WebSphere® software

IBM

Deploying and Customizing IBM Sales Center for WebSphere Commerce V6

- Automated deployment with IBM Tivoli Configuration Manager and IBM WebSphere Everyplace Deployment
- User interface and role-based customization
- Customer Care integration with Sametime

Rufus Credle
Rajesh Adukkadukkath
Amit Jain
Lorilee Jarosinski
Ravindra Pratap Singh
Mojca Spazzapan
Dagmara Ulanowski

Redbooks

ibm.com/redbooks

International Technical Support Organization

IBM Sales Center for WebSphere Commerce V6

April 2007

SG24-7249-00

Note: Before using this information and the product it supports, read the information in “Notices” on page ix.

First Edition (April 2007)

This edition applies to Microsoft Windows XP, Microsoft Windows 2000 Server, Microsoft Windows 2003 Server, IBM Rational Application Developer V6.0.1.1, WebSphere Application Server Test Environment V6.0.2.5, IBM Sales Center for WebSphere Commerce V6.0, IBM WebSphere Commerce Developer V6.0, WebSphere Commerce Enterprise V6.0, DB2 Universal Database V8.2.3, IBM HTTP Server V6.0, WebSphere Application Server Network Deployment V6.0.

Contents

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

Redbooks (logo)  ™
developerWorks®
eServer™
i5/OS®
xSeries®
AFS®
BladeCenter®
Cloudscape™
Domino Designer®
Domino®
DB2 Universal Database™
DB2®
Everyplace®
IBM®
Lotus Notes®
Lotus®
Notes®
Rational®
Redbooks™
Sametime®
System x™
Tivoli Enterprise™
Tivoli®
WebSphere®
Workplace™
Workplace Managed Client™

The following terms are trademarks of other companies:

Oracle, JD Edwards, PeopleSoft, Siebel, and TopLink are registered trademarks of Oracle Corporation and/or its affiliates.

Oracle, JD Edwards, PeopleSoft, and Siebel are registered trademarks of Oracle Corporation and/or its affiliates.

Enterprise JavaBeans, EJB, Java, Javadoc, JavaBeans, JavaServer, JavaServer Pages, JDBC, JSP, JVM, J2EE, Solaris, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Internet Explorer, Microsoft, Windows Server, Windows, Win32, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, Pentium, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Preface

The IBM Sales Center for WebSphere Commerce V6 is an application for customer service representatives to capture and manage customer orders. This IBM® Redbook helps you understand IBM Sales Center for WebSphere Commerce and provides you with how-to instructions to deploy the business solution, customize it, and integrate the Sales Center with other applications.

This IBM Redbook helps you install, tailor, and configure the Sales Center development environment and production environment for creating and deploying the Sales Center customizations. In addition, this book discusses the use of IBM Tivoli Configuration Manager and IBM WebSphere Everyplace Deployment, to perform automated deployment.

This book discusses how to plan and design Sales Center customizations. Examples are provided to help you through this process. The customization scenarios that include the integration of additional IBM software and original equipment manufacturer (OEM) software are described.

This book provides user interface and role-based customization examples to demonstrate customization within the user interface framework and the role-based tools.

This book also provides code sample that you can use to integrate IBM Lotus Sametime V7.5 into Sales Center, where live help and customer care functionality are achieved.

IBM WebSphere Commerce Analyzer allows you to view analytical data and provide customer service reports. This book provides instructions about how to use this tool to gather information for analyzing data in order to help the marketing, sales, and customer service representative supervisors take more informed business decisions.

This book is useful for IT architects, IT specialists, application designers, application developers, application deployers, and consultants because it contains information that is necessary to design, develop, deploy, and customize.

The team that wrote this IBM Redbook

This IBM Redbook was produced by a team of specialists from around the world working at the International Technical Support Organization (ITSO), Poughkeepsie Center, USA.

Rufus Credle is a Certified Consulting IT Specialist at the ITSO, Raleigh Center. In his role as Project Leader, he conducts residencies and develops IBM Redbooks™ on network operating systems, enterprise resource planning (ERP) solutions, voice technology, high availability, and clustering solutions, Web application servers, pervasive computing, IBM and OEM e-business applications, IBM System x™, IBM eServer™ xSeries®, and IBM BladeCenter®. The various positions he has held during the course of his career at IBM include assignments in administration and asset management, systems engineering, sales and marketing, and IT services. He holds a BS degree in Business Management from Saint Augustine's College. He has been employed at IBM for 26 years.

Rajesh Adukkadukkath is a Staff Software Engineer in India Software Labs, Bangalore, India. He has six years of experience in software design and development of e-business, network management systems, and client-server technologies. He holds a degree in Master of Computer Applications from Bharathiar University, Coimbatore, India. He has worked extensively on IBM Sales Center development for WebSphere Commerce and his areas of expertise include Java™ and Java 2 Platform, Enterprise Edition (J2EE™) technologies, including plug-in development on the Eclipse framework.

Amit Jain is a Technical Architect in the Portal and eCommerce competency at IBM India. He has nine years of experience in consulting, software design, and development of e-business and client-server technologies. He has carried out extensive work on WebSphere Commerce customizations for several IBM customers. He has been with IBM since 2000. He holds a degree in Computer Science from Rohailkhand University, India. His areas of expertise include solution design, analysis, and development of Java and J2EE applications, and IBM WebSphere® technologies. He has practical experience in problem determination and resolution.

Lorilee Jarosinski is a Staff Software Developer at the IBM Toronto Lab. She is responsible for programming and customizing tutorials for WebSphere Commerce. She has five years of experience working on the build, development, and technical writing teams. She has written extensively on IBM Sales Center customization, including tutorials, samples, and a white paper. Lorilee holds a degree in Computer Science from York University, Toronto.

Ravindra Pratap Singh is a Software Engineer in IBM India. He has over three years of experience in the WebSphere Commerce field. He holds a Master of Computer Applications degree from Jawaharlal Nehru University, New Delhi, India. His areas of expertise include WebSphere Commerce Analyzer. He has written several articles and tutorials in IBM developerWorks® about WebSphere Commerce and WebSphere Commerce Analyzer.

Mojca Spazzapan is an Advisory Product Services Specialist for WebSphere Commerce in the Europe, Middle East, and Africa (EMEA) support team, working in IBM Slovenia since 2001. She has four years of experience in diverse WebSphere Commerce products areas, with practical experience in problem determination and resolution. She holds a Master's degree in Electrical Engineering from the University of Ljubljana, Slovenia. Mojca's areas of expertise include software programming, middleware applications, and e-commerce. She has co-authored an IBM Redbook, *WebSphere Commerce V5.4 Handbook: Architecture and Integration Guide*, SG24-6567, and an IBM Redpaper, *WebSphere Commerce V5.4 for Solaris and Oracle9i, Infrastructure and Deployment Patterns*, REDP-0316.

Dagmara Ulanowski is a WebSphere Commerce Consultant with the IBM Software Services for WebSphere team, working in IBM Canada. She has several years of experience in consulting and developing e-commerce solutions using WebSphere Commerce. She has worked on implementing the WebSphere Commerce solution for various IBM customers. She holds an Honors degree in Computer Science from York University, Toronto, Canada. Her areas of expertise include IBM Sales Center and IBM WebSphere Commerce Accelerator customization.

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

Carolyn Sneed, Tamikia Barrow
ITSO, Poughkeepsie Center

Brian Nolan, IT Architect, WebSphere Business Integration Services Planning
IBM Research Triangle Park

Bill MacIver, WebSphere Commerce Suite Sr Development Manager
IBM Markham, Canada

Carl Kaplan, Worldwide e-Commerce Sales
IBM Waltham

Anthony Tjong, Manager, WebSphere Commerce Development
IBM Markham, Canada

Michael Au, Manager, WebSphere Commerce Foundation Development
IBM Markham, Canada

Peter Swithinbank, ITSO Project Leader
IBM Hursley, UK

Andy Kovacs, Support, Quality, and Measurements
IBM Markham, Canada

Tack Tong, Markham Lab
IBM Markham, Canada

Judy Chan, WebSphere Commerce Business-to-Business Solutions
IBM Markham, Canada

Brian Thomson, STSM Performance, Scalability, Availability
IBM Markham, Canada

Glenn Jones, SWG VoIP Infrastructure, Backup AFS® Cell Admin
IBM Markham, Canada

Wai-Kong Ho, Senior IT Specialist
IBM, Australia

Jegathasan Thambipillai, End User Support
IBM Toronto

Ramya Rajendiran, Associate Software Engineer
IBM India

Become a published author

Join us for a two-week to six-week residency program! Help write an IBM Redbook 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 IBM Redbooks to be as helpful as possible. Send us your comments about this or other IBM Redbooks in one of the following ways:

- Use the online **Contact us** review IBM Redbook form found 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

Introduction

This part introduces IBM Sales Center for WebSphere Commerce V6 and discusses the functionality and value of this product.

1

IBM Sales Center for WebSphere Commerce V6.0

This chapter describes the most common call center pains and how you can resolve them using IBM Sales Center for WebSphere Commerce.

1.1 Introduction

IBM Sales Center for WebSphere Commerce is a new and separately orderable feature that leverages the catalog, order management, promotions, and merchandising capabilities of WebSphere Commerce to provide call center representatives with the functionalities they require to service and up-sell to cross-channel customers.

More than two-thirds of customers were unsatisfied with agent-assisted phone support. Many companies are finding that their existing call center applications are failing to accommodate the high volume of requests they receive and that the call center representatives have inadequate access to customer data and order data from other channels. In addition, most call center representatives are not equipped to perform cross-sell and up-sell activities, losing additional revenue opportunities. IBM Sales Center for WebSphere Commerce helps address these issues and contributes to a more efficient call center operation.

Figure 1-1 shows a sample IBM Sales Center customer editor.

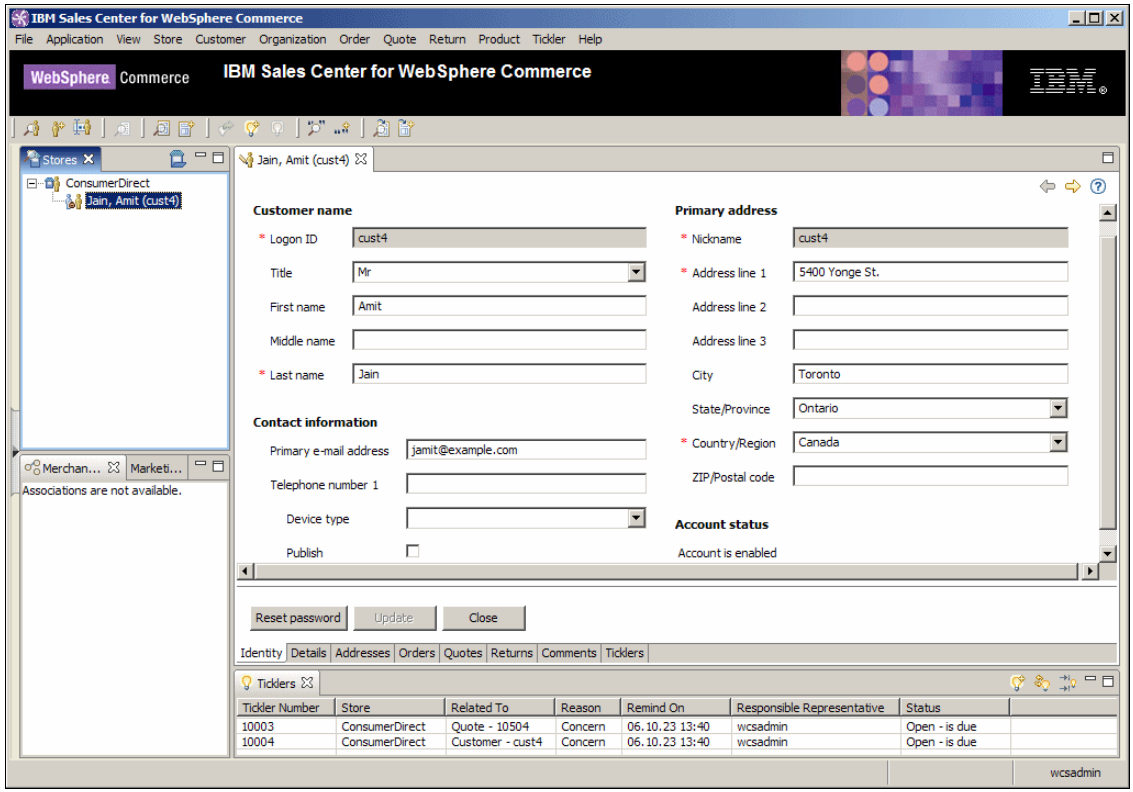


Figure 1-1 IBM Sales Center customer editor

Figure 1-2 shows a sample IBM Sales Center order editor.

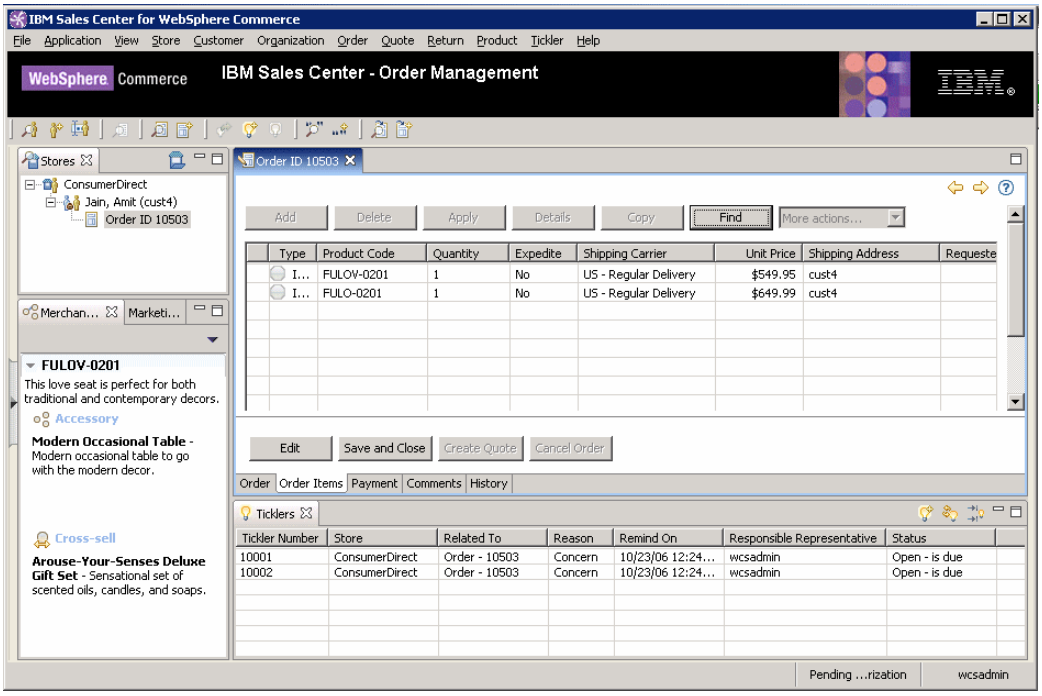


Figure 1-2 IBM Sales Center order editor

1.2 IBM Sales Center features

IBM Sales Center's key features include the following:

- Works with multiple stores, orders, and customers simultaneously
- Views cross-sell, up-sell, and promotion information
- Finds and compares products, and views product availability
- Views and takes control of customers' shopping carts
- Creates quotes and turns them into orders
- Overrides contract and list pricing
- Creates, updates, cancels orders, and processes payments
- Creates and manages profiles and ticklers (reminders)
- Integrates with other applications

1.3 IBM Sales Center benefits

IBM Sales Center provides the following benefits:

- Improves the productivity of call center employees
- Increases sales in call center through cross-sell and up-sell
- Improves service for cross-channel customers
- Reduces IT cost and complexity by using a central server for both the call center and the Web, and by reducing the number of systems requiring replicated catalog, customer, promotion, and order data

2

Overview of the IBM Sales Center environment

This chapter discusses the IBM Sales Center architecture and the functions of the IBM Sales Center environment. It also provides a comparison with the IBM WebSphere Commerce Accelerator tool.

2.1 IBM Sales Center's high-level architecture

IBM Sales Center for WebSphere Commerce consists of the IBM Sales Center client component and the WebSphere Commerce server. A large number of IBM Sales Center clients can connect to the WebSphere Commerce server using Web services. Multiple customer service representatives (CSRs) can use IBM Sales Center clients simultaneously to perform their daily tasks. IBM Sales Center accesses and updates the data in the WebSphere Commerce database. This data, for example, order and product information, is the same data accessed and updated through the storefront and the WebSphere Commerce Accelerator tool.

Figure 2-1 shows a high-level view of the IBM Sales Center clients and the WebSphere Commerce server. The IBM Sales Center clients are installed directly on the CSRs' machines unlike browser-based tools.

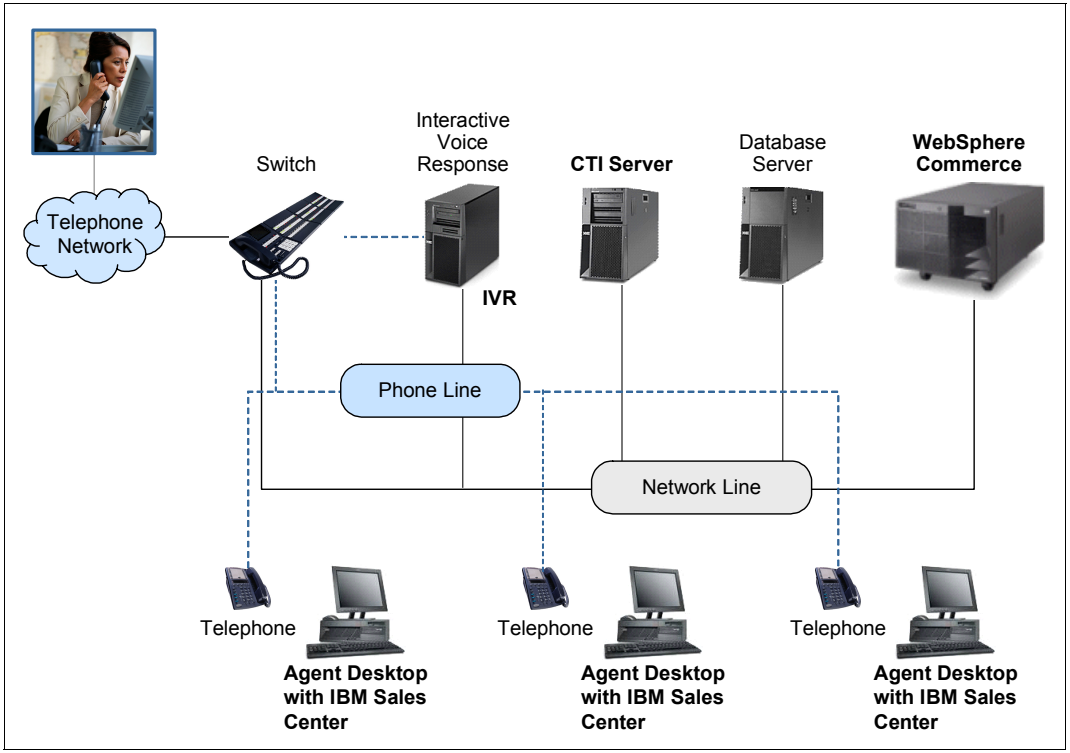


Figure 2-1 A high-level view of IBM Sales Center

2.2 IBM Sales Center's functionality

IBM Sales Center provides various functionalities for CSRs to work easily and efficiently with WebSphere Commerce. The following sections describe some of these functionalities.

2.2.1 Working with stores

With IBM Sales Center, you can work with any of the stores your role has access to, for example, WebSphere Commerce might contain a consumer direct store and a business-to-business direct store (Figure 2-2). If your role is to service the customers of the consumer direct store, you may be restricted from viewing the business-to-business direct store. Any of the stores with which you are working are listed in the Stores view. Figure 2-2 shows an example of the Stores view with three stores open, Business-to-business Direct, Consumer Direct, and Sample Business-to-business Reseller.

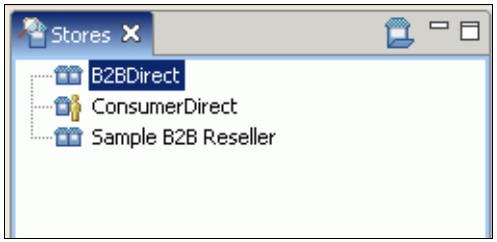


Figure 2-2 A Stores view

2.2.2 Creating new customers and working with existing customers

You can create new customers and work with existing customers in IBM Sales Center. Existing customers might have registered themselves through the storefront or might have been created by another CSR.

When the Customer editor opens, the customer name and login ID is displayed in the Stores view, as shown in Figure 2-3.

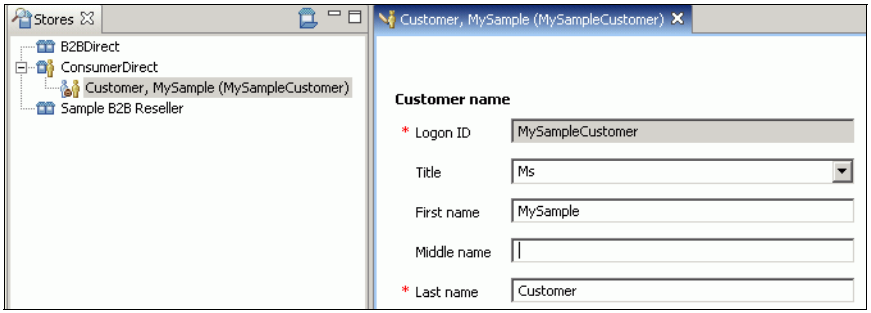


Figure 2-3 The Stores view and the Customer editor

2.2.3 Performing order-related actions

In IBM Sales Center, you can perform the following order-related actions:

- Create an order for a new or existing customer by beginning with a new order or by using the information from an existing order or quote
- Modify many of the details in an existing order such as item quantities, shipping addresses, and payment methods after the order is submitted, but before all the items in the order are fulfilled
- View and work with the merchandising associations that are related to the items in an order, and view the marketing promotions that might be of interest to the customer
- Create guest orders for direct customers who have not registered or do not want to register with the consumer direct store

  Note: Guest orders cannot be created in a business-to-business store.

- Manage manually blocked orders (A CSR can resolve manual blocks, but automatic blocks must be removed by an authorized administrator using the WebSphere Commerce Accelerator tool.)

2.2.4 Performing quote-related activities

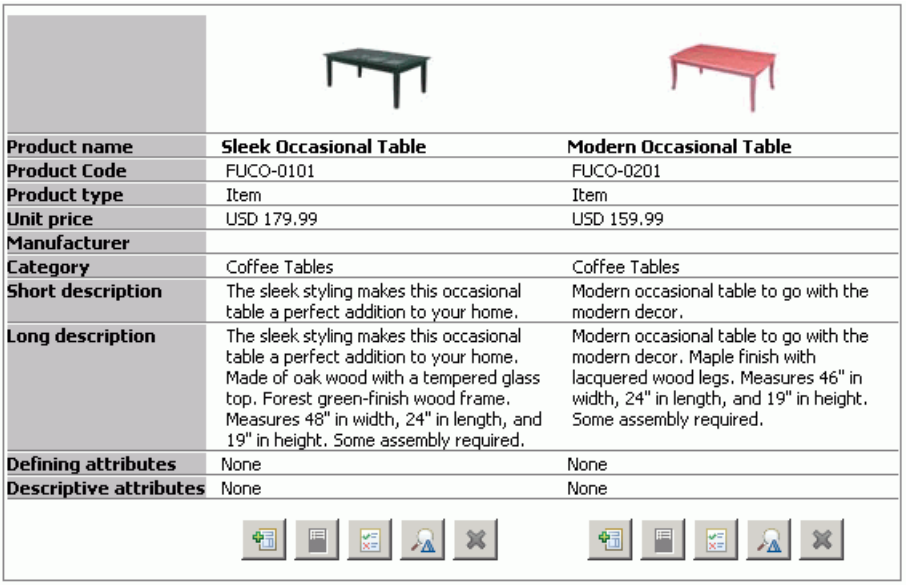
A customer might want to get a cost quotation before deciding about whether to place an order. You can perform the following quote-related activities:

- Create a quote for a new or existing customer by beginning with an empty quote or based on the information in an existing quote or order. You can also modify many of the details in an existing quote, such as item quantities and shipping addresses.
- View and work with merchandising associations relating to the items in a quote, and view marketing promotions that might be of interest to the customer.
- Blocks can be placed on a quote automatically, and you can work with these blocks to resolve any issues. Blocks on quotes do not halt the quote life cycle. However, when an order gets generated from a quote, any blocks that are applicable are transferred to the order and prevent it from being released to fulfilment until the blocks are removed by an authorized person.

2.2.5 Performing product-related activities

You can search for products, add products to an order, and perform side-by-side product comparisons (Figure 2-4) in the IBM Sales Center. The products might be grouped into bundles, packages, and static and dynamic kits, in the same way in which they are grouped in the online store. Products can also be related to each other using merchandising associations such as cross-sells and up-sells.

Merchandising associations are displayed in a view next to the Order editor so that CSRs can see both at the same time. This functionality allows CSRs to drive revenue by suggesting associated products to customers at the same time as working on an order.



Product name	**Sleek Occasional Table**	**Modern Occasional Table**
Product Code	FUCO-0101	FUCO-0201
Product type	Item	Item
Unit price	USD 179.99	USD 159.99
Manufacturer		
Category	Coffee Tables	Coffee Tables
Short description	The sleek styling makes this occasional table a perfect addition to your home.	Modern occasional table to go with the modern decor.
Long description	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.	Modern occasional table to go with the modern decor. Maple finish with lacquered wood legs. Measures 46" in width, 24" in length, and 19" in height. Some assembly required.
Defining attributes	None	None
Descriptive attributes	None	None

Figure 2-4 Side-by-side comparison of two products

2.2.6 Understanding ticklers

Ticklers are notifications and reminders for CSRs to take action. Ticklers can be created automatically, for example, when a CSR exceeds the price override limit, the order is automatically blocked and the CSR Supervisor receives a tickler to follow up on the blocked order. Ticklers can also be created manually, for example, to remind a CSR to follow up with a dissatisfied customer, a tickler can be created and assigned to the corresponding CSR. Ticklers can be assigned to self, to a specific person, to a specific group, or to a specific person within a group, depending on the system setup.

2.2.7 Understanding returns

A return, which is also referred to as a Return Merchandise Authorization, is created when a customer wants to return a product purchased earlier from a store. A CSR can create, view, and edit the returns. Editing a return includes adding items to the return, removing items from the return, and changing the credit method, credit amount, and other key information, and approving it for further processing.

2.2.8 User experience features

In addition to typical features such as logging in, logging out, and changing passwords, the IBM Sales Center offers the following functionalities:

- Views and perspectives

  Views support the editors and are used to navigate information or display the properties for an active editor. In IBM Sales Center, for example, the Stores view displays open stores, customers, orders, and quotes. Each view has its own toolbar, and can have its own context menu. Views can be opened, closed, resized, and moved to suit the work environment and habits.

  Perspectives contain a set of views used to accomplish a specific type of task, for example, in the IBM Sales Center, the IBM Sales Center - Order Management Perspective contains the views that are required by a CSR to perform daily tasks in a call center.

- Setting preferences

  Several preferences can be set in the IBM Sales Center to facilitate daily tasks, for example, if you work frequently with one store, you can set a preference so that this store opens immediately after login.

- Keyboard navigation and shortcut keys

  IBM Sales Center is accessible from a keyboard instead of the mouse to support people with disabilities and those who prefer the keyboard to the mouse. The F10 key in the keyboard, for example, accesses the menus on the main menu bar, and the combination of Ctrl+L opens the logon dialog box. You can customize the keyboard shortcuts based on your requirements. The default keyboard shortcuts are listed next to the menu items they represent, as shown in Figure 2-5.

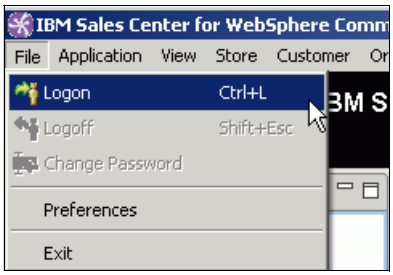


Figure 2-5 The Ctrl+L combination opens the logon dialog box

For a complete list of the default keyboard shortcuts, refer to the following Web site:

http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.telesales.tsr.doc/concepts/ctrhotkeys.htm

2.2.9 IBM Support Assistant

The IBM Support Assistant is an application that helps you find answers to your questions regarding IBM software products. Launch the IBM Support Assistant by selecting **Help** → **IBM Support Assistant** in the IBM Sales Center client running in the administrator mode. Use the IBM Support Assistant to perform the following tasks:

- Search technical notes and IBM Web resources, including newsgroups and developerWorks
- Submit service requests through a link to the IBM software support Web site, where you can create a new service request or problem management record (PMR) using the Electronic Service Request (ESR) tool.

