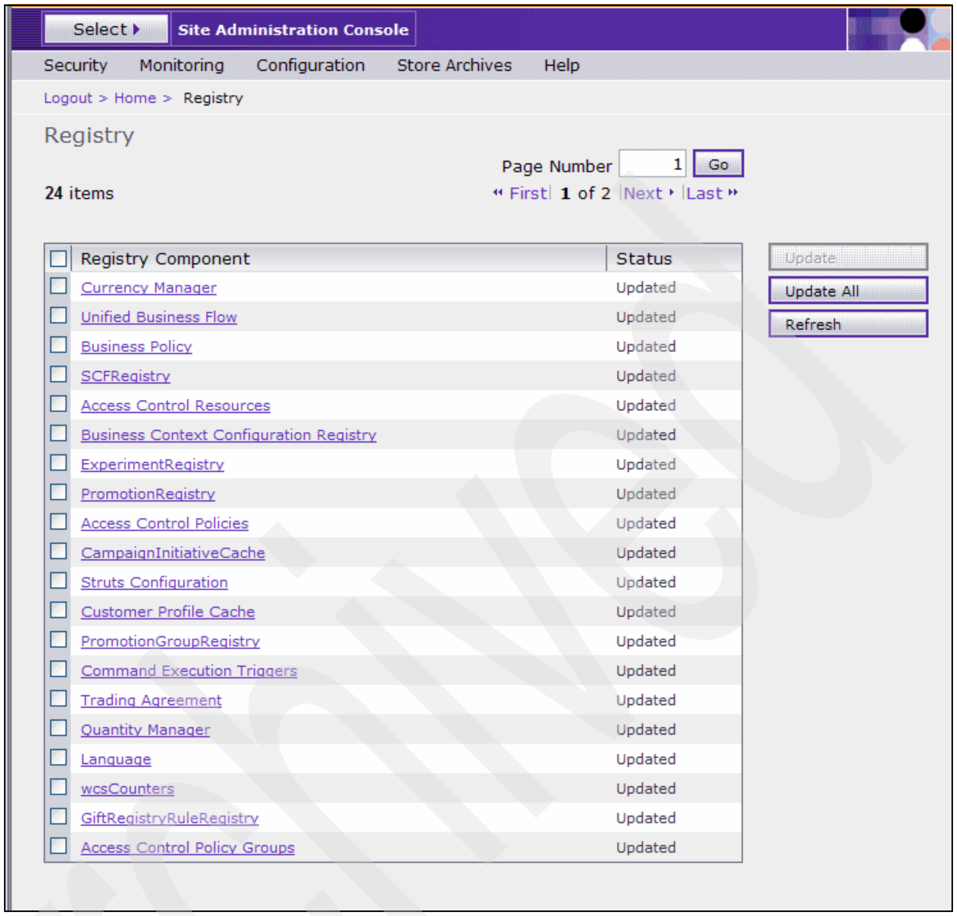
- Set up message transports and types, such as e-mail or plain text
- Schedule common jobs that the system should run at particular intervals
- Update registry components
- Enable or disable WebSphere Commerce components for the current session
- View the version of WebSphere Commerce you are using

Use this menu to publish a st include:

- Publish a store and creat starter store
- Configure the store featu disable

Figure 10-7 WebSphere Commerce: Administration Console registry selection

5. Select **Struts Configuration** from the registry list and click **Update**. See Figure 10-8. Wait for a minute and then click **Refresh** to see if the Struts configuration has been refreshed.



	Registry Component	Status
☐	Currency Manager	Updated
☐	Unified Business Flow	Updated
☐	Business Policy	Updated
☐	SCFRegistry	Updated
☐	Access Control Resources	Updated
☐	Business Context Configuration Registry	Updated
☐	ExperimentRegistry	Updated
☐	PromotionRegistry	Updated
☐	Access Control Policies	Updated
☐	CampaignInitiativeCache	Updated
☐	Struts Configuration	Updated
☐	Customer Profile Cache	Updated
☐	PromotionGroupRegistry	Updated
☐	Command Execution Triggers	Updated
☐	Trading Agreement	Updated
☐	Quantity Manager	Updated
☐	Language	Updated
☐	wcsCounters	Updated
☐	GiftRegistryRuleRegistry	Updated
☐	Access Control Policy Groups	Updated

Figure 10-8 WebSphere Commerce: Administration console update registry

Tip: If you do not wait for sufficient amount of time as specified or if you click the **Refresh** button too early, it might result into an exception error being thrown. You can ignore the exception message and click the browser **Refresh** button to reload the page again.

From this point onwards, the WebSphere Commerce starter store, Consumer-Direct, starts to point to the WCDSTabbedNav JSP file for search results instead of the search results provided by the WCDSSearchApplication JSP file.

These steps are useful in your customization, as they provide a quick and easy way to dynamically change the search results server at the back end without bringing the whole WebSphere Commerce Server down. You can change back and forth the parameters in the struts-ext-config.xml during your development and customization efforts.

10.3.4 Code customization using WCDSTabbedNav: main.jsp

Once the Struts configuration changes are successfully applied using the steps described in section 10.3.3, “Struts configuration” on page 219, all places in the WebSphere Commerce Consumer-Direct store’s code that use SearchCatalogView would now be directed to the program control to main.jsp under the *WCDSTabbedNav* directory.

Open a browser and point it to the following URL:

```
http://<server_name>/webapp/wcs/stores/servlet/TopCategoriesDisplay?lan
gId=-1&storeId=10001&catalogId=10001
```

The Consumer-Direct store appears as shown in Figure 10-3 on page 215.

Perform a search for *tables* within this store. A Web page similar to Figure 10-9 appears.

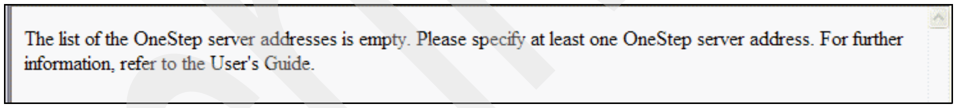

The list of the OneStep server addresses is empty. Please specify at least one OneStep server address. For further information, refer to the User's Guide.

Figure 10-9 WCDSTabbedNav server address empty error page

Notice that the new page does not look like the default search result page shown in Figure 10-4 on page 216. Let us try to understand the components of *main.jsp* and make the appropriate changes.

main.jsp code is responsible for rendering the search result page and for providing navigational controls along with the rendered search results. It calls a number of JSP files as listed in Table 10-1.

Table 10-1 JSP files that are called by main.jsp

JSP file	File description
taglibdecl.jsp	Contains declarations of tag libraries used by the Web search application.
config.jsp	Contains configuration templates.

JSP file	File description
globalForm.jsp	Contains JavaScript functions, form controls, query input box, the **New Search** and **Search Within** radio buttons.
voiceover.jsp	Contains code to format and render the voiceover component in the search result page.
tabbedDrillDown.jsp	Contains code to format and render the tabs display.
refineByList.jsp	Contains code to format and render the refinement options.
drillSideways.jsp	Contains code to format and render the breadcrumb component.
directAnswer.jsp	Contains code to control the display of direct answers (if any exist).
pageContext.jsp	Contains code to control the format of the range of results.
mainResultsTable.jsp	Contains code to control the main results table rendering.
pagination.jsp	Determines the format of the page navigation controls.
viewByOptions.jsp	Displays the following controls: ► group by ► sort by ► summarize by ► auto summarize ► (results) per page
sidebars.jsp	Displays business rules results (if any exist). We do not use this in our book.

Refer to Figure 10-1 on page 199 and Figure 10-2 on page 200 for graphical representations of the search result page and its associated JSP files.

To modified the code so that Figure 10-9 on page 225 looks like Figure 10-4, we leverage the existing code that come along with the integration solution. The code, including JSPF code fragments and wrappers for the integration solution, can be found under the application directory.

Copy the files under the application directory from WCDSSearchApplication to WCDSTabbedNav area. Specifically, copy from:

```
<WCS_HOME>\Stores.war\ConsumerDirect\WCDSSearchApplication\application
```

to:

```
<WCS_HOME>\Stores.war\ConsumerDirect\WCDSTabbedNav\application
```

Customizing main.jsp

We have to modify main.jsp such that WCDSTabbedNav returns the correct search result set. This includes the following steps:

1. Comment out the page encoding code.
2. Add application\jstl.jspf as an include directive to main.jsp.
3. Add application\resourceBundle.jspf as an include directive to main.jsp.
4. Add application\mainInit.jspf to main.jsp.
5. Set BaseDir parameter value.
6. Add application\preInitializeWithPageContext.jspf.
7. Add application\preQuery.jspf.

Step 1. Commenting out the page encoding code

Locate and remove the pageEncoding part of the code in main.jsp. Because WebSphere Commerce already uses UTF-8 page encoding, OmniFind Discovery Edition tries to override the existing page encoding with ISO-8859-1. This creates ambiguity and causes exceptions to be thrown. We comment out this part of the code by doing as shown in Example 10-14.

Example 10-14 Remove pageEncoding directive from main.jsp

```
<%/*
VG: removed
@page pageEncoding="ISO-8859-1" contentType="text/html;
charset=ISO-8859-1"
*/%>
```

Important: Make sure that you search for pageEncoding across *all* JSP files under WCDSTabbedNav and comment out this line of code. Special attention should be paid for new code or JSP that will be inserted into the current code base in the future. If there are any pageEncoding commands, they should be commented out.

Step 2. Adding application\jstl.jspf as include directive to main.jsp

The jstl.jspf code fragment lists the standard JSTL core and fmt taglibs and the proprietary wcbase taglib directives used throughout the user interface design.

Add the include directive as follows to *main.jsp* just after the include directive of taglibdecl.jspf:

```
<%@ jsp include file=”application\jstl.jspf”%>
```

The contents of application\jstl.jspf are shown in Example 10-15.

Example 10-15 application\jstl.jspf

```
<%@ taglib uri="http:java.sun.comjspjstlcore" prefix="c" %>
<%@ taglib uri="http:java.sun.comjspjstlfmt" prefix="fmt" %>
<%@ taglib uri="http:commerce.ibm.combase" prefix="wcbase" %>
<%@ taglib uri="flow.tld" prefix="flow" %>
```

Step 3. Adding application\resourceBundle.jspf to main.jsp

Resource bundle is a property file called WCDSTabbedNav.properties. It resides under <*WCS_HOME*>\Stores.war\WEB-INF\classes.

Copy WCDSSearchApplication.properties and rename it to WCDSTabbedNav.properties. This properties file contains information such as OmniFind Discovery Edition server address, VoiceOver Parameters, and BreadCrumb Parameters.

Add the include directive to *main.jsp* just after the include directive of jstl.jspf:

```
<%@ jsp include file=”application\resourceBundle.jspf”%>
```

Modify the contents of application\resourceBundle.jspf as in Example 10-16. Make sure the resource bundle points to WCDSTabbedNav.

Example 10-16 application\resourceBundle.jspf

```
<%
java.util.ResourceBundle rb =
java.util.ResourceBundle.getBundle("WCDSTabbedNav");
%>
```

Step 4. Adding application\mainInit.jspf to main.jsp

mainInit.jspf provides code for initializing list of explicit parameters that are passed along and are mapped one to one, from WebSphere Commerce to OmniFind Discovery Edition in the HTTPRequest Object. It also set up the actionpage JSTL variables. The parameters that are passed along with the mapping (WebSphere Commerce → OmniFind Discovery Edition) are:

- ip_text → ip_text
- catalogId → ip_catalogId
- ip_categoryId → ip_categoryId
- ip_requestUri → ip_requestUri

Add the include directive as follows to *main.jsp* just after the include directive of resourceBundle.jspf:

```
<%@ jsp include file="application\mainInit.jspf"%>
```

Change the contents of application\mainInit.jspf is as shown in Example 10-17:

- Replace all occurrences of WCDSSearchApplication with WCDSTabbedNav.
- Add additional check for empty string values.

Example 10-17 application\mainInit.jspf

```
<c:set var="basedir" value ="${jspStoreDir}WCDSTabbedNav"
scope="request" >
<c:forTokens items="${requestScope.requestURIPath}" delims=""
var="URLtoken">
  <c:set var="actionpage" value="${URLtoken}" scope="request">
<c:forTokens>

<%  get from ip_param if it exists, else WCParam... bec. it may be
encrypted by WCS %>

<c:set var="ip_text" value ="" scope="request" >
<c:if test="${! empty WCParam.ip_text}">
  <c:set var="ip_text" value ="${WCParam.ip_text}" scope="request" >
<c:if>
<c:if test="${! empty param.ip_text}">
  <c:set var="ip_text" value ="${param.ip_text}" scope="request" >
<c:if>

<%  mirror into session if it exists %>
<c:if test="${! empty WCParam.catalogId}">
  <!-- VG non null catalog  Id -->
   <c:set var="ip_catalogId" value="${WCParam.catalogId}"
scope="session">
<c:if>
<c:if test="${! empty WCParam.ip_categoryId}">
  <!-- VG non null ip_category  Id -->
   <c:set var="ip_categoryId" value="${WCParam.ip_categoryId}"
scope="session">
<c:if>
<c:if test="${! empty WCParam.ip_requestUri}">
     <!-- VG non null ip_requestUri -->
   <c:set var="ip_requestUri" value="${WCParam.ip_requestUri}"
scope="session">
<c:if>
```

Step 5. Setting basedir parameter value

Setting the basedir parameter value is important because all image files references are made absolute to the value set for basedir.

Add basedir related lines to main.jsp to set basedir. See Example 10-18.

Example 10-18 Set basedir in main.jsp after the mainInit include

```
<c:if test='${basedir == null}'>
<c:set var="basedir" value ="." scope="request" />
</c:if>
```

Step 6. Adding application\preInitializeWithPageContext.jspf

preInitializeWithPageContext.jspf handles initialization of the OmniFind Discovery Edition server IP address which it reads from the resource bundle file.

Locate and find the code where queryHandler.intializeWithPageContext() is called in main.jsp. Add the following include directive to main.jsp:

```
<%@ jsp include file="application\preInitializeWithPageContext.jspf"%>
```

The contents of the preInitializeWithPageContext.jspf are shown in Example 10-19.

Example 10-19 application\preInitializeWithPageContext.jspf

```
<%
// we want a bigger default
queryHandler.setTimeout(timeout != null ? new
Float(timeout).floatValue() : (float)30);

// servers property
String servers = rb.getString("servers");
if(servers != null && servers.length() > 0){
    StringTokenizer st = new StringTokenizer(servers, ";");
    while (st.hasMoreTokens()) {
      queryHandler.addServerAddress(st.nextToken());
    }
} else {
    queryHandler.addServerAddress("127.0.0.1:8777");
}
%>
```

Step 7. Adding application\preQuery.jspf

WebSphere Commerce supports Contract Entitlements, also known as Trading Agreements, which limit buyers to a subset of catalog. This structure can be enforced based on individual login, organization membership, and other variables. WebSphere Commerce provides a list of trading agreements, which the current user is entitled to purchase. For each trading agreement, WebSphere Commerce provides:

- Included product sets (all others are excluded)
- Excluded product sets (all others are included)
- Included and excluded product sets (included sets less excluded sets)
- Blank (everything is included)

Locate and find the code where queryHandler.execute() is called in main.jsp. Add the include directive to *main.jsp* just before this code:

```
<%@ jsp include file="application\preQuery.jspf"%>
```

preQuery.jspf takes the set union of the products allowed by each trading agreement, performs a store ID filtering, generates the appropriate constraints, and sets up the queryHandler constraints. For example, to set up wcsrequesturi and wcscatalogid with session attributes, you can use code as shown in Example 10-20.

Example 10-20 Code snippet from preQuery.jspf

```
/* session info for custom query parameters */
  Object o;
  o = session.getAttribute("ip_requestUri");
  if(o != null) {
   String stringVal = o.toString();
   if(!stringVal.equals("")) {
     queryHandler.getRuntimeQuery().setData("wcsrequesturi",stringVal);
   }
  }
  o = session.getAttribute("ip_catalogId");
  if(o != null) {
   String stringVal = o.toString();
   if(!stringVal.equals("")) {
     queryHandler.getRuntimeQuery().setData("wcscatalogid",stringVal);
   }
  }
```

In addition to setting up the required runtime query parameters, preQuery.jspf also construct a union of constraints based on trade agreement parameters if they are passed along. Constraints are constructed by prefixing the productID by keyword productset and suffixing it by keyword wcs. A sample code snippet is shown in Example 10-21.

Example 10-21 Concatenating TradingAgreements and queryHandler

```
<jsp:useBean id="sdb"
class="com.ibm.commerce.common.beans.StoreDataBean" scope="request" >
<% com.ibm.commerce.beans.DataBeanManager.activate(sdb, request); %>
</jsp:useBean>
<%@include file="CurrentOrderContext.jspf"%>

<%
  /* special for WCS */
  /* add in productset filter */
  java.util.Vector v = null;
  com.ibm.commerce.command.CommandContext commandContext =
(com.ibm.commerce.command.CommandContext)
request.getAttribute(com.ibm.commerce.server.ECConstants.EC_COMMANDCONT
EXT);
  com.ibm.commerce.contract.objects.TradingAgreementAccessBean []
tradingAgreements = commandContext.getCurrentTradingAgreements();
  com.ibm.commerce.contract.util.ProductSetSelection
productSetSelection = new
com.ibm.commerce.contract.util.ProductSetSelection(tradingAgreements,
commandContext.getUserId());
  boolean isAll = false;
  String allConstraints = null;
  for (int i = 0; i < tradingAgreements.length; i++) {
     boolean blank = true;
     String constraint = null;
     v =
productSetSelection.getTradingAgreementInclusionProductSetIds()[i];
     if (v != null) {
       blank = false;
       for(int j=0; j<v.size();j++) {
    String s = "productset" + v.get(j).toString() + "wcs";
         if(constraint != null) {
           constraint = constraint + " " + s;
         } else {
           constraint = s;
         }

       }
     }
     v =
productSetSelection.getTradingAgreementExclusionProductSetIds()[i];
     if (v != null) {
       blank = false;
```

```
          for(int j=0; j<v.size();j++) {
        String s = "productset" + v.get(j).toString() + "wcs";
             if(constraint != null) {
               constraint = constraint + " -" + s;
             } else {
               constraint = "-" + s;
             }
          }
        }
        if (blank) {
          isAll = true;
          break;
        }
        if(allConstraints != null) {
          allConstraints = allConstraints + "." + constraint;
        } else {
          allConstraints = constraint;
        }
      }
      if(!isAll) {

queryHandler.getRuntimeQuery().setData("wcsproductconstraints",allConst
raints);
        out.println("<!-- WCS preQuery.jspf ProductSet constraints from
TradingAgreements: " + allConstraints + " -->");
      } else {
        out.println("<!-- WCS preQuery.jspf ProductSet constraints from
TradingAgreements: none -->");
      }
      String storeId = commandContext.getStoreId().toString();
      out.println("<!-- WCS preQuery.jspf StoreID: " + storeId + " -->");

      queryHandler.getRuntimeQuery().setData("wcsstoreid",storeId);

      Object ipTextObj = request.getAttribute("ip_text");
      if(ipTextObj != null) {
        String ipText = ipTextObj.toString();
        queryHandler.getRuntimeQuery().setText(ipText);
      }
```

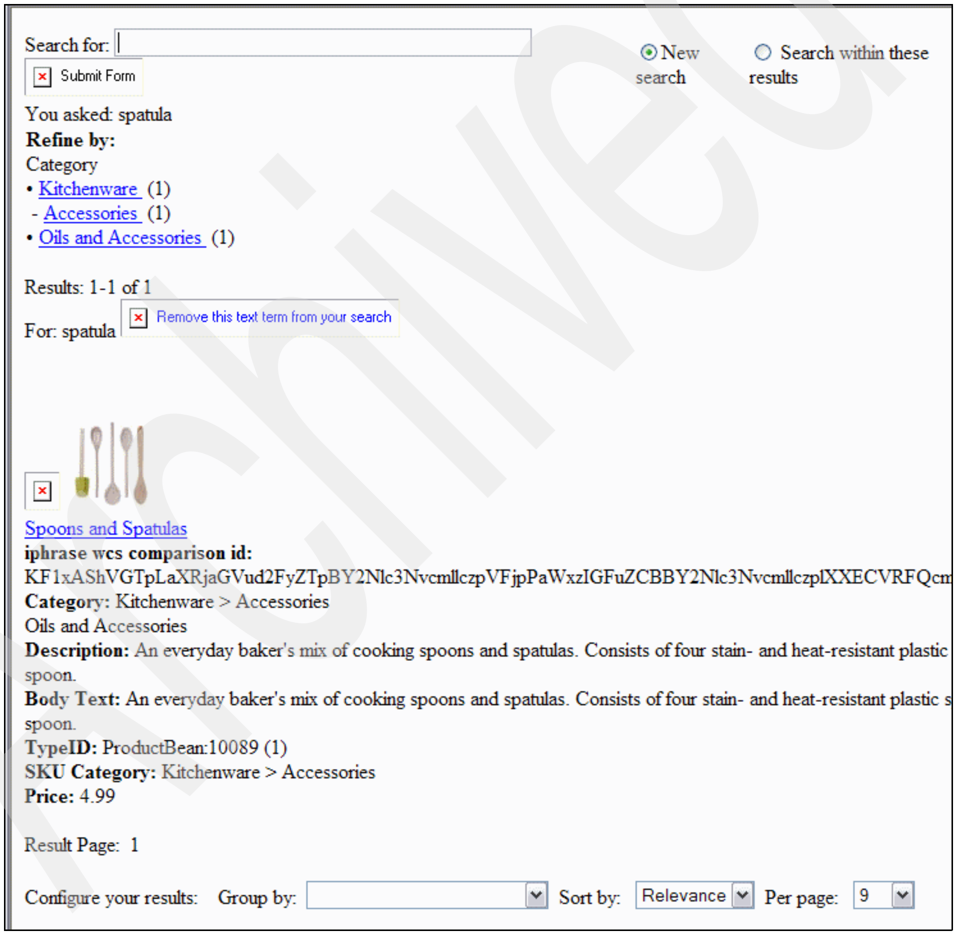
Tip: Whenever you are changing anything in a JSPF file, make sure that you re-save the JSP file that includes the JSPF to see the effect of the changes. Re-saving the JSP file forces the browser to recompile the JSP file.

At this point, we are finished with our first set of customization to main.jsp. After performing all of the foregoing steps, we take a look at a TabbedNav skin.

Open a browser and point it to the following URL:

```
http://<server_name>/webapp/wcs/stores/servlet/TopCategoriesDisplay?lan
gId=-1&storeId=10001&catalogId=10001
```

You should see a Web page that looks similar to Figure 10-10. This has the basic appearance and behavior of TabbedNav skins. We discuss the components that construct this appearance, and the different ways we can use the TabbedNav skin in the following sections.


Search for:
Submit Form
New search
Search within these results

You asked: spatula
Refine by:
Category
• Kitchenware (1)
- Accessories (1)
• Oils and Accessories (1)

Results: 1-1 of 1
For: spatula

Remove this text term from your search

Spoons and Spatulas
iphrase wcs comparison id:
KF1xAShVGTpLaXRjaGVud2FyZTpBY2Nlc3NvcmllczpVFjpPaWxzIGFuZCBBY2Nlc3NvcmllczplXXECVRFQcm
Category: Kitchenware > Accessories
Oils and Accessories
Description: An everyday baker's mix of cooking spoons and spatulas. Consists of four stain- and heat-resistant plastic
spoon.
Body Text: An everyday baker's mix of cooking spoons and spatulas. Consists of four stain- and heat-resistant plastic s
spoon.
TypeID: ProductBean:10089 (1)
SKU Category: Kitchenware > Accessories
Price: 4.99

Result Page: 1

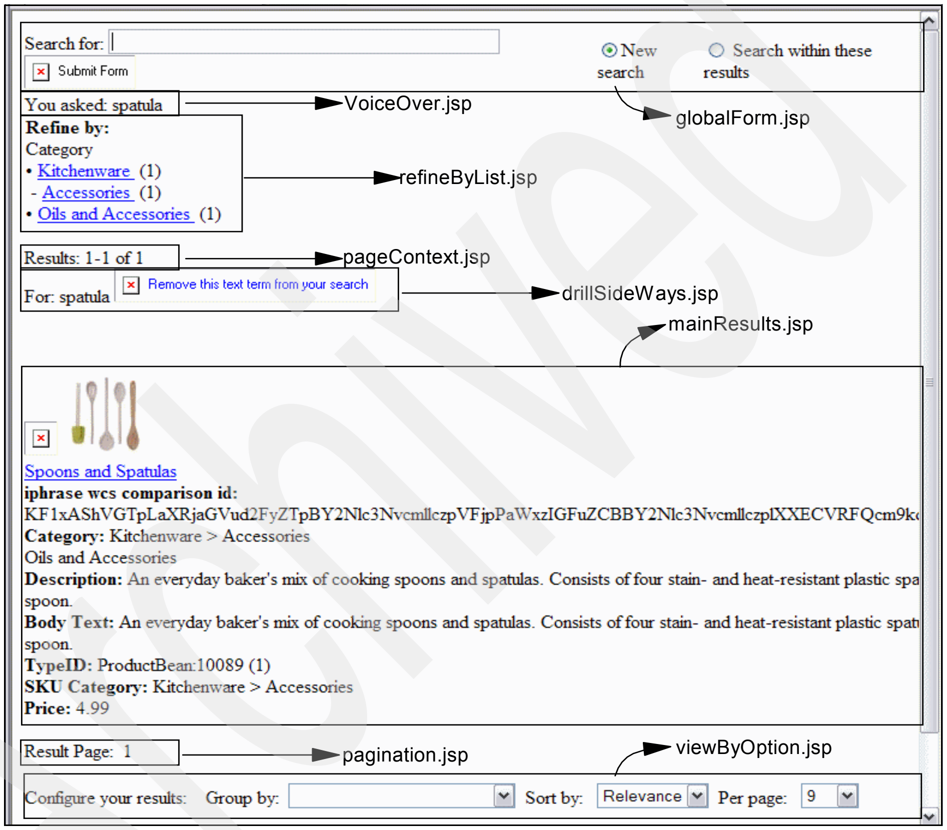
Configure your results: Group by: Sort by: Relevance Per page: 9

Figure 10-10 Basic TabbedNav search result page

10.3.5 Designing and customizing the TabbedNav page

In this section, we explain the components that help construct a search result page using the TabbedNav skin. We explain customization that has to be done for Figure 10-10 on page 234 to look like Figure 10-4 on page 216.

The TabbedNav search result page consists of a layout pattern and has various components to it. See Figure 10-11.



Spoons and Spatulas
iphrase wcs comparison id:
KF1xAShVGTpLaXRjaGVud2FyZTpBY2Nlc3NvcmllczpVFjpPaWxzIGFuZCBBY2Nlc3NvcmllczplXXECVRFQcm9k
Category: Kitchenware > Accessories
Oils and Accessories
Description: An everyday baker's mix of cooking spoons and spatulas. Consists of four stain- and heat-resistant plastic spa
spoon.
Body Text: An everyday baker's mix of cooking spoons and spatulas. Consists of four stain- and heat-resistant plastic spat
spoon.
TypeID: ProductBean:10089 (1)
SKU Category: Kitchenware > Accessories
Price: 4.99

Figure 10-11 Components of a TabbedNav search result page

Important: It is important that you complete 10.3.4, “Code customization using WCDSTabbedNav: main.jsp” on page 225 before you start this section.

The main components or the JSP files of the rendered search results are:

- main.jsp
- globalForm.jsp
- voiceOver.jsp
- refineByList.jsp
- tabbedDrillDown.jsp
- mainResults.jsp
- viewByOptions.jsp

Let us go over each one of these components and look into the customization that are required to get to our end search results page.

We begin with looking at customization for each of the JSP files listed above.

main.jsp customization

main.jsp is responsible for the following operations:

1. Sets up the connection with the OmniFind Discovery Edition search engine.
2. Sets up the required query parameters passed from WebSphere Commerce.
3. Sets up the pageContext object.
4. Executes the query.
5. Renders the query result sets.

In 10.3.4, "Code customization using WCDSTabbedNav: main.jsp" on page 225, we covered the customization with respect to steps 1 through 4. In this section we cover the customization related to step 5.

In main.jsp, after the queryHandler.execute() is run, the queryResultRqst attribute in the pageContext object is set with the query results. The queryResultRqst object is processed further to set other pageContext attributes. A code snippet for processing is shown in Example 10-22.