Note: You require a valid ESR user ID and password to use the ESR tool.

2.3 IBM Sales Center default workflows

IBM Sales Center supports a variety of functions. To understand these functions, many documented business process workflows exist. Workflows are a pictorial representation of processes that help users understand the steps and their sequence in the process. An orange box in a workflow represents a high-level grouping of activities that take place in a business process. It may contain other task objects and subprocesses, creating a hierarchy. The details of the subprocesses are documented in a separate page.

As an example, the Sales Center workflow (Figure 2-6) describes the CSR tasks, the objectives and features of this process, and the customization and links to its subprocesses. CSRs inquire about how they may help customers and respond with the most appropriate actions.

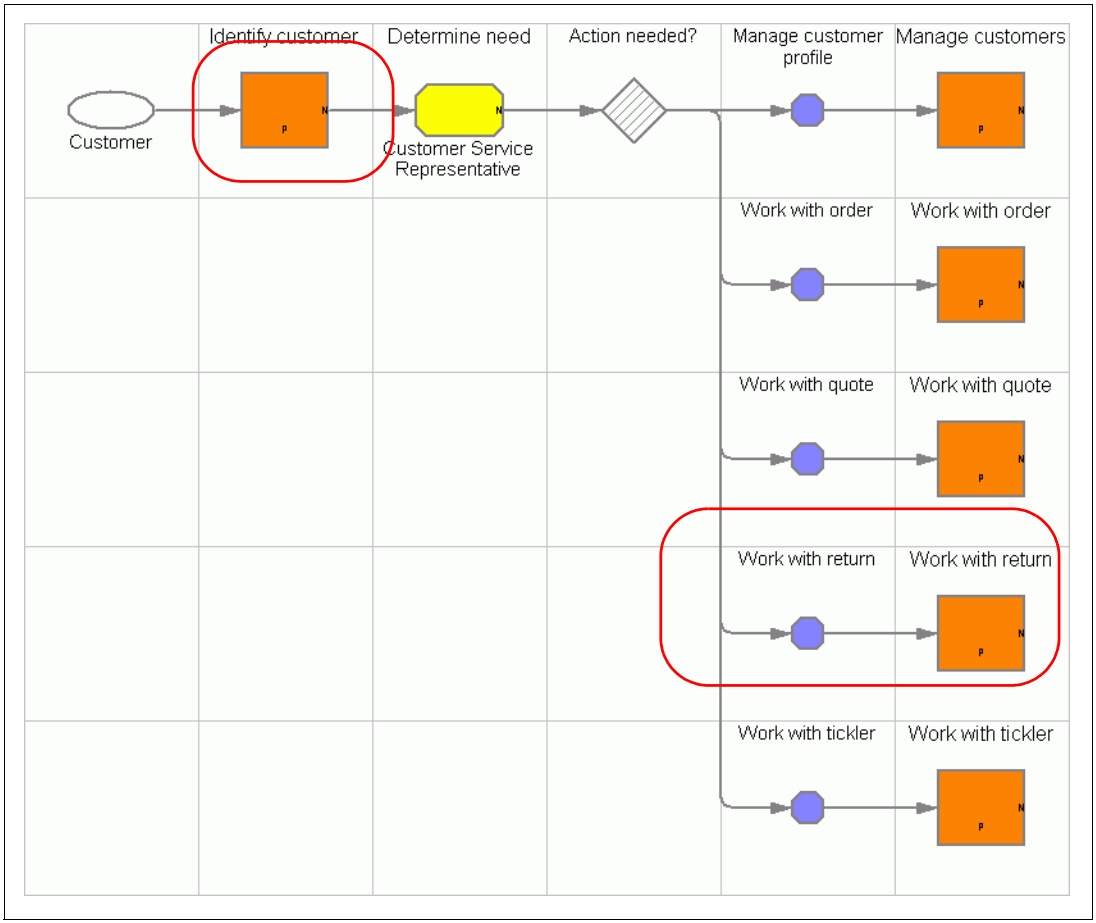


Figure 2-6 The Sales Center workflow

The *identify customer workflow* (Figure 2-7) describes the steps a CSR follows to find a customer, given certain search criteria.

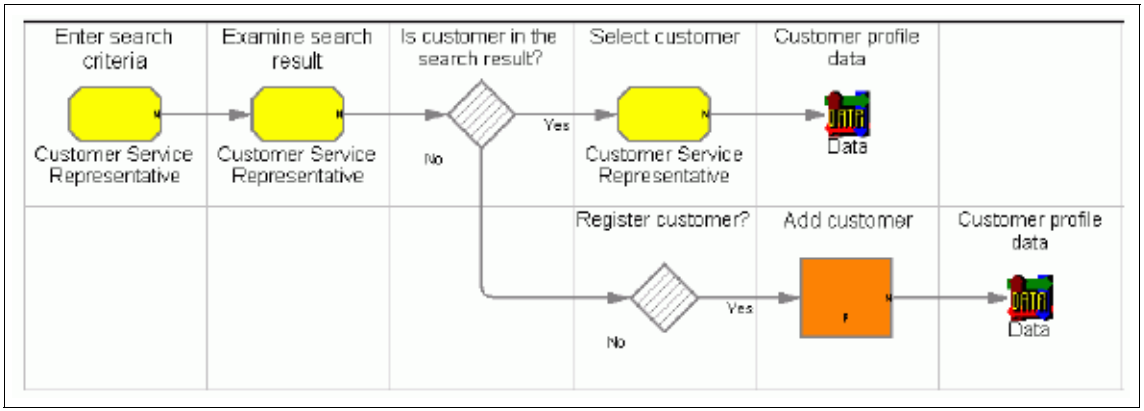


Figure 2-7 The identify customer workflow

Following is the identify customer process:

1. The CSR enters the search criteria.

   The CSR chooses the key information that is to be used to find a customer profile, and then enters the key information about a customer in order to determine whether the customer profile has already been entered into the system.

2. The CSR examines the search result.

   The CSR looks at the customer profiles that are a potential match to the customer.

3. If the customer is in the search result, the CSR performs the *Select customer* task.

   The CSR chooses the customer profile that matches the customer from among the list of potential matches, and the customer profile loads.

4. If the customer is not in the search result, the CSR can perform the *Add customer* task.

   The orange box indicates that the Add Customer workflow is described in a separate document.

The *work with return* workflow (Figure 2-8) describes the steps a CSR follows to work with a return.

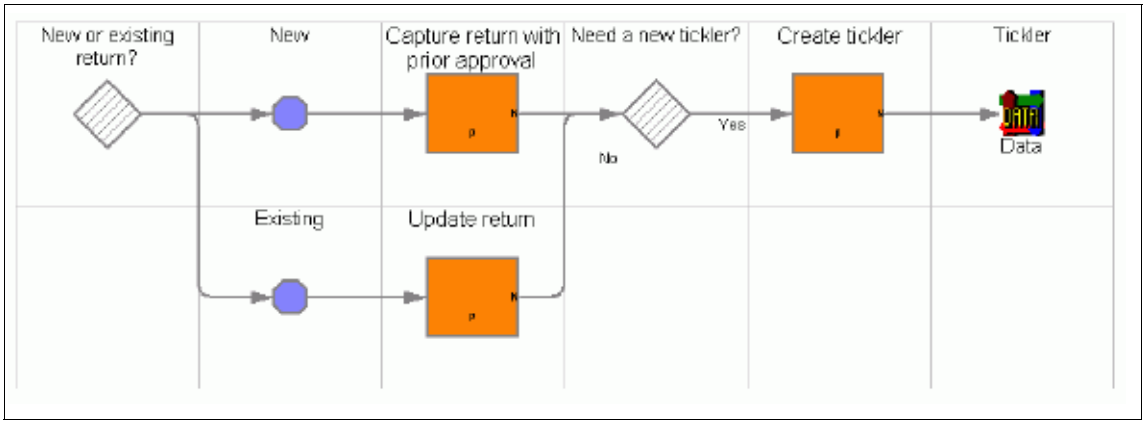


Figure 2-8 The work with return workflow

Following is the work with return process:

- If the return is a new return, the *Capture return with prior approval* process is followed. The orange box indicates that this process may contain other task objects and subprocesses that are explained in a separate page.

  If a new tickler is required, the *Create tickler* process is followed. The orange box indicates that this process may contain other task objects and subprocesses that are explained in a separate page.

- If the return is an existing return, the CSR can update the return.

  If a new tickler is required, the *Create tickler* process is followed. The orange box indicates that this process may contain other task objects and subprocesses that are explained in a separate page.

Note: To view all the workflows of the business processes, refer to the WebSphere Commerce V6 Information Center:

`http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.business_process.doc/concepts/processSales_Center.htm`

2.4 Comparing IBM Sales Center with WebSphere Commerce Accelerator

There may be some amount of confusion about when to use IBM Sales Center and when to use the WebSphere Commerce Accelerator. Both are WebSphere Commerce tools, but they are targeted at different users.

IBM Sales Center must be used by CSRs who require a highly efficient user interface for day-to-day call center tasks. It can also be used for in-store customer service tasks, where in-store staff can use it to find a user account quickly and work with a customer's privileges, for example, promotions and coupons. IBM Sales Center helps implement online order pick-up in-store and provide the in-store staff with all the information pertaining to cross-sell, promotions, and so on.

The WebSphere Commerce Accelerator tool is targeted at roles that maintain online stores, hubs, and catalogs by completing various store operations, from managing the look and feel of an online store, to creating and maintaining orders, and tracking store activities. With the WebSphere Commerce Accelerator tool, you can, for example, manage the catalogs, create marketing campaigns, and view operational reports. The WebSphere Commerce Accelerator can be used for some call center tasks. However, it does not provide the same rich and high-performance user experience as the IBM Sales Center.

Many of the functionalities of IBM Sales Center are also available in WebSphere Commerce Accelerator. However, not all the functionalities of WebSphere Commerce Accelerator are available in IBM Sales Center.

Table 2-1 shows a comparison of the IBM Sales Center and WebSphere Commerce Accelerator functionalities.

Table 2-1 Comparison of IBM Sales Center and WebSphere Commerce Accelerator functionalities

Category	IBM Sales Center	WebSphere Commerce Accelerator
Rich client-based or browser-based	Eclipse rich client that is installed on the user's machine	Browser-based Web application
Audience	CSRs who require an efficient graphical user interface (GUI)	Roles responsible for store maintenance such as Marketing Managers, Product Managers, and Returns Administrators

Category	IBM Sales Center	WebSphere Commerce Accelerator
Multitasking	Can multitask by working on multiple stores, customers, and orders simultaneously. Can see multiple views (orders, ticklers, up-sells, and so on) at the same time.	Multitasking is not possible. The user must work on one customer, order, product, and so on at a time.
Efficiency tools and role-based GUIs	Efficiency tools such as drag-and-drop, closing and hiding views that are not wanted or required for viewing, and hot keys. GUI elements can be suppressed or shown depending on the user's role.	Menu items can be displayed or hidden based on the user's role
Creating customers	Can create customers in IBM Sales Center	Cannot create a customer in Accelerator. Must use Organization Administration Console and have the proper user authority to create customers.
Performing customizations	Customizations performed by using the Eclipse framework and Standard Widget Toolkit (SWT) or by using the WebSphere Commerce's own customization framework, and Extensible Markup Language (XML) files to manipulate the user interface.	Customizations performed by using JavaServer™ pages (JSP™) and XML files
Viewing merchandising associations and promotions when taking an order	When creating an order in Sales Center, appropriate promotions and merchandising associations (up-sells, cross-sells) display in an adjacent view. CSRs do not have to remember what promotions and merchandising associations are available.	Not available in Accelerator
Reminders	Ticklers are reminders or tasks that can be assigned to other users or departments	Not available in Accelerator
Quotes	Can create quotes for a customer	Not available in Accelerator

Part 2

Installation

This part discusses and describes the installation and building of the IBM Sales Center for WebSphere Commerce development environment and production environment.

3

IBM Sales Center development environment installation

IBM Sales Center development environment and WebSphere Commerce development environment are the recommended tools for creating the IBM Sales Center client and the WebSphere Commerce server customizations.

This chapter describes the hardware, software, and networking requirements. It also addresses the development environment installation prerequisites.

This chapter describes how to install the development environment with the IBM Sales Center toolkit and the WebSphere Commerce toolkit, both as components of WebSphere Commerce Developer product installation. It then outlines the installation steps involved in different installation scenarios.

Refer to Chapter 7, “Developing customizations for IBM Sales Center” on page 119 to determine which installation scenario is appropriate for the type of customization that you plan to develop.

3.1 WebSphere Commerce Developer requirements

The development environment must comply with the hardware requirements, operating system requirements, and networking requirements described in this section.

3.1.1 Hardware requirements

This section describes the hardware requirements of the WebSphere Commerce Developer.

Tip: For updates, refer to the Technote *IBM WebSphere Commerce Developer, V6.0 hardware prerequisites*, which is available in the following Web site:

http://www-1.ibm.com/support/docview.wss?uid=swg27007490

Depending on which development environment component you are installing (IBM Sales Center, WebSphere Commerce, or both), ensure that the minimum hardware requirements for WebSphere Commerce Developer are as follows:

- An Intel® Pentium® III IBM-compatible personal computer with a minimum of 800 MHz processor

  Note: A 1.6 GHz Intel Pentium 4 processor is recommended.

- A minimum of 1.5 GB of RAM

  Note: A 2.0 GB of RAM is recommended.

- A minimum of 6.1 GB of free disk space on the target installation drive, broken down as follows:
  - IBM Rational Application Developer V6.0 requires 4.2 GB
  - WebSphere Commerce Developer requires 1.9 GB

  Note: If you choose to install WebSphere Commerce Developer only with the IBM Sales Center toolkit component (without the WebSphere Commerce toolkit), the free disk space that is required is about 100 MB.

- Enough free disk space to install the Rational Application Developer 6.0.1.1 Fix Pack:
  - 906 MB to install the fix pack directly from the IBM update server or 1.7 GB to download, extract, and install the fix pack from a compressed file if you already have Rational Application Developer Refresh Pack 6.0.1 or later installed.
  - 3.5 GB to install the fix pack directly from the IBM update server or 6.5 GB to download, extract, and install the fix pack from a compressed file if you do not already have Rational Application Developer Refresh Pack 6.0.1 or later installed.
- A graphics-capable monitor with a screen resolution of 800 x 600 display resolution

  Note: A 1024 x 768 display resolution is recommended.

- A CD-ROM drive or another media for installing the software, that is, network drive

3.1.2 Operating system requirements

This section describes the operating system requirements of the WebSphere Commerce Developer.

Tip: For updates, refer to the Technote *IBM WebSphere Commerce Developer Version 6.0 operating system prerequisites*, *w*hich is available on the Web at:

`http://www-1.ibm.com/support/docview.wss?uid=swg27007488`

Ensure that the WebSphere Commerce Developer system is running on one of the following operating systems:

- Microsoft® Windows® 2000, Server Edition with Service Pack 4
- Microsoft Windows 2000, Advanced Server Edition with Service Pack 4
- Microsoft Windows 2000, Professional Edition with Service Pack 4
- Microsoft Windows XP Professional with Service Pack 2

Note: In addition to the required operating system service pack levels, ensure that the system has the following:

- All the latest critical fixes issued by Microsoft
- Microsoft Internet Explorer® 6.0 Service Pack 1 or higher with the latest critical security updates

To obtain the latest service packs and critical fixes, refer to the Microsoft Windows Update Web site:

`http://windowsupdate.microsoft.com`

3.1.3 Networking requirements

This section describes the networking requirements of the WebSphere Commerce Developer.

Tip: For updates, refer to the Technote *WebSphere Commerce Developer Version 6.0 Networking Prerequisites*, which is available on the Web at:

`http://www-1.ibm.com/support/docview.wss?uid=swg27007489`

Depending on which development environment component you are installing (IBM Sales Center, WebSphere Commerce, or both), ensure that the network configuration of the system meets the following requirements:

- The system must have a resolvable, fully qualified host name. This host name is the host name combined with the domain name. If, for example, the host name is *system1* and the domain is *mydomain.com*, the fully qualified host name is *system1.mydomain.com*

  Issue the following command from a command prompt to return the IP address of the system:

  `nslookup` *`fully_qualified_host_name`*

  The result must be a reply with the correct IP address of the system.

- The IP address of the system must resolve to a host name, including a domain. To determine if the IP address is mapped to a fully qualified host name, start a command prompt session and issue the following command:

  `nslookup` *`IP_address`*

  The result must be a reply with the correct, fully qualified host name of the system.

- Ensure that all the nodes in the configuration can be reached from other machines in the network by pinging the fully qualified host name of each node in the configuration.

> **Note:** If you choose to install both the toolkits on the same system, you might have only one node in your configuration.
>
> If you choose to install only the IBM Sales Center toolkit, the IBM Sales Center client in the IBM Sales Center development environment is required to connect to the WebSphere Commerce server (testing environment or development environment) for customization development testing purposes, possibly running on another node in your configuration.

3.2 Prerequisites for WebSphere Commerce Developer installation

Depending on which development environment component you are installing (IBM Sales Center, WebSphere Commerce, or both), have the following software installed and configured at a minimum fix pack level:

- IBM Rational Application Developer V6.0 updated to 6.0.1.1 Fix Pack level
- WebSphere Application Server Test Environment V6.0 updated to a minimum 6.0.2.5 Fix Pack level, with the necessary fixes installed (required for WebSphere Commerce toolkit only)
- Eclipse-based IBM Sales Center for WebSphere Commerce V6.0 rich client (required for IBM Sales Center toolkit only)

> **Note:** IBM Sales Center is only available for WebSphere Commerce Professional and WebSphere Commerce Enterprise.

In our case, we followed the installation guide *WebSphere Commerce Developer Enterprise and Professional Version 6.0 Installation Guide* (GC10-4255-03) to install IBM WebSphere Commerce Developer Enterprise 6.0 with the WebSphere Commerce toolkit and the IBM Sales Center toolkit, both on the same system, running the Windows XP SP2 operating system. The following installation guide describes how to install and configure IBM WebSphere Commerce Developer V6.0, and is available for download from the IBM Publications Center site:

`http://www.elink.ibmlink.ibm.com/public/applications/publications/cgibin/pbi.cgi?CTY=US&FNC=SRX&PBL=GC1O4255`

Tip: All the product documentation is available for download from the WebSphere Commerce Developer product library page:

http://www-306.ibm.com/software/genservers/commerce/commercestudio/lit-tech-general-be-en.html#v60

The following sections outline the installation process of the WebSphere Commerce Developer V6.0 prerequisite software.

3.2.1 IBM Rational Application Developer V6.0 installation

Regardless of whether you install only the IBM Sales Center toolkit, only the WebSphere Commerce toolkit, or both the toolkits on a machine, the first step in preparing the environment for WebSphere Commerce Developer installation is to install the Rational Application Developer V6.0 on the same machine.

Tip: If you already have Rational Application Developer installed, proceed to 3.2.2, “Applying the IBM Rational Application Developer fixes” on page 34.

During the installation process, you might see the options to install software that do not ship with WebSphere Commerce Developer. The only options of Rational Application Developer that are available for install from the WebSphere Commerce Developer CDs are:

- Rational Application Developer 6.0
- WebSphere Application Server 6.0 Integrated Test Environments
- Remote Agent

To install Rational Application Developer V6.0, perform the following tasks:

1. Log in as a user with Administrator privileges.
2. Run launchpad.exe and select **Install IBM Rational Application Developer V6.0** (Figure 3-1).

Note: You can install Rational Application Developer by running launchpad.exe in the disk1 directory or by running setup.exe in the disk1\setup directory.

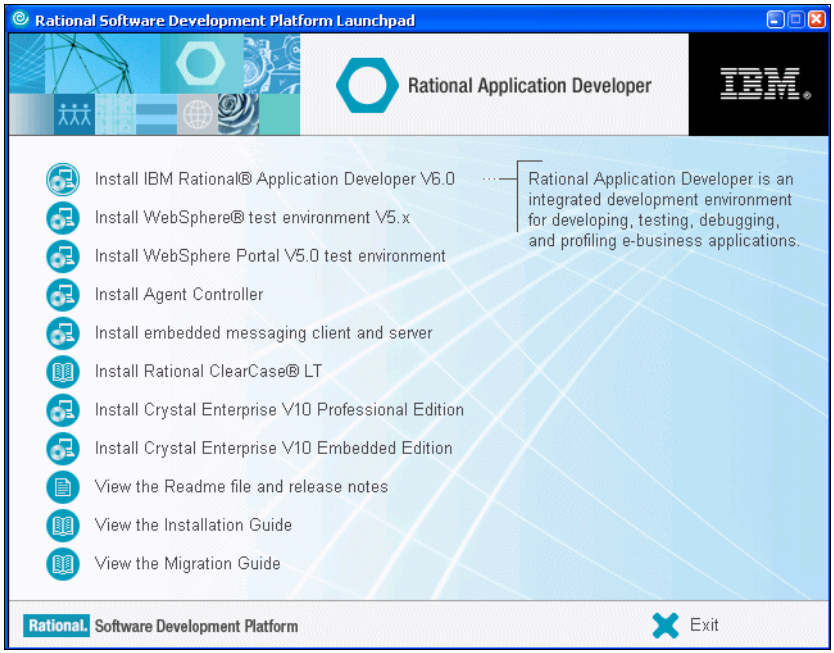


Figure 3-1 Selecting Install IBM Rational Application Developer V6.0

3. In the Welcome window, click **Next**.
4. In the window that appears, ensure that you specify a short directory name such as `C:\RAD601` and click **Next** (Figure 3-2).

Notes:

- Do not accept the default installation path for IBM Rational Application Developer V6.0. The default installation path is too long for configuration with the WebSphere Commerce development environment.
- Specify a short directory name such as `C:\RAD601`. Avoid using periods (.), spaces, or dollar signs ($) in the directory names.

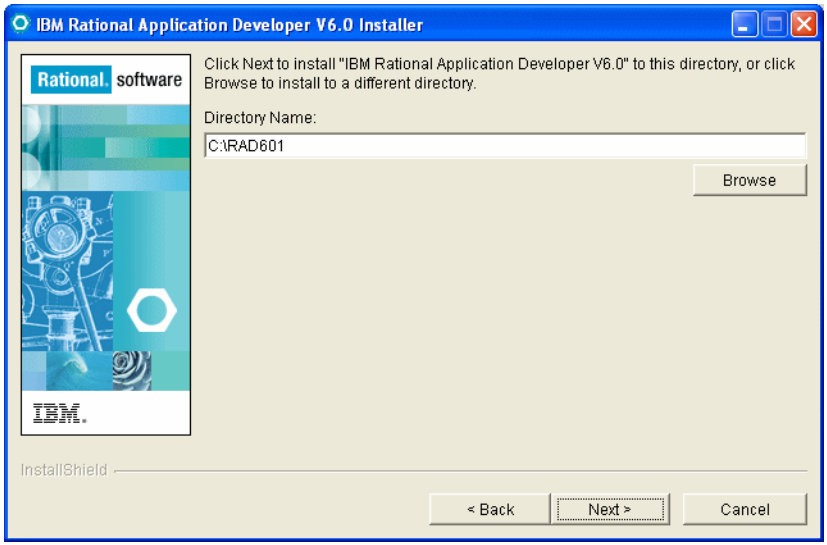


Figure 3-2 Specifying a short directory name to install Rational® Application Developer

5. When prompted to select the features that are to be installed, ensure that you select **IBM WebSphere Application Server V6.0 Integrated Test Environment**, and click **Next** (Figure 3-3).

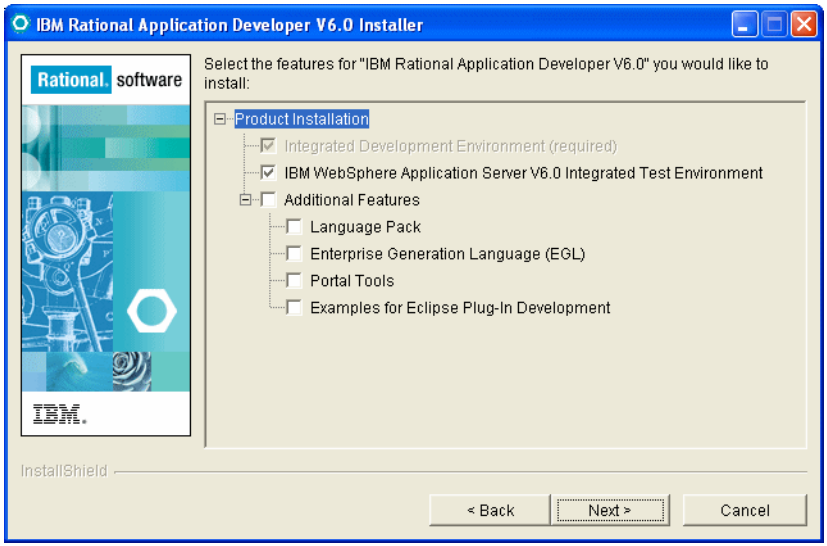


Figure 3-3 Selecting IBM WebSphere Application Server V6.0 Integrated Test Environment

6. Review the summary information (Figure 3-4) and click **Next** to begin the installation.

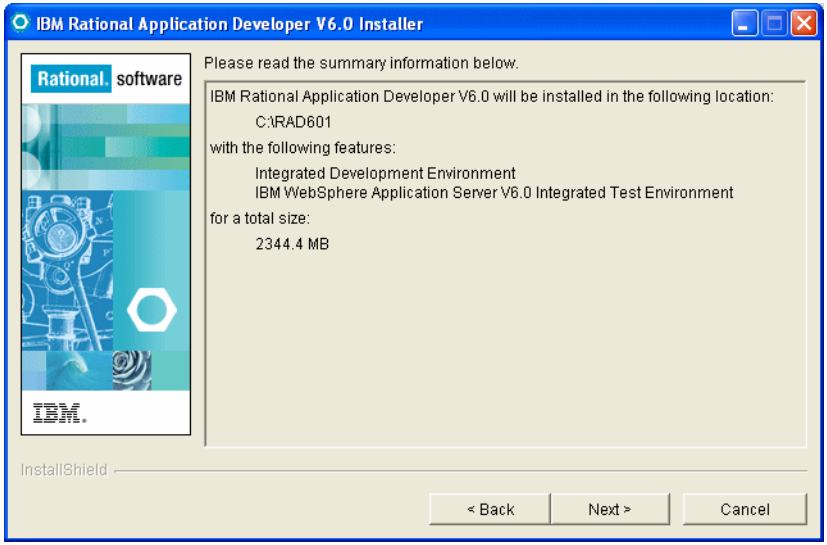


Figure 3-4 Summary information of the IBM Rational Application Developer V6.0 installation

7. After the Rational Application Developer installation is completed, you are prompted to install the Agent Controller feature (Figure 3-5). If you do not want to install this component at this juncture, deselect **Launch Agent Controller Install** and click **Finish** to exit the wizard. (In our case, we did not install this feature.)

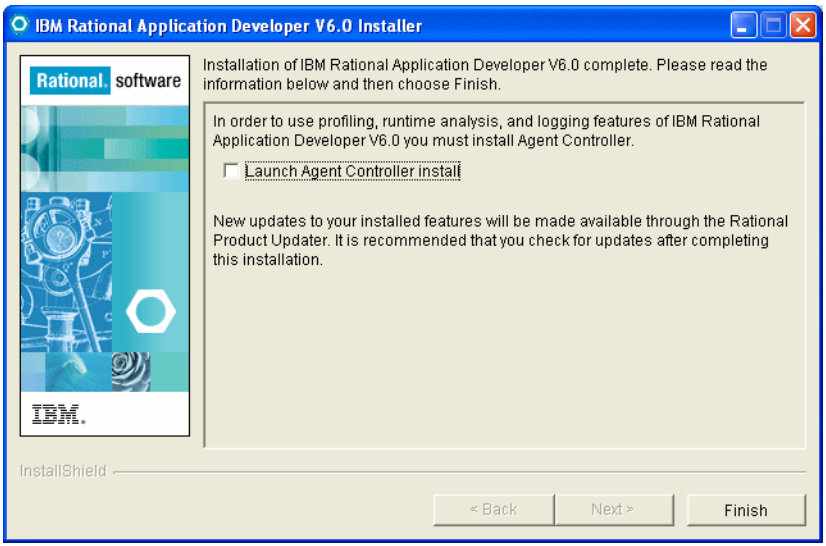


Figure 3-5 Prompt to launch the Agent Controller install

3.2.2 Applying the IBM Rational Application Developer fixes

After the installation of IBM Rational Application Developer V6.0, apply the Rational Application Developer fixes to update it to a minimum level of 6.0.1.1.