Example 10-22 Query results processing after queryHandler.execute() is run

```
<%
QueryResult queryResultRqst =
(QueryResult)pageContext.getAttribute("queryResultRqst",
PageContext.REQUEST_SCOPE);
if(queryResultRqst.getResultSets() != null){
   for (int j = 0; j < queryResultRqst.getResultSets().length; j++){
      ResultSet set = queryResultRqst.getResultSets()[j];
      String url = set.getDirectNavUri();
      if(url != null){
         response.sendRedirect(response.encodeURL(url));
```

```
            break;
         }
      }
   }

pageContext.setAttribute("mainResSetRqst",
queryResultRqst.getMainResultSet(), PageContext.REQUEST_SCOPE);
%>
```

We now look into the customization required inside the HTML code in main.jsp. We present our description in three parts:

- Title and style sheet changes
- WebSphere Commerce Header Insertion
- WebSphere Commerce Footer Insertion

Title and style sheet changes

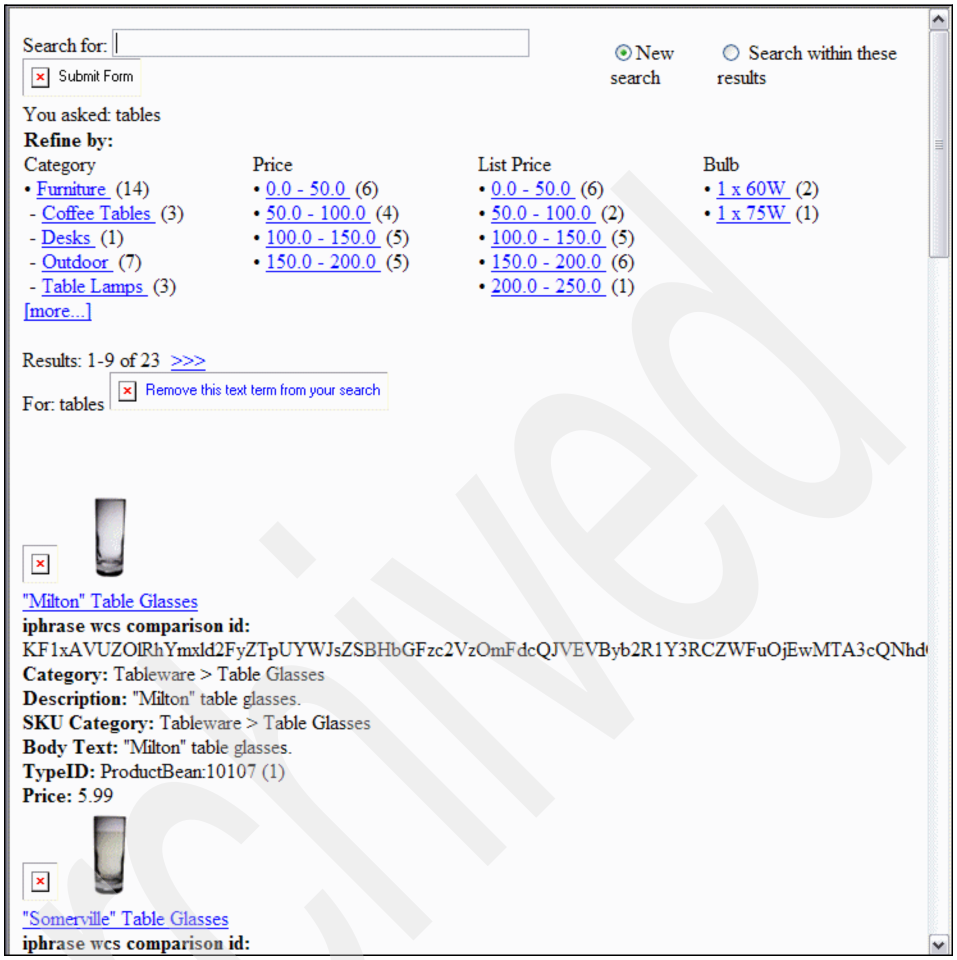
Immediately after the foregoing code, the HTML code for rendering the Web page starts as shown in Example 10-23. We make two changes in this code:

- We replace the title from *Tabbed Navigation Layout* to *WebSphere Content Discovery Server Search*.
- We comment out the existing style sheet and add the WebSphere Commerce style sheet.

Example 10-23 .HTML code for search results rendering

```
<html>
<head>
   <title>WebSphere Content Discover Server Search</title>
   <!--link type="text/css" rel="stylesheet"
href="<%=pageContext.getAttribute("LAYOUT_PATH",
PageContext.REQUEST_SCOPE)%>CSS/ip.css"-->
  <!-- Added WCS style sheet -->
  <link rel="stylesheet" href="<c:out
value="${jspStoreImgDir}${vfileStylesheet}"/>" type="text/css"/>
</head>
```

After making the foregoing changes, a search for tables on the Consumer-Direct store returns a Web page similar to Figure 10-12.



Search for:

Submit Form

[✓ New search] Search within these results

You asked: tables

Refine by:

Category	Price	List Price	Bulb
• Furniture (14)	• 0.0 - 50.0 (6)	• 0.0 - 50.0 (6)	• 1 x 60W (2)
- Coffee Tables (3)	• 50.0 - 100.0 (4)	• 50.0 - 100.0 (2)	• 1 x 75W (1)
- Desks (1)	• 100.0 - 150.0 (5)	• 100.0 - 150.0 (5)	
- Outdoor (7)	• 150.0 - 200.0 (5)	• 150.0 - 200.0 (6)	
- Table Lamps (3)		• 200.0 - 250.0 (1)	
[more...]			

Results: 1-9 of 23 >>>

For: tables Remove this text term from your search

"Milton" Table Glasses

iphrase wcs comparison id:
KF1xAVUZOlRhYmxld2FyZTpUYWJsZSBHbGFzc2VzOmFdcQJVEVByb2R1Y3RCZWFuOjEwMTA3cQNhd

Category: Tableware > Table Glasses

Description: "Milton" table glasses.

SKU Category: Tableware > Table Glasses

Body Text: "Milton" table glasses.

TypeID: ProductBean:10107 (1)

Price: 5.99

"Somerville" Table Glasses

iphrase wcs comparison id:

Figure 10-12 The search result page with correct title and WCS style sheet

Next we add the Consumer-Direct store header and footer information to the search result page.

Header insertion

In main.jsp immediately after the <body> tag of the HTML page, insert the following code:

```
<jsp:include page="application/HeaderDisplay.jsp"/>
```

Create a new file under the application directory called HeaderDisplay.jsp. The contents of HeaderDisplay.jsp are shown in Example 10-24. After setting up the JSTL environment, HeaderDisplay.jsp includes HeaderDisplay.jspf.

Example 10-24 Content of application\HeaderDisplay.jsp

```
<%@include file="../taglibdecl.jsp"%>
<%@include file="jstl.jspf"%>
<%@ include file="../../include/JSTLEnvironmentSetup.jspf"%>
<%
/* Resets environment so that we can successfully include
JSTLEnvironmentSetup.jspf (which conflicts with preQuery.jspf)
 * if that conflict can be fixed, then this indirection can be removed
 */
%>

<%@ include file="HeaderDisplay.jspf"%>
```

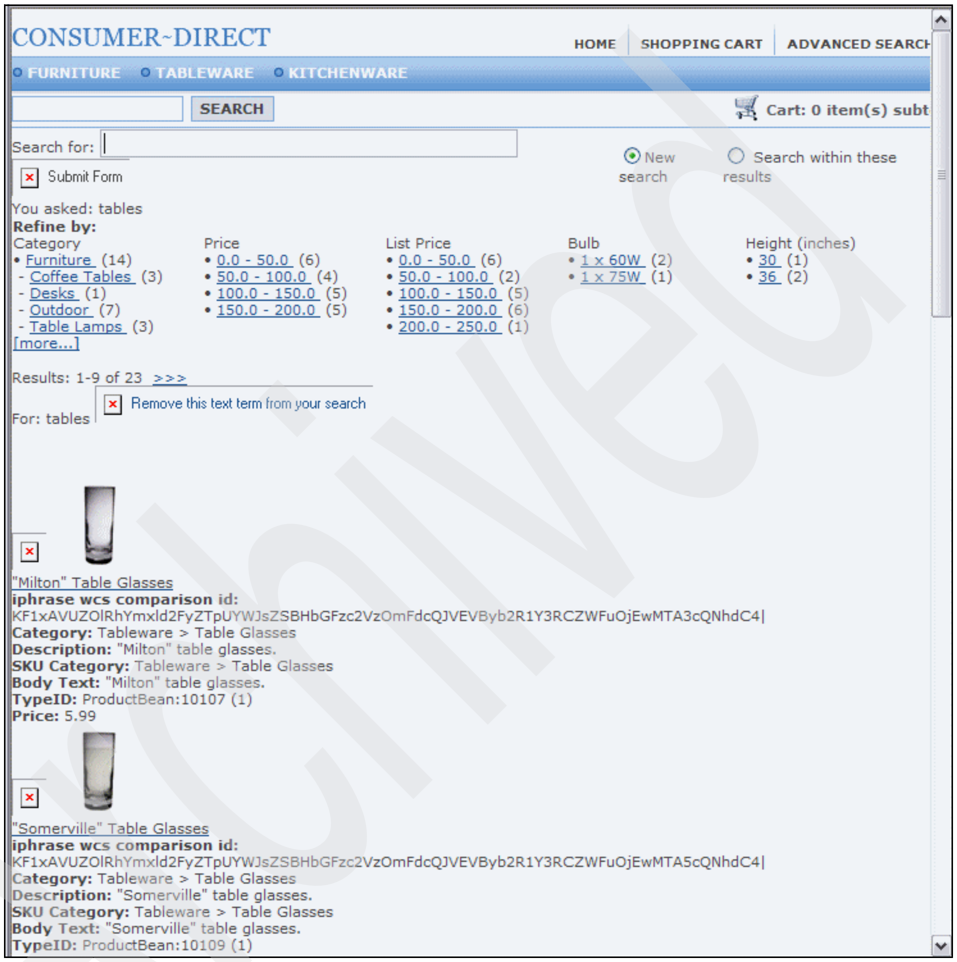
HeaderDisplay.jspf includes the WebSphere Commerce CachedHeaderDisplay.jsp. The contents of HeaderDisplay.jspf are shown in Example 10-25.

Attention: The HeaderDisplay.jspf code will be changed to import a new version of CachedHeaderDisplay.jsp called CachedHeaderWCDS.jsp. It would not have the WebSphere Commerce search form. See “Final modifications: Images and Search box” on page 258 for an explanation.

Example 10-25 Content of application\HeaderDisplay.jspf

```
<link rel="stylesheet" href="<c:out
value="${jspStoreImgDir}${vfileStylesheet}"/>" type="text/css"/>

<!-- BEGIN HeaderDisplay.jspf -->

<% out.flush(); %>
<div>
<c:import url="${jspStoreDir}${StyleDir}CachedHeaderDisplay.jsp">
   <c:param name="storeId" value="${storeId}" />
   <c:param name="catalogId" value="${catalogId}" />
   <c:param name="langId" value="${langId}" />
   <c:param name="userType" value="${userType}" />
   <c:param name="userState" value="${userState}" />
   <c:param name="liveHelp" value="${liveHelp}" />
   <c:param name="rfqLinkDisplayed" value="${rfqLinkDisplayed}" />
</c:import>
</div>
```

After making the foregoing changes, do a search for tables for the Consumer-Direct store. A search result page should return a Web page that looks similar to Figure 10-13. Notice that now you have the WebSphere Commerce' Consumer-Direct store header on the page.



Search for:

Submit Form

(●) New search ( ) Search within these results

You asked: tables
Refine by:

Category	Price	List Price	Bulb	Height (inches)
• Furniture (14)	• 0.0 - 50.0 (6)	• 0.0 - 50.0 (6)	• 1 x 60W (2)	• 30 (1)
- Coffee Tables (3)	• 50.0 - 100.0 (4)	• 50.0 - 100.0 (2)	• 1 x 75W (1)	• 36 (2)
- Desks (1)	• 100.0 - 150.0 (5)	• 100.0 - 150.0 (5)		
- Outdoor (7)	• 150.0 - 200.0 (5)	• 150.0 - 200.0 (6)		
- Table Lamps (3)		• 200.0 - 250.0 (1)		
[more...]				

Results: 1-9 of 23 >>>

For: tables   Remove this text term from your search

"Milton" Table Glasses
iphrase wcs comparison id:
KF1xAVUZOlRhYmxld2FyZTpUYWJsZSBHbGFzc2VzOmFdcQJVEVByb2R1Y3RCZWFuOjEwMTA3cQNhdC4|
Category: Tableware > Table Glasses
Description: "Milton" table glasses.
SKU Category: Tableware > Table Glasses
Body Text: "Milton" table glasses.
TypeID: ProductBean:10107 (1)
Price: 5.99

"Somerville" Table Glasses
iphrase wcs comparison id:
KF1xAVUZOlRhYmxld2FyZTpUYWJsZSBHbGFzc2VzOmFdcQJVEVByb2R1Y3RCZWFuOjEwMTA5cQNhdC4|
Category: Tableware > Table Glasses
Description: "Somerville" table glasses.
SKU Category: Tableware > Table Glasses
Body Text: "Somerville" table glasses.
TypeID: ProductBean:10109 (1)

Figure 10-13 The search result page with WCS header

Footer insertion

In main.jsp immediately before the end of </body> tag of the HTML page, insert the following code:

```
<jsp:include page="application/FooterDisplay.jsp"/>
```

Create a new file under the application directory called FooterDisplay.jsp. The contents of FooterDisplay.jsp are shown in Example 10-24. After setting up the JSTL environment, FooterDisplay.jsp includes HeaderDisplay.jspf.

Example 10-26   Content of application\FooterDisplay.jsp

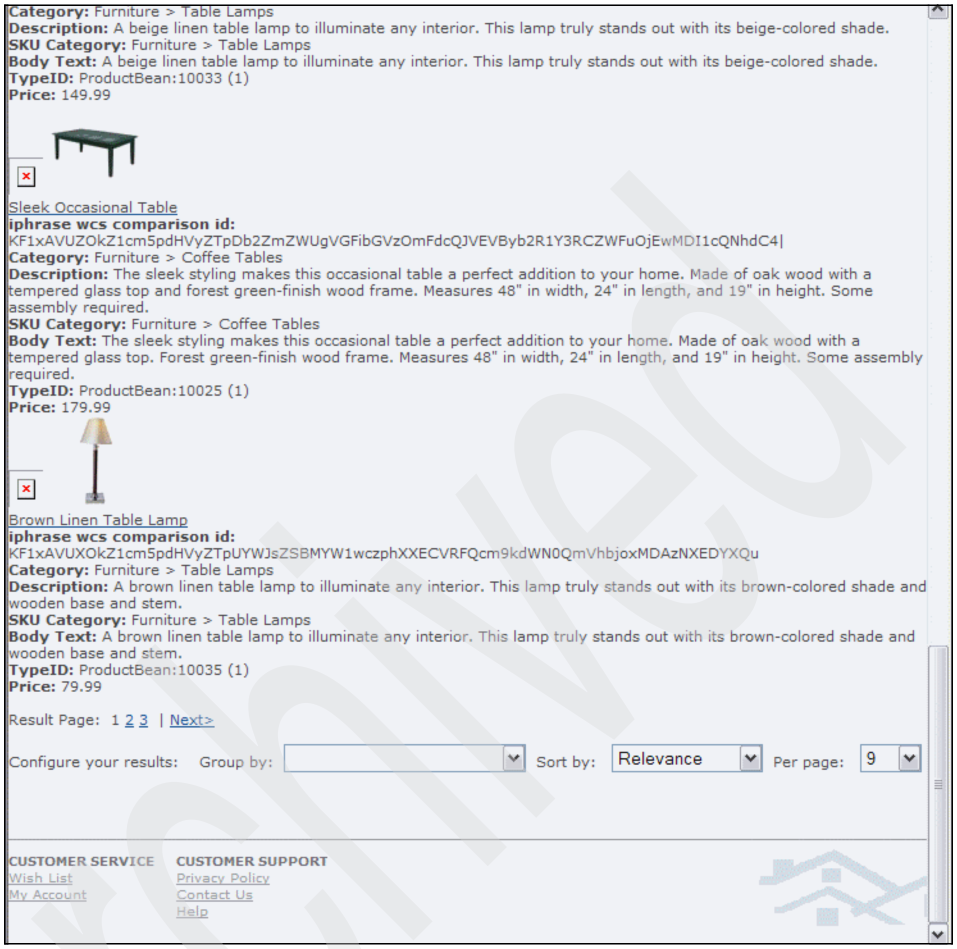
```
<%@include file="../taglibdecl.jsp"%>
<%@include file="jstl.jspf"%>
<%@ include file="../../include/JSTLEnvironmentSetup.jspf"%>
<%
/* Resets environment so that we can successfully include
JSTLEnvironmentSetup.jspf (which conflicts with preQuery.jspf)
 * if that conflict can be fixed, then this indirection can be removed
 */
%>

<%@ include file="FooterDisplay.jspf"%>
```

FooterDisplay.jspf includes the WebSphere Commerce CachedFooterDisplay.jsp. The contents of FooterDisplay.jspf are shown in Example 10-27.