Tip: If you already have the Rational Application Developer V6.0 fixes installed, proceed to 3.2.3, “Applying the WebSphere Application Server Test Environment fixes” on page 35.

To install the IBM Rational Application Developer fix pack and fixes, perform the following tasks:

1. Open the Rational Product Updater by selecting **Start** → **Programs** → **IBM Rational** → **Rational Product Updater**.
2. Select **IBM Rational Application Developer** → **Find Updates**.
3. The Rational Product Updater detects all the applicable fixes. Choose **Select All** → **Install Updates**.

Note: Incremental updates might be required. To apply these updates, run the Rational Product Updater multiple times.

4. Accept the license agreement and click **Finish**.
5. Follow the prompts to install all the updates.

Note: Install the fix pack and the fixes directly from the IBM update server as described earlier. However, updates are also available for download. Download and install the fix pack in certain situations, such as the following:

- You have a slow or unstable Internet connection and want to use a download manager that can resume the download
- You have difficulty accessing the live IBM update servers from behind a firewall
- You prefer to download the fix pack and install it later

For information about how to download and install the fix pack, refer to the Technote *IBM Rational Application Developer Fix Pack 6.0.1.1*, which is available on the Web at:

`http://www-1.ibm.com/support/docview.wss?uid=swg24010926`

3.2.3 Applying the WebSphere Application Server Test Environment fixes

If you choose to install the WebSphere Commerce toolkit, update the WebSphere Application Server Test Environment V6.0 to the minimum 6.0.2.5 fix pack level and apply the required fixes before installing the WebSphere Commerce toolkit. The recommended way to update the WebSphere Application Server Test Environment 6.0 is by using the WebSphere Commerce installation wizard.

Tip: If you are installing only the WebSphere Commerce toolkit, and you already have the IBM WebSphere Application Server V6.0 Test Environment 6.0.2.5 or later Fix Pack, and have all the required fixes installed, proceed to 3.3, “WebSphere Commerce Developer install” on page 38.

If you are installing only the IBM Sales Center toolkit or both the toolkits, and you already have the required WebSphere Application Server Test Environment fixes installed, proceed to 3.2.4, “IBM Sales Center for WebSphere Commerce installation” on page 38.

There are several WebSphere Application Server Test Environment fix pack and fix installation scenarios that are possible. Each scenario requires the application of specific fixes in order to bring it to the appropriate level. Apply the WebSphere Application Server Test Environment fixes that are specific to your installation scenario. Refer to the Technote *WebSphere Commerce Developer, V6.0 required maintenance*, which is available on the Web at:

http://www-1.ibm.com/support/docview.wss?uid=swg21236356

If you have, for example, manually applied the 6.0.2.5 Fix Pack to your WebSphere Application Server Test Environment, and no required fixes were applied, refer to the Technote *Required Maintenance Scenario: WebSphere Application Server was manually installed at the version 6.0.2.5 level*, which is available on the Web at:

http://www-1.ibm.com/support/docview.wss?uid=swg21237197

Furthermore, WebSphere Application Server Test Environment can be updated to a Fix Pack V6.0.2.5 or later. If this update is made to 6.0.2.10 or later, apply one of the three solutions as described in the Technote *Installation of WebSphere Commerce Developer with WebSphere Application Server Fix Pack 6.0.2.11 fails,* before proceeding to 3.3, “WebSphere Commerce Developer install” on page 38. The Technote is available on the Web at:

http://www-1.ibm.com/support/docview.wss?uid=swg21243206

Perform the following tasks to install the required fixes to the WebSphere Application Server V6.0 Test Environment by using the WebSphere Commerce installation wizard:

1. Log in as a user with Administrator privileges.
2. Stop all the Java applications running on the machine.
3. Insert the WebSphere Application Server CD disk2 from the WebSphere Commerce package, which is provided with WebSphere Commerce Developer, into the CD-ROM drive of the machine on which Rational Application Developer is installed. Run install.exe and click **Next** in the welcome window.
4. Accept the terms and conditions shown in the agreement window and click **Next**.
5. After verifying that your system has completed the prerequisites check successfully, click **Next**.

6. When prompted by the install wizard, select **Apply maintenance and add features** (Figure 3-6**)**. Ensure that you select the WebSphere Application Server that is used by Rational Application Developer, for example, **<RAD_*installdir*>\runtimes\base_v6**. Click **Next**.

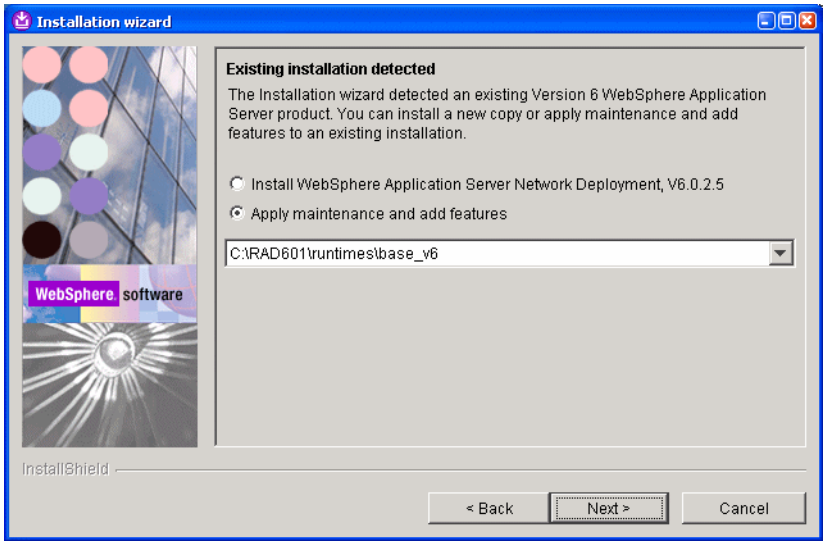


Figure 3-6 Selecting the Apply maintenance and add features option

> **Tip:** If your WebSphere Application Server V6.0 Test Environment is not detected properly as <RAD_*installdir*>\runtimes\base_v6, that is, C:\RAD*601*\runtimes\base_v6, verify whether the WebSphere Application Server Test Environment was selected to be installed at the time of Rational Application Developer installation (Figure 3-3 on page 33).
>
> If the WebSphere Application Server Test Environment was not installed during the Rational Application Developer installation, install the IBM WebSphere Application Server V6.0 Test Environment manually.
>
> Refer to WTE V6 MANUAL INSTALL instructions in the Technote *Install of the Rational Application Developer 6.x WebSphere Test Environment 6.0 server failed on Windows Operating System*, which is available on the Web at:
>
> http://www-1.ibm.com/support/docview.wss?uid=swg21209120

7. When the update installation wizard is completed, click **Finish**.

3.2.4 IBM Sales Center for WebSphere Commerce installation

If you choose to install the IBM Sales Center toolkit, install the IBM Sales Center for WebSphere Commerce client before installing the IBM Sales Center toolkit.

Install the IBM Sales Center for WebSphere Commerce client by using interactive install:

1. Insert the IBM Sales Center for WebSphere Commerce CD into the CD-ROM drive.
2. Browse the CD-ROM drive and run the setup.exe.
3. Follow the prompts in the installation wizard.

For detailed information about installing the client IBM Sales Center for WebSphere Commerce, refer to Chapter 4, “IBM Sales Center production environment installation” on page 45.

Note: Microsoft Windows XP is the only supported platform for the IBM Sales Center client. For updates, refer to the Technote *IBM WebSphere Commerce, version 6.0 operating system prerequisites*, which is available on the Web at:

http://www-1.ibm.com/support/docview.wss?uid=swg27007429

3.3 WebSphere Commerce Developer install

IBM Sales Center toolkit and WebSphere Commerce toolkit are the components of the WebSphere Commerce Developer product, and are used to create IBM Sales Center and WebSphere Commerce customizations, as described in Chapter 7, “Developing customizations for IBM Sales Center” on page 119.

Note: The IBM Sales Center toolkit is only available for WebSphere Commerce Developer Professional and WebSphere Commerce Developer Enterprise. Use the WebSphere Commerce Developer that matches your WebSphere Commerce server version:

- WebSphere Commerce Developer Enterprise with WebSphere Commerce Enterprise
- WebSphere Commerce Developer Professional with WebSphere Commerce Professional
- WebSphere Commerce Developer Express with WebSphere Commerce - Express

Depending on the type of the customizations you will be developing, you can install only the IBM Sales Center toolkit, the WebSphere Commerce toolkit, or both the toolkits. Also, depending on your hardware performance, you can install the IBM Sales Center toolkit and the WebSphere Commerce toolkit on the same system, or can have the two development environments installed on two different machines.

Note: The IBM Sales Center development environment is launched in another workspace that is separate from the WebSphere Commerce development environment workspace. If you choose to install both the toolkits in the same Rational Application Developer (on the same system), you will be able to launch each development environment separately.

The WebSphere Commerce Developer is built on Rational Application Developer V6.0. In addition to Rational Application Developer V6.0.1.1, which is required for both the toolkits, the WebSphere Commerce development environment requires a WebSphere Application Server Test Environment at a minimum 6.0.2.5 Fix Pack level, with all the required fixes installed. The IBM Sales Center development environment requires IBM Sales Center for WebSphere Commerce (Eclipse-based rich client) to be installed on the same machine.

The IBM Sales Center development environment is configured to launch the plug-in development environment perspective (default). The target platform is set to the IBM Sales Center client, and a runtime workbench configuration is available to launch the IBM Sales Center application (either in debug or normal mode). No projects are available in the workspace initially.

The WebSphere Commerce development environment is configured to launch the Java 2, Enterprise Edition (J2EE) perspective (default). The WebSphere Commerce Test Server is created as part of the WebSphere Commerce Developer installation in order to deploy and test the customized code. You can start and stop the WebSphere Commerce Test Server within the Rational Application Developer graphical user interface (GUI). However, you might want to run the WebSphere Commerce Test Server without running the Rational Application Developer GUI. You might, for example, want to test the code in your development environment and do not want to examine or change the code. Running the WebSphere Commerce Test Server without the Rational Application Developer GUI reduces the resources used on your system and may result in better performance.

Note: For information about starting the WebSphere Commerce Test Server from the command prompt, refer to the topic “Starting and stopping the WebSphere Commerce Test Server via command line” in the WebSphere Commerce information center, which is available on the Web at:

http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.developer.doc/tasks/tsrwcdevnogui.htm

For information about launching the IBM Sales Center development environment, refer to the topic “Launching the IBM Sales Center development environment” in the WebSphere Commerce information center, which is available on the Web at:

http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.telesales.developer.doc/tasks/ttrdevlaunch.htm

3.3.1 Installing both the toolkits on the same machine

For moderate and extensive customizations, install both the toolkits, the IBM Sales Center toolkit and the WebSphere Commerce toolkit, either on two separate machines, or on the same machine, depending on your hardware performance.

To install WebSphere Commerce Developer Enterprise V6.0 with the WebSphere Commerce toolkit and the IBM Sales Center toolkit components, both on the same machine using GUI installation, perform the following tasks:

1. Log in as a user with Administrator privileges.
2. Install the prerequisite software as outlined in 3.2, “Prerequisites for WebSphere Commerce Developer installation” on page 29.
3. Ensure that no Java applications are running and insert the WebSphere Commerce Developer CD in the CD-ROM of the Rational Application Developer machine. (If the WebSphere Commerce toolkit installation wizard does not start automatically, run setup.exe in the root of the WebSphere Commerce Developer CD.)
4. Select the **Language** to be used for the install wizard, and click **OK**.
5. In the welcome window click **Next**.
6. Accept the terms in the agreement window and click **Next**.
7. Specify a short directory name such as `C:\WCToolkit60` and click **Next**.

Note: Long installation paths can cause errors with some WebSphere Commerce additional software. Use a short directory name such as `C:\WCToolkit60` or `C:\WebSphere\WCToolkit60`.

8. Select the WebSphere Commerce development environment and the IBM Sales Center development environment components (Figure 3-7) and click **Next**.

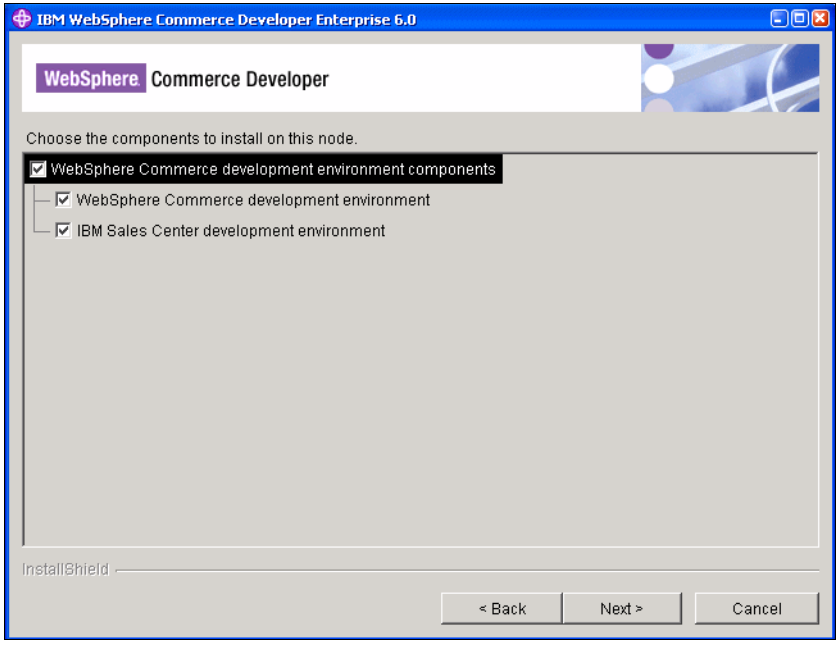


Figure 3-7 Selecting the components to install

9. Verify the summary information and click **Next**.
10. Verify that the installation is successful and click **Finish**.

Tip: To verify whether the installation is successful, perform the following tasks:

1. For the WebSphere Commerce development environment installation, examine the contents of the <*WCDE_installdir*>/logs/setup.log

   For the IBM Sales Center development environment installation, examine the contents of <*WCDE_installdir*>/logs/setupmsc.log

2. If these logs do not exist, are empty, or have errors, try running <*WCDE_installdir*>/bin/setup.bat and <*WCDE_installdir*>/bin/setupmsc.bat manually from the command line.

3. Examine the <*WCDE_installdir*>/bin/setenv.bat file to ensure that the RAD_HOME environment variable points to the correct Rational Application Developer installation location, and that the WED_HOME environment variable points to the correct IBM Sales Center for WebSphere Commerce (rich client) location, for example, setenv.bat sets these variables to:

   ```
   RAD_HOME=C:\RAD601
   WED_HOME=C:\WebSphere\SalesCenter60
   ```

3.3.2 Installing the IBM Sales Center toolkit in the WebSphere Commerce development environment

To develop moderate and extensive IBM Sales Center customizations, you may choose to install the IBM Sales Center toolkit on a machine with an existing WebSphere Commerce Developer, where you already have the WebSphere Commerce toolkit installed.

To add the IBM Sales Center toolkit to an existing WebSphere Commerce Developer, perform the following tasks:

1. Preinstall IBM Sales Center for WebSphere Commerce client. For more information, refer to 3.2.4, “IBM Sales Center for WebSphere Commerce installation” on page 38.
2. Run the WebSphere Commerce Developer installation wizard as described in 3.3.1, “Installing both the toolkits on the same machine” on page 40. When performing the task described in step 8 on page 41, opt to install only the IBM Sales Center development environment component.
3. Open the file <*WCDE_installdir*>/bin/setenv.bat in a text editor. Ensure that the RAD_HOME environment variable points to the correct Rational Application Developer installation location, for example, RAD_HOME=*C:\RAD601*.

Ensure that the WED_HOME environment variable points to the correct IBM Sales Center for WebSphere Commerce client location, for example, WED_HOME=*C:\WebSphere\SalesCenter60*

4. Save the file and exit.

3.3.3 Installing only the IBM Sales Center toolkit

For moderate and extensive customizations where you choose to install the IBM Sales Center toolkit and the WebSphere Commerce toolkit on two separate machines, and for simple customizations where you only require the IBM Sales Center toolkit, you can install the IBM Sales Center toolkit on a machine only as part of the WebSphere Commerce Developer installation.

Note: You can create the IBM Sales Center development environment without the WebSphere Commerce toolkit component. However, in order to test the customized IBM Sales Center code, you must log in from the IBM Sales Center client, which is a part of the IBM Sales Center development environment, to the WebSphere Commerce server. This can be a WebSphere Commerce Test Server running within the WebSphere Commerce toolkit on the same system or on another system as the IBM Sales Center toolkit. It can also be a runtime WebSphere Commerce server, running on the same system or on another system as the WebSphere Commerce Developer when no WebSphere Commerce customizations are required, for example, when developing simple IBM Sales Center customizations.

To install only the IBM Sales Center toolkit, perform the following tasks:

1. Preinstall Rational Application Developer as outlined in 3.2, “Prerequisites for WebSphere Commerce Developer installation” on page 29.
2. Preinstall IBM Sales Center for WebSphere Commerce. For more information about this, refer to 3.2.4, “IBM Sales Center for WebSphere Commerce installation” on page 38.
3. Run the WebSphere Commerce Developer installation wizard as described in 3.3.1, “Installing both the toolkits on the same machine” on page 40. When you perform the task described in step 8 on page 41, opt to install only the IBM Sales Center development environment component.

4

IBM Sales Center production environment installation

The IBM Sales Center production environment may involve anywhere from tens to thousands of clients. The enterprise requires a scalable and repeatable way in which to distribute and update IBM WebSphere Everyplace® Deployment for Windows and Linux® (WED4WL) and the IBM Sales Center components, and to distribute their customizations.

This chapter describes the basic hardware, software, and networking requirements for the IBM Sales Center production environment. It provides details about the production environment installation prerequisites, security considerations, and different distribution mechanisms.

The chapter then outlines the quick install of the client with manual installation tasks, and the manual installation of IBM Sales Center customizations and IBM Sales Center updates, for example, fix pack installation using the Eclipse Update Manager.

The last part of this chapter describes how to create a fully automated production environment using automated deployment with IBM Tivoli Configuration Manager and IBM WebSphere Everyplace Deployment.

4.1 IBM Sales Center client requirements

The production environment must comply with the hardware requirements, operating system requirements, and networking requirements described in this section.

4.1.1 Hardware requirements

This section describes the hardware requirements for installing IBM Sales Center for WebSphere Commerce:

- An Intel Pentium III IBM-compatible personal computer with the following minimum hardware requirements:
  - A minimum of 384 MB of RAM
  - A minimum of 350 MB of free disk space on the target installation drive
- A graphics-capable monitor with a screen resolution of 1024 x 768 display resolution

Tip: For updates, refer to the Technote *IBM WebSphere Commerce, version 6.0 hardware prerequisites*, which is available on the Web at:

`http://www-1.ibm.com/support/docview.wss?uid=swg27007428`

4.1.2 Operating system requirements

This section describes the operating system requirements to install the IBM Sales Center for WebSphere Commerce.

Tip: For updates, refer to the Technote IBM *WebSphere Commerce V6.0 operating system prerequisites*, which is available on the Web at:

`http://www-1.ibm.com/support/docview.wss?uid=swg27007429`

Ensure that you have Microsoft Windows XP with Service Pack 1, which is the minimum operating system, installed.

Note: In addition to the required operating system service pack levels, ensure that your system has all the latest critical fixes issued by Microsoft. To obtain the latest service packs and critical fixes, refer to the Microsoft Windows Update Web site:

`http://windowsupdate.microsoft.com`

4.1.3 Networking requirements

This section describes the networking system requirements that are required to create the IBM Sales Center production environment.

In addition to the hardware and software requirements, ensure that the network configuration of the systems involved meets the following requirements:

- The WebSphere Commerce server in your configuration can be reached from all IBM Sales Center clients in the network by pinging the fully qualified host name of the server from the client computer.
- The communication port is opened.

> **Note:** The default port for communication between the IBM Sales Center client and the WebSphere Commerce server is 8000.

4.2 Prerequisites to use the IBM Sales Center client

The prerequisites for using the IBM Sales Center client in a production environment are as follows:

- The WebSphere Commerce V6.0 server must be installed and running
- At least one store must exist

> **Note:** IBM Sales Center is only available for WebSphere Commerce Professional and WebSphere Commerce Enterprise.

Also consider the security precautions with regard to the machines on which the IBM Sales Center client is installed. For more information about this, refer to the IBM Sales Center client security considerations.

The next section briefly describes the installation and configuration processes pertaining to the WebSphere Commerce V6.0 server environment.

4.2.1 WebSphere Commerce server

The first step in creating the IBM Sales Center production environment is to ensure that WebSphere Commerce V6.0 is installed, and that at least one store exists.

> **Tip:** If you already have WebSphere Commerce V6.0 installed and configured, proceed to 4.2.2, “IBM Sales Center client security considerations” on page 49.

In our case, we followed the *IBM WebSphere Commerce V6.0 Enterprise and Professional Installation Guide for Windows* (GC10-4261-00) to install WebSphere Commerce Enterprise V6.0 on the Windows 2003 Server Standard Edition operating system platform. This installation guide describes how to install and configure IBM WebSphere Commerce V6.0, and is available for download from the IBM Publications Center site:

http://www.elink.ibmlink.ibm.com/public/applications/publications/cgibin/pbi.cgi?CTY=US&FNC=SRX&PBL=GC104261

We installed WebSphere Commerce V6.0 on a clean system, as a stand-alone one-tier server, by following the instructions in the product installation guide, using the WebSphere Commerce Quick Install. The Quick Install performs the following tasks:

- Installs DB2® Universal Database™ V8.2.3
- Installs IBM HTTP Server V6.0
- Installs WebSphere Application Server Network Deployment V6.0.2.5
- Installs Web server plug-ins for the WebSphere Application Server
- Installs WebSphere Commerce V6.0
- Creates a WebSphere Commerce instance named *demo*

After the server was installed, we published two starter stores using the WebSphere Commerce Administration Console:

- Consumer Direct B2C starter store (ConsumerDirect.sar) with housewares catalog. In our case, we chose the Available to Promise-enabled inventory option for the store in the store publish wizard.
- Advanced Business-to-business Direct starter store (AdvancedB2BDirect.sar) under Seller Organization with the electronics catalog. In our case, we chose the Available to Promise-enabled inventory option for the store in the store publish wizard.

Tip: By default, Windows 2003 Internet Explorer enables tight security. If you see a blank page when accessing the tools (Accelerator, Adminconsole, Organization Adminconsole), verify and modify the Internet Explorer security settings, for example, add the server site to your trusted sites.

For the most current list of the required maintenance, refer to the Technote *WebSphere Commerce required maintenance*, which is available on the Web at:

http://www-1.ibm.com/support/docview.wss?uid=swg21232042

For details about the recommended fixes for the IBM WebSphere Application Server, refer to the Technote *Recommended fixes for WebSphere Application Server*, which is available on the Web at:

http://www-1.ibm.com/support/docview.wss?uid=swg27004980#ver60

4.2.2 IBM Sales Center client security considerations

Security is an important concern whenever technology and commerce are involved. The IBM Sales Center client is no exception. It is important that you take the following precautions with regard to the machines on which the IBM Sales Center client is installed:

- Ensure that the client is behind a firewall
- Physically secure the machine from intruders
- Use a password-protected screen saver

Without these precautions, an intruder can install software on the client machine, which can, in turn, compromise the security of the information in the IBM Sales Center for WebSphere Commerce.

Note: Treat the files under the IBM Sales Center installation path <*SC_installdir*> with care because their exposure can cause security problems. Ensure that proper file permissions and access control settings are set to protect these files.

4.2.3 IBM Sales Center distribution mechanisms

Depending on the number of clients and the frequency of the updates required, you have several choices for installing and updating the IBM Sales Center.

The IBM Sales Center for WebSphere Commerce Quick Install installer provides a one time (one-off) install capability for the Eclipse-based rich client and the IBM Sales Center components. Installation scenarios requiring more scalability or

automation use servers that provide local area network (LAN) booting, reimaging, software installation, and inventory tracking so that the software installed on a large number of clients can be remotely managed completely.

Updates are distributed and managed through a similar group of manual and automated technologies, as are IBM Sales Center client preference settings.

Table 4-1 indicates the different tasks for which the different technologies described in this chapter are suitable.

Table 4-1 Different technologies to distribute code

	IBM Sales Center Quick Install	**Image install mechanisms**	**Manual install with Eclipse Update Manager**	**IBM Tivoli® Device Management Server**	**Other software distribution systems**
Initial client installation	Easy, limited flexibility, manual	Better for high volumes	N/A	N/A	N/A
Installing customizations	N/A	N/A	Yes, manual	Yes, automated	Yes, automated
Applying updates	N/A	N/A	Yes, manual	Yes, automated	Yes, automated
Centrally managing the client configuration	N/A	N/A	N/A	Yes	Yes

4.3 IBM Sales Center Quick Install

For simple deployment, IBM provides a Quick Install package that installs the complete IBM Sales Center for WebSphere Commerce client and the full WebSphere Everyplace Deployment for Windows and Linux V6.0 (WED4WL) runtime on a single machine.

Quick Install has a graphical user interface-based (GUI-based) installer packaged on a single CD, and supports manual installation. It is appropriate for the following circumstances:

- Irregular and occasional installation, including installing for evaluation purposes
- Generally, ad hoc installation and uninstallation

- Production use only in very limited deployments
- Under difficult circumstances, for example, to replace or rebuild a client on site when the usual software deployment mechanisms are not available
- For developers, to install an instance of IBM Sales Center on the development environment, as described in Chapter 3, “IBM Sales Center development environment installation” on page 25

Note: IBM Sales Center Quick Install *cannot* be used to install user customizations. The customizations must be built in the form of an Eclipse update site. Any of the postdeployment provisioning mechanisms can then be used to install custom extensions to the IBM Sales Center installed using Quick Install. If Quick Install was chosen to perform an ad hoc install in the absence of a deployment infrastructure, the manual installation of customizations using the Eclipse Update Manager can be used because this method does not require installation servers or deployment infrastructure to be in place.

This type of installation is not appropriate under the following circumstances:

- You require manual installation of only a subset of WED4WL.

  The WED4WL installer provides the flexibility that the IBM Sales Center Quick Install does not expose. You can, for example, initially install a minimal WED4WL, and then add the optional features.

  IBM Sales Center for WebSphere Commerce requires the WED4WL International Components for Unicode for Java (ICU4J) optional feature, which must be installed before IBM Sales Center can be added.

  For more information, refer to the installation instructions in *WebSphere Everyplace Deployment System Administrator's Guide*, which is available on the Web at:

  `http://www-1.ibm.com/support/docview.wss?uid=swg27006861`

- You have to install WED4WL with the Enterprise Management Agent already enabled.

  As part of the IBM Sales Center installation, WebSphere Everyplace Deployment for Windows and Linux V6.0 (WED4WL) containing the Enterprise Management Agent is installed. However, by default, this component is not enabled in IBM Sales Center when Quick Install is used.

You can install IBM Sales Center for WebSphere Commerce manually in the following ways:

- Install IBM Sales Center for WebSphere Commerce interactively from a CD
- Install IBM Sales Center for WebSphere Commerce silently from a CD, the instructions for which are available in the following Web site:

  http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.telesales.admin.doc/tasks/ttrin_silent.htm

For this book, we installed IBM Sales Center for WebSphere Commerce using an interactive GUI install, as described in the next section.

4.3.1 IBM Sales Center for WebSphere Commerce interactive install

To interactively install IBM Sales Center for WebSphere Commerce from a CD, complete the following tasks:

1. Ensure that the computer on which you are installing the IBM Sales Center for WebSphere Commerce meets the hardware, operating system, and networking requirements (4.1, “IBM Sales Center client requirements” on page 46).
2. Ensure that you are logged in as a user with sufficient system privileges to install new software, and that the basic IBM Sales Center client security is set (4.2.2, “IBM Sales Center client security considerations” on page 49).
3. Collect the following information:
   - The directory in the computer on which you want to install the IBM Sales Center
   - WebSphere Commerce server information:
     - Fully qualified host name for WebSphere Commerce

       This is the host name that you will use when accessing WebSphere Commerce tools such as WebSphere Commerce Accelerator
     - Port number for communication with WebSphere Commerce

       The default port is 8000
   - (Optional) The Uniform Resource Locator (URL) for the Update Manager site. This URL is used to update the IBM Sales Center components and can be set at any time.
   - The starting language for IBM Sales Center. This is the language that IBM Sales Center displays in when it starts.