Example 10-27   Contents of application\FooterDisplay.jspf

```
<link rel="stylesheet" href="<c:out
value="${jspStoreImgDir}${vfileStylesheet}"/>" type="text/css"/>
<div>
<c:import url="${jspStoreDir}${StyleDir}CachedFooterDisplay.jsp">
   <c:param name="storeId" value="${storeId}" />
   <c:param name="catalogId" value="${catalogId}" />
   <c:param name="langId" value="${langId}" />
   <c:param name="userType" value="${userType}" />
   <c:param name="userState" value="${userState}" />
   <c:param name="liveHelp" value="${liveHelp}" />
   <c:param name="rfqLinkDisplayed" value="${rfqLinkDisplayed}" />
</c:import>
</div>
```

After making the foregoing changes, a search for tables on the Consumer-Direct store should return a Web page similar Figure 10-14. Notice the footer that appears on the bottom of the page.



Category: Furniture > Table Lamps
Description: A beige linen table lamp to illuminate any interior. This lamp truly stands out with its beige-colored shade.
SKU Category: Furniture > Table Lamps
Body Text: A beige linen table lamp to illuminate any interior. This lamp truly stands out with its beige-colored shade.
TypeID: ProductBean:10033 (1)
Price: 149.99

Sleek Occasional Table
iphrase wcs comparison id:
KF1xAVUZOkZ1cm5pdHVyZTpDb2ZmZWUgVGFibGVzOmFdcQJVEVByb2R1Y3RCZWFuOjEwMDI1cQNhdC4|
Category: Furniture > Coffee Tables
Description: The sleek styling makes this occasional table a perfect addition to your home. Made of oak wood with a tempered glass top and forest green-finish wood frame. Measures 48" in width, 24" in length, and 19" in height. Some assembly required.
SKU Category: Furniture > Coffee Tables
Body Text: The sleek styling makes this occasional table a perfect addition to your home. Made of oak wood with a tempered glass top. Forest green-finish wood frame. Measures 48" in width, 24" in length, and 19" in height. Some assembly required.
TypeID: ProductBean:10025 (1)
Price: 179.99

Brown Linen Table Lamp
iphrase wcs comparison id:
KF1xAVUXOkZ1cm5pdHVyZTpUYWJsZSBMYW1wczphXXECVRFQcm9kdWN0QmVhbjoxMDAzNXEDYXQu
Category: Furniture > Table Lamps
Description: A brown linen table lamp to illuminate any interior. This lamp truly stands out with its brown-colored shade and wooden base and stem.
SKU Category: Furniture > Table Lamps
Body Text: A brown linen table lamp to illuminate any interior. This lamp truly stands out with its brown-colored shade and wooden base and stem.
TypeID: ProductBean:10035 (1)
Price: 79.99

Result Page: 1 2 3 | Next>

Configure your results: Group by: Sort by: Relevance Per page: 9

Figure 10-14 The search result page with WebSphere Commerce footer

globalForm.jsp customization

globalForm.jsp defines the HTML inquiry form used by users to enter a query with its format and fields (including hidden fields). If you use only the OmniFind Discovery Edition search engine to render a Web page for the search query and navigation, this form would be the start of the query page. Since, we integrate OmniFind Discovery Edition search engine to the existing WebSphere Commerce store, this search form is duplicate in its basic functionality, but with a subtle difference. OmniFind Discovery Edition search inquiry form comes along with two additional radio button; one enables a user to start a new search and one enables the user to a search within the result set.

For our modification, we disable the HTML code inside globalForm.jsp and disable the JavaScript functions that submits the form. All other JSP files which submit user inquiries, use the JavaScript functions defined in globalForm.jsp to perform the submission.

Attention: This is a useful feature and we would keep this instead of the WebSphere Commerce search form in our final search result page. Later on in section "Final modifications: Images and Search box" on page 258, we reverse this choice to take advantage of search within feature provided by OmniFind Discovery Edition such that once we enter the primary search text using WebSphere Commerce search inquiry form, the OmniFind Discovery Edition search results page would render results and put its own inquiry form and hide the WebSphere Commerce inquiry form. We explain this in "Final modifications: Images and Search box" on page 258.

Customization of *globalForm.jsp* is summarized as follows:

- Add the include files.
- Set actionName to the actionpage attribute of the contextpage.
- Code comments for HTML form.
- Add include files.

Adding include files

Include two files in globalForm.jsp. This is required for setting up the JSTL tag library and the JSTL environment. To do so, locate and find the JSP include code for taglibdecl.jsp. Insert the following lines at the end of the existing include code:

```
<%@include file="application/jstl.jspf"%>
<%@include file="application/resourceBundle.jspf"%>
```

Setting actionName to actionpage attribute of the context page

We require the action name for the form. The actionName is not set correctly in the in the getRequest() for the pagecontext. So instead, we use pageContext's actionPage. See the code changes in Example 10-28.

Example 10-28 Changing actionName to refer to actionpage of the page context

```
// Changed actionName to get from actionPage instead of actionName
// String actionName =
pageContext.getRequest().getParameter("actionName");
String actionName = (String)pageContext.getAttribute("actionpage",
PageContext.REQUEST_SCOPE);
```

Commenting out the HTML form code

The code shown in Example 10-29 is responsible for displaying the OmniFind Discovery Edition search form along with radio buttons for **New Search** and **Search Within**. These controls are designed and made available if OmniFind Discovery Edition is run as a standalone product to enhance the search experience for a store. Since this is duplicated functionality, you can comment this code as shown as Example 10-29. Later on, we uncomment this code and comment out the WebSphere Commerce code responsible for this functionality.

Example 10-29 Commenting out the HTML form code

```
   <!-- start ipSearchTable table in globalForm.jsp -->
<!-- Comment Out
   <table border="0" cellspacing="0" cellpadding="0"
id="ipSearchTable">
      <tr>
         <td>
            <span
class="ipEmphasis"><%=prose.getString("searchForStr")%></span>
            <input type="text" name="ip_row_text" class="ipSearchInput"
size="50">
            <input type="image"
src="<%=pageContext.getAttribute("LAYOUT_PATH",
PageContext.REQUEST_SCOPE)%>images/<%=prose.getString("searchButton")%>
" alt="Submit Form" class="ipSearchInput">
         </td>
            <%if((queryResultRqst != null) &&
(resultSet.getIsRenderSearchWithin() == true)){%>
               <td>
                  <input type="radio" class="ipRadio"
name="ip_searchWithin" value="Off"
checked="true"><%=prose.getString("newSearchStr")%></input>
               </td>
               <td>
                  <input type="radio" class="ipRadio"
name="ip_searchWithin" value="On">
<%=prose.getString("searchWithinStr")%></input>
               </td>
            <%}%>
         <td>
      </tr>
   </table>--> <%// End of Comment %>
   <%// end ipSearchTable %>
```

Adding include files

Locate the initForm() function inside the JavaScript <script> tag. Add the following include code after all the document.globalform parameters are set:

```
<% /* Included this file */ %>
<%@include file="application/globalFormInit.jspf" %>
```

globalFormInit.jspf initializes storeId, catalogId, and LanguageId parameters passed from WebSphere Commerce. The contents of application\globalFormInit.jspf are shown in Example 10-30.

Example 10-30 Content of application\globalFormInit.jspf

```
<!-- start: application/globalFormInit.jspf -->
       document.globalForm.storeId.value="<c:out
value="${CommandContext.storeId}" />";
       document.globalForm.catalogId.value="<c:out
value="${WCParam.catalogId}" />";
       document.globalForm.langId.value="<c:out
value="${CommandContext.languageId}" />";
       document.globalForm.wcsCompareConstraint.value="";
<!-- end: application/globalFormInit.jspf -->
```

Just before the end of form </form> tag in globalForm.jsp, add the following include lines:

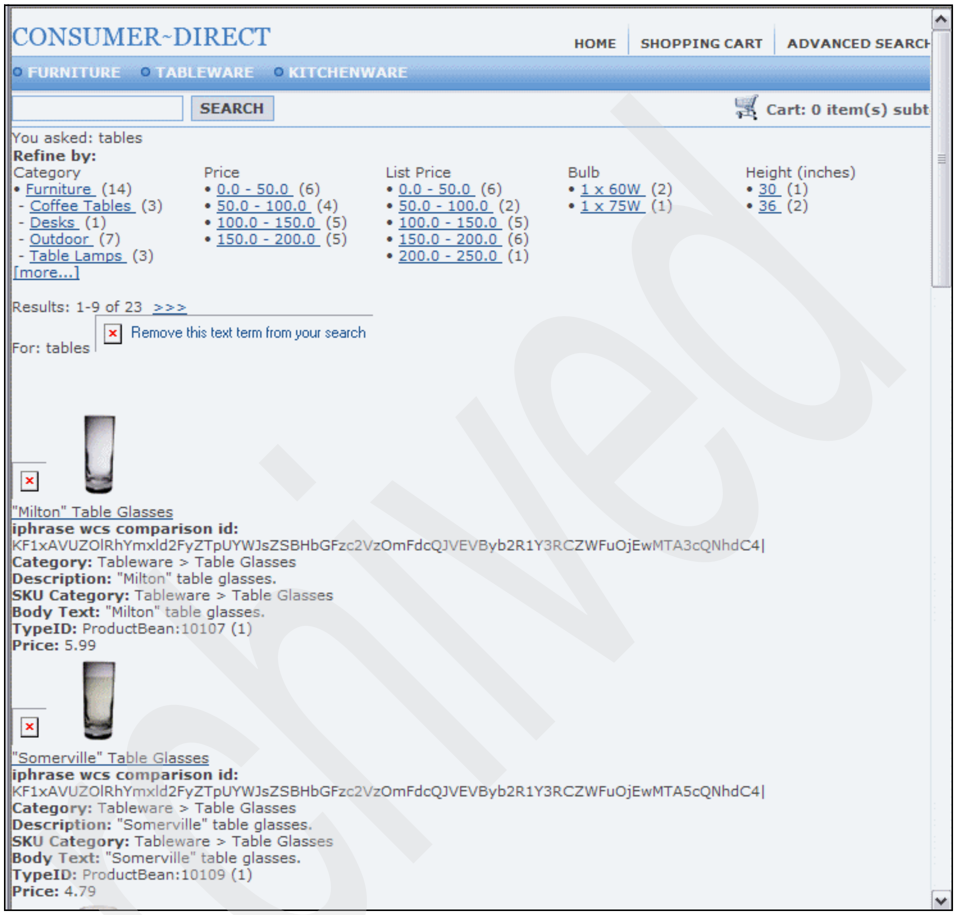
```
<% /* Included this file */ %>
<%@include file="application/globalFormForm.jspf" %>
```

globalFormForm.jspf restores the values of storeId, catalogId, and LanguageId parameters passed from WebSphere Commerce. The contents of application\globalFormForm.jspf are shown in Example 10-31.

Example 10-31 Content of application\globalFormForm.jspf

```
!-- start: application/globalFormForm.jspf -->
        <input type="hidden" name="storeId" value="<c:out
value="${CommandContext.storeId}" />">
        <input type="hidden" name="catalogId" value="<c:out
value="${WCParam.catalogId}" />">
        <input type="hidden" name="langId" value="<c:out
value="${CommandContext.languageId}" />">
        <input type="hidden" name="wcsCompareConstraint" value="">
<!-- end: application/globalFormForm.jspf -->
```

After making the foregoing changes, with a search for tables, the new search result page looks as shown in Figure 10-15.



You asked: tables
Refine by:

Category	Price	List Price	Bulb	Height (inches)
• Furniture (14)	• 0.0 - 50.0 (6)	• 0.0 - 50.0 (6)	• 1 x 60W (2)	• 30 (1)
- Coffee Tables (3)	• 50.0 - 100.0 (4)	• 50.0 - 100.0 (2)	• 1 x 75W (1)	• 36 (2)
- Desks (1)	• 100.0 - 150.0 (5)	• 100.0 - 150.0 (5)		
- Outdoor (7)	• 150.0 - 200.0 (5)	• 150.0 - 200.0 (6)		
- Table Lamps (3)		• 200.0 - 250.0 (1)		
[more...]				

Results: 1-9 of 23 >>>

For: tables
Remove this text term from your search

"Milton" Table Glasses
iphrase wcs comparison id:
KF1xAVUZOlRhYmxld2FyZTpUYWJsZSBHbGFzc2VzOmFdcQJVEVByb2R1Y3RCZWFuOjEwMTA3cQNhdC4|
Category: Tableware > Table Glasses
Description: "Milton" table glasses.
SKU Category: Tableware > Table Glasses
Body Text: "Milton" table glasses.
TypeID: ProductBean:10107 (1)
Price: 5.99

"Somerville" Table Glasses
iphrase wcs comparison id:
KF1xAVUZOlRhYmxld2FyZTpUYWJsZSBHbGFzc2VzOmFdcQJVEVByb2R1Y3RCZWFuOjEwMTA5cQNhdC4|
Category: Tableware > Table Glasses
Description: "Somerville" table glasses.
SKU Category: Tableware > Table Glasses
Body Text: "Somerville" table glasses.
TypeID: ProductBean:10109 (1)
Price: 4.79

Figure 10-15 The search result page after globalForm.jsp customization

refineByList.jsp customization

We now discuss the way we can show the results rendered by refineByList on the left sidebar in a vertical fashion as opposed to the current display of the results rendered in a horizontal fashion. There are two ways of doing this:

- By making changes to the table format in existing refineByList.jsp
- By making use of the refineByList.jsp from Commerce skin

We make use of the refineByList.jsp from Commerce, since it is much easier and less cumbersome to integrate it into our TabbedNav skin.