Note: The application starts the first time using the default language properties file if no locale setting is detected on the client machine. The language that IBM Sales Center uses to display menus and labels is set by the -nl argument on the command line or, if no such argument is provided, by the operating system language setting (Regional Settings in Windows).

Refer to the information center topic “Globalization in the IBM Sales Center”, which is available on the Web at:

http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.telesales.developer.doc/concepts/ctrglobalization.htm

4. Insert the IBM Sales Center for WebSphere Commerce CD into the CD-ROM drive, browse the CD-ROM drive, and run setup.exe.
5. Select the **Language** to be used for the install wizard, and click **OK**.
6. In the welcome window, click **Next**.
7. In the agreement window, accept the terms and click **Next**.
8. In the page that appears, specify a short directory name such as `C:\WebSphere\SalesCenter60` and click **Next**.

9. Verify the summary information (Figure 4-1) and click **Next**.

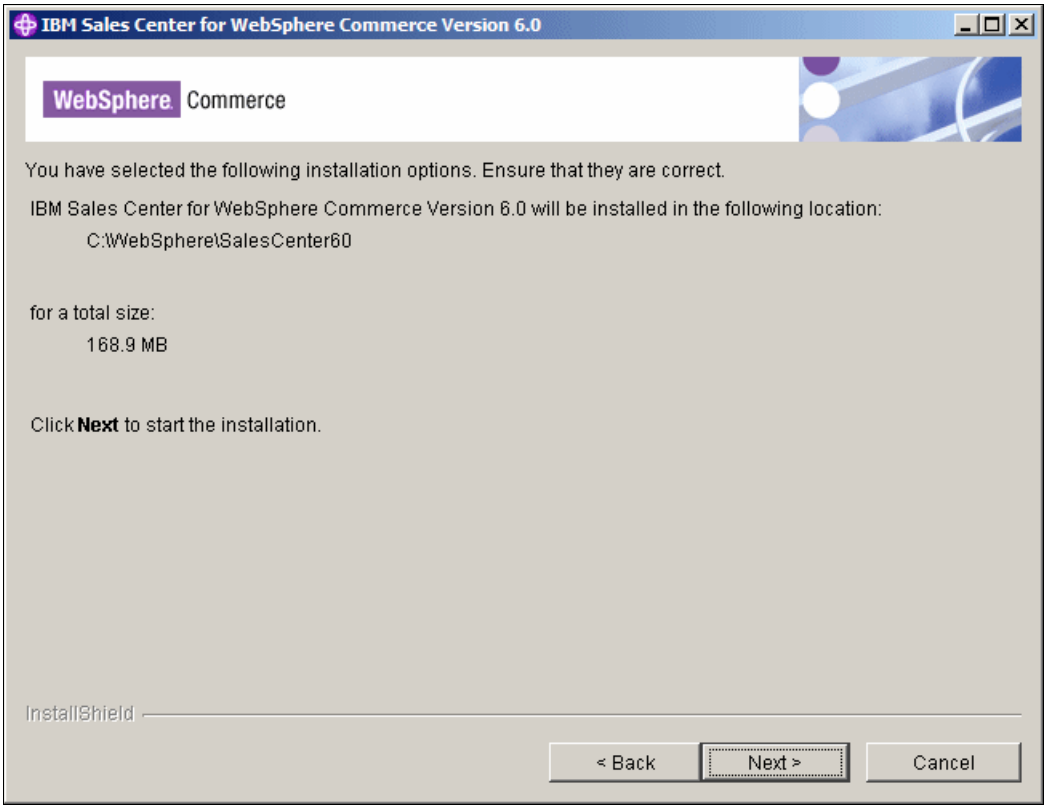


Figure 4-1 Summary information of IBM Sales Center for WebSphere Commerce V6 installation

10. The installation starts. After the installation is complete, verify if the installation is successful, and click **Finish** (Figure 4-2).

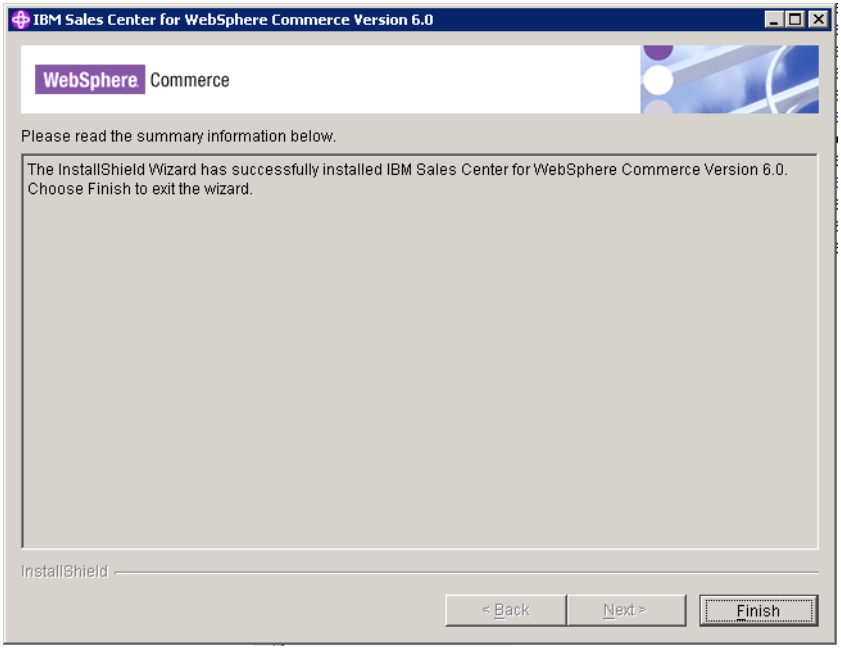


Figure 4-2 IBM Sales Center for WebSphere Commerce installation completed

After the installation process is complete, review the <*SC_installdir*>\log\tsinstall.log file to ensure that no errors occurred during the installation. (In our example, the <*SC_installdir*> install directory was C:\WebSphere\SalesCenter60.)

4.3.2 Manual installation of the IBM Sales Center updates using the Eclipse Update Manager

To install the IBM Sales Center client updates, for example, the IBM Sales Center fix pack using the Eclipse Update Site, perform the following tasks:

1. Open the IBM Sales Center client in administrator mode, using the -clientAdmin parameter.

   Add the -clientAdmin parameter inside the startup.bat file (if you are using it) in order to start the IBM Sales Center client, or add to the command that is used to start IBM Sales Center.

This mode provides access to administrative functions that are normally hidden at start. It is neither necessary to log in to IBM Sales Center nor to launch the Order Capture application to install the fix pack.

We created a copy of startup.bat located in the <*SC_installdir*>, C:\WebSphere\SalesCenter60, and edited it to add the -clientAdmin parameter. Our copy of startup.bat, renamed to, for example, Adminstartup.bat, contains the following command in one line:

```
"%~dp0rcp\rcplauncher.exe" -product
com.ibm.commerce.telesales.TelesalesWorkbenchProduct -clientAdmin
```

2. From the menu, select **Application** → **Install** → **Add Remote Location**.
3. The New Remote location window appears with the Field Name and the URL.
   - In the Name field, enter a name for the update location, for example, `IBM Sales Center v6.0 updates`
   - In the URL field, enter the following:

     ```
     ftp://ftp.software.ibm.com/software/websphere/commerce/60/salescenter/update/
     ```
4. The new update site is defined in the Location List (Figure 4-3). Select the newly created update site and click **Next**.

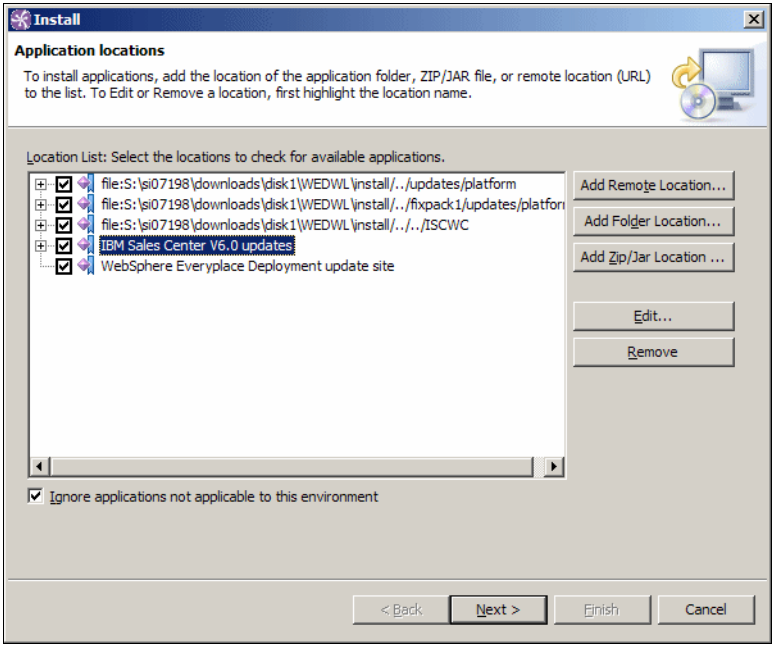


Figure 4-3 Application install locations

5. Select the features to install. You can, for instance, select all of them. Click **Next** (Figure 4-4).

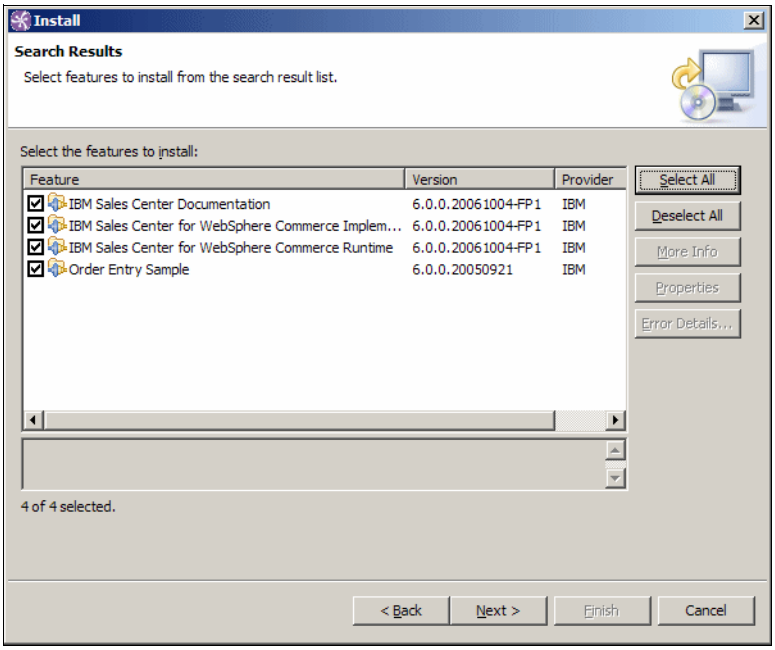


Figure 4-4 Search Results window

6. Accept the terms in the license agreements and click **Next**.
7. If you are presented with an optional features page, click **Next**.

8. Select the installation location for the features (Figure 4-5). Click **Finish** to perform the update.

> **Note:** IBM Sales Center fix pack updates are not enabled to change the location of the installed feature.

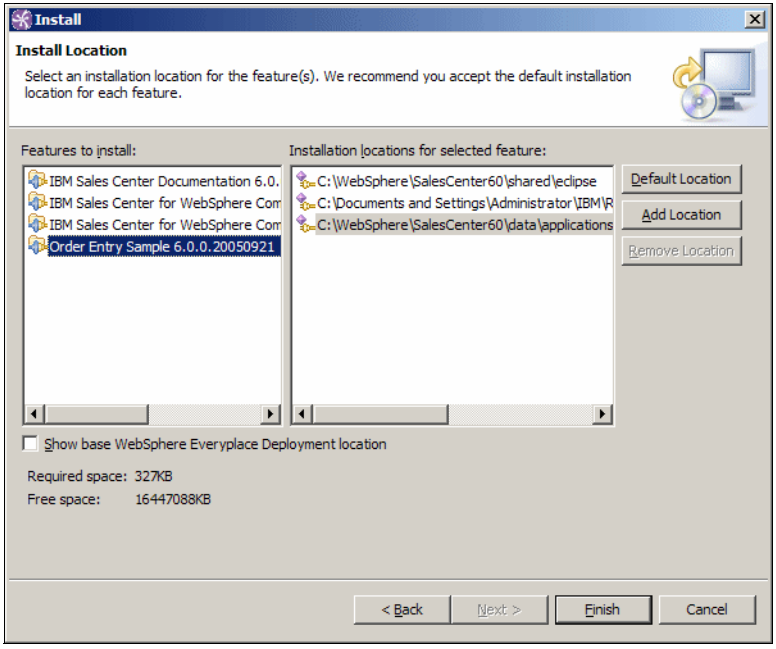


Figure 4-5 Install Location window

9. If presented with a Jar Verification window (Figure 4-6) after feature verification, click **Install**.

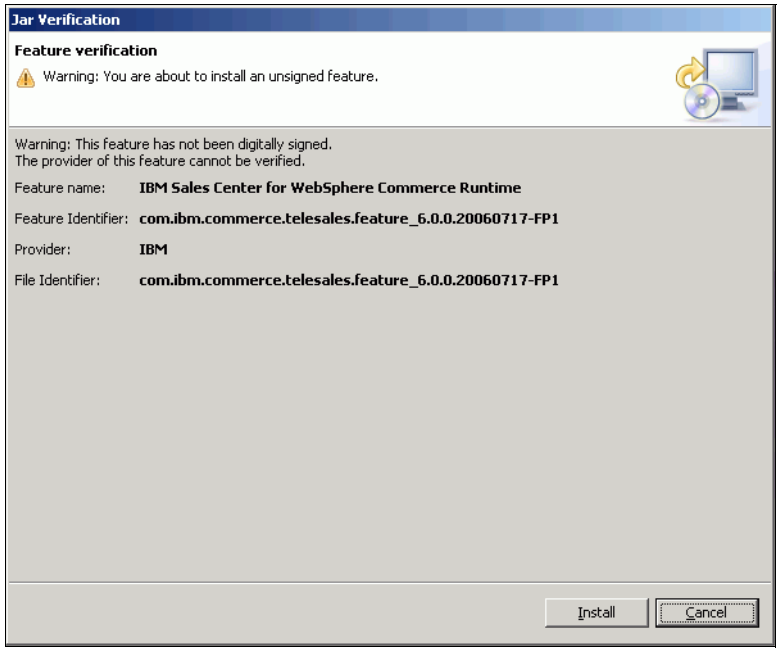


Figure 4-6 Feature Verification window

10. When the installation is completed, you will be prompted to restart the WebSphere Everyplace Deployment workbench. Click **Restart**.

4.4 Manual installation of customizations using the Eclipse Update Manager

The WED4WL platform has the built-in ability to install the features packaged in an Eclipse Update Site. The installation of updates can be initiated manually by an IBM Sales Center client user if the administrator mode option is enabled.

The following list includes the prerequisites for the manual installation of customizations using the Eclipse Update Manager:

- WED4WL must be installed on the client

  WED4WL can be installed using the WED4WL installer or the IBM Sales Center Quick Install. For production install by client imaging, install WED4WL

on the prototype client using either of these methods, so that it becomes a part of the initial client image.

- WED4WL must not be using Enterprise Management Agent

  The update manager's GUI is not available after the Enterprise Management Agent is enabled.

- The update site must be available to the client

  It may be on the client's local hard disk, CD-ROM, a file server, the Web, or the File Transfer Protocol (FTP) server accessible from the client.

To manually install your customizations and updates, perform the following tasks:

1. Open the IBM Sales Center client in administrator mode using the -clientAdmin parameter. (For instructions about this, refer to step 1 on page 55.)
2. Select **Application** → **Install**.
3. If the update site directory is available on the client hard disk, CD-ROM, or on a shared file server, click **Add Folder Location** in the Application Locations dialog box and enter the path to the update site directory. Otherwise, click **Add Remote Location** and enter the URL to the update site if it is available on an FTP server, Hypertext Transfer Protocol (HTTP) server or Hypertext Transfer Protocol Secure (HTTPS) server. Click **OK**.
4. In the Application Locations dialog box, ensure that the new update site is selected. Click **Next**.
5. From the update site, select the features you want to install, and click **Next**.
6. Accept the license agreement and click **Next**.
7. At the Install Location prompt, select the features to install and the location. (Do *not* select the check box labeled Show base WebSphere Everyplace Deployment location.) Click **Finish**.

   Note: If you select Show base WebSphere Everyplace Deployment location, other base WebSphere Everyplace Deployment locations are displayed. However, if you choose to install features to the base WebSphere Everyplace Deployment location, they cannot be updated, disabled, nor uninstalled using the Application Manager.

8. The feature verification window is shown. Click **Install**.
9. A dialog box will ask you whether you want to restart the workbench. Click **Yes**.

Tip: To uninstall or disable updates, perform the following tasks:

1. Launch IBM Sales Center in administrator mode (see instructions in step 1 on page 55).
2. Select **Application** → **Application Management** → ***<the feature>*** to uninstall.
3. In the right panel click **Uninstall**.

4.5 Automatic installation of customizations and updates

As discussed earlier, the production use of IBM Sales Center is expected to involve anywhere from tens to thousands of IBM Sales Center clients. The enterprise requires a scalable and repeatable way of distributing and updating WebSphere Everyplace Deployment for Windows and Linux (WED4WL) and IBM Sales Center components, and to distribute their customizations. This chapter provides the recommendations pertaining to this activity. These recommendations involve exploiting the capabilities of Eclipse and the management capabilities built into WED4WL.

The automation of this process requires additional software, which is available from IBM, external parties, and from the Eclipse Web site:

http://www.eclipse.org

This process also requires additional setup, including installing and configuring the management and software distribution servers. The automation process is attractive when this activity is spread over a sufficient number of clients. Otherwise, manual installation is recommended.

4.5.1 The production installation of IBM Sales Center

The production installation of the IBM Sales Center client is performed in two stages:

1. First, the base image consisting of Windows XP and an initial load of the client software is installed.
2. The client is then deployed and further provisioned with the remaining client application (if any), using one of the supported automated software distribution mechanisms. (Over the lifetime of the client, updates and additional features are installed using the same automated mechanisms.)

Deciding about where to draw the line between the initial load of the client software that occurs before client deployment and the further provisioning that happens afterwards is a customer design decision. The competing factors that contribute to this decision are:

- The initial load must include a minimal enabling software for postdeployment provisioning. What this enabling software consists of depends on the provisioning method.
- The initial load might require a significant setup effort that is repeated for each choice of initial load content. This argues the point that the initial load must contain only components that are relatively stable and common to all the supported configurations in order to minimize the number of times the setup is to be performed.
- If postdeployment provisioning involves a low bandwidth channel or a channel whose bandwidth is already committed to a high-priority business function, it is advantageous to make the initial load as complete as possible. In an extreme situation, the initial load may contain all the software that is necessary for a fully functioning client, and postdeployment provisioning only applies updates to sections of the client that have changed since the time the initial load image was created.
- If the deployed client has a high bandwidth channel to the provisioning server, it is possible to deploy the bare hardware and perform both the initial load and the subsequent provisioning at the deployment site. This method is also useful to reimage a client that has been corrupted in the field.

The next section describes how to automatically deploy customizations to IBM Sales Center.

4.5.2 Automatically deploying customizations using IBM Tivoli Configuration Manager

Many third-party enterprise software management systems are capable of moving a set of files to a managed system and issuing a command on that system or running a native installer there. Use these management systems to install and maintain IBM Sales Center for WebSphere Commerce and the customizations you build.

IBM Tivoli Configuration Manager is used to illustrate this. Tivoli Configuration Manager is a highly scalable system for monitoring, installing, and controlling IT resources across an enterprise. Its software distribution and inventory monitoring can be implemented over large numbers of geographically distributed clients.

Relevant concepts are referred to using their Tivoli Configuration Manager nomenclature. The following descriptions of the nomenclature will help you identify the corresponding terms in your software management product:

- Tivoli management region

  In a Tivoli environment, a Tivoli server and the set of clients that it serves form a Tivoli management region. An organization can have more than one region.

- Tivoli server

  The server for a specific Tivoli management region that holds or references the complete set of Tivoli software, including the full object database.

- Managed node

  A management server on which the Tivoli Management Framework is installed, and which can therefore participate in providing management services to endpoints.

- Gateway

  The software that provides communication services between endpoints and the rest of the Tivoli environment. The gateway is a managed node. In a minimal configuration, a single machine hosts a *Tivoli server* and a *gateway.*

- Repeater

  A managed node or gateway that caches and transmits data through a repeater hierarchy to designated targets. Repeaters are used for multiplexed distribution and collection operations. A repeater is a managed node.

- Endpoint

  The final recipient of any type of Tivoli operation. IBM Sales Center clients in a Tivoli management region are endpoints, not managed nodes. For scalability reasons, endpoints communicate with gateways instead of directly with the Tivoli server.

  Endpoints require an endpoint agent installed as a Windows service. The agent communicates with a gateway to send and receive data and to perform actions. Tivoli Configuration Manager uses the Tivoli Management Framework endpoint agent, lcfd.

  In contrast to the Device Management Server whose agent is included as part of the Workplace™ Managed Client platform, a Tivoli Management Framework agent is not included in the client and must be installed separately. Although this involves an extra component, it has the advantage of the agent being active and managing the client even when the IBM Sales Center client is not running. It is also possible to use Tivoli Configuration Manager to perform the base Workplace Managed Client™ install, in addition to distributing Eclipse update sites to the client.

There are several ways of installing an agent, including CD installation and remote installation methods. For more information about this, refer to the “Tivoli Enterprise™ Installation Guide” section under “Installing endpoints”. The Tivoli Enterprise Installation Guide is part of the Tivoli Management Framework documentation, and is available for download from the following Web site:

http://publib.boulder.ibm.com/infocenter/tivihelp/v3r1/topic/com.ibm.tivoli.frmwrk.doc/instguid.htm

In the rest of this section, it is assumed that the Tivoli Configuration Manager server and the gateway is installed and configured, and that an endpoint agent is installed and running on the client.

Environment

In our environment, we installed and configured the following components:

- The Tivoli server and gateway are installed and configured with at least the Tivoli Configuration Manager Software Distribution, Inventory, Activity Planner, and Change Manager features. Our Tivoli server ran on a Windows 2000 Server machine.
- An endpoint agent is running on the target client and successfully connecting with the gateway. We used a desktop machine with Windows XP Professional as the target machine to run the endpoint.

  Note: The endpoint agent must be configured to run under the Windows user identity, and not under the Local Server identity. Eclipse Update Site installations fail if the agent runs as Local Server.

- The endpoint machine or the machine where the server is running, or any other machines connected to the same network that can also be used as a the development client, has a copy of the Tivoli Configuration Manager Software Package Editor component installed. The Software Package Editor is a part of the software distribution function of the Tivoli Configuration Manager. We used the same machine on which the Tivoli server was running to run the Software Package Editor and the Tivoli desktop.
- The IBM Sales Center client install CD image is available to the development client, which, in our case, was installed and configured on the Tivoli server.

Figure 4-7 shows our laboratory environment.

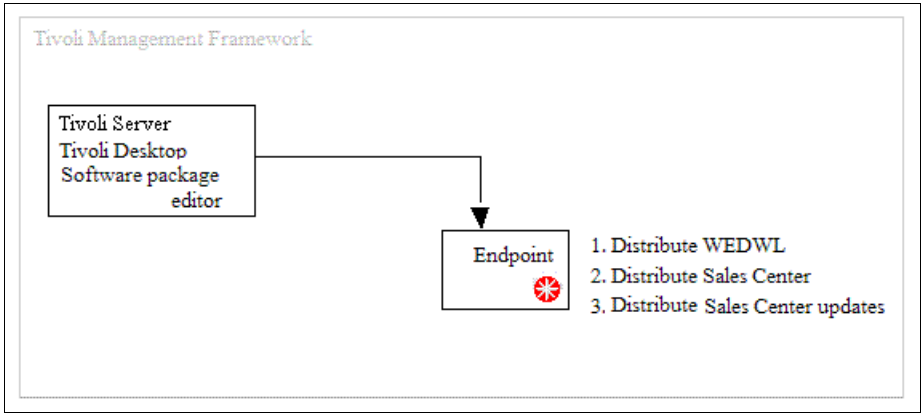


Figure 4-7 Tivoli Management Framework environment

Creating and distributing a WED4WL software package

This section discusses how to create and distribute a WED4WL software package using the Tivoli Configuration Manager. Use the method described here to remotely distribute a large number of WED4WL clients in a Tivoli environment.

To create, test, and distribute the WED4WL software package, perform the following tasks:

1. On the development client, run the WED4WL installer with the option to generate a prototype response file:

   ```
   setupwin32.exe -options-template c:\response.txt
   ```

2. Use a text editor to customize the generated response file by following the instructions provided in the WED4WL System Administrator's Guide. Do not set the installLocation product bean property. Set the following product bean properties in the responses file:

   - `-G licenseAccepted=true`
   - `-W launchPlatformPanel.launchPlatform="0"`

3. Launch the Software Package Editor on the development client.

4. In the Software Package Editor Selector window (Figure 4-8), select **Generic Software Package** and click **OK**.

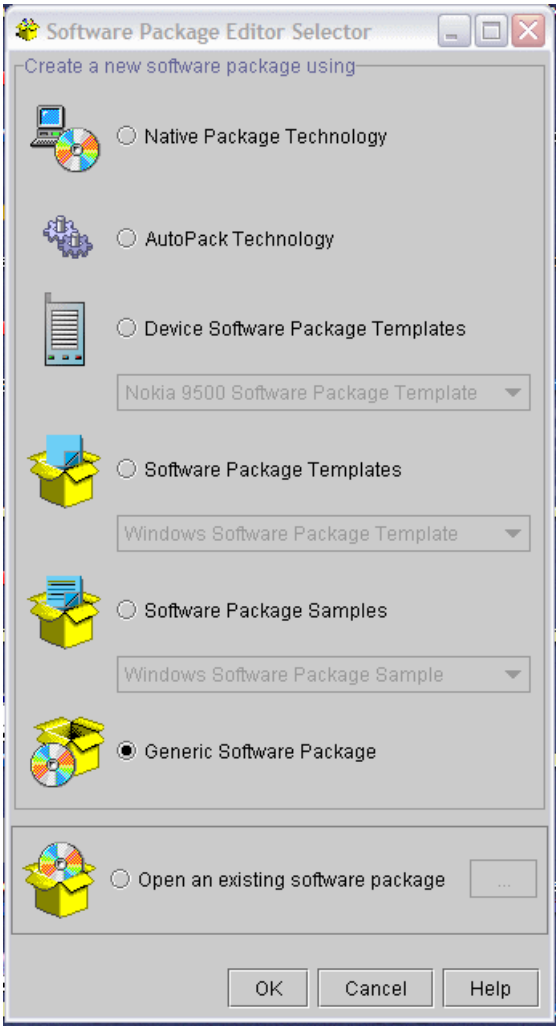


Figure 4-8 Software Package Editor Selector window

5. In the page that opens, right-click **Noname package** in the left-hand navigator and select **Properties**.
6. In the General tab, set the Package Name to `WMC` and the Version to `6.0`. Click **Condition**.

7. In the Condition Editor window, double-click **os_name**, click **==**, and type `Windows_NT` in the text window, as shown in Figure 4-9. Click **OK**.

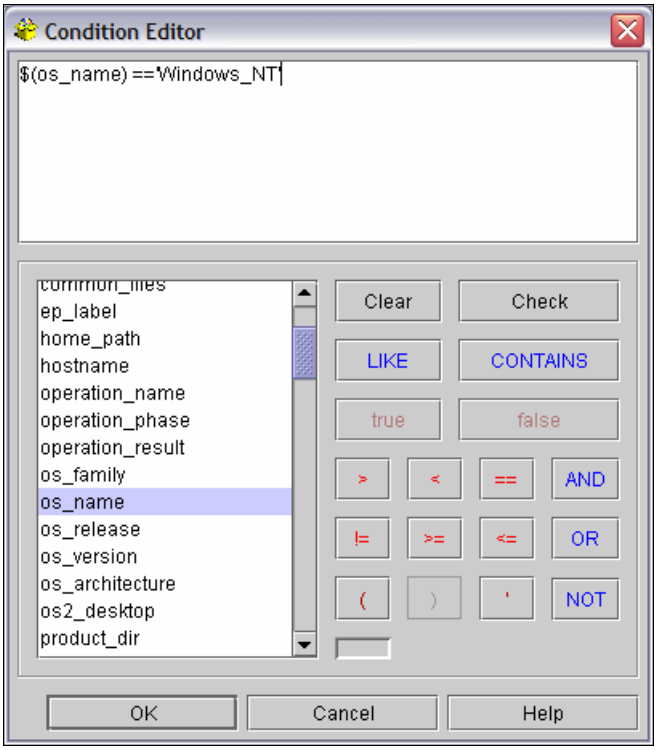


Figure 4-9 Condition Editor

Note: In Tivoli Configuration Manager, Windows_NT is the name used to refer to all the supported Windows operating systems. Refer to the Tivoli Configuration Manager Reference Manual for Software Distribution for information about the condition variables.