To do this, copy over the following three files:

- refineByList.jsp
- constants.jsp
- formatValue.jsp

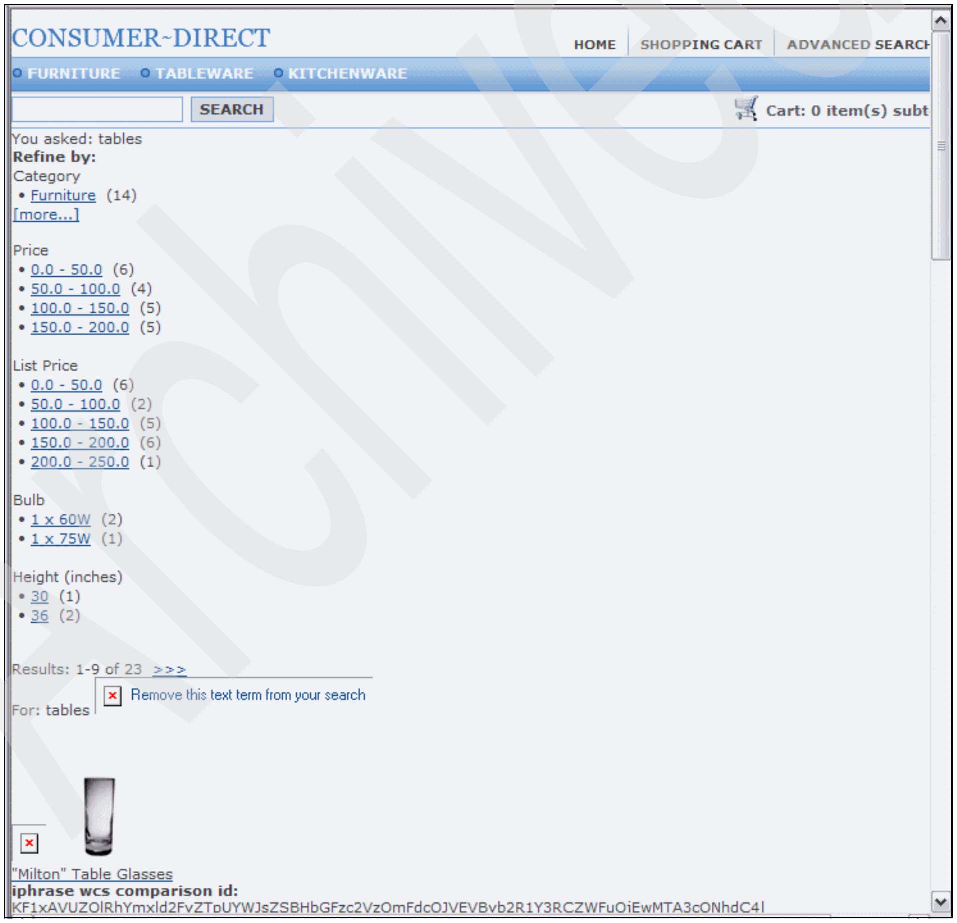
from:

`<`*`IPHRASE_HOME`*`>\webapps\jspapps\layout-commerce\jsp`

to:

`<`*`WCS_HOME`*`>\Stores.war\ConsumerDirect\WCDSTabbedNav`

After making the foregoing changes, a search on tables would return a search result page looks similar to Figure 10-16.



You asked: tables
Refine by:
Category
- Furniture (14)

[more...]

Price
- 0.0 - 50.0 (6)
- 50.0 - 100.0 (4)
- 100.0 - 150.0 (5)
- 150.0 - 200.0 (5)

List Price
- 0.0 - 50.0 (6)
- 50.0 - 100.0 (2)
- 100.0 - 150.0 (5)
- 150.0 - 200.0 (6)
- 200.0 - 250.0 (1)

Bulb
- 1 x 60W (2)
- 1 x 75W (1)

Height (inches)
- 30 (1)
- 36 (2)

Results: 1-9 of 23 >>>

Remove this text term from your search

For: tables

"Milton" Table Glasses
iphrase wcs comparison id:
KF1xAVUZOlRhYmxld2FvZTpUYWJsZSBHbGFzc2VzOmFdcOJVEVBvb2R1Y3RCZWFuOiEwMTA3cONhdC4l

Figure 10-16 The search result page after refineByList.jsp customization

mainResultsTable.3a.jsp Customization

We now have to render the main results on the search result page. The TabbedNav skins display the main results in a N X 1 (N rows by 1 column) table format. This means that if the search results yielded N matching products, all N products would be shown on the main page one after another vertically. There is only one product display for each horizontal section. Also TabbedNav shows all the features of a product that are set in the OmniFind Discovery Edition Management Console.

We want to make the search result page looks like the one that is rendered with the default integration solution. We want to display results (N products) as an M X 5 (M rows by 5 columns) matrix and remove all the features except for Product Description and Price shown along with the product.

mainResultsTable.3a.jsp renders the main results table in the 5 across the layout. The main results table in this layout displays five products in a row. For each product, an image is displayed with the product features, such as name, description, and price. We have to do three sets of customization to mainResultsTable.3a.jsp:

1. Copy file from Commerce skin and make changes to main.jsp.
2. Modify the code to eliminate feature display.
3. Modify the code to resize rows and other display changes.

Copying file from Commerce skin and making changes to main.jsp

To start, we first copy over the mainResultsTable.3a.jsp from:

```
<IPHRASE_HOME>\webapps\jspapps\layout-commerce\jsp
```

to:

```
<WCS_HOME>\Stores.war\ConsumerDirect\WCDSTabbedNav
```

Attention: We just copied over a new JSP file. To see the effect of this JSP, we have to modify main.jsp. See the instructions following.

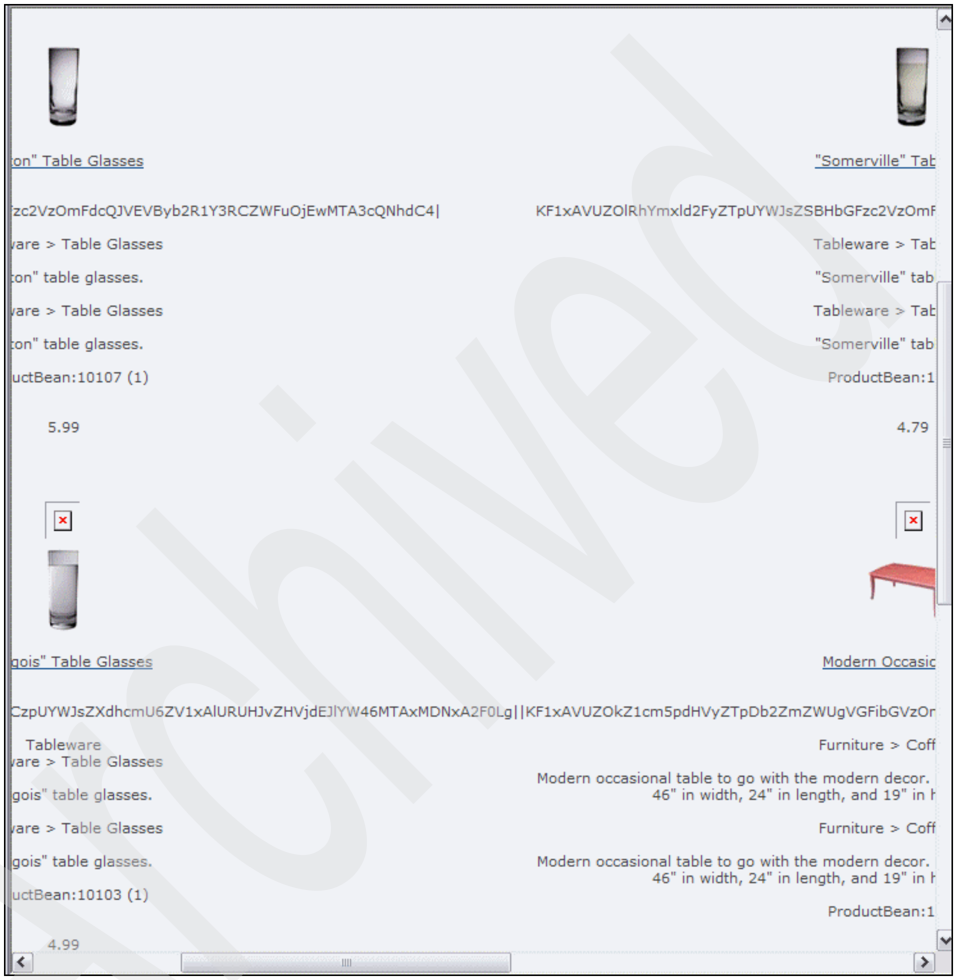
To include this new JSP (mainResultsTable.3a.jsp) in main.jsp, search and locate the code as shown in Example 10-32.

Example 10-32 include code for mainResultsTable.jsp in main.jsp

```
            <!-- render Main Results Table -->
            <jsp:include page="mainResultsTable.jsp">
             <jsp:param name="resultSetName" value="mainResSetRqst"/>
            </jsp:include>
```

Replace the bold text mainResultsTable.jsp with mainResultsTable.3a.jsp.

After making the foregoing changes, a search for tables would produce a search result page that looks as shown in Figure 10-17.



```
on" Table Glasses                                                    "Somerville" Tab
zc2VzOmFdcQJVEVByb2R1Y3RCZWFuOjEwMTA3cQNhdC4|        KF1xAVUZOlRhYmxld2FyZTpUYWJsZSBHbGFzc2VzOmF
are > Table Glasses                                                  Tableware > Tab
on" table glasses.                                                   "Somerville" tab
are > Table Glasses                                                  Tableware > Tab
on" table glasses.                                                   "Somerville" tab
uctBean:10107 (1)                                                    ProductBean:1
    5.99                                                                  4.79

gois" Table Glasses                                                  Modern Occasic
CzpUYWJsZXdhcmU6ZV1xAlURUHJvZHVjdEJlYW46MTAxMDNxA2F0Lg||KF1xAVUZOkZ1cm5pdHVyZTpDb2ZmZWUgVGFibGVzOr
  Tableware                                                          Furniture > Coff
are > Table Glasses
                                                     Modern occasional table to go with the modern decor.
gois" table glasses.                                             46" in width, 24" in length, and 19" in h
are > Table Glasses                                                  Furniture > Coff
gois" table glasses.                                 Modern occasional table to go with the modern decor.
                                                                 46" in width, 24" in length, and 19" in h
uctBean:10103 (1)
                                                                     ProductBean:1
    4.99
```

Figure 10-17   The search result page after mainResultsTable.3a.jsp customization

Modifying the code to eliminate feature display

When the result product is displayed, it is shown along with other features of the product such as taxonomy and product description. You might not want to display these product features because they are not essential to what you want to show to the users and might disrupt the overall presentation of the search result page.

To hide the non-essential features from being displayed, we modify mainResultsTable.3a.jsp as shown in Example 10-33. Find the `if` condition shown in the example and add the extra expression (highlighted as bold text) in the end of this condition. A hash set called noDisplayIds is constructed in constants.jsp. This condition checks for feature Ids matching features declared in the hash set and do not display them.

Example 10-33 Addition of expression at end of IF clause in mainResultsTable.3a.jsp

```
        if (!currentFeature.getId().equals(PRODUCT_ID) &&
!currentFeature.getId().equals(CATEGORY_ID) &&
!noDisplayIds.contains(currentFeature.getId())) {
```

Important: Make sure to do the customization to constants.jsp to actually see the results as shown.

Modifying the code to resize rows and other display changes

The existing mainResultsTable.3a.jsp displays only 3 products per row and has hard coded values. We remove that in our implementation and instead use a variable `ROWS_PER_PAGE_3A` from constants.jsp.

To do so, replace all occurrences of `3` by `ROWS_PER_PAGE_3A`. Replace the hard coded value of `33%` to `100/ROWS_PER_PAGE_3A` as shown in Example 10-34.

Example 10-34 Logic to recalculate the width based on ROWS_PER_PAGE_3A

```
<!-- Logic -->

     <td width="<%=100/ROWS_PER_PAGE_3A%>%" class="ipResultsCopy"
align=center valign=top>
     <!--td width="33%" class="ipResultsCopy" align=center
valign=top-->
```

In addition, add the following include statements after taglibdecl.jsp as shown in Example 10-35.

Example 10-35 List of includes in mainResultsTable.3a.jsp

```
<%@include file="taglibdecl.jsp"%>
<%@include file="application/jstl.jspf"%>
<%@include file="formatValue.jsp"%>
<%@include file="constants.jsp"%>
```

Finally, comment out the code for rendering a shopping cart image as shown in Example 10-36. Add a product to the shopping cart by clicking the product.

Example 10-36 Comment shopping cart image to be displayed

```
<% /* Commented Out */ %>
   <!-- a href=""><img src="<%=pageContext.getAttribute("LAYOUT_PATH", 
PageContext.REQUEST_SCOPE)%>images/add.gif" border=0></a-->
</td>
```

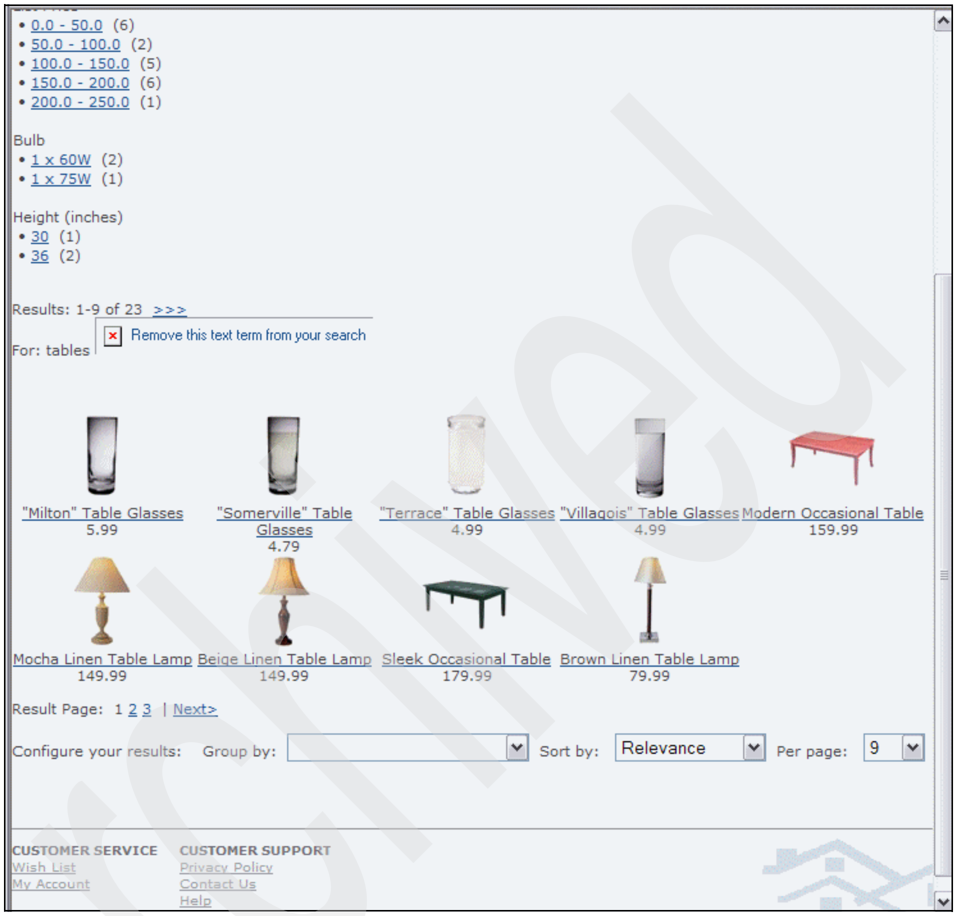
constants.jsp customization

Make changes to constant.jsp as shown in Example 10-37. These variables are used extensively in mainResultsTable.3a.jsp to dynamically render search results.