8. In the Package Properties editor (Figure 4-10), select **Variable list**. In the variable editor, type `install_location` in the Name field and `$(program_files)\IBM\WED` in the Value field, and click **Set**. This creates a variable and its default value, which you can use in the installation script to control the place where the WED4WL will install. Also create another variable `install_temp` with the value `$(install_location)\temp` (Figure 4-10). This directory will be used to temporarily hold the install image files. Click **OK**.

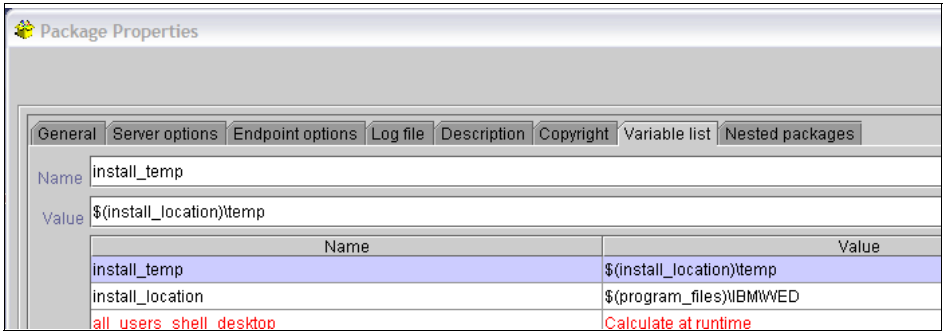


Figure 4-10 Package Properties editor

9. In the Software Package Editor page, right-click the package icon again and select **Insert** → **Program** → **Execute program** (Figure 4-11).

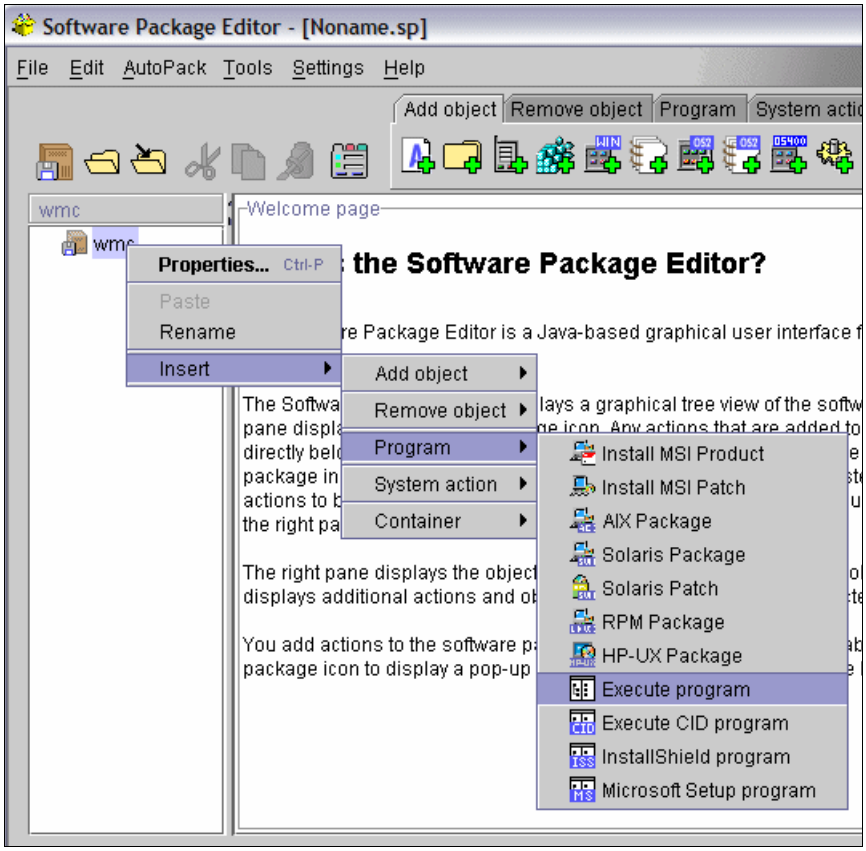


Figure 4-11 The Software Package Editor

10. The Execute Program Properties editor opens. In the Install tab of the Execute Program Properties editor, type `cmd` in the Path field. This is the program that will run the Windows Command Interpreter.

11. Click **Advanced** and fill in the properties listed in Table 4-2, including the quotes that are provided. Click **OK**.

Table 4-2 Property information

Arguments	“$(install_location)”
Inhibit parsing	Not checked
Working directory	$(install_temp)
Reporting standard error file on server	Check

12. In the Execute Program Properties window, click **Add** in the Corequisite Files section. Navigate to the local file system location where the WED4WL install images are present, and select the **install** directory. Click **Open**. Select the directory icon that has just been created and click **Edit**.

13. In the Add Directory Properties window (Figure 4-12), change the Destination Location to `$(install_temp)`. This temporarily copies the install directory and all the files and subdirectories to the path $(install_temp)/install. Click **OK**.

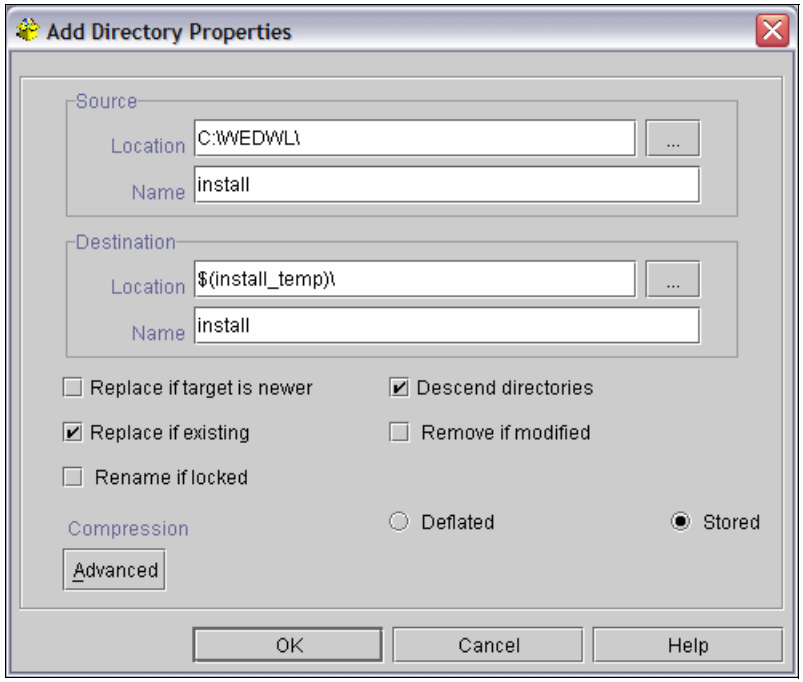


Figure 4-12 Add Directory Properties window

14. Ensure that the Directory icon is not selected and click **Add** again. This time, copy the updates directory to $(install_temp)/updates.

15. Deselect any unwanted icons once again and click **Add** one more time. Navigate to the response.txt file created in step 1 on page 65. Edit the created file entry to again set the Destination location to `$(install_temp)`. The editor looks similar to Figure 4-13. Click **OK**.

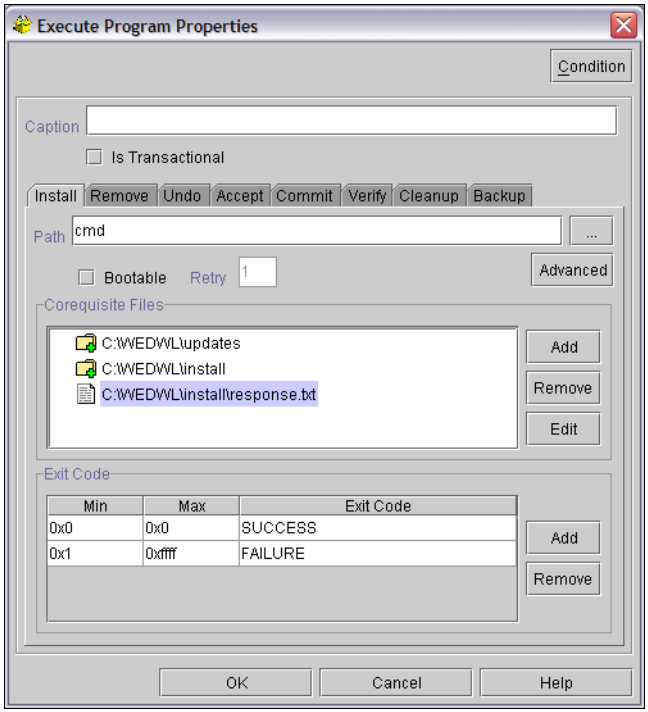


Figure 4-13 Execute Program Properties settings

16. In the main editor window, from the menu, select **File** → **Save as**. Select the Software Package Block file type and provide the name `wmc6` for the file.

17. To test the package locally on the development client, open a command shell window, with the software distribution command line environment established. Your Start menu must contain a shortcut named SWDIS ENV that will open this. Issue the following command:

    ```
    wdinstsp -Dinstall_location=c:\WMC6 wmc6.spb
    ```

    In this command, wmc6.spb is the software package block file created earlier. The property C:\WMC6 overrides the default install location of C:\Program Files\IBM\WED and the package installs in the C:\WMC6 directory.

18. When the software package is ready, it is distributed to the endpoint using the Tivoli desktop.

> **Tip:** Refer to the IBM Tivoli Configuration Manager Reference Manual for Software Distribution for more information about disconnected command-line interface commands and the software package change management states.

Creating and distributing an Eclipse Update Site software package

This section discusses how to create and distribute an IBM Sales Center feature using Tivoli Configuration Manager. Use the method described here to remotely distribute a large number of IBM Sales Center clients in a Tivoli environment. A software package bundle is created in a Tivoli environment for packaging and distributing any software you want to distribute.

Perform the following tasks to create a software package from an Eclipse Update Site. For this book, the IBM Sales Center Update Site was used. However, the same procedure can be used to distribute update sites built with your own customizations:

1. Launch the Software Package Editor on the development client.
2. In the Software Package Editor Selector window, select **Generic Software Package** and click **OK**.
3. In the window that opens, right-click the **Noname package** icon in the left-hand navigator and select **Properties**.
4. Enter the Name as `Sales Center` and the Version as `1.0`. Click **Condition**.
5. In the Condition Editor window, double-click **os_name**, click the **==** button, and type `Windows_NT` in the text window. Click **OK**.
6. In the Package Properties editor, click the **Variable list** tab. In the variable editor, type `install_location` in the Name field and `$(program_files)\IBM\WED` in the Value field, and click **Set**. When the distribution is installed, this variable must be set to the path where WED4WL has already been installed. Also create a variable *install_temp* with the value `$(install_location)\temp`. This directory will be used to temporarily hold the install image files. Click **OK**.
7. In the Software Package Editor, right-click the **Package** icon again and select **Insert** → **Program** → **Execute**.

8. An Execute Program Properties editor opens. In the Install tab of the Execute Program Properties editor, type `installfeature.bat` in the Path field. This is the name of the Windows batch file that will be executed. Click **Advanced** and fill in the properties as shown in Table 4-3, including the quotes that are provided. Click **OK**.

Table 4-3 Property information

Arguments	“$(install_location)”
Inhibit parsing	Not checked
Working directory	$(install_temp)
Reporting standard error file on server	Check

9. Create a copy of installfeature.bat (Example 4-1) and copy it to a location in the development client.

   Edit this file by replacing the text FEATUREID with the name of the feature to install and the VERSION with its version number.

   To install Sales Center, replace FEATUREID with the name of the primary feature, `com.ibm.commerce.telesales.impl.feature`, and VERSION with `6.0.0`. Any nested features in the update site will also be installed. If your site has more than one primary feature, replicate those lines and replace FEATUREID and VERSION with the corresponding values for the other features as many times as necessary.

   This script takes one command-line argument, that is, the path where an instance of WED4WL is already installed. The script assumes that it will be in the same directory in which the update site's site.xml file is located. It implicitly creates an install site named sc. You might encounter errors if you attempt to use an existing install site instead. In particular, the site in %WED_HOME%\shared\eclipse path cannot be installed into using this mechanism.

Example 4-1 installfeature.bat

```
setlocal enableextensions
set WED_HOME=%1
set CUR_DIR=%~dp0

%WED_HOME%\rcp\rcplauncher -application
org.eclipse.update.core.standaloneUpdate -nosplash -os win32 -arch x86
-command install -featureId FEATUREID -version VERSION -from
"file:/%CUR_DIR%/"  -to %WED_HOME%\sc\eclipse  -rcpLauncherWait
-consolelog -data "%CUR_DIR%/data"
set rc=%ERRORLEVEL%
if errorlevel 1 goto :error
```

```
rem replicate previous lines to add other features

:error
rem endlocal is implied by end of script
exit %rc%
```

10. In the Execute Program Properties window, select **Add** in the Corequisite Files section. Navigate to the local file system location of the Eclipse update site and select its directory. Click **Open**. Select the directory icon just created, and click **Edit**.
11. In the Add Directory Properties window (Figure 4-14), change the Destination Location to `$(install_temp)` and the Destination Name to `"."`, and select the **Descend directories** option. This maps the Update Site directory and all the files and subdirectories it contains, to the path $(install_temp). Click **OK**.

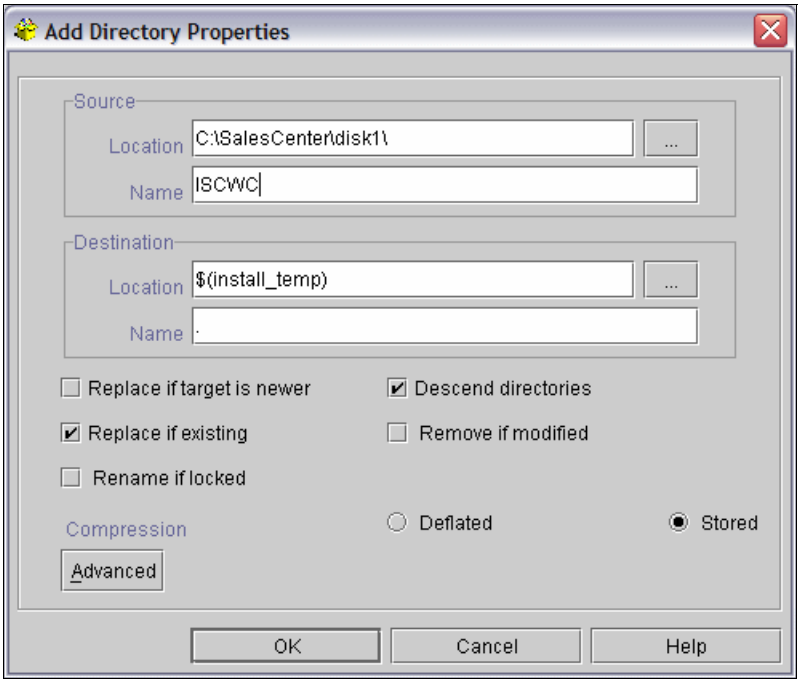


Figure 4-14 Add Directory Properties window

12. In the Execute Program Properties window (Figure 4-15), deselect all the icons and click **Add**. Navigate to the customized installfeature.bat file and select **Open**. Click **Edit** and set the Destination Location as `$(install_temp)`. Click **OK**.

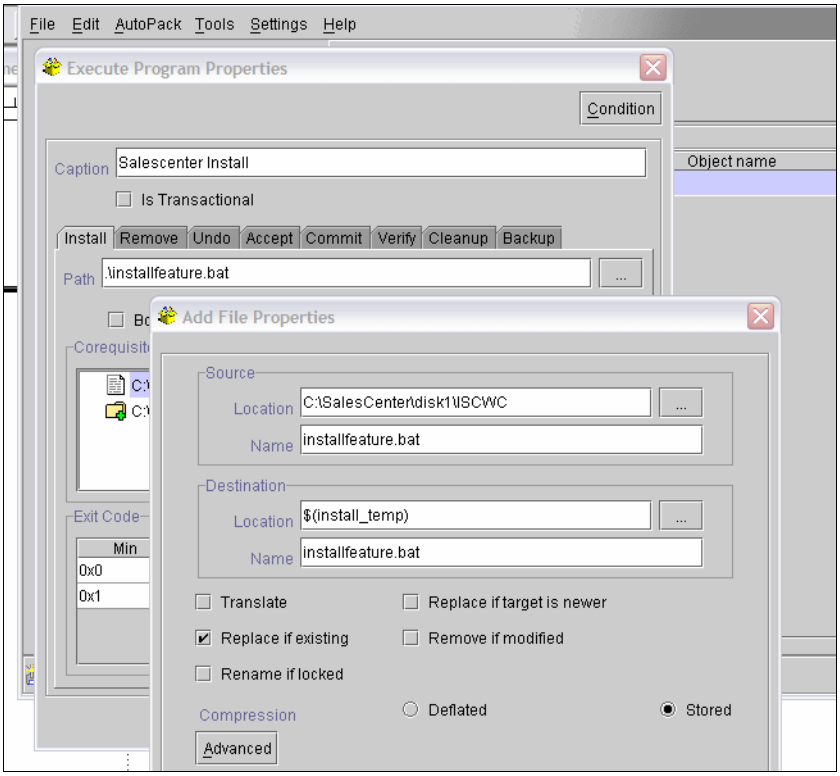


Figure 4-15 Adding installfeature.bat file properties

13. Configure the actions to be uninstalled. Select the **Remove** tab and type `uninstallfeature.bat` in the Path field. Click **Advanced** and set the fields in such a way that they are identical to the way they were set for the Install action.

14. Create a copy of uninstallfeature.bat (Example 4-2) and copy it to a location in the development client. Again, customize the FEATUREID and VERSION and the install site location. Duplicate the text if more than one feature is to be uninstalled. Because features must be disabled before they can be uninstalled, this script performs two actions for each feature.

Example 4-2 uninstallfeature.bat

```
setlocal enableextensions

set WED_HOME=%1
set CUR_DIR=%~dp0

%WED_HOME%\rcp\rcplauncher -application
org.eclipse.update.core.standaloneUpdate -nosplash -os win32 -arch x86
-command disable -featureId FEATUREID -version VERSION -to
%WED_HOME%/sc/eclipse  -rcpLauncherWait -consolelog -data
"%CUR_DIR%/data"
set rc=%errorlevel%
if errorlevel 1 goto :error

%WED_HOME%\rcp\rcplauncher -application
org.eclipse.update.core.standaloneUpdate -nosplash -os win32 -arch x86
-command uninstall -featureId FEATUREID -version VERSION -to
%WED_HOME%/sc/eclipse -rcpLauncherWait -consolelog -data
"%CUR_DIR%/data"
set rc=%errorlevel%
if errorlevel 1 goto :error

:error
rem endlocal implied by end of script
exit %rc%
```

15. Select **Add** in the Execute Program Properties editor and map the customized uninstallfeature.bat file to the directory $(install_temp). Click **OK**.
16. As in the previous example, save in the Software Package Block format.
17. Test using the **wdinstsp** command on the development client to install features into a previously installed WED4WL. The install succeeds irrespective of whether the client is running or not, but the installed features become accessible only after the client is restarted.

    Test the uninstall action on the development client using the following software distribution disconnected command:

    ```
    wdrmvsp "Sales Center^1.0"
    ```

Note: If the features are removed when the client is running, the results are unpredictable. Therefore, uninstalls must normally be scheduled when the client is not in use.

The Tivoli desktop can now be used to import the software distribution packages built in this section into a Profile Manager in the management region, and activities can be scheduled to distribute them to endpoints. There are other variations in the distribution mechanism where, for example, the corequisite files are served from a file server instead of being copied to each endpoint, or the package is served from a repeater depot.

4.5.3 Automatically deploying customizations using WebSphere Everyplace Deployment

IBM Workplace Managed Client includes an Enterprise Management Agent that can be used in conjunction with a Tivoli Device Management Server to install, upgrade, and configure applications running on the WED4WL client. The agent and the server can also manage other aspects of the client machine, such as installing or upgrading native applications or manipulating the contents of the Windows registry.

Note: The Tivoli Device Management Server is not supplied with WebSphere Commerce, but is a component of several IBM products that may be separately acquired. The Enterprise Management Agent in WED4WL requires a Device Management Server V1.8 or later. An appropriate Device Management Server is available in the IBM products WebSphere Everyplace Deployment V6.0 and WebSphere Everyplace Device Manager V6.0.

For information about installing and configuring the Device Management Server, and the full set of capabilities it provides, refer to the documentation supplied with the product. In this case, we assume that the server is already configured and the objective is to use it to install or uninstall the Eclipse features in a WED4WL runtime.

The Device Management Server is capable of running a number of different types of management operations on a variety of different device types. In Device Management Server terms, the distribution of IBM Sales Center or of customized Eclipse features to install on a WED4WL Windows XP client uses a native software distribution job targeting a Windows 32-bit device. This device is a subset of the Open Services Gateway initiative (OSGi) devices, which in turn, are a type of Open Mobile Alliance Device Management (BaseOMA DM) device.

To distribute software to the client platform, use the NativeAppBundle tool that is provided as part of the Device Management Server distribution to wrap the Eclipse update site in an OSGi bundle. After this, create and schedule a software distribution job targeting the client. When the created bundle is received on the client by the agent, it programmatically invokes the Eclipse Update Manager to install the features packaged in the update site.

Prerequisites

The following components were installed and configured on our environment:

- WebSphere Everyplace Deployment 6.0 Server

  This is installed and configured on a server whose port 80 is accessible from the client. Enable WebSphere Application Server security on the Device Management Server. Although the Device Manager Installation Guide indicates that enabling security is optional, security is mandatory to correctly support Windows 32-bit devices.

  Before starting the WebSphere Everyplace Deployment Server, apply IBM WebSphere Everyplace Deployment V6.0 Interim Fix 2.

  For information about applying IBM WebSphere Everyplace Deployment V6.0 Interim Fix 2 and the location information, refer to the following Web site:

  http://www-1.ibm.com/support/docview.wss?rs=2314&context=SSNLT6&dc=D400&q1=wedfpintfx&uid=swg24013683&loc=en_US&cs=utf-8&lang=en

- WebSphere Everyplace Deployment for Windows and Linux (WED4WL) 6.0

  Install WED4WL on the client either by using the WED4WL installer or the IBM Sales Center Quick Install.

  Apply the following fixes on the WED4WL:

  - Fix Pack 1 for WebSphere Everyplace Deployment for Windows and Linux 6.0

    For information about applying Fix Pack 1 for WebSphere Everyplace Deployment for Windows and Linux 6.0 and the location information, refer to the following Web site:

    http://www-1.ibm.com/support/docview.wss?rs=2314&context=SSNLT6&dc=D400&q1=wedfpintfx&uid=swg24012062&loc=en_US&cs=utf-8&lang=en

– WebSphere Everyplace Deployment for Windows and Linux 6.0 Interim Fix 3

  For information about applying the WebSphere Everyplace Deployment for Windows and Linux 6.0 Interim Fix 3 and the location information, refer to the following Web site:

  http://www-1.ibm.com/support/docview.wss?rs=2314&context=SSNLT6&dc=D400&q1=wedfpintfx&uid=swg24013807&loc=en_US&cs=utf-8&lang=en

Following are the tasks that we performed (each of these tasks are described subsequently):

- Enabling the Enterprise Management Agent
- Creating an Eclipse NativeAppBundle
- Registering the NativeAppBundle with the Device Management Server
- Scheduling a software distribution job
- Uninstalling using a bundle control job

Enabling the Enterprise Management Agent

To enable the Enterprise Management Agent, perform the following tasks:

1. Start the IBM Sales Center client with the -clientAdmin command-line option. For instructions, refer to step 1 on page 55.
2. In the Preferences window (Figure 4-16):
   – Select the **Enterprise Management Agent** preferences page.
   – Select the check box against **Enable Enterprise Management Agent**. A dialog box pops up asking you to confirm whether you want to enable the Enterprise Management Agent. Click **OK**.
   – In the Server IP Address field, enter the following:

     `http://<hostname>/dmserver/SyncMLDMServerAuthRequired`

   – *<hostname>* represents the Device Management Server host name.
   – For the Device User Name and Device User Password fields, provide valid information for a user authorized to access the Device Management Server. These users are defined in the user registry selected during the WebSphere Application Server configuration. You can test a user and password combination by accessing the server URL with a Web browser.
   – Review the polling configuration. The Enterprise Management Agent contacts the Device Management Server based on a start and stop polling window. When the Enterprise Management Agent is enabled for the first time, it attempts to contact the Device Management Server regardless of the polling window settings. On each subsequent restart, the agent will not contact the Device Management Server if it is outside the polling window.

The polling configuration is not changeable in this window. It can be changed by a Device Management Server Configuration Job.

Click **OK**.

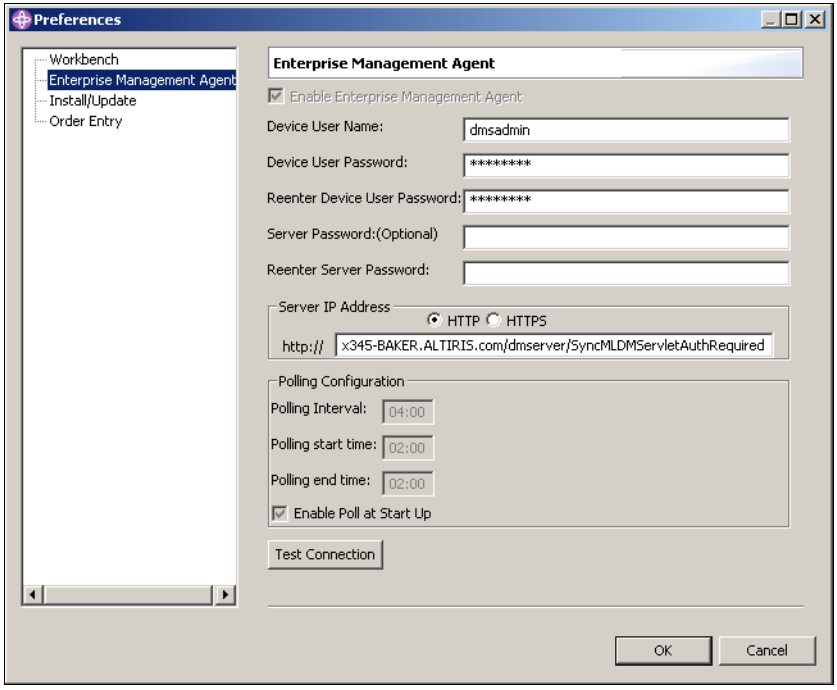


Figure 4-16 Agent Preferences page

Creating an Eclipse NativeAppBundle

Following is the process involved in creating an Eclipse NativeAppBundle:

1. Log in to the server where the Device Management Server is installed and copy the Eclipse update site to a directory on that machine.
2. Change the current directory to `<DM Server install directory>/bin`.
3. Run the command shown in Example 4-3 to create your software bundle.

Example 4-3 Creating a software bundle

```
NativeAppBundle -BundleName=<bundle_name> -InputDirectory=<path to your
update site> -InstallDirectory=<temporary_install_directory>
-BuildDirectory=<HTTP server document directory>/bundles
-Eclipse=default -BundleVersion=<version> -CleanupAfterInstall=yes
-RemoveOnUninstall=no
```

In this command:

– <*bundle_name*> is a name of the bundle that will be generated to wrap the Eclipse update site.

  The bundle name may not contain spaces or periods. Each bundle that you define using the NativeAppBundle tool must use a unique bundle name, for example, if you have created a distribution bundle with a BundleName of `MyApp`, containing V1.0.0 of your application, you must use a different BundleName when creating a distribution bundle containing V2.0.0. This refers only to the BundleName specified as parameters to the NativeAppBundle program, and not to the features or plug-ins contained in the update site.

– <*path to your update site*> is the location of the input update site on the Device Management Server machine.

– <*temporary_install_directory*> is the name of the directory that will be used on the client, and is operating system-specific. This value may contain a variable reference that will be replaced on the client during bundle installation. The manifest file generated for this bundle uses % as the delimiter surrounding the variable name. The search order for the variable value is the operating system environment, followed by the Java system properties.

  If the NativeAppBundle command is being run on a Windows system, escape characters are required, for example, in order to use the value of the environment variable TEMP on the client system, you must specify `-InstallDirectory=^%^%TEMP^%^%`.

  The variable name is case-sensitive, unlike the usual behavior on Windows systems.