Example 10-37 Modify constants.jsp to reflect these values

```
//needed to send to server for product comparison
String PRODUCT_ID = "iphrase bundle id";

//needed to sort by and display department name in Dept Previews
String CATEGORY_ID = "iphrase bundle taxonomy";

//String DEFAULT_LAYOUT = "fl";
String DEFAULT_LAYOUT = "3a";

// the default number of rows per page for 3 across layout
int ROWS_PER_PAGE_3A = 5;

//the default number of rows per page for flat list layout
int ROWS_PER_PAGE_FL = 10;

HashSet noDisplayIds = new HashSet();
//
// a list of product keys NOT to show
//
noDisplayIds.add("iphrase wcs comparison id");
noDisplayIds.add("sitemap taxonomy");
noDisplayIds.add("iphrase bundle description");
noDisplayIds.add("iphrase bundle taxonomy");
noDisplayIds.add("sitemap teaser");
noDisplayIds.add("sitemap text");
noDisplayIds.add("iphrase wcs cart up sell");
noDisplayIds.add("iphrase wcs cart cross sell");
noDisplayIds.add("iphrase wcs cart accessory");
%>
```

After making the foregoing changes, a search for tables would return a search result page that looks similar to Figure 10-18.


• 0.0 - 50.0 (6)
• 50.0 - 100.0 (2)
• 100.0 - 150.0 (5)
• 150.0 - 200.0 (6)
• 200.0 - 250.0 (1)

Bulb
• 1 x 60W (2)
• 1 x 75W (1)

Height (inches)
• 30 (1)
• 36 (2)

Results: 1-9 of 23 >>>

For: tables   Remove this text term from your search

"Milton" Table Glasses 5.99	"Somerville" Table Glasses 4.79	"Terrace" Table Glasses 4.99	"Villagois" Table Glasses 4.99	Modern Occasional Table 159.99
Mocha Linen Table Lamp 149.99	Beige Linen Table Lamp 149.99	Sleek Occasional Table 179.99	Brown Linen Table Lamp 79.99	

Result Page: 1 2 3 | Next>

Configure your results: Group by: Sort by: Relevance Per page: 9

Figure 10-18 Search Results page after all modifications to the mainResultsTable.3a.jsp

Notice that in Figure 10-18, there are five products displayed across each row of the search result page.

main.jsp customization

We are very close to getting the final appearance and behavior that we want from our TabbedNav skin. As you can see from Figure 10-18, we have to move the rendering of the main results table from being rendered below the left hand sidebar to render it next to the left hand sidebar display.

To do this, we have to customize table properties within main.jsp so as to position the various components that we talked about in 10.3.5, "Designing and customizing the TabbedNav page" on page 235.

On the main search results page, there are three layers of components that get rendered as shown in Figure 10-19.

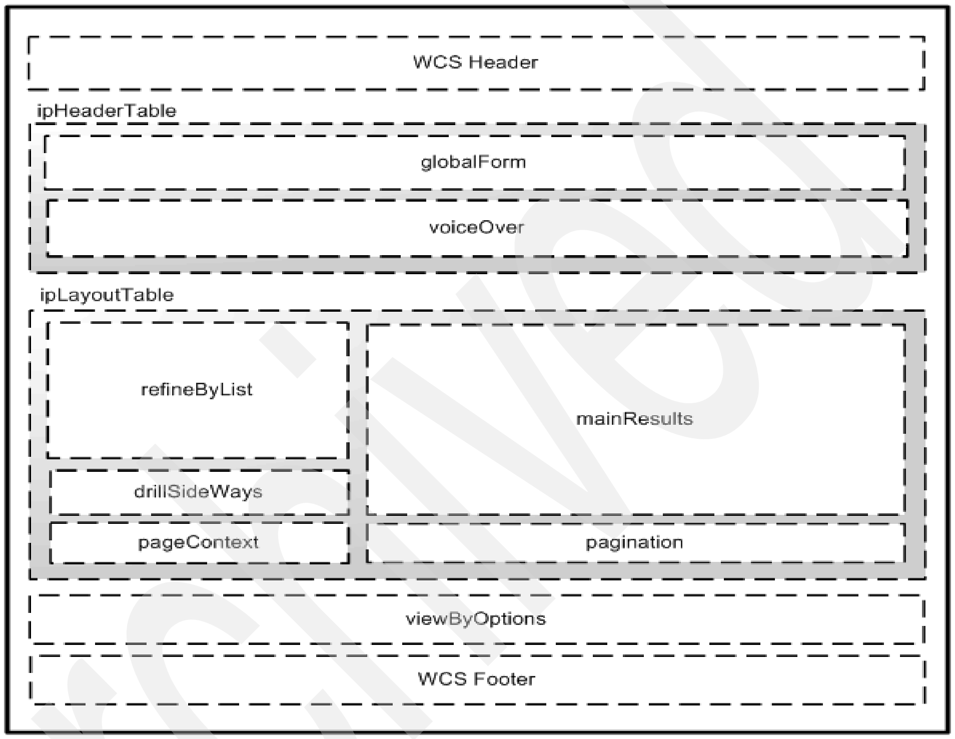


Figure 10-19 Component Layout in the search results page

The first layer of components that gets rendered is called *ipHeaderTable*. It consists of two components:

- *globalForm* — globalForm.jsp renders the OmniFind Discovery Edition HTML form for query string. We disable this code as already explained in "globalForm.jsp customization" on page 242.
- *voiceOver* — voiceOver.jsp renders the voiceover components such as: you asked, within, you clicked, you removed, stopwords, backoff phrases, related terms, synonyms, and respelled words.

The second layer of components that gets rendered is called *ipLayoutTable*. It consists of five components:

- *refineByList* — refineByList.jsp formats results categorization and refinement options. We cover customization of this list using the Management Console in 10.3.7, "Customization using Management Console" on page 264.
- *drillSideWays* — drillSideWays.jsp renders breadcrumbs and provides extra flexibility of conditional navigation.
- *pageContext* — pageContext.jsp displays the page numbers in the format *Results 1-N of NN*, where *1-N* is the range of results and *NN* is total number of results.
- *mainResults* — mainResults.jsp renders the search results in a M X 5 matrix.
- *pagination* - pagination.jsp displays the previous page, next page, and the page numbers as links. The page numbers are displayed as a sliding row of 10 numbers.

The third layer of components that gets rendered is called *ipLayoutTable*. It consists of just one component:

- *viewByOptions* — viewByOptions.jsp renders page controls such as group by, sort by, and per page.

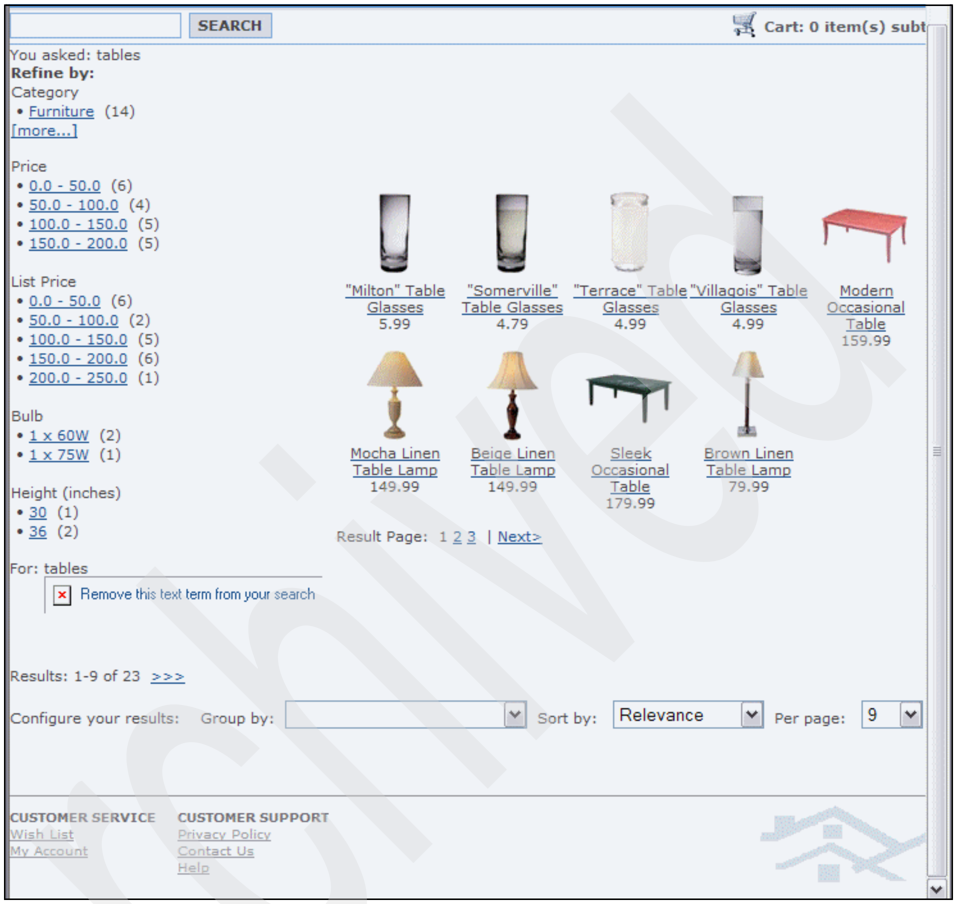
Modify the main.jsp code as shown in Example 10-38.

Example 10-38 main.jsp components rendering code changes

```
<!-- start ipHeaderTable table in main.jsp -->

<table border="0" cellspacing="0" cellpadding="0" id="ipHeaderTable">
   <tr>
      <td>
         <jsp:include page="globalForm.jsp">
            <jsp:param name="resultSetName" value="mainResSetRqst"/>
            <jsp:param name="actionName" value="<%=actionName%>"/>
         </jsp:include>
      </td>
   </tr>

   <tr>
      <td class="VOUser" align="left" width="100%" colspan="4">
         <!-- renderVoiceover -->
         <jsp:include page="voiceover.jsp">
            <jsp:param name="resultSetName" value="mainResSetRqst"/>
         </jsp:include>
      </td>
   </tr>


</table><%// end ipHeaderTable %>
<!-- end ipHeaderTable table in main.jsp -->

<!-- start ipLayout table in main.jsp -->

         <!-- render drillDown tabs -->
         <jsp:include page="tabbedDrillDown.jsp">
            <jsp:param name="resultSetName" value="mainResSetRqst"/>
         </jsp:include>

   <table border="0" cellspacing="0" cellpadding="0" id="ipLayout">

      <tr>

      <td id="ipContentCol1"> <!-- start ipContentCol1 in main.jsp -->

         <!-- render refineBy tabs -->
         <%/* Holds attributes.render_refine_full_page flag.
         The attribute is set in the refineByList.jsp file    */%>
         <%pageContext.setAttribute("renderRefineFullPageRqst",
"false", PageContext.REQUEST_SCOPE);%>
         <jsp:include page="refineByList.jsp">
            <jsp:param name="resultSetName" value="mainResSetRqst"/>
         </jsp:include>
         <%String renderRefineFullPageRqstStr =
(String)pageContext.getAttribute("renderRefineFullPageRqst",
PageContext.REQUEST_SCOPE);
         if(!Tools.isTrue(renderRefineFullPageRqstStr)){%>

            <!-- render page Controls: number of results -->


            <!-- render Breadcrumbs -->
            <jsp:include page="drillSideways.jsp">
               <jsp:param name="resultSetName" value="mainResSetRqst"/>
            </jsp:include>

            <jsp:include page="pageContext.jsp">

                  <jsp:param name="resultSetName"
value="mainResSetRqst"/>
```

```
            </jsp:include>

            <!-- render Direct Answer-->
            <jsp:include page="directAnswer.jsp" />
       </td>

      <td id="ipContentCol2" align="left"> <!-- start ipContentCol2 in
main.jsp -->
            <!-- render Main Results Table -->
            <jsp:include page="mainResultsTable.3a.jsp">
             <jsp:param name="resultSetName" value="mainResSetRqst"/>
            </jsp:include>

            <!-- render page Controls: page navigation -->
            <jsp:include page="pagination.jsp">
             <jsp:param name="resultSetName" value="mainResSetRqst"/>
            </jsp:include>
       </td>

         <%}%> <%// end if (renderRefineFullPage != 1) %>

         <!-- render SideBars -->
         <%if(queryResultRqst.getOtherResultSets() != null){%>
            <%if(queryResultRqst.getOtherResultSets().length > 0) {%>
               </td> <!-- end td id="ipContentCol1" -->
               <td id="ipContentCol3"> <!-- start ipContentCol2 in
main.jsp -->
               <jsp:include page="sidebars.jsp"/>
            <%}%>
         <%}%>
      </td> <!-- end ipContentCol1 or ipContentCol2 td in main.jsp -->
   </tr>
</table> <!-- end ipLayout table in main.jsp -->
<br>
            <!--render ViewBy: perPage, viewBy, groupBy    etc. -->
  <jsp:include page="viewByOptions.jsp">
    <jsp:param name="resultSetName" value="mainResSetRqst"/>
  </jsp:include>
```

After making the foregoing changes, a search for tables would result in a search result page that looks similar to Figure 10-17.



SEARCH

Cart: 0 item(s) subt

You asked: tables
Refine by:
Category
• Furniture (14)
[more...]

Price
• 0.0 - 50.0 (6)
• 50.0 - 100.0 (4)
• 100.0 - 150.0 (5)
• 150.0 - 200.0 (5)

List Price
• 0.0 - 50.0 (6)
• 50.0 - 100.0 (2)
• 100.0 - 150.0 (5)
• 150.0 - 200.0 (6)
• 200.0 - 250.0 (1)

Bulb
• 1 x 60W (2)
• 1 x 75W (1)

Height (inches)
• 30 (1)
• 36 (2)

For: tables
Remove this text term from your search

"Milton" Table Glasses 5.99
"Somerville" Table Glasses 4.79
"Terrace" Table Glasses 4.99
"Villagois" Table Glasses 4.99
Modern Occasional Table 159.99
Mocha Linen Table Lamp 149.99
Beige Linen Table Lamp 149.99
Sleek Occasional Table 179.99
Brown Linen Table Lamp 79.99

Result Page: 1 2 3 | Next>

Results: 1-9 of 23 >>>

Configure your results: Group by: Sort by: Relevance Per page: 9

CUSTOMER SERVICE
Wish List
My Account

CUSTOMER SUPPORT
Privacy Policy
Contact Us
Help

Figure 10-20 Search Results Page after final main.jsp Customization

Final modifications: Images and Search box

For the final modifications, we cover the following changes:

- Fix the GIF image links on the search result page.
- Switch the search inquiry HTML form from WebSphere Commerce to OmniFind Discovery Edition search form.