– <*HTTP Server document directory*> is the destination where the OSGi bundle jar will be written. Because the client has to access it using either the HTTP or the FTP protocol, it is most convenient to generate it directly into a document directory served by a Web server or an FTP server. An example path would be C:/Program Files/IBM HTTP Server/htdocs/en_US/bundles.

– The -Eclipse parameter indicates that the target bundle contains an Eclipse update site for installation into an Eclipse extension framework. Its value defines the Eclipse install site into which the features will be installed. The special value default installs the features into the default install site for WED4WL. Any other value must be an absolute file system path to the Eclipse subdirectory of the install site, for example, -Eclipse=c:/Program Files/IBM/WED/SalesCenter/eclipse.

Example 4-4 shows the command that we ran.

Example 4-4 Targeted bundle command

```
NativeAppBundle -BundleName=MCSC1 -InputDirectory=C:\WED-SW-Dump\ISCWC
-installDirectory=C:\Temp\mcsc -BuildDirectory=c:\IBM\IBM HTTP
Server\htdocs\en_US\bundles -Eclipse=default -BundleVersion=1.0.0
-CleanupAfterInstall=yes -RemoveOnUninstall=noand on successful
execution generates the response
```

After the command has run successfully, the output is as follows:

```
SMF bundle: c:\Program Files\IBM HTTP
Server\htdocs\en_US\bundles\MCSC1+1_0_0.jar successfully created
```

Registering the NativeAppBundle with the Device Management Server

To register the NativeAppBundle with the Device Management Server, perform the following tasks:

1. Start the Device Manager console. The console is automatically installed on the Device Management Server, but may also be installed on other Windows systems using a browser. For instructions, refer to the topic “Installing the Device Manager console” in the Device Management Server documentation.

   Ensure that the Device Manager Server field contains the Device Manager Server host name.

2. Log in to the Device Manager console with the appropriate user ID and password:

   – In a WebSphere Everyplace Deployment server, use the WebSphere Everyplace Deployment administrator user ID and password.

   – In a WebSphere Everyplace Device Management server, the default Subscription Manager authorizes the user ID dmadmin with the password dmadmin.

   If you have implemented a custom Subscription Manager on the server, it will determine which IDs and passwords are valid.

3. In the Device Manager page, right-click **Software** and select **New Software** (Figure 4-17).

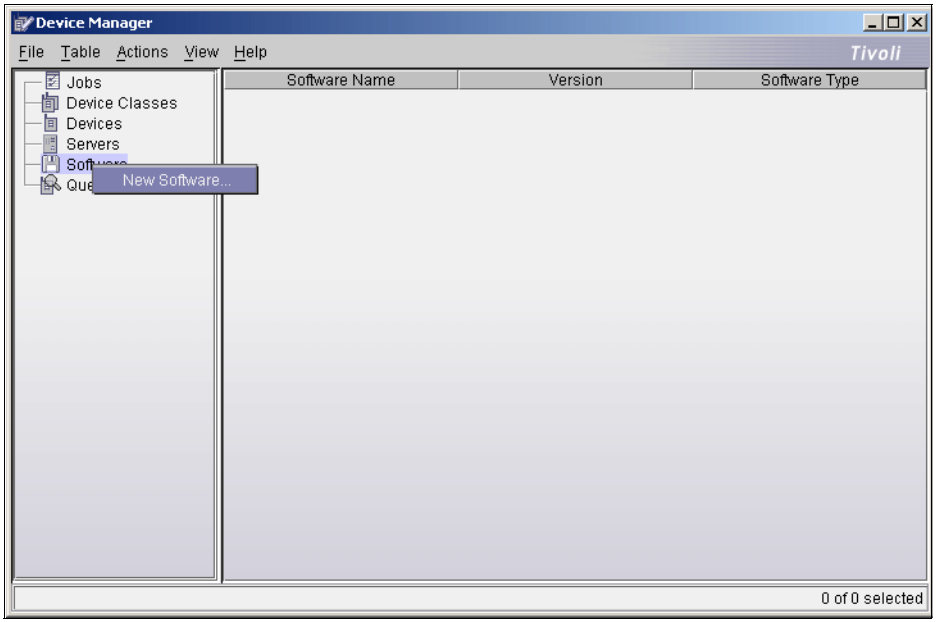


Figure 4-17 Adding new software bundle in Device Manager

4. In the page that appears (Figure 4-18), perform the following tasks:
   - Select **NativeWin32OSGiBundle** software type.
   - Specify the URL where the previously created native application bundle is served from.
   - Click **Fetch**.

   Click **Next**.

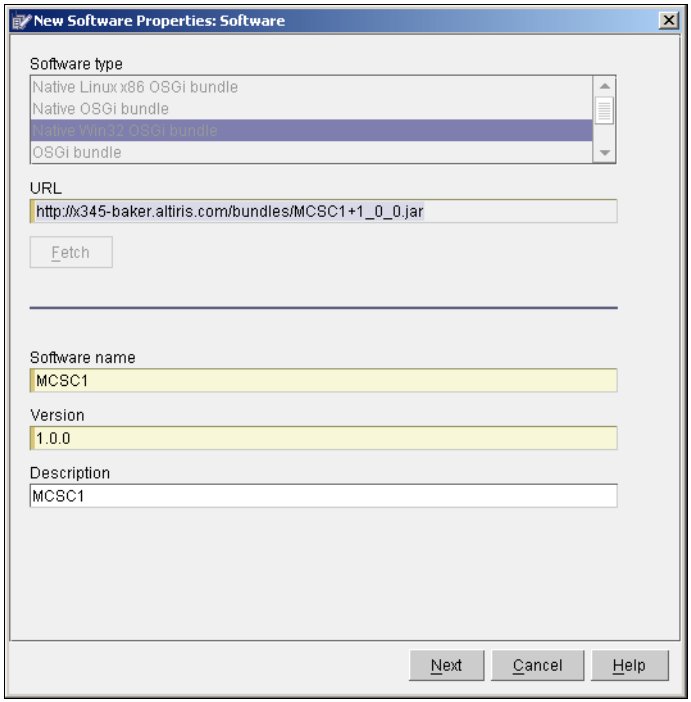


Figure 4-18 Fetching the software-created bundle

5. In the page that appears (Figure 4-19), select all the operations for the Device Class = Win32®, and click **OK**.

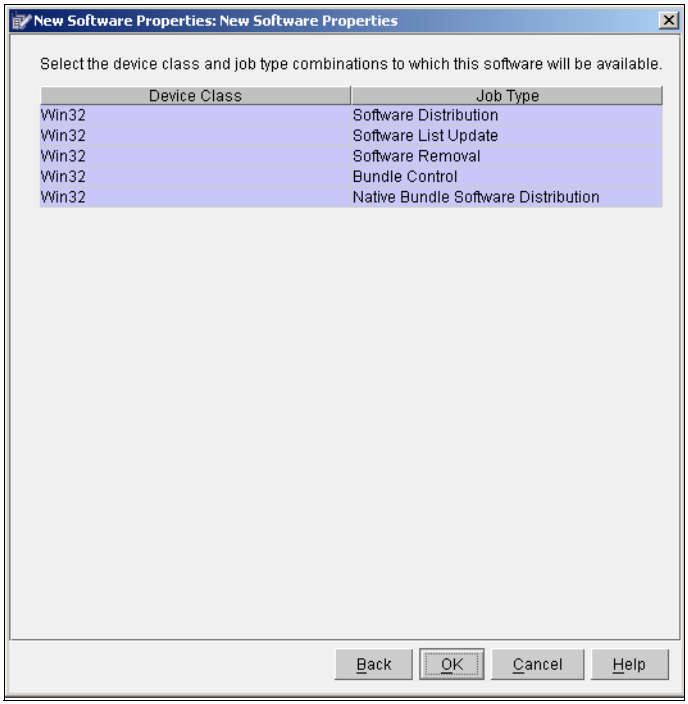


Figure 4-19 Selecting all the devices

Scheduling a software distribution job

To schedule a software distribution job, perform the following tasks:

1. Start and log in to the Device Manager console.
2. In the Device Manager console, perform the following tasks:
   - Select **Devices**.
   - Select **Use New Query and Return anything** as your search criteria.

   Click **OK**.

3. The Device Manager console (Figure 4-20) shows a list of the enrolled devices. Select the device or devices you want to distribute the software package to, right-click it, and select **Submit Job**.

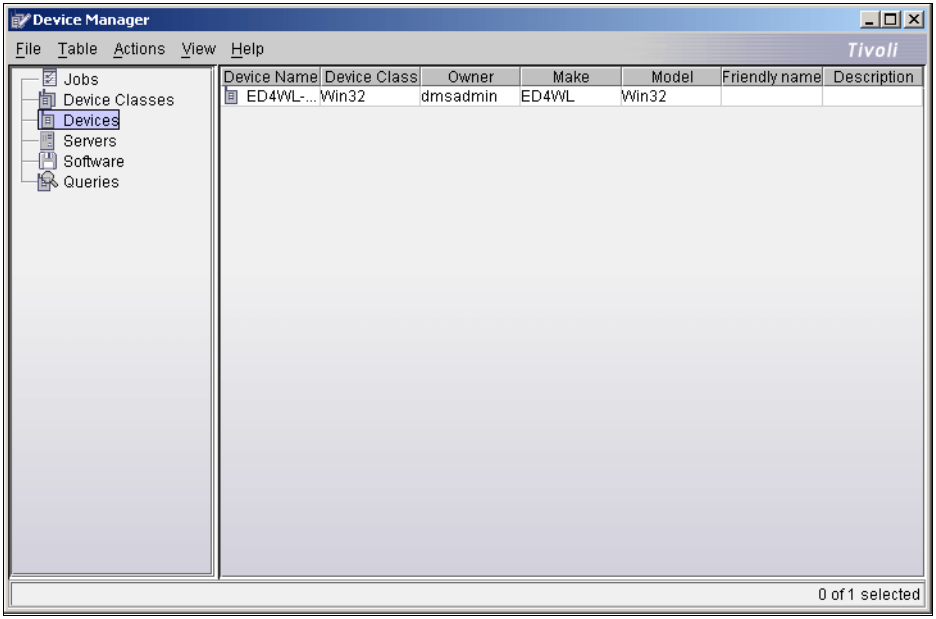


Figure 4-20 The list of devices enrolled

4. In the page that appears (Figure 4-21), under Job Type, select **Native Bundle Software Distribution**. Use the default settings for all the other job attributes, and click **Next**.

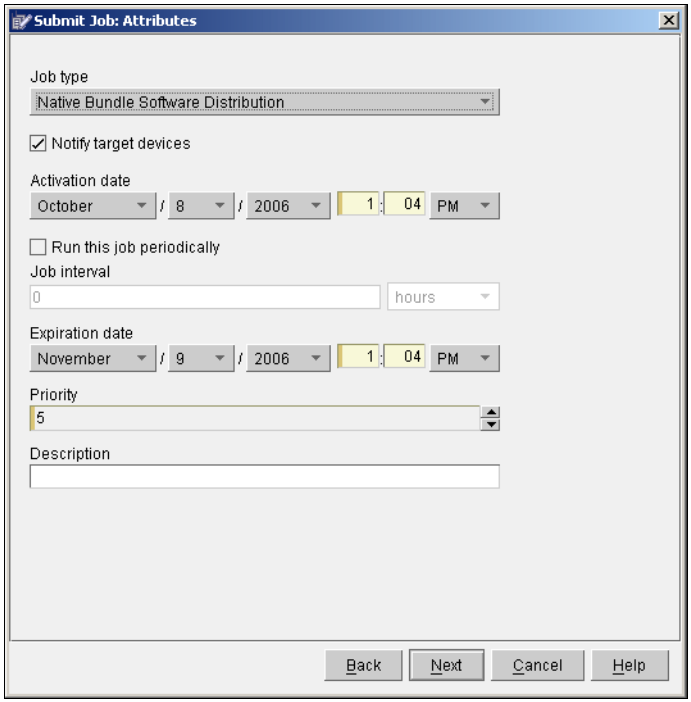


Figure 4-21 Submit Job: Attributes window

5. In the Submit Job:Job Parameters page (Figure 4-22), perform the following tasks:
   - Select **Add Group**.
   - Select the relevant registered software package from the list of available software.
   - Set Auto start install program to **True**.

   Click **Next**.

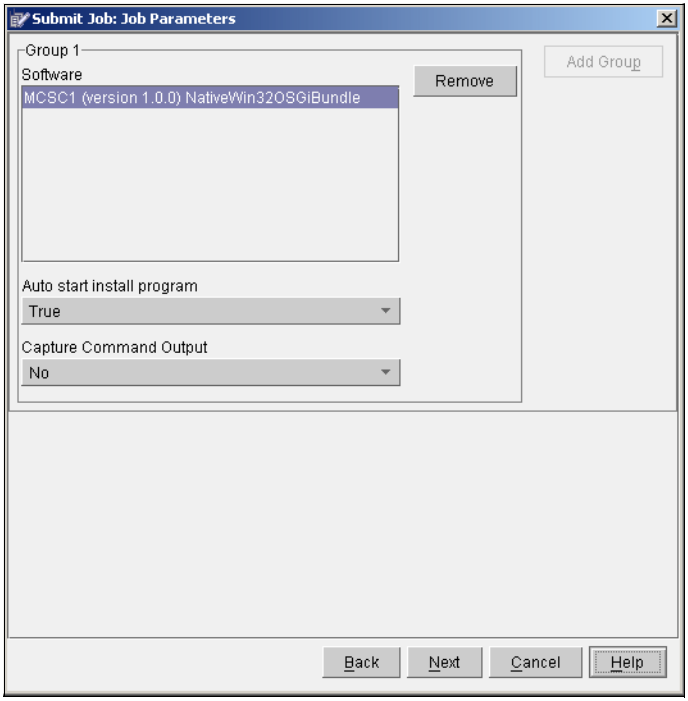


Figure 4-22 Adding a group

6. In the Submit Jobs:Summary page (Figure 4-23) click **OK**.

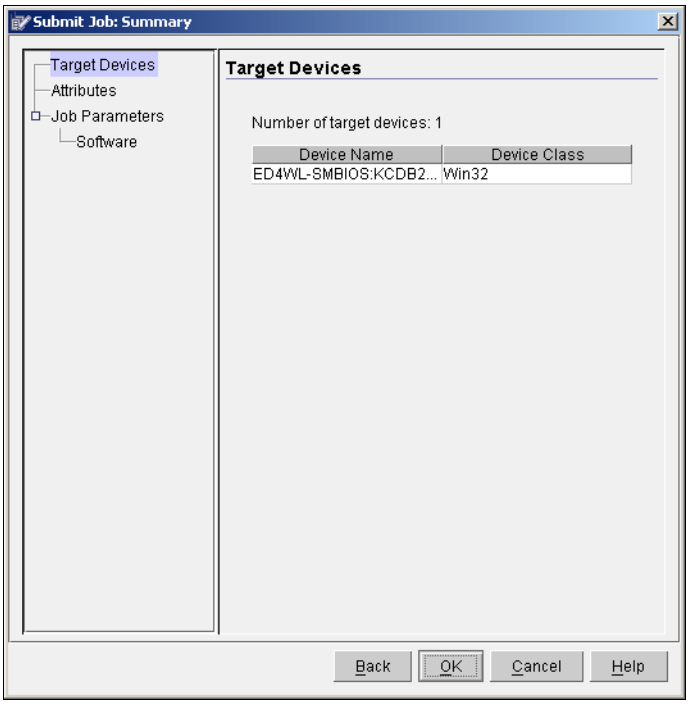


Figure 4-23 Job summary

7. After the job is submitted, click **Close**.

The job runs on each targeted client according to the parameters set in its polling configuration. (WED4WL must be running for the polling to occur.)

To check the status of the Job in the Device Manager console, perform the following tasks:

1. Select **Jobs** in the left-hand navigation pane.
2. Select **Return anything** and click **OK**.
3. Right-click the **Native Bundle Software Distribution** job and select **View Job Progress**... or **View Job Progress Summary**...

After the distribution job is completed, the features provided in the update site is installed on the file system on the client. The WEDWL prompts the user for an update when it finds a new update in the server (Figure 4-24).

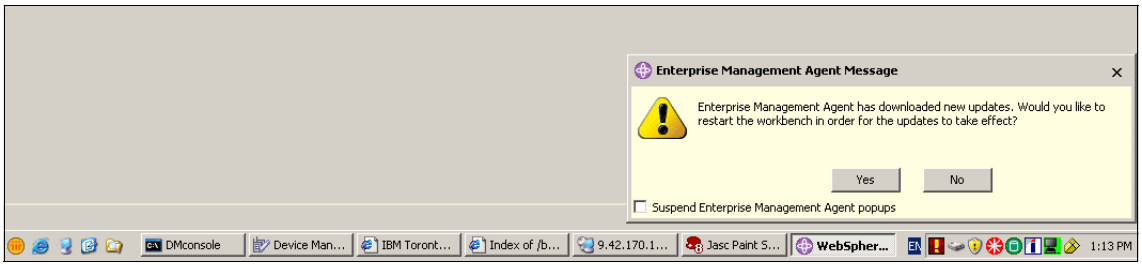


Figure 4-24 Agent prompt for new update install

After the update is completed, WEDWL restarts the client. A few features will then be accessible.

Uninstalling using a bundle control job

To uninstall the features, perform the following procedure within the Device Manager console:

1. Right-click **Device** and select **Submit Job**.
2. Select **Bundle Control** as the Job type (Figure 4-25) and click **Next**.

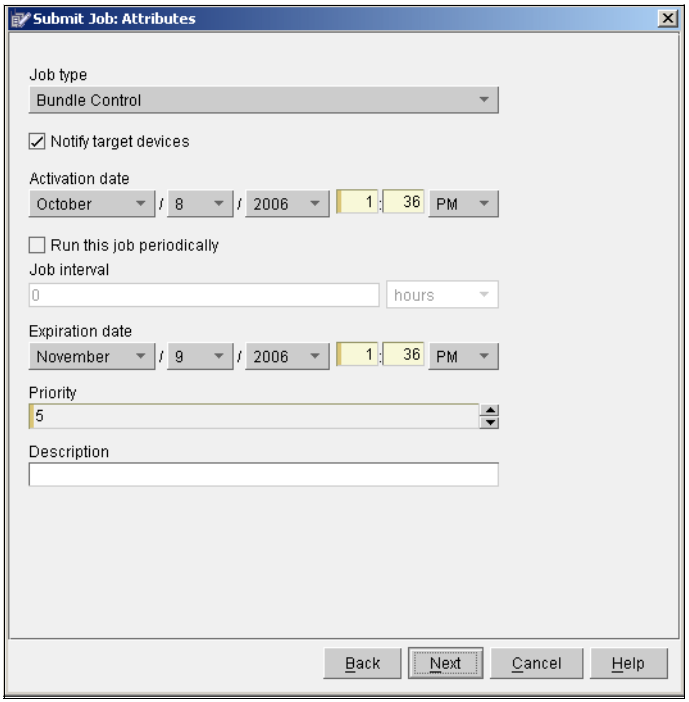


Figure 4-25 Selecting bundle control

3. In the page that appears, click **Add Step**.
4. In the page that appears, select **Uninstall** against Action.

5. Select the feature or features to be uninstalled from either the OSGi bundles from Inventory field or the OSGi bundles from repository not listed in inventory field selection list, and click **Next** (Figure 4-26).

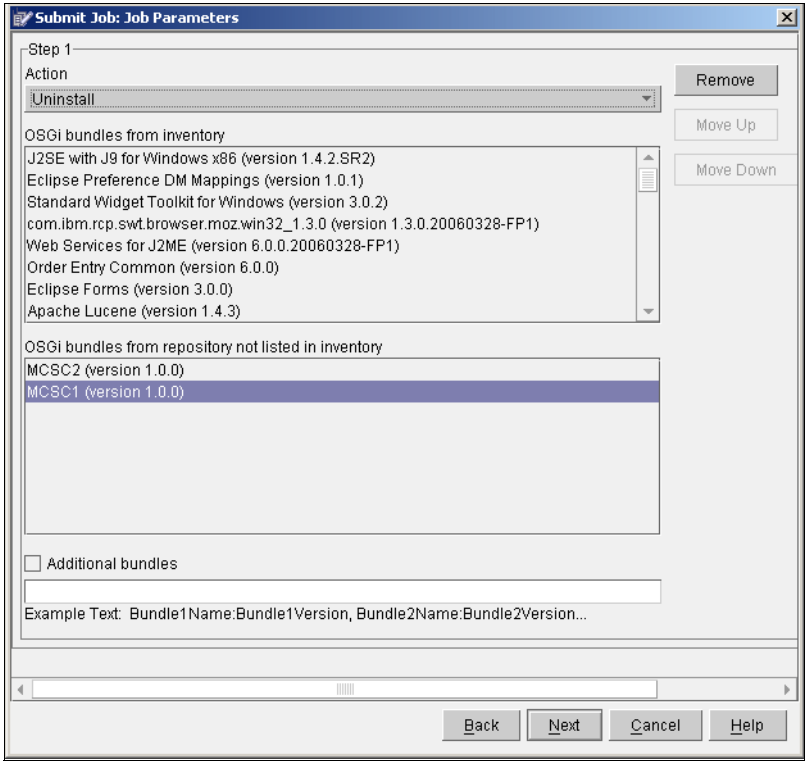


Figure 4-26 Selecting Uninstall and the bundle name

6. Click **OK**.
7. Click **Close**.
8. The feature is uninstalled when the job is received at the client end. Stop and restart the client after the feature is uninstalled.

You can distribute any additional customizations by using the procedure described in this section.

Part 3

IBM Sales Center customizations

This part lists and discusses the requirements and the design of IBM Sales Center and the process involved in installing and building the IBM Sales Center for WebSphere Commerce development environment and production environment.

5

Requirements and design

This chapter explains how to plan IBM Sales Center customizations. The first section explains the phased approach to planning customizations. The second section demonstrates the use of the phases within the example of a call center environment, including customer requirement scenarios and solution design.

5.1 Planning and designing IBM Sales Center customizations

Following are the planning and design phases of IBM Sales Center customizations:

1. Requirements gathering
2. Fit-gap analysis
3. Solution design
4. Macro design and micro design
5. Postdesign activities

5.1.1 Phase 1: Requirements gathering

In the requirements gathering phase, participants provide the requirements pertaining to the customization.

Participants include the following categories of people:

- Technical staff who understand the software and hardware systems used in the business process. These include architects, developers, and database administrators. This group may include people from the company that requires the customizations, people from a partner organization that is planning to implement customizations, and people from a software vendor organization, such as IBM.
- Businesspersons who understand the business and its processes such as business analysts and project managers
- Other stakeholders such as users, CEOs, and CIOs

The topics that are discussed in this phase include the following:

- Integration with systems outside WebSphere Commerce such as an interactive voice response (IVR) software or a product inventory system
- Existing business processes and the vision for the solution
- Project milestones, target milestones, and launch dates
- Nonfunctional requirements such as performance targets

The following requirements are considered specifically for IBM Sales Center:

- Business requirements
  - Changing the workflow within IBM Sales Center to match the flow of the existing business processes and to minimize the impact on customer service representatives (CSRs) when learning the new workflow
  - Viewing reports to analyze CSR productivity, for example, the number of calls processed by a CSR in 60 minutes
- Integration requirements
  - Integrating with a customer relationship management (CRM) system to load the additional customer data into IBM Sales Center
  - Providing *live help* capability to allow CSRs to send and receive instant messages to and from customers
- Nonfunctional requirements
  - A CSR must be able to find a product within 15 seconds and place an order within 60 seconds
  - Two hundred CSRs must be able to access IBM Sales Center concurrently with no performance degradation

5.1.2 Phase 2: Fit-gap analysis

In the fit-gap analysis phase, participants analyze each requirement to determine if the requirement already exists by default (“fit”) or if customization is required (“gap”).

Participants include technical implementors or product architects or both.

Fit-gap analysis includes the following:

- Determining which requirements are already met from the base install, and which are customizations. The participants might consider using as much of the out-of-the-box functions as possible in order to reduce customization costs and migration complexity. It might be easier for a business process to adapt to the product’s default process rather than implement extensive product customizations.
- Categorizing each customization as small, medium, or large according to the complexity
- Deciding on high-level implementation details

5.1.3 Phase 3: Solution design

The solution design describes the customizations at a high level, including any assumptions, dependencies, and the end goals of the solution. The solution

design must contain enough details to continue the design because it is the basis for the rest of the design phases.

The solution design document is presented back to the stakeholders from the requirements gathering phase for approval in order to ensure that everybody's requirements are fulfilled and that everyone understands what will be implemented.

Solution design includes the following information:

- Architecture, including components and their interaction, and software version levels
- Storyboards outlining any new or modified user interfaces
- Modified business processes and high-level use cases
- Sizings and time lines for the implementation

5.1.4 Phase 4: Macro design and micro design

In this phase, macro design and micro design documents are created to outline the technical aspects of the implementation in different levels of detail.

Macro design includes the high-level details of the solution. The information in the macro design document might include the following:

- A site map of a newly designed Web site
- Functional use cases
- A short description of any new or extended commands
- Summary diagrams detailing integration with original equipment manufacturer (OEM) software
- Database schema changes
- User interface mockups, wire frames, and screen captures

The micro design includes the lower-level technical details required for the customizations. The goal of micro design is to contain enough technical details for coding and unit testing to begin. A micro design document might, for example, contain the following:

- Specific information about new and extended commands, including input and output parameters, command logic, and the exceptions thrown
- New entity beans, finder methods, and data beans
- Any messages created for OEM software integration
- Eclipse extension points to be used
- New or modified Business Object Documents (BODs) required to pass custom data between the WebSphere Commerce server and the client and

any code that is required to handle this data, such as message mappers, request handlers, and response builders

5.1.5 Phase 5: Post-design activities

After the macro design and micro design phase, the following activities take place. The participants of these activities can include business partners, users, software vendor representatives, and people from the company that requires the customizations:

1. Development and unit test

   In this phase, developers write the code to implement the customizations. Developers also perform unit tests on their code to ensure that it works as expected.

   Developers must ensure that they write the proper Javadoc™ so that their code can be understood and extended by other developers.

2. Build

   In this phase, the build team collects all the code from multiple developers and packages the code to make it available for testing

3. Test

   In this phase, testers test the package created by the build team to ensure that all the different developers' customizations work together as expected. After testing is completed successfully, the package can be deployed to the production systems. Depending on the software, there might be multiple packages. In the context of IBM Sales Center customizations, two packages might be required, one to update the IBM Sales Center client, and one to update the WebSphere Commerce server.

4. Information development

   In this phase, information developers or technical writers write documentation for the users. They may, for example, create a user guide for the new functions or a customization guide so that developers can, in the future, further enhance the new code.

5. Deploy and verify

   In this step, the customizations are deployed to the IBM Sales Center rich clients and to the WebSphere Commerce server. Whether the customizations work as expected after deployment must be verified because of possible differences between the test environment and the production environment.

5.2 An example using IBM Sales Center

This section demonstrates the design phases described in the previous section, using WebSphere Commerce and IBM Sales Center.

5.2.1 Requirements gathering

In this example, the business users and the development team meet to discuss enhancements to the IBM Sales Center. The participants who are involved in the requirements gathering phase include architects, developers, business analysts, and a project manager from a call center company, and architects and developers from IBM, who help implement WebSphere Commerce and IBM Sales Center solutions.

The business analyst and the project manager provide the following requirements:

- CSRs must be able to find a customer record by searching for the customer's e-mail address
- CSRs must be prompted, possibly with a pop-up, to mention the promotions available at the store, for example, "Free shipping on orders over $50".
- A new loyalty points program requires that each customer record is associated with a loyalty points number and balance. Each time a customer makes a purchase, points are added to their loyalty points balance. The points can be redeemed against free merchandise.
- If a customer's order contains a back-ordered item, the customer must send an e-mail when the back-ordered product becomes available.

The technical users provide the following software and hardware requirements:

- The database system is DB2
- The operating system for the Sales Center clients is Windows XP, and the operating system for the WebSphere Commerce server is Windows 2003 Server
- Enhance the tax calculation process by integrating with a OEM taxation software package

5.2.2 Fit-gap analysis

In this example, the participants involved in this phase are technical implementors or product architects or both. Table 5-1 shows a fit-gap analysis summary.

Table 5-1 Fit-gap analysis summary

Requirement	Fit or Gap	Implementation details
Find customer by e-mail address	Fit	None
Prompt CSR to mention promotions	Gap	Involves customizing the IBM Sales Center client only
Loyalty points number and balance	Gap	Involves customizing both the IBM Sales Center and the WebSphere Commerce server
Send e-mail to customer when back-ordered product becomes available	Gap	Involves customizing the WebSphere Commerce server only
The database system is DB2	Fit	None
The operating system is Windows XP	Fit	None
Integration with a third-party taxation software	Gap	Involves customizing the IBM Sales Center client and the WebSphere Commerce server, and integration with OEM software

5.2.3 Solution design

The solution design is the basis for the rest of the design phase and is presented to all the stakeholders for approval before the implementation begins. The solution design for this set of requirements is described in detail in the following sections.

Approach

For each requirement, the methodology for the solution is described at a high level, for example:

- Some requirements are available from the base install, and therefore, no work is required

- Some requirements require customization of only IBM Sales Center, and not of WebSphere Commerce server. These types of customizations do not require custom data to be transferred between the client and the server. Also, this type of customization does not require any change to the commands that run on the WebSphere Commerce server side. The requirement of a pop-up prompting CSRs to mention store promotions is an example of this type of customization. This approach section must mention that an implementation detail from the fit-gap section, the promotion listed in the pop-up, is relative to the items in the order and not related to other attributes such as the customer's demographic information.
- Some requirements require customization of only WebSphere Commerce server. These type of customizations do not affect the user interface in the IBM Sales Center. Sending an e-mail notification when a back-ordered product is in stock is an example of this type of customization. This section must mention any implementation details, for example, the necessity for a user to have a valid e-mail address, and a description of the software used to manage the outbound e-mails.
- Some requirements require customization of both the IBM Sales Center client and the WebSphere Commerce server. The loyalty points requirement is an example of this type of customization. This type of customization requires changes to the business logic or Web storefront JavaServer Pages™ (JSP) on the WebSphere Commerce server side and changes to the business logic or user interface on the IBM Sales Center side. This section must describe implementation details, including the following:
  - Requirement details
    - Loyalty points are automatically calculated based on the order's total price
    - Loyalty points are deducted if an order is returned
    - Loyalty points redemption is handled by an OEM company and is out of scope of this project
    - CSRs must be able to view a customer's loyalty points number and balance
    - CSRs must be able to update a customer's loyalty points number
    - Customers must be able to register for a loyalty points number or add a loyalty points number to their profile on the store's Web page
    - Customers must be able to view their loyalty points balance on the store Web page
  - WebSphere Commerce server customizations
    - The name of the database table where the loyalty points number and balance are stored

- Any new entity beans required to fetch and update the database table
- How the point value is calculated when an order is completed, for example, before or after tax is added to the item
- Calculating and deducting the loyalty points when an item is returned, for example, whether to deduct the points when the customer calls to create the return or when the physical goods are received
- Changes to the JSP pages in the storefront to display or collect information
- Code required to handle any new or modified BODs

– IBM Sales Center customizations

- Modified or new editor pages to capture and display the loyalty points number and balance
- Modified or new BODs to transfer custom data between the WebSphere Commerce server and the IBM Sales Center client

Scope of the work

Following is the scope of the work of the project (this is to ensure that everyone understands what will and will not be completed):

- The work provided as part of the customization includes design, coding, testing, and documentation. Deployment to client machines will be tested but not implemented to the actual users.
- Deployment and user training of the new functionality is not within the scope of this work
- The redemption of loyalty points is not within the scope of this project

Testing requirements beyond the functional test

Functional test ensures that each new customized function works as expected. In addition to the functional test, following are the other types of test that might be required:

- User acceptance test

  This ensures that the functions implemented by the technical team is what the other stakeholders expect, for example, is the number of loyalty points calculated as expected?

- Usability test

  This ensures that IBM Sales Center is easy to use for a user, for example, in the graphical user interface (GUI), if the user expects the keyboard shortcut Ctrl+Q to quit the program, but the actual function creates a new quote, the user might be confused.

- Performance and scalability test

  Because IBM Sales Center will be used by many users in a call center, and the success of the call center is partially judged by how quickly customer calls are resolved, it is important to test the performance of the IBM Sales Center under different conditions, such as adding more CSRs or handling a high call volume.

Architecture and back-end system integration

Architecture and back-end system integration includes the following:

- System architecture details are explained here. This example does not require much architectural change. However, other scenarios may include information about how asset stores are used and the catalog hierarchy in the implementation.
- The requirement to use OEM software for tax calculation is considered as back-end integration, that is, this change must be invisible to the user. In this example, architectural information about how integration is implemented is explained.

Physical topology

This section details the physical hardware required for the solution. The hardware and software for the WebSphere Commerce server, the database, the Web server, the load balancers and firewalls, and the source control repositories for production, development, and test are described, including machine specifications such as types, operating systems, and the amount of RAM (Figure 5-1).

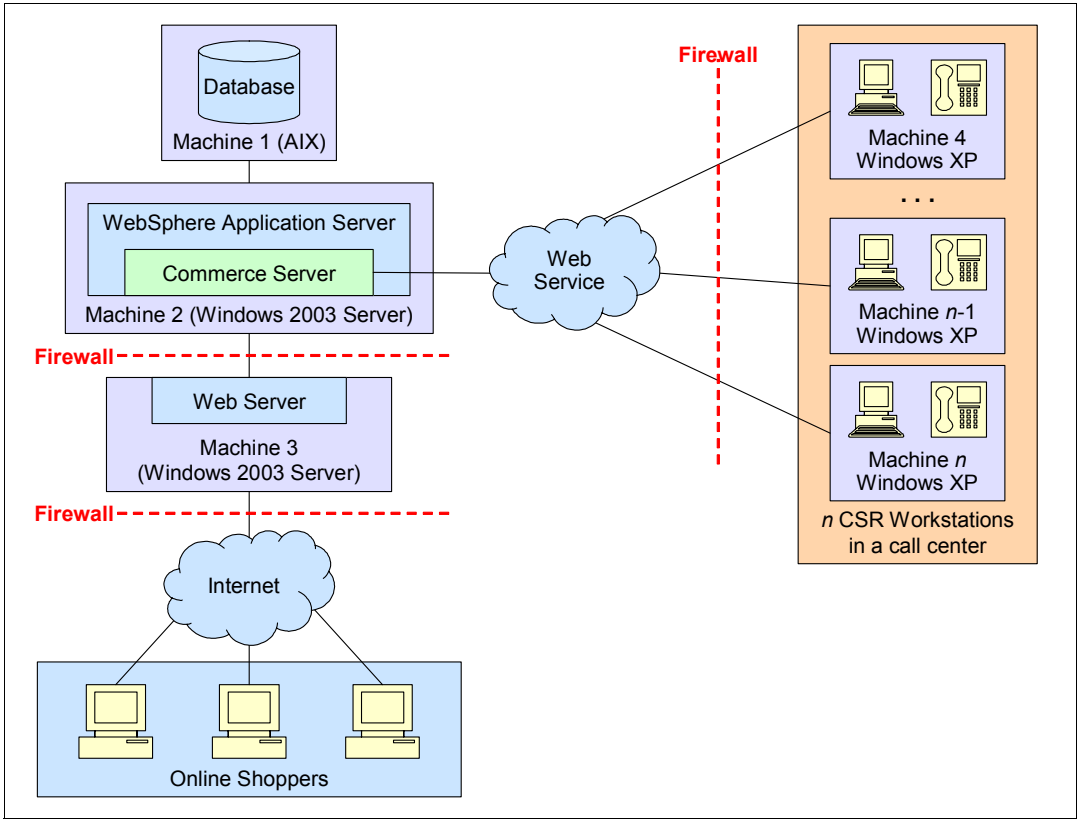


Figure 5-1 Sample diagram of physical topology

Deployment plan

The deployment plan includes the organization, strategy, resources, and methods used to deploy the customization. In this example, the term deployment means to make the customizations available in the production environment for use by the users. In this example, deployment is tested in a test environment, but not rolled out to the user CSRs. This section provides recommendations about how best to deploy the customizations. Deployment can also include user training or notification about the new functions.

5.2.4 Macro design and micro design

This section describes macro design and micro design by using the loyalty points requirement as an example.

Macro design

Macro design includes the order capture flow in which a CSR encounters the loyalty points option. The flow diagram shown in Figure 5-2 helps you understand where the loyalty points option fits in a macro design.

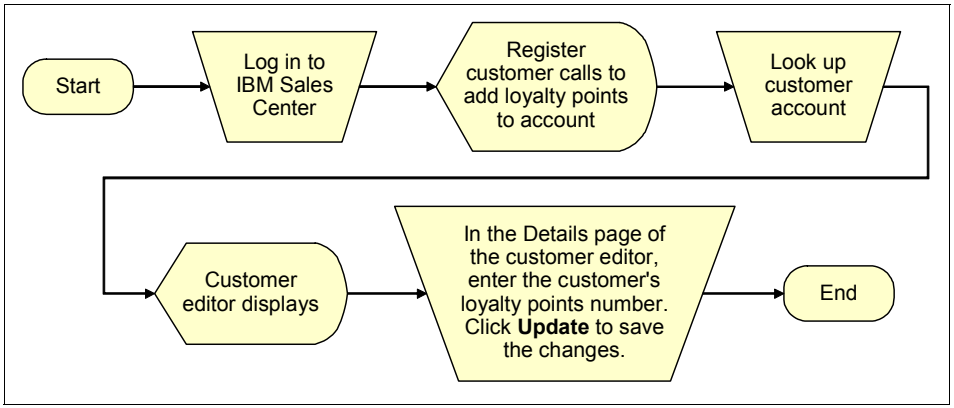


Figure 5-2 Example of IBM Sales Center flow with the loyalty points option

Mockups of the user interface are included here to provide you with a general idea about what the users will see. In this scenario, customers see the modified JSP pages, and the CSRs see the modified IBM Sales Center pages, as shown in Figure 5-3.

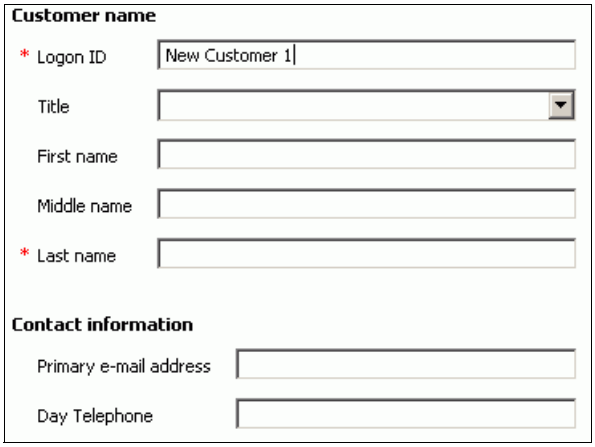


Figure 5-3 IBM Sales Center user interface mock-up (The label "Telephone number 1" has been changed to "Day Telephone")

Use cases are provided in the macro design to detail the tasks to be performed by a CSR in order to use the loyalty points functionality. These use cases must be used by the test teams to ensure that the function behaves as expected. Following are the use cases:

- Storefront
  - Registered customer adds loyalty points number to the existing account
  - New customer registers in the store and creates a new loyalty points number
  - Registered customer views the loyalty points balance
  - Registered customer makes purchase through the store Web site. Loyalty points balance must be updated appropriately.
- IBM Sales Center
  - CSR adds loyalty points number to the existing account on behalf of a registered customer
  - CSR registers and creates a new loyalty points number on behalf of a new customer
  - CSR looks up loyalty points balance for customer

- CSR places an order on behalf of customer. Loyalty points balance must be increased appropriately.
- CSR creates a return on behalf of customer. Loyalty points balance must be decreased appropriately.

Micro design

The micro design for the loyalty points requirement contains low-level details about implementation, including new or extended controller and task commands, database tables, entity beans, JSP pages, and any messages required for integration with systems that are external to WebSphere Commerce.

In the loyalty points example, the following customizations are required on the WebSphere Commerce server:

- A new database table is required to store the loyalty points number and balance. The database table name is X_LOYALTY.
- A new entity bean, access bean, and a data bean are required to access the data from the X_LOYALTY table.
- The following store JSPs must be modified to capture and display loyalty points information:
  - UserRegistrationAddForm.jsp
  - MyAccountDisplay.jsp
- New commands are required to calculate the number of loyalty points and to add or subtract from a customer's points total as required, including input and output and exceptions that can be thrown.

The following customizations are required for the IBM Sales Center:

- The Customer editor must be modified to display the customer's loyalty number and balance.
- When the customer editor is open for editing, there must be an input text box for a CSR to input a customer's loyalty number.
- BODs will be modified to use the user data method of transferring custom data between the WebSphere Commerce server and the IBM Sales Center in order to transfer the loyalty points information.
- Code must be written on the WebSphere Commerce server to send and receive the loyalty points information.

After the macro design and micro design documents are complete, the implementation phase and the test phase begin.

6

Customization scenarios

Customization implicitly references the set of features that are currently available by default. In this context, there are two types of customizations in WebSphere Commerce:

- The first type is more a configuration than a customization. This type includes setting up custom groups and templates, associating the existing policies to groups, and creating new promotion types using the existing components.
- The second type requires Java code to be written. An implied step in the second type of customization is that all your new or changed Java code must be available on the classpath of your WebSphere Commerce server for the Java Virtual Machine class loader to load the customized code properly.

In IBM Sales Center for WebSphere Commerce, there is a set of user interface panels, views, and dialog boxes that contain the most commonly used features. You can add to or reconfigure the user interface to suit your requirements. You can also change the content and the format of the data that is passed between the client and the server.

Thus, in addition to the WebSphere Commerce customization types mentioned at the beginning of this chapter, IBM Sales Center customization may be a simple, moderate, or extensive modification of the IBM Sales Center code, as described in Chapter 7, “Developing customizations for IBM Sales Center” on page 119.

This chapter describes three different customization scenario types depending on which components are required to be customized, IBM Sales Center client only, WebSphere Commerce server only, or both, and their high-level implementation.

This chapter also describes an additional type of IBM Sales Center customization scenario, the integration customization scenario, where integration with additional or OEM software is required.

6.1 IBM Sales Center client changes

This is the type of customization scenario where only the IBM Sales Center client changes are required. To implement some of the requirements (see Chapter 5, “Requirements and design” on page 95), only the IBM Sales Center client must be customized, and no modification of the WebSphere Commerce server is required. These may be simple, or even extensive modifications of the IBM Sales Center client code, as discussed in Chapter 7, “Developing customizations for IBM Sales Center” on page 119.

User interface changes do not always affect the WebSphere Commerce server. In other words, in some cases, the server does not have to be involved in the user interface display, the way data is captured, or any processing that may be implemented on the client side. These customizations do not require any additional (existing or custom) data to be transferred between the client and the server.

Following are the customizations that affect only the IBM Sales Center:

- Changing the look and feel
  - Implementing the new images and modifying the existing images
  - Implementing the new images and modifying the existing text (property files)
- Reorganizing the layout
- Changing the user interface logic
- Hiding and showing the existing widgets
- Creating new widgets that rely on data that is already available to the IBM Sales Center client

Refer to the information center topic “Tutorial: Customizing the appearance of the IBM Sales Center”, which is available on the Web at:

http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.telesales.developer.doc/tutorial/ttrsalescentercustomization_1.htm

A specific example of this type of customization is to prompt and remind the CSR to mention promotions to customers during order creation. The CSR will be prompted about all the available promotions in a pop-up before the CSR asks customers for payment information. The promotions listed in the pop-up will be relative to the items in the order that is placed, and will not be related to other attributes such as the customer’s demographic information.

This customization involves IBM Sales Center client code changes only because the marketing promotions view and data are already available in IBM Sales Center. The customization simply displays all the applicable promotions in a new pop-up window in order to remind the CSR to offer them to customers.

Another specific example of this type of customization is the IBM Sales Center language. IBM Sales Center has a large set of displayable text in various parts of its user interface, such as window titles, dialog box messages, labels, button text, and table column headers. These text strings are externalized from the code and placed in properties files called resource bundles. The properties files exist for the default language and the various supported locales.

By default, the English language IBM Sales Center resources are located in the com.ibm.commerce.telesales.resources plug-in in the resources directory in the telesalesResources_en_US.properties file. Alternative locale files are located in the com.ibm.commerce.telesales.resources.nl1 plug-in in the resources directory. However, note that not all the modifiable and translatable text and images are in the telesalesResources file.

When adding a new translation, you can define a fragment that extends the plug-in containing the original properties file. IBM Sales Center provides fragments with names containing nl1, which illustrates this. You can follow this pattern in these fragments to add fragments of your own, extending the same plug-ins.

Note that the data coming to the client from the server side may have to be translated on the server side. Thus, adding additional languages to the WebSphere Commerce and IBM Sales Center may require server side customization as well.

Refer to the information center topic “Globalization in the IBM Sales Center”, which is available on the Web at:

http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.telesales.developer.doc/concepts/ctrglobalization.htm

Refer to the information center topic “Resources”, which is available on the Web at:

http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.telesales.developer.doc/concepts/ctrresources.htm

6.2 WebSphere Commerce server changes

This is the type of customization scenario where only the WebSphere Commerce server changes are required. To implement some of the requirements in some cases only, as discussed in Chapter 5, “Requirements and design” on page 95, the WebSphere Commerce server side must be changed. However, no changes to the IBM Sales Center client are required.

Business logic changes do not necessarily affect the IBM Sales Center user interface, for example, the client side does not have to be involved in a scheduled job that runs on the server side. Another example is when the tax calculation on the server side is changed, and this modification does not affect the way the result is displayed on the client side; only the calculated result is changed on the client side. Some customizations may require changes only to the commands that run on the WebSphere Commerce server side.

These are the customizations that are required when only the WebSphere Commerce server must be modified, and no additional data has to be collected from the customer, as a result of which no additional data (either existing or custom) is to be transferred between the client and the server.

Following are the customizations that affect only the WebSphere Commerce server side:

- Sending an e-mail to the customer when a back-ordered product becomes available or when the order is shipped
- Changing the tax calculation, which does not affect the display of the result on the client side
- Automatic updating of the order state

Refer to the information center topic “Tutorial: Conducting an e-mail campaign”, which is available on the Web at:

http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.samples.doc/tutorial/tcpemail1.htm

Refer to the information center topic “Tutorial: Importing and exporting contracts”, which is available on the Web at:

http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.samples.doc/tutorial/tctcontractimportexport_1.htm

A specific example of this type of customization is sending an e-mail notification to all the affected customers when a back-ordered product becomes available. A scheduled job is running on the server and is checking the inventory table. When the inventory level becomes sufficient, the server sends e-mails to all the corresponding customers through the outbound messaging system, assuming that valid e-mail IDs have been provided to the CSR.

These customizations involve changing the WebSphere Commerce server side only. There is no requirement for any additional data to be collected from the customer in the IBM Sales Center client. This type of customization does not affect or require changes to the IBM Sales Center client user interface.

6.3 IBM Sales Center and WebSphere Commerce changes

This is the type of the customization scenario where IBM Sales Center client and WebSphere Commerce server changes are required. These may be moderate or extensive modifications, as discussed in Chapter 7, “Developing customizations for IBM Sales Center” on page 119.

To implement most of the requirements (see Chapter 5, “Requirements and design” on page 95), the IBM Sales Center client side user interface must be changed along with the business logic and the Web storefront JSP pages on the WebSphere Commerce server side.

These customizations are required when changes in the WebSphere Commerce server side affect the IBM Sales Center client behavior and display.

Also, when additional data (custom or existing) is to be collected from the customer and the data is to be transferred between the client and the server, these customizations affect both the display in the IBM Sales Center client and the processing in the WebSphere Commerce server side.

Following are the examples of customizations that affect both the IBM Sales Center client and the WebSphere Commerce server:

- Gift wrap order
- Loyalty points
- Adding customer pet information to the customer record (Chapter 9, “User interface customization” on page 171)
- Changing user interface behavior based on the IBM Sales Center user role (Chapter 10, “Role-based customizations” on page 257)

Refer to the information center topic “Tutorial: Adding a new search option in the IBM Sales Center”, which is available on the Web at:

http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.telesales.developer.doc/tutorial/ttrfinddialog.htm

Refer to the information center topic “Tutorial: Modifying a page in an editor”, which is available on the Web at:

http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.telesales.developer.doc/tutorial/ttradvanced_1.htm

Refer to the information center topic “IBM Sales Center-Adding a column to the order items table” in the information center, which is available on the Web at:

http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.telesales.developer.doc/tutorial/ttravaildate.htm

Refer to the information center topic “Tutorial: Creating new business logic” in the information center, which is available on the Web at:

http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.developer.tutorial.doc/tutorial/ttd09.htm

Refer to the information center topic “Tutorial: Creating a multicultural store” in the information center, which is available on the Web at:

http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.samples.doc/tutorial/tgbglobilization1.htm

A specific example of this type of customization is to enable the gift wrap offer for an order. The entire order is gift-wrapped and the gift-wrapping charge is calculated automatically based on the size of the ordered items. When creating the order, the CSR is given the option of selecting gift wrap to be added to the order. Based on the size of the items, the charge is calculated automatically on the server side and transferred to the client side. The gift wrap is then added to the order as a separate product.

Another specific example of this type of customization is loyalty points:

- The loyalty points are automatically calculated based on the order’s total price before tax, and added when an order is completed.
- Loyalty points are deducted if an order is returned.
- The customer editor page is modified to enable the CSR to view and display the customer’s loyalty points number and balance. When the customer editor page is open for editing, there is an input text box for a CSR to input a customer’s loyalty number.

- Business Object Documents (BODs) are modified to transfer custom data (the loyalty points information) between the WebSphere Commerce server and the IBM Sales client. A new code is created to handle new or modified BODs.
- New commands are created to calculate the number of loyalty points and to add or subtract from a customer's point total as required.
- The loyalty points number and loyalty points balance are stored in a new database table. A new entity bean, access bean, and a data bean are created to fetch and update the new database table.
- Loyalty points are deducted when the returned physical goods are received.
- Storefront JSPs are changed to allow the customer to register for the loyalty points program and to view and display the loyalty points number and loyalty points balance on the store Web page.

These customizations require both IBM Sales Center client and WebSphere Commerce server modifications.

6.4 Integration customization scenarios

Integration customization scenario is a customization scenario where, in addition to the three different customization scenarios types already discussed (IBM Sales Center client only, WebSphere Commerce server only, or both), integration with an additional software or an additional system might be required.

Following are the customizations that involve integration:

- Integration with a OEM taxation software
- Customer Care integration with Lotus Sametime (See Chapter 11, "Customer Care integration with Lotus Sametime" on page 287)
- Customer Service Report generation using WebSphere Commerce Analyzer (See Chapter 12, "Installing, configuring, and running the WebSphere Commerce Analyzer" on page 315 and Chapter 13, "Developing and customizing customer service reports" on page 347)

Refer to the information center topic "Integrating with back-end systems and external applications", which is available on the Web at:

http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.integration.doc/concepts/ccvcapabilities.htm

A specific integration customization scenario example is integration with a OEM taxation software. More complicated taxing structures may be implemented on the server and displayed on the client side, such as ship-to state, city, or zip code level. After integrating the WebSphere Commerce server with a third-party taxation software, the additional tax breakdown information is displayed on the IBM Sales Center client user interface.

7

Developing customizations for IBM Sales Center

This chapter introduces IBM Sales Center customizations and provides an overview of the IBM Sales Center components and the customization framework.

7.1 Skill prerequisites

In order to start customizing IBM Sales Center, the developer must be familiar with the following:

- Working with Extensible Markup Language (XML)
- Java programming
- Understanding the IBM Sales Center user interface (UI) framework
- Understanding the Eclipse framework
- Knowledge of WebSphere Commerce programming model
- Familiarity with IBM Sales Center development environment and WebSphere Commerce development environment.

The knowledge required for IBM Sales Center customization depends on those parts of the IBM Sales Center that are to be customized.

Performing simple modifications

To perform simple modifications to the UI, for example, removing a page from an editor or changing a key binding, you only have to know how to work with XML files.

Performing moderate modifications

To perform moderate modifications to IBM Sales Center, for example, customizing an editor and changing the data that is passed between the IBM Sales Center client and the WebSphere Commerce server, you require a knowledge of Java programming and must be able to work with XML files.

Performing extensive modifications

To perform extensive modifications to the UI, for example, adding a new editor and moving several UI elements to the pages of this editor, a general knowledge of the Eclipse platform is highly recommended. In particular, it is critical to understand the use of extension points. Passing additional data between the IBM Sales Center client and the WebSphere Commerce server and adding new business logic requires a knowledge of the WebSphere Commerce programming model.

7.2 IBM Sales Center architecture

The IBM Sales Center architecture comprises the IBM Sales Center client, the WebSphere Commerce server, and a messaging architecture. The information is transferred between the IBM Sales Center client and the WebSphere Commerce server in the form of Business Object Documents (BODs).

Business Object Documents

A BOD is an open standard of a common, horizontal message architecture developed by the Open Application Group. For more information about this, refer to the following Web site:

http://www.oagi.org

BODs are business messages exchanged between software applications or components. In this case, they are exchanged between the IBM Sales Center client and the WebSphere Commerce server.

The BOD informs the receiving system about what kind of message is there in the data area and the status and error conditions. The BOD comprises two parts, a noun and a verb. The noun is a common business object. The actions performed on the noun are the verbs. BODs are designed to be extensible when providing a common underlying architecture for integration (Figure 7-1).

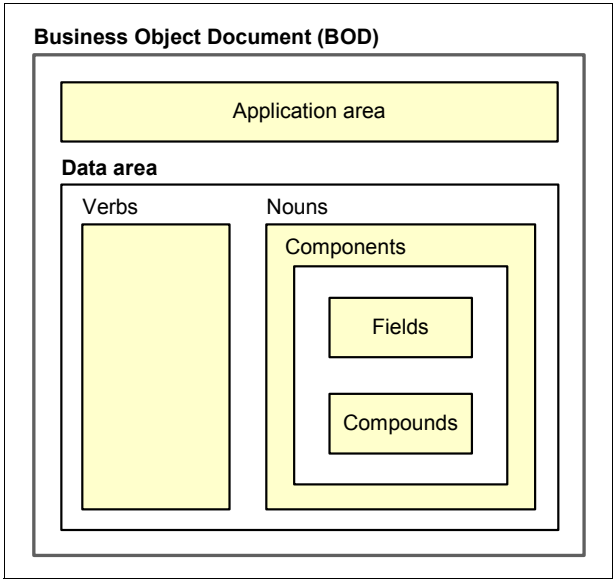


Figure 7-1 BOD structure

The BOD architecture is independent of the communication mechanism. The default transport protocol is SOAP over Hypertext Transfer Protocol Secure (HTTPS), but the BOD architecture can be used with many transport mechanisms, including the following:

- Hypertext Transfer Protocol (HTTP)
- Simple Mail Transfer Protocol (SMTP)
- SOAP
- electronic business XML (ebXML)

IBM Sales Center messaging architecture

The IBM Sales Center client contains a UI, a view into the WebSphere Commerce server data. The WebSphere Commerce server contains the business logic and data. The messaging architecture provides the integration between the two components.

Figure 7-2 illustrates the IBM Sales Center architecture and shows how the messages are passed between the IBM Sales Center client and the WebSphere Commerce server.

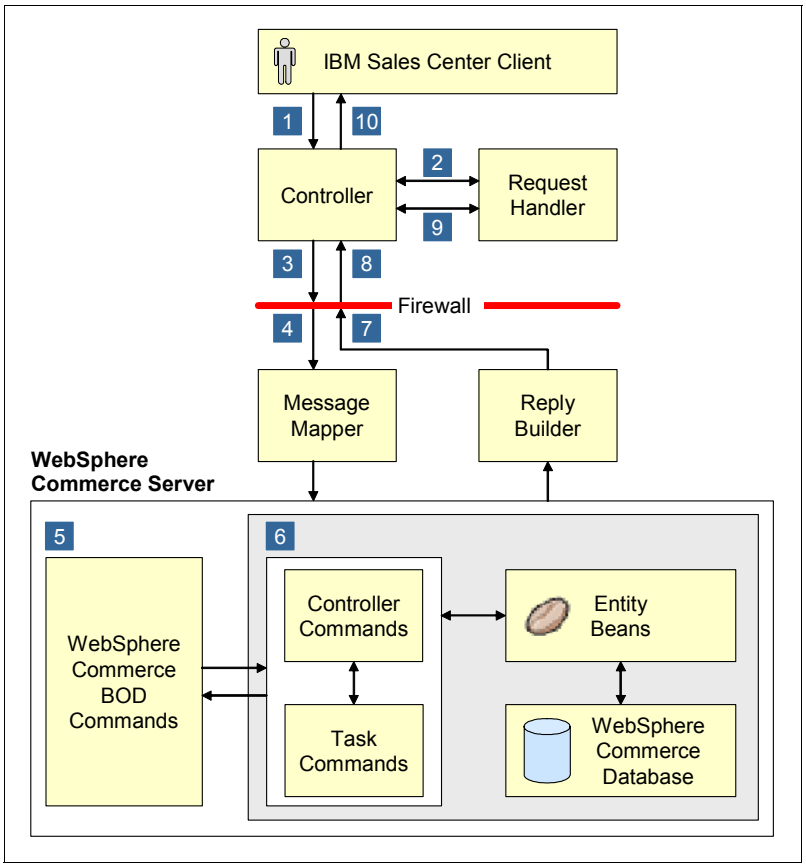


Figure 7-2 IBM Sales Center architecture

Following is the process involved:

1. The IBM Sales Center client performs a service request.
2. The service request handler prepares a BOD message.
3. The message is sent from the client to the server.
4. The message mapper receives the message and maps the BOD to a WebSphere Commerce BOD command.
5. The WebSphere Commerce BOD command is invoked.
6. The WebSphere Commerce BOD command calls a WebSphere Commerce Controller command, which might in turn call one or more task commands.