Fixing GIF image links on the search result page

In main.jsp, as explained in “main.jsp customization” on page 236, we defined the *basedir* variable pointing to *ConsumerDirect/WCDSTabbedNav*. All image source references inside the WCDSTabbedNav area are relative to the value of *basedir* variable followed by images directory.

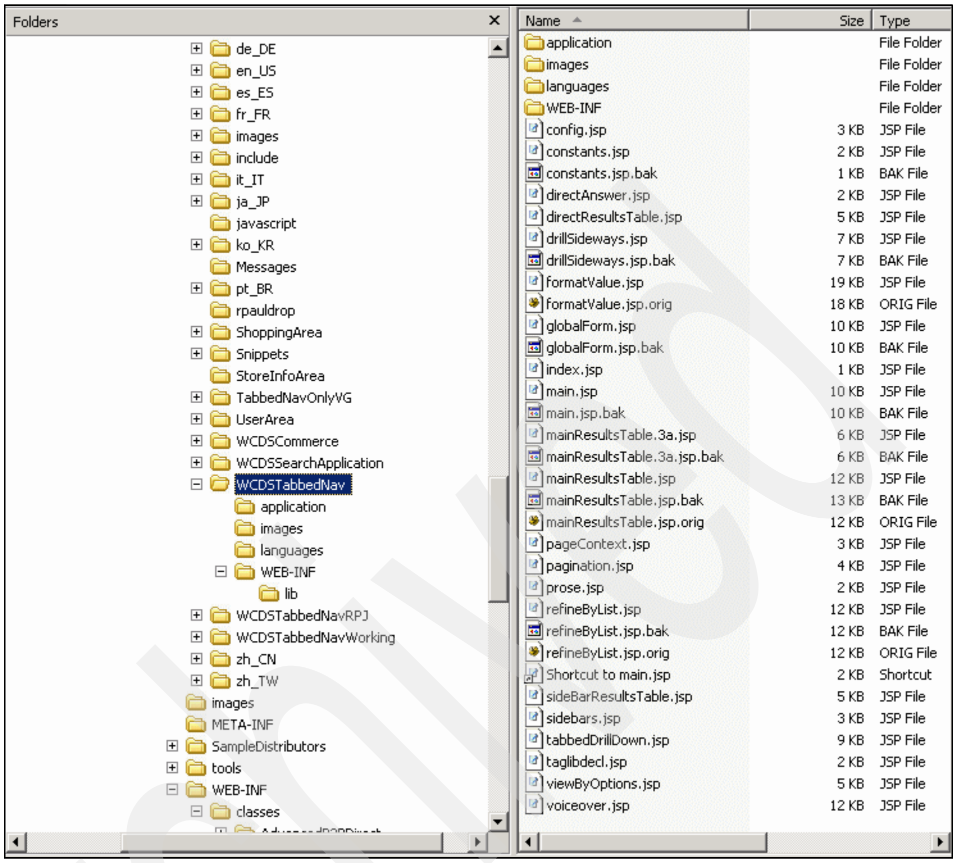
We now have to move our images directory one level up by copying images folder from:

```
<WCS_HOME>\ConsumerDirect\WCDSTabbedNav\layout-tabbednav\
```

to:

```
<WCS_HOME>\ConsumerDirect\WCDSTabbedNav\
```

Because we do not use the CSS under the layout-tabbednav directory, we can remove the entire layout-tabbednav directory. Your directory structure should look similar to Figure 10-21.



Folders

- de_DE
- en_US
- es_ES
- fr_FR
- images
- include
- it_IT
- ja_JP
- javascript
- ko_KR
- Messages
- pt_BR
- rpauldrop
- ShoppingArea
- Snippets
- StoreInfoArea
- TabbedNavOnlyVG
- UserArea
- WCDSCommerce
- WCDSSearchApplication
- WCDSTabbedNav
  - application
  - images
  - languages
  - WEB-INF
    - lib
- WCDSTabbedNavRPJ
- WCDSTabbedNavWorking
- zh_CN
- zh_TW
- images
- META-INF
- SampleDistributors
- tools
- WEB-INF
  - classes

Name	Size	Type
application		File Folder
images		File Folder
languages		File Folder
WEB-INF		File Folder
config.jsp	3 KB	JSP File
constants.jsp	2 KB	JSP File
constants.jsp.bak	1 KB	BAK File
directAnswer.jsp	2 KB	JSP File
directResultsTable.jsp	5 KB	JSP File
drillSideways.jsp	7 KB	JSP File
drillSideways.jsp.bak	7 KB	BAK File
formatValue.jsp	19 KB	JSP File
formatValue.jsp.orig	18 KB	ORIG File
globalForm.jsp	10 KB	JSP File
globalForm.jsp.bak	10 KB	BAK File
index.jsp	1 KB	JSP File
main.jsp	10 KB	JSP File
main.jsp.bak	10 KB	BAK File
mainResultsTable.3a.jsp	6 KB	JSP File
mainResultsTable.3a.jsp.bak	6 KB	BAK File
mainResultsTable.jsp	12 KB	JSP File
mainResultsTable.jsp.bak	13 KB	BAK File
mainResultsTable.jsp.orig	12 KB	ORIG File
pageContext.jsp	3 KB	JSP File
pagination.jsp	4 KB	JSP File
prose.jsp	2 KB	JSP File
refineByList.jsp	12 KB	JSP File
refineByList.jsp.bak	12 KB	BAK File
refineByList.jsp.orig	12 KB	ORIG File
Shortcut to main.jsp	2 KB	Shortcut
sideBarResultsTable.jsp	5 KB	JSP File
sidebars.jsp	3 KB	JSP File
tabbedDrillDown.jsp	9 KB	JSP File
taglibdecl.jsp	2 KB	JSP File
viewByOptions.jsp	5 KB	JSP File
voiceover.jsp	12 KB	JSP File

Figure 10-21 Final WCDSTabbedNav directory structure

Find all occurrences of `img src` code under the WCDSTabbedNav directory and replace them with the code shown in Example 10-39.

Example 10-39 Image Source values relative to basedir

```
img src="${basedir}/images/add.gif"
OR
src="<c:out value="${basedir}"/>/images/add.gif
```

Switching the search inquiry form to OmniFind Discovery Edition

OmniFind Discovery Edition provides rich sets of functionality in its search form. At this step, we switch the search form from WebSphere Commerce to OmniFind Discovery Edition.

To do so, perform the following steps:

1. Make a copy of the following file from the <WCS_HOME>\ConsumerDirect\include\styles\style1 directory:

   `CachedHeaderDisplay.jsp`

   Rename it to:

   `CachedHeaderDisplayWCDS.jsp`

2. Edit this new file: Find the code as shown in Example 10-40 and comment out the table definition section.

Example 10-40 Comment WCS Search Form in CachedHeaderDisplayWCDS.jsp

```
<% /* Commented
         <td id="WC_CachedHeaderDisplay_TableCell_61">
            <table cellpadding="0" cellspacing="0" border="0"
id="WC_CachedHeaderDisplay_Table_4">
            <tbody><tr>
               <form name="CatalogSearchForm"
action="CatalogSearchResultView" method="get" id="CatalogSearchForm">
               <input type="hidden" name="storeId" value='<c:out
value="${WCParam.storeId}" />'
id="WC_CachedHeaderDisplay_FormInput_storeId_In_CatalogSearchForm_1"/>
               <input type="hidden" name="catalogId" value='<c:out
value="${WCParam.catalogId}"/>'
id="WC_CachedHeaderDisplay_FormInput_catalogId_In_CatalogSearchForm_1"/
>
               <input type="hidden" name="langId" value='<c:out
value="${langId}"/>'
id="WC_CachedHeaderDisplay_FormInput_langId_In_CatalogSearchForm_1"/>
               <input type="hidden" name="pageSize" value="12"
id="WC_CachedHeaderDisplay_FormInput_pageSize_In_CatalogSearchForm_1"/>
               <input type="hidden" name="beginIndex" value="0"
id="WC_CachedHeaderDisplay_FormInput_beginIndex_In_CatalogSearchForm_1"
/>
               <input type="hidden" name="sType" value="SimpleSearch"
id="WC_CachedHeaderDisplay_FormInput_sType_In_CatalogSearchForm_1"/>
               <input type="hidden" name="resultCatEntryType" value="2"
id="WC_CachedHeaderDisplay_FormInput_resultType_In_CatalogSearchForm_1"
/>
```

```
              <td valign="middle"
id="WC_CachedHeaderDisplay_TableCell_611">
                 <label
for="WC_CachedHeaderDisplay_FormInput_searchTerm_In_CatalogSearchForm_1
">
                    <input class="input"
id="WC_CachedHeaderDisplay_FormInput_searchTerm_In_CatalogSearchForm_1"
type="text" size="20" maxlength="254" id="search" class="srch"
name="ip_text" title="searchTerm" />
                                              <c:forTokens
items="${requestScope.requestURIPath}" delims="/" var="URLtoken">
                                              <c:set var="ReloadURL"
value="${URLtoken}"/>
                                              </c:forTokens>
                 <input type="hidden" name="ip_requestUri"
value="<c:out value="${ReloadURL}" />">
                 <input type="hidden" name="ip_categoryId"
value="<c:out value="${categoryId}"/>">
                 <input type="hidden" name="ip_catalogId"
value="<c:out value="${catalogId}"/>">
                 <!--ITSO Added's lines-->
                 <input type="hidden" name="ip_text" value="<c:out
value="$(ip_text)"/>">
                 <input type="hidden" name="ip_constrain" value="" />

                 </label>
              </td>
              <td id="WC_CachedHeaderDisplay_TableCell_612">
                 <a
href="javascript:document.CatalogSearchForm.submit()"
id="WC_CachedHeaderDisplay_Link_3" class="button"><fmt:message
key="SEARCH" bundle="${storeText}" /></a>
              </td>
               </form>
           </tr></tbody>
           </table>
        </td>
*/ %>
```

3. Edit HeaderDisplay.jspf under the directory, <WCS_HOME>\ConsumerDirect\WCSTabbedNav\application, to import the new CachedHeaderDisplayWCDS.jsp, as shown in Example 10-41.

Example 10-41 New application\HeaderDisplay.jspf

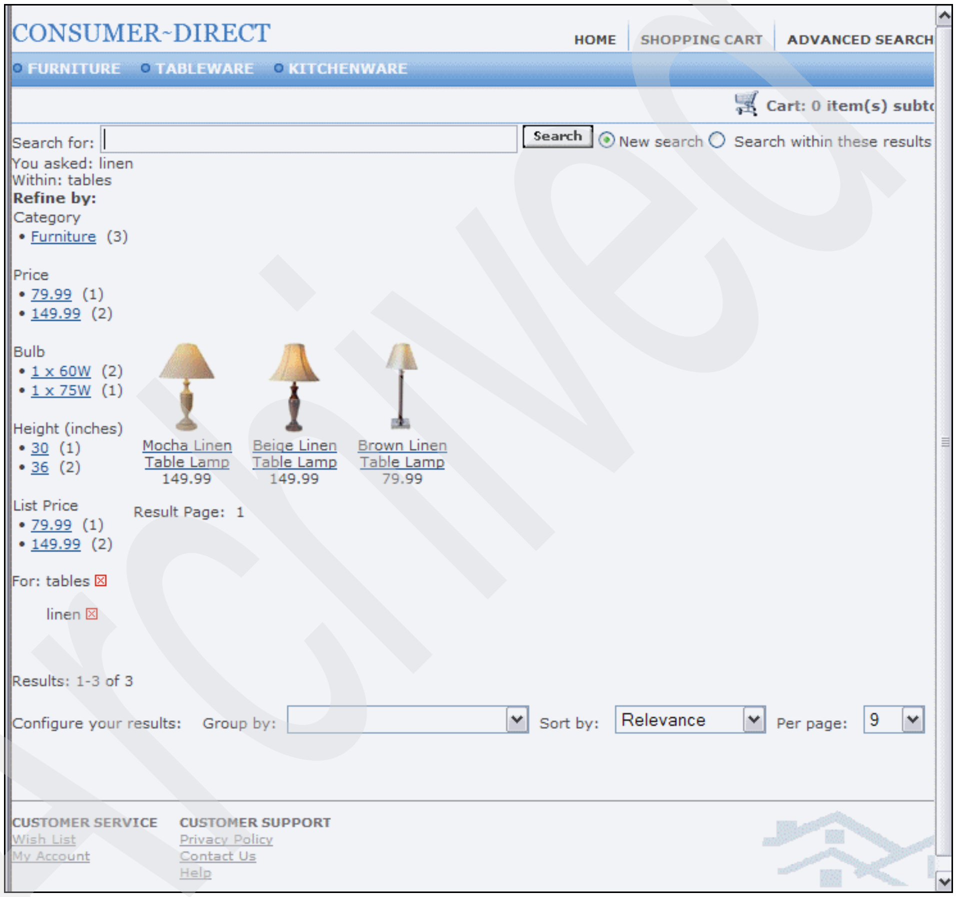
```
<link rel="stylesheet" href="<c:out
value="${jspStoreImgDir}${vfileStylesheet}"/>" type="text/css"/>

<!-- BEGIN HeaderDisplay.jspf -->

<% out.flush(); %>
<div>
<c:import url="${jspStoreDir}${StyleDir}CachedHeaderDisplayWCDS.jsp">
   <c:param name="storeId" value="${storeId}" />
   <c:param name="catalogId" value="${catalogId}" />
   <c:param name="langId" value="${langId}" />
   <c:param name="userType" value="${userType}" />
   <c:param name="userState" value="${userState}" />
   <c:param name="liveHelp" value="${liveHelp}" />
   <c:param name="rfqLinkDisplayed" value="${rfqLinkDisplayed}" />
</c:import>
</div>
```

4. Edit the globaForm.jsp file in <WCS_HOME>\ConsumerDirect\WCSTabbedNav directory and uncomment the code that was commented in “Commenting out the HTML form code” on page 244.

Tip: You might want to check for the img source location code — use the explanation in “Fixing GIF image links on the search result page” on page 258.

10.3.6 Final search result page

We are ready for the final search result page as shown in Figure 10-22. In this page, we see that all images are linked correctly. We show a search results using the *search within* function. That is, in this example, we perform a search on tables, and then search Linen within the previous result set. See the drillSideways rendering on the upper left corner. The WebSphere Commerce search form is hidden. The OmniFind Discovery Edition search form is shown.



Search for: [ ] [✓ New search] Search within these results

You asked: linen
Within: tables
Refine by:
Category
- Furniture (3)

Price
- 79.99 (1)
- 149.99 (2)

Bulb
- 1 x 60W (2)
- 1 x 75W (1)

Height (inches)
- 30 (1)
- 36 (2)

List Price
- 79.99 (1)
- 149.99 (2)

For: tables ☒

linen ☒

Mocha Linen Table Lamp 149.99
Beige Linen Table Lamp 149.99
Brown Linen Table Lamp 79.99

Result Page: 1

Results: 1-3 of 3

Configure your results: Group by: [ ] Sort by: Relevance Per page: 9

Figure 10-22 Final TabbedNav search result page

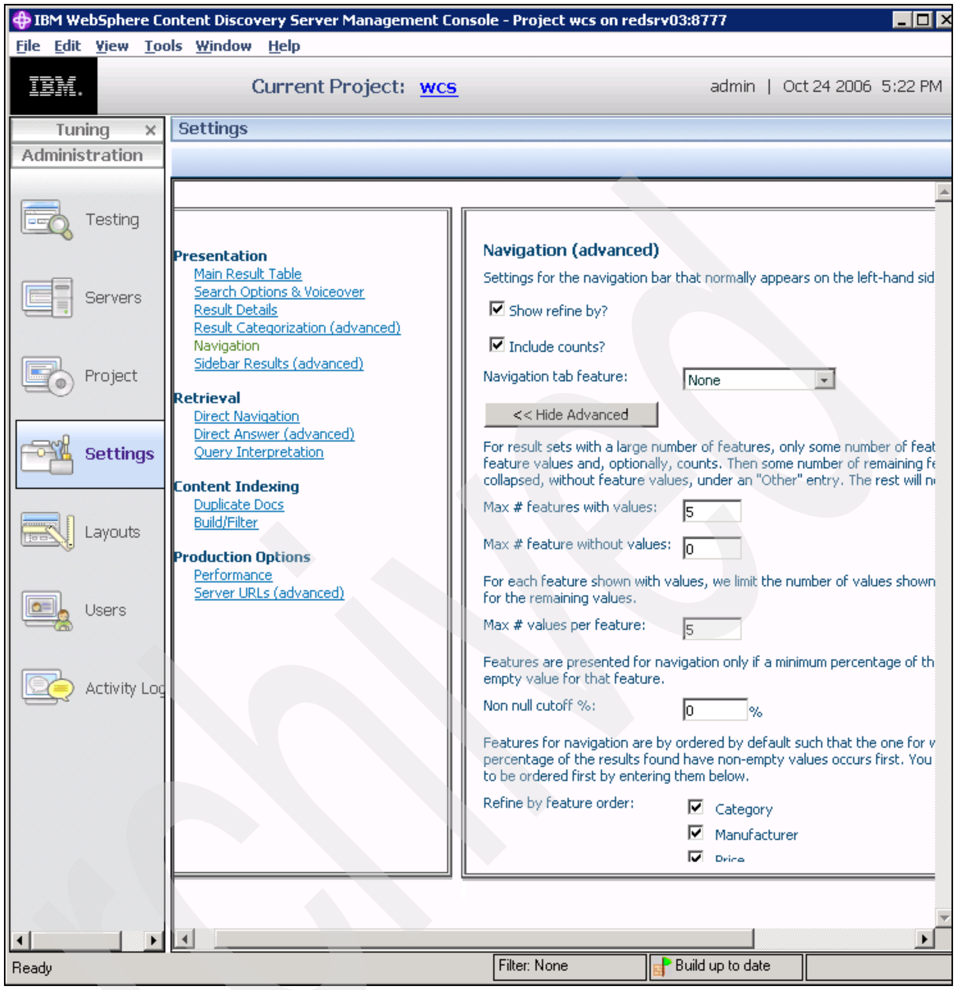
10.3.7 Customization using Management Console

In this section, we show you how you can use OmniFind Discovery Edition Management Console to customize the results displayed in the refine-by list on the left hand side panel. For more details on how to start the Management Console and edit the administration properties, refer to 7.3.1, “Starting the Management Console” on page 136.

Note: Make sure OmniFind Discovery Edition is running in the edit mode.

Do the following steps:

1. Select the Management Console, by selecting **Start** → **Programs** → **IBM WebSphere Content Discovery Server 8.3** → **Management Console**.
2. Select **Administration** → **Settings** → **Navigation** → **Advanced**.
3. Set the property variables shown in Figure 10-23. This allows OmniFind Discovery Edition server to do the following:
   - Renders a maximum of five features which have non-null values.
   - Render a maximum of five values per feature.
   - Do not render any features with zero values.


IBM WebSphere Content Discovery Server Management Console - Project wcs on redsrv03:8777

Presentation
- Main Result Table
- Search Options & Voiceover
- Result Details
- Result Categorization (advanced)
- Navigation
- Sidebar Results (advanced)

Retrieval
- Direct Navigation
- Direct Answer (advanced)
- Query Interpretation

Content Indexing
- Duplicate Docs
- Build/Filter

Production Options
- Performance
- Server URLs (advanced)

Navigation (advanced)

Settings for the navigation bar that normally appears on the left-hand sid

[✓] Show refine by?

[✓] Include counts?

Navigation tab feature: None

<< Hide Advanced

For result sets with a large number of features, only some number of feat
feature values and, optionally, counts. Then some number of remaining fe
collapsed, without feature values, under an "Other" entry. The rest will n

Max # features with values: 5

Max # feature without values: 0

For each feature shown with values, we limit the number of values shown
for the remaining values.

Max # values per feature: 5

Features are presented for navigation only if a minimum percentage of th
empty value for that feature.

Non null cutoff %: 0 %

Features for navigation are by ordered by default such that the one for v
percentage of the results found have non-empty values occurs first. You
to be ordered first by entering them below.

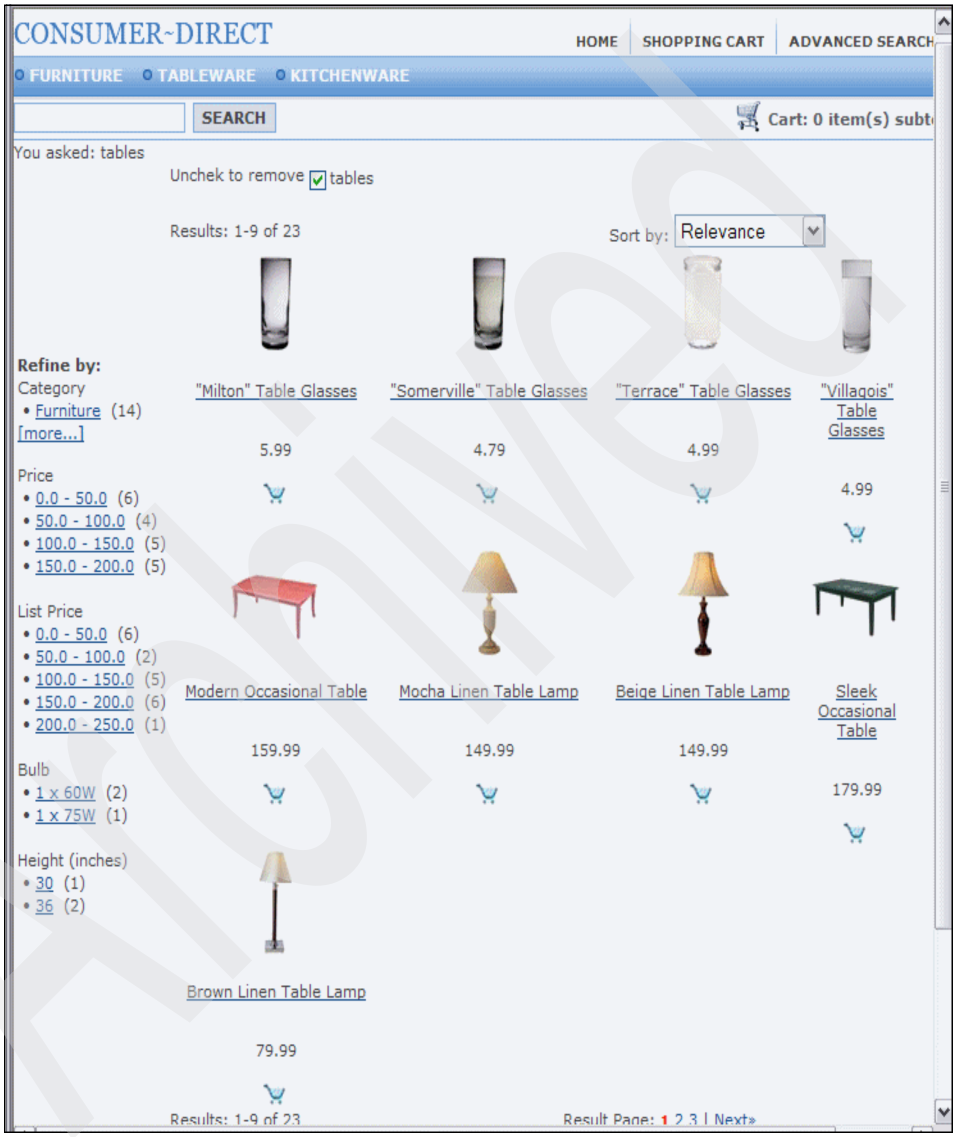
Refine by feature order:
- [✓] Category
- [✓] Manufacturer
- [✓] Price

Figure 10-23 Setting Management Console RefineBy List property variables

10.4 Customization using Commerce Layout skin

OmniFind Discovery Edition product also comes along with an out-of-the-box Commerce Layout skin, just like the out-of-the-box TabbedNav skin that we used to do our customization. The Commerce Layout skin is designed to give more of a Commerce appearance and behavior, as opposed to the TabbedNav skin, which is generic and contains the default rendering search results returned by OmniFind Discovery Edition.

We managed to get the Commerce Layout skin to work with WebSphere Commerce with a similar customization, as we described in 10.3, “Customization using TabbedNav skin” on page 216. The Commerce layout search results page is shown in Figure 10-24.


CONSUMER~DIRECT

HOME | SHOPPING CART | ADVANCED SEARCH

FURNITURE | TABLEWARE | KITCHENWARE

SEARCH | Cart: 0 item(s) subt

You asked: tables

Unchek to remove ✓ tables

Results: 1-9 of 23

Sort by: Relevance

Refine by:

Category
- Furniture (14)

[more...]

Price
- 0.0 - 50.0 (6)
- 50.0 - 100.0 (4)
- 100.0 - 150.0 (5)
- 150.0 - 200.0 (5)

List Price
- 0.0 - 50.0 (6)
- 50.0 - 100.0 (2)
- 100.0 - 150.0 (5)
- 150.0 - 200.0 (6)
- 200.0 - 250.0 (1)

Bulb
- 1 x 60W (2)
- 1 x 75W (1)

Height (inches)
- 30 (1)
- 36 (2)

"Milton" Table Glasses 5.99

"Somerville" Table Glasses 4.79

"Terrace" Table Glasses 4.99

"Villagois" Table Glasses 4.99

Modern Occasional Table 159.99

Mocha Linen Table Lamp 149.99

Beige Linen Table Lamp 149.99

Sleek Occasional Table 179.99

Brown Linen Table Lamp 79.99

Results: 1-9 of 23

Result Page: 1 2 3 | Next»

Figure 10-24   Commerce Layout skin working with WebSphere Commerce

10.5 Troubleshooting and debugging hints and tips

There are a varieties of log files that you can look into, both from the WebSphere Commerce and OmniFind Discovery Edition sides, in case you encounter problems and exceptions during your customization. Here, we list the log files that are helpful in troubleshooting and debugging problems.

WebSphere Commerce specific log files:

- <WAS_HOME>\profiles\demo\logs\server1\SystemOut.log
- <WAS_HOME>\profiles\demo\logs\server1\systemErr.log

OmniFind Discovery Edition specific log files:

- <ODE_HOME>\deployment\wcs\wcs-logs\server_logs\serverChild.0.stderr.log
- <ODE_HOME>\deployment\wcs\wcs-logs\server_logs\serverChild.0.stdout.log

In addition to the foregoing log files, you can run the OmniFind Discovery Edition integration code in debug mode to see the results of all features and their values based on the parameters and property variables set in the Management Console.

Part 3

Appendixes

A

Additional material

This book refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this book is available in softcopy on the Internet from the Redbooks Web server. Point your Web browser to:

```
ftp://www.redbooks.ibm.com/redbooks/SG247358
```

Alternatively, you can go to the Redbooks Web site at:

```
ibm.com/redbooks
```

Select the **Additional materials** and open the directory that corresponds with the Redbooks form number, SG247358.

Using the Web material

The additional Web material that accompanies this book includes the following files:

File name	*Description*
SG247358.zip	Source code and data for the sample starter store

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space:	200 MB minimum
Operating System:	Windows
Processor:	Pentium® IV or higher
Memory:	512 MB

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

For information about ordering these publications, see "How to get IBM Redbooks" on page 274. Note that some of the documents referenced here might be available in softcopy only.

- *Best Practices and Tools for Creating WebSphere Commerce Sites,* SG24-6699
- *e-Commerce Hosting Solutions Guide, Using WebSphere Commerce V5.5 Business Edition*, SG24-7018

Other publications

These publications are also relevant as further information sources:

- *IBM OmniFind Discovery Edition Installation Guide*, GC18-9815
- *IBM OmniFind Discovery Edition Management Console Guide*, GC18-9814
- *IBM OmniFind Discovery Edition User's Guide*, GC18-9813
- *IBM Commerce Module for OmniFind Discovery Edition User's Guide*, GC18-9818

Online resources

These Web sites are also relevant as further information sources:

- Information center for OmniFind Discovery Edition Version 8.4:

  `http://publib.boulder.ibm.com/infocenter/discover/v8r4/topic/com.ibm.discovery.ds.nav.doc/cdsnav_welcome.html`

- Primary product support resource for OmniFind Discovery Edition:

  `http://www.ibm.com/software/data/enterprise-search/omnifind-discovery/support.html`

- Publication library for OmniFind Discovery Edition Version 8.4:

  `http://www.ibm.com/support/docview.wss?rs=3035&uid=swg27008552`
- Product support for OmniFind Discovery Edition:

  `http://www.ibm.com/software/data/enterprise-search/omnifind-discovery/support.html`
- Information center for WebSphere Commerce Version 6.0:

  `http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0`

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads:

ibm.com/support

IBM Global Services:

ibm.com/services

Index

IBM

Redbooks

Getting Started with Commerce Module for OmniFind Discovery Edition

(0.5" spine)
0.475"<->0.875"
250 <-> 459 pages

Getting Started with Commerce Module for OmniFind Discovery Edition Customization 101

Concepts and overview

Introducing basic APIs

Customizing search and navigation

IBM Commerce Module for OmniFind Discovery Edition integrates IBM OmniFind Discovery Edition with IBM WebSphere Commerce. The integration helps online retail and catalog companies convert shoppers into buyers by making it easier for people to find the right products and services that match their specific requirements from the Web.

This IBM Redbooks publication provides an introduction to customizing the Commerce Module for OmniFind Discovery Edition using OmniFind Discovery Edition APIs. We introduce some of the basic APIs and show examples of using them to run simple searches, refine search results, and navigate within the search results.

Using a sample start store as the case study, we explain how to use the APIs to add or modify existing menus and navigation options and change the appearance and behavior of the existing sample store Web site. We show you how to work with OmniFind Discovery Edition search engine and pass multiple constraints to narrow searches for the online store.

This book is intended to be used by system integrators and solution developers who will be working with the Commerce Module for OmniFind Discovery Edition and performing customization for the online stores powered by OmniFind Discovery Edition.

SG24-7358-00 ISBN 0738486124