7. The reply or response builder constructs the response BOD.
8. The response is returned to the client machine.
9. The request handler receives and handles the response BOD.
10. The client UI is updated on the screen.

IBM Sales Center client architecture details

Figure 7-3 demonstrates the interaction between the IBM Sales Center client components and their response, resulting in the UI display changes.

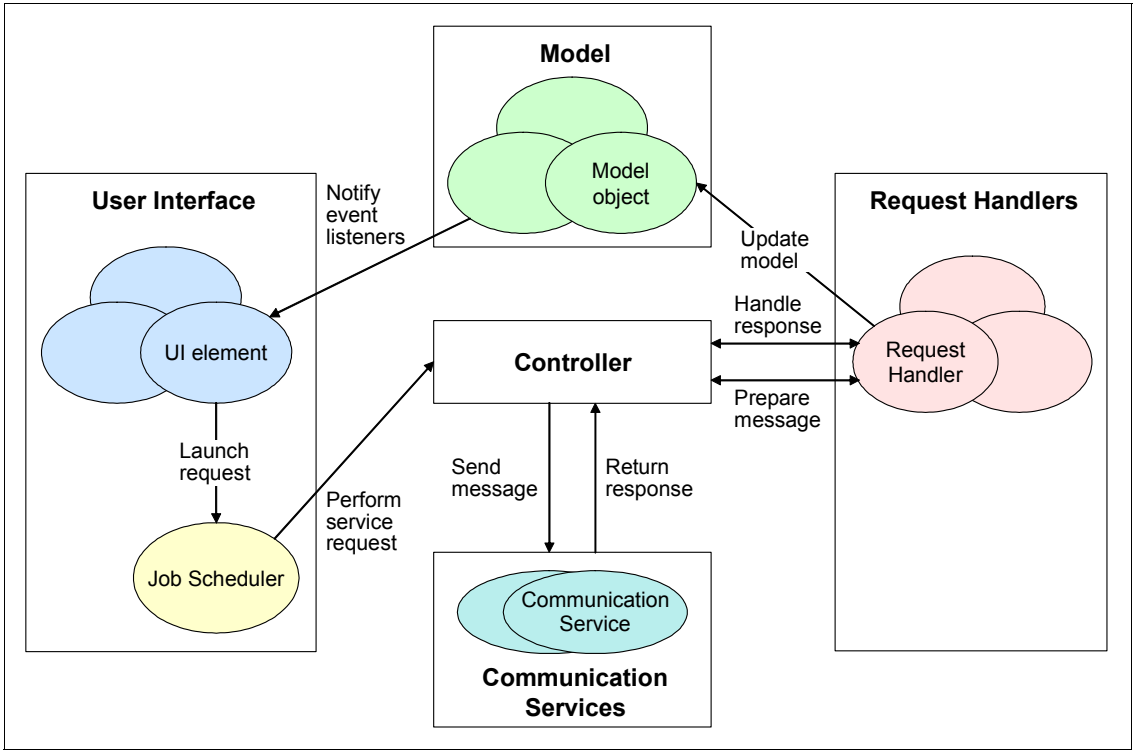


Figure 7-3 IBM Sales Center client architecture

Following is the process involved:

1. A UI element launches a request.
2. The Job Scheduler sends the request to the Controller.
3. The Controller performs the service request by passing it to the Request Handler.

4. The Request Handler handles the request, prepares the BOD message, and sends it to the Controller.
5. The Controller sends the message and receives the response message using the Communication Services.
6. The Request Handler handles the response message (reply BOD) and updates the model.
7. The event listeners notify the UI elements to update the data with the model change.

7.2.1 The Eclipse framework

The IBM Sales Center client relies heavily on the use of the Eclipse platform. Eclipse is a platform that has been designed for building integrated Web and application development tooling. The platform is extremely popular based on what it supports. It encourages the rapid development of integrated features based on a plug-in model.

Eclipse provides a common UI model for working with tools. Plug-ins can program to the Eclipse portable application programming interfaces (APIs) and run unchanged on any of the supported operating systems. At the core of Eclipse is an architecture for dynamic discovery, loading, and running of plug-ins.

You can customize IBM Sales Center by defining an extension in a customization plug-in and then using the system configurator to indicate that the application will use this new definition. Changes to the extensions in the XML files are read at the start and maintained in the Eclipse plug-in runtime registry.

Plug-ins contain units of function called extensions. You can customize the IBM Sales Center by adding new extensions defined in the customization plug-in by removing (suppressing) the existing extensions or by replacing the existing extensions with new ones.

7.2.2 The IBM Sales Center user interface framework

The IBM Sales Center UI framework is built on top of the Eclipse framework to make the customization of IBM Sales Center easier.

Wherever the Eclipse framework requires Java coding to develop UI elements, the IBM Sales Center framework allows the writing of simple XML tags to define the widgets and their layout.

The framework simplifies the sending and receiving of simple data (name-value pairs) by automating the common coding tasks.

The framework supports a system configurator that allows each plug-in to identify which extensions will be used in addition to or instead of the default IBM Sales Center extensions. The system configurator file is a text file that allows you to provide a substitute extension ID that will be used to replace a standard IBM Sales Center extension ID. The framework provides an extension point to specify the location of the system configurator file in each plug-in.

The IBM Sales Center framework must always be used to extend the existing IBM Sales Center UIs, for example, to reorder the layout, to remove a section, or to add new sections to an existing UI.

When creating new UI elements, there is a choice between using the base Eclipse framework or the IBM Sales Center UI framework. Create the new UI components by extending from the base Eclipse Java classes or from the Sales Center base classes. Customizing IBM Sales Center using the Eclipse framework involves creating Java classes to represent UI elements and the UI behavior when using the Sales Center UI framework.

7.3 Steps to develop customizations

Development of IBM Sales Center customizations involves customizing the two parts, the IBM Sales Center client and the WebSphere Commerce server.

Customizing the IBM Sales Center client

The customization of the IBM Sales Center client involves the following steps:

1. Create a plug-in project in the Sales Center development environment to contain the customizations.

   When customizing IBM Sales Center, place all the extensions in one or more plug-ins that you created specifically for your customizations.

   Note: Do not place your customizations in the default IBM Sales Center plug-ins. These plug-ins may be changed in a migration or fix pack and your customizations might be lost.

To create a new plug-in for the IBM Sales Center workspace, perform the following tasks:

a. Ensure that the current perspective is the Plug-in Development perspective. If it is not, select the **Window** → **Open Perspective** menu item, and switch to the **Plug-in Development** perspective (under Other).

b. Create a new plug-in project. Click **New** in the toolbar and select **Plug-in Development** → **Plug-in Project**. This launches the Plug-in Project wizard.

   Alternately, from the Eclipse File menu, launch the Plug-in Project wizard in the Eclipse environment.

c. Enter the required information using the Eclipse platform documentation as a guide. The Project name for your plug-in must follow the Java package conventions, for example, com.mycompany.salescenter.extensions.

   Ensure that you select the check box against **Create an OSGi bundle manifest for the plug-in**. Click **Next.**

d. Validate the information and click **Finish**.

2. Define the new system configurator for your plug-in. The system configurator identifies which extensions will be used over the default IBM Sales Center extensions for this plug-in.

   a. Create the system configurator config.ini file in the config directory:

      i. In the Package Explorer view, right-click the plug-in project and select **New** → **Folder**.

      ii. In the Folder name field, enter `config`, and click **Finish**.

      iii. Right-click the config folder and select **New** → **File**.

      iv. In the **File** name field, enter `config.ini`, and click **Finish**.

   b. Add the system configurator extension point to the plug-in to identify which extensions will be used over the default IBM Sales Center extensions:

      i. From the Package Explorer view, double-click the **plugin.xml** file.

      ii. Click the **plugin.xml** tab.

      iii. Add the text shown in Example 7-1 to define the config.ini as the new system configurator.

Example 7-1 Defining the config.ini as the new system configurator

```
<extension point="com.ibm.commerce.telesales.configurator">
<configurator path="config"/>
</extension>
```

3. Develop new extensions extending the IBM Sales Center client components. In this step, all the customization code will be developed to implement the new IBM Sales Center features. Refer to 7.4, “Developing the IBM Sales Center client components” on page 128 for customization details.

Customizing the WebSphere Commerce server

The customization of the WebSphere Commerce server involves developing WebSphere Commerce server components. In this step, all the customization code will be developed to implement the new IBM Sales Center features on the server side. Refer to 7.5, “Developing IBM Sales Center server components” on page 148 for customization details.

7.4 Developing the IBM Sales Center client components

This section discusses the development of various IBM Sales Center client elements that you may have to use in your day-to-day customization activities.

7.4.1 User interface organization

Figure 7-4 shows how the IBM Sales Center UI is organized. Most of these UI elements can be extended or replaced according to customer requirements. The Workbench consists of the following:

- Title bar

  This displays the program title and the icon.

- Menu bar

  This contains a set of actions provided either by the default Workbench or by the IBM Sales Center.

- Banner bar

  This optionally displays a graphic and an application name.

- Switcher bar

  This lists each running application as an icon from which a user can select applications.

- Data area

  This is the primary data area that contains the editors and the views for an application.

- Cool bar/Tool bar

  This optionally displays icons for the available actions.

- Status bar

  This is used by the application to display its status. It is at the bottom of the window.

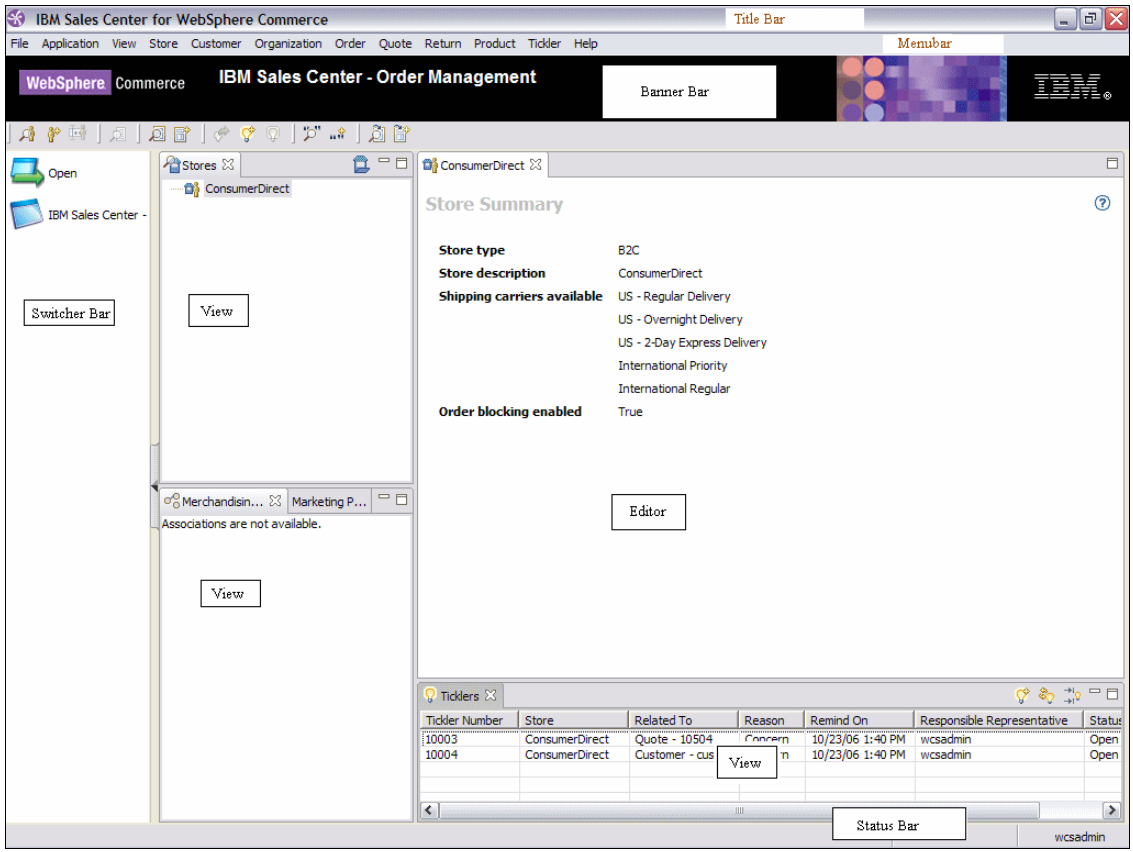


Figure 7-4 IBM Sales Center UI organization

7.4.2 User interface elements

IBM Sales Center has a workbench that contains the following high-level UI elements:

- Editors
- Dialogs
- Views
- Perspectives
- Preference pages
- Menus

Editors

Editors can consist of one or more editor pages. You can open any number of editors at a time, but only one can be active. In addition, multiple instances of the same editor can be opened. However, if an editor is opened for a specific store or order, a second instance of the editor for that store or order cannot be opened.

Many editors are provided with IBM Sales Center, such as the Store Summary editor, the Order editor, the Customer editor, the Quote editor, and so on. Define an editor by using the org.eclipse.ui.editors extension point.

To add a new editor and editor pages to IBM Sales Center, perform the following tasks:

1. Define a new editor using the org.eclipse.ui.editors extension point, which is a base Eclipse extension point, as shown in Example 7-2.

Example 7-2 Defining a new editor using the org.eclipse.ui.editors extension point

```
<extension point="org.eclipse.ui.editors">
<editor
name="<editor name>"
icon="<editor icon>"
class="<editor class>"
id="<editor identifier>">
</editor>
```

2. Define a new method in the factory (extended from TelesalesEditorFactory) to instantiate and return the new editor that is defined, as shown in Example 7-3.

Example 7-3 Defining a new method in the factory

```
public static IEditorPart open<New>Editor(IEditorInput input) {
return openEditor(input, "<Newly created editor identifier>");
//$NON-NLS-1$
}
```

3. To ensure that the pages your editor uses can be modified using the com.ibm.commerce.telesales.editorPages extension point, create an editor implementation Java class that extends the TelesalesMultiPageEditor abstract class. The IBM Sales Center has a number of base classes in the com.ibm.commerce.telesales.ui.editors package that can be subclassed to create editors and editor pages:
   - TelesalesMultiPageEditor

     This is a base class for a multipage editor.

- EditorPage

  This is a base class that all the pages in a TelesalesMultiPageEditor must subclass.

- TelesalesEditorPage

  This is a subclass of the EditorPage that provides support for the common toolbar and the button bar area.

- TelesalesConfigurableEditorPage

  This is a subclass of the TelesalesEditorPage that provides support for defining the editor page using managed composites.

- TelesalesEditorPart

  This is a base class for a single page editor.

- TelesalesConfigurableEditorPart

  This is a subclass of the TelesalesEditorPart that provides support for defining the editor using managed composites.

4. Create the page you want to add to the editor. TelesalesEditorPage or TelesalesConfigurableEditorPage are useful superclasses to extend. Add editor pages to the new editor (note that these pages must implement the EditorPage interface), as shown in Example 7-4.

Example 7-4 Creating a page to add to the editor

```
<extension
id="com.ibm.commerce.telesales.ui.impl.editors"
name="<editor pages name>"
point="com.ibm.commerce.telesales.ui.editorPages">
<editor
editorId="com.ibm.commerce.telesales.orderEditor"
id="com.ibm.commerce.telesales.orderEditorPages">
<page
name="<editor page name>"
class="<editor page class name>"
id="<editor page identifier>">
</page>
<page>
<!-- Next page -->
</page>
<!--More pages -->
</editor>
</extension>
```

5. Use the following TelesalesEditorFactory Java class to open your editor:

```
<New>TelesalesEditorFactory.open<New>Editor(<Model object
instance>);
```

Dialogs

Dialogs are common UI elements in IBM Sales Center. They are used when a user performs an action and has to provide additional information. The contents of a dialog are UI elements known as controls and managed composites.

To define and invoke a new dialog from IBM Sales Center, perform the following tasks:

1. Define a new dialog using the dialogs extension point. Using this extension point allows others to replace or extend your dialog. It allows you to reuse the same mechanism for launching and managing the dialog that the IBM Sales Center uses. Create a new plugin.xml (or add to your plugin.xml) as shown in Example 7-5.

Example 7-5 Defining a new dialog using the dialogs extension point

```
<extension point="com.ibm.commerce.telesales.ui.dialogs">
<dialog
class="<Custom dialog class>"
id="<Custom dialog identifier>">
</dialog>
</extension>
```

2. Create a new Java class extending the Eclipse framework org.eclipse.jface.dialogs.Dialog or the Sales Center UI framework dialog base classes in the com.ibm.commerce.telesales.ui.dialogs package:
   - Dialog

     This is a base class for a dialog.
   - ConfiguredDialog

     This is a subclass of a dialog that provides support for the managed composites.
   - ConfiguredMessageLineDialog

     This is a subclass of dialog that provides support for the managed composites and message-line area.

– TitleAreaDialog

  This is a base class for dialogs with a title area.

– ConfiguredTitleAreaDialog

  This is a subclass of TitleAreaDialog that provides support for the managed composites.

3. Provide a way to open your new dialog. You can open it programmatically or add a menu action or button. Dialogs are instantiated using the getDialog method in the com.ibm.commerce.telesales.ui.dialogs.DialogFactory class, as shown in Example 7-6.

Example 7-6 Opening a new dialog

```
IDialog myDialog =
      DialogFactory.getDialog("extension.customDialog");
myDialog.open(); //show the dialog
```

Views

Different views are available for customization in IBM Sales Center. These views are defined in the com.ibm.commerce.telesales.ui.impl plug-in using the base Eclipse product org.eclipse.ui.views extension point. The different views available in IBM Sales Center are Store view, Marketing Promotions, Ticklers, and so on.

Perform the following tasks to create a new view:

1. Define an extension to the org.eclipse.ui.views extension point, as shown in Example 7-7.

Example 7-7 Defining an extension to the org.eclipse.ui.views extension point

```
<view
name="<view label>"
icon="<icon file>"
category="<category of view>"
class="<view class>"
id="<view id>">
</view>
```

2. To have this new view appear on the IBM Sales Center perspective, extend the IBM Sales Center perspective.

Perspectives

A perspective is a particular rendering of IBM Sales Center that contains a predefined combination of views and editors. There are two perspectives in IBM Sales Center, the Orders perspective and the Web Browser perspective, which enables different sets of tasks. Both perspectives can be open and visible, but only one can be active at a time.

Perspectives are represented in the WebSphere Everyplace Deployment platform (and in the IBM Sales Center client) as applications, and are launched from the Application menu.

The Orders perspective is used to create and manage orders, quotes, customers, and organizations, and to view and work with merchandising associations, marketing promotions, and ticklers. It is the default perspective that opens each time you log in to IBM Sales Center and it is where users perform a majority of their work.

To add a new perspective, perform the following tasks:

1. Define an extension to the base Eclipse extension point org.eclipse.ui.perspectives (this declaration contains the basic elements such as ID, name, and class), as shown in Example 7-8.

Example 7-8 Defining an extension to the base Eclipse extension point

```
<extension
  point="org.eclipse.ui.perspectives">
  <perspective
  name="<New perspective name>"
  class="<New perspective class>"
  id="<New perspective id">
  </perspective>
</extension>
```

2. Create a new perspective class to define the initial layout of the perspective. You can write this class from scratch or from the subclass com.ibm.commerce.telesales.ui.impl.OrdersPerspective. (Refer to the class API information (Javadoc) for more details.) This class must implement org.eclipse.ui.IPerspectiveFactory and its method, createInitialLayout.

Implementers can add views, folders, actions, action sets, and other objects to the page layout. In this example, com.xyz.perspective.NewPerspective is the class that defines the initial layout of the perspective, as shown in Example 7-9.

Example 7-9 Creating a new perspective class to define the initial layout of the perspective

```
package com.xyz.perspective;
import org.eclipse.ui.IPageLayout;
import org.eclipse.ui.IPerspectiveFactory;
public class PerspectiveFactory implements IPerspectiveFactory {
public void createInitialLayout(IPageLayout layout) {
defineActions(layout);
 }
}
```

Note: The method defineActions defines the actions associated with this new perspective. An alternate method is to supply an IPerspectiveFactory that does nothing, and then use the perspective extension mechanism to define its contents in XML instead of in code.

3. Define an application to select the perspective. In the WebSphere Everyplace Deployment platform, perspectives are accessed by selecting an application. Applications are defined using the WebSphere Everyplace Deployment com.ibm.eswe.workbench.WctApplication extension point, as shown in Example 7-10.

Example 7-10 Defining an application to select the perspective

```
<extension
id="<New application extension id>"
point="com.ibm.eswe.workbench.WctApplication">
<DisplayName>New application</DisplayName>
<Icon>icons/full/eview16/new_perspective.gif</Icon>
<PerspectiveId>
<!-- Perspective identifier-->
</PerspectiveId>
<Version>6.0.0</Version>
<Description>This is a new application</Description>
</extension>
```

Preference pages

You can set preferences in the IBM Sales Center UI to make the process of working with orders, customers, and stores easier. You can, for example, set a preference to auto select a particular store or create customers by default with certain country, currency, and language attributes. There are preference pages for communication preferences, login preferences, customer preferences, order preferences, store preferences, search preferences, and image preferences. Preference page definitions for default preference pages are in the com.ibm.commerce.telesales.config plug-in manifest file (plugin.xml).

Preference pages in IBM Sales Center are defined using the org.eclipse.ui.preferencePages extension point.

Menus

The default IBM Sales Center actions are defined by the com.ibm.commerce.telesales.ui.actionSetGroups extension point. This extension point allows you to specify a list of actionSet IDs and an ID for the whole group. When an IBM Sales Center perspective is loaded, it uses a predefined ID to determine which action set group to use. You can use the system configurator plug-in to override an entire action set group or a specific action set that is already defined as part of the action set. All the out-of-the-box IBM Sales Center menu items delegate their actual work to an action that is registered with the com.ibm.commerce.telesales.ui.actions extension point. This allows you to use the system configurator to replace the implementation of a single action.

Perform the following tasks to define a new menu action:

1. Define your actions using the base Eclipse extension point, org.eclipse.ui.actionSets. The sample shown in Example 7-11 defines a single action in an action set.

Example 7-11 Defining actions using the base Eclipse extension point

```
<extension point="org.eclipse.ui.actionSets">
<actionSet
label="<Action set name>"
id="extensions.extendedActionSet">
<action
label=""
class="<Extended workbench action delegate identifier>"
```

```
menubarPath="<Menu bar path>"
id="<Action identifier>">
</action>
</actionSet>
</extension>
```

2. When writing a new workbench action delegate class, ensure that you extend from the com.ibm.commerce.telesales.ui.actions.TelesalesWorkbenchActionDelegate class. The real work for an IBM Sales Center workbench action is performed in an action registered with the com.ibm.commerce.telesales.ui.actions extension point, as shown in Example 7-12.

Example 7-12 Writing a new workbench action delegate class

```
package extensions;
import org.eclipse.jface.action.Action;
public class ExtendedAction extends Action {
public void run() {
System.out.println("running extended action");
}
}
```

The ExtendedWorkbenchActionDelegate.java looks as shown in Example 7-13.

Example 7-13 ExtendedWorkbenchActionDelegate.java

```
package extensions;
import org.eclipse.jface.action.IAction;
import
com.ibm.commerce.telesales.ui.actions.TelesalesWorkbenchActionDelegate;
public class ExtendedWorkbenchActionDelegate extends
TelesalesWorkbenchActionDelegate {
public void init(IAction action) {
//Initialize the actions
}
public String getDelegateActionId() {
return "extensions.ExtendedAction";
}
}
```

3. Indicate the ID of the delegate action in your getDelegateActionId method.
4. To include the new action in the action set group for a perspective, perform the following tasks:
   a. Define a new action set group
   b. Use the system configurator to indicate that your action set group will be used instead of the default one. Under normal circumstances, you must include all the default action sets and your new one. This can be done by referencing the default action set group in your action set group definition. Example 7-14 is a new action set group definition for the Orders perspective, which includes the test.actionSet action set.

Example 7-14 New action set group definition for the Orders perspective

```
<extension
name="<Extended Orders Perspective ActionSet Group>"
point="com.ibm.commerce.telesales.ui.actionSetGroups">
<actionSetGroup id="extensions.actionSetGroup.orders">
<actionSetGroupContribution
actionSetGroupId="com.ibm.commerce.telesales.actionSetGroup.orders"/>
<actionSetContribution
actionSetId="extensions.extendedActionSet"/>
</actionSetGroup>
</extension>
```

This example also requires you to add the following entry to your system configurator file:

```
com.ibm.commerce.telesales.actionSetGroup.orders=<new action set
group defined>
```

7.4.3 IBM Sales Center framework user interface elements

The elements in the layout of IBM Sales Center dialog boxes and editors are constructed using configured control.

Configured control

A configured control is an UI extension point that is declared by using the controls extension point. The definition of a configured control includes a unique identifier, the type of control, and additional attributes and properties that are specific to the type of control being defined. Configured control can be a button, a text field, or a text area. The control type is used to locate the control factory for creating a new configured control. Define a configured control, as shown in Example 7-15.

Example 7-15 Defining a configured control

```
<extension point="com.ibm.commerce.telesales.widgets.controls">
<control
id="<control identifier>"
type="text"
tooltip="<tooltip text>"
label="<text>"
managerType="<widget manager>"
required="true"
editable="true">
<property name="<custom property name>" value="<custom property
value>"/>
</control>
</extension>
```

Composite control

A composite control is a control that has child controls. The layout of the children of a composite control are defined using the compositeDefinition extension point. Composite definitions can be arranged as either a grid layout or a form layout. A composite definition looks as shown in Example 7-16.

Example 7-16 Composite definition

```
<extension
point="com.ibm.commerce.telesales.widgets.compositeDefinitions">
<gridCompositeDefinition id="<Composite definition identifier>"
layoutId="defaultLayout">
```

```
<row id="<row identifier>">
<control controlId="<control identifier>"
dataId="fillHorizontalGridData"/>
</row>
</gridCompositeDefinition>
</extension>
```

Managed composites and widget managers

A managed composite is a composite control along with one or more widget managers. A managed composite is generated using the managed composite factory. The initialization information is passed to the managed composite factory and is available to the widget managers. The composite and its child controls are constructed and the life cycle events delegated to the widget managers. There are several levels of composite controls, including other composite controls, before you reach a top-level composite control that will be the composite referred to in the managed composite. The widget managers are responsible for defining the behavior of all the controls under this top-level composite.

The behavior of the controls on the IBM Sales Center UI is managed by the widget managers. These managers initialize, refresh, save, and dispose the configured controls. They are also responsible for providing the content of table cells. Widget managers can add complex behavior to controls by adding listeners and handlers to the controls during initialization.

A standard widget manager is provided with the IBM Sales Center, and is used with standard controls and tables. When a configured control is defined, it can be declared with a manager type. The manager type is read by the widget manager to decide if the control must be managed by the widget manager. Widget managers will only manage the controls that have a manager type they recognize. If a control does not declare a manager type, it is assumed to be standard. Define a managed composite as shown in Example 7-17.

Example 7-17 Defining a managed composite

```
<managedComposite id="<managed composite identifier>"
compositeId="<composite identifier extended from>"><widgetManager
id="<widget manager identifier>"/><widgetManager id="<default standard
widget manager>"/></managedComposite>
```

The defined managed composite identifier must be referred to in the configurable page of the editor. The code fragment in Example 7-18 shows how the managed composite is referred to in the associated configurable page.

Example 7-18 Referring a managed composite in the associated configurable page

```
public static final String <CUSTOM>_MANAGED_COMPOSITE_ID =
"com.ibm.commerce.telesales.ui.impl.<managed composite identifier>";
//$NON-NLS-1$

protected String getPageContentManagedCompositeId() {
return <CUSTOM>_MANAGED_COMPOSITE_ID;

}
```

The XML fragment displayed in Example 7-19 shows how the widget manager is defined.

Example 7-19 Defining a widget manager

```
<widgetManager id="<widget manager identifier>" managerClass="<Custom
widget manager>"/>
```

Figure 7-5 shows the relationship between the editor and dialog and their managed composite.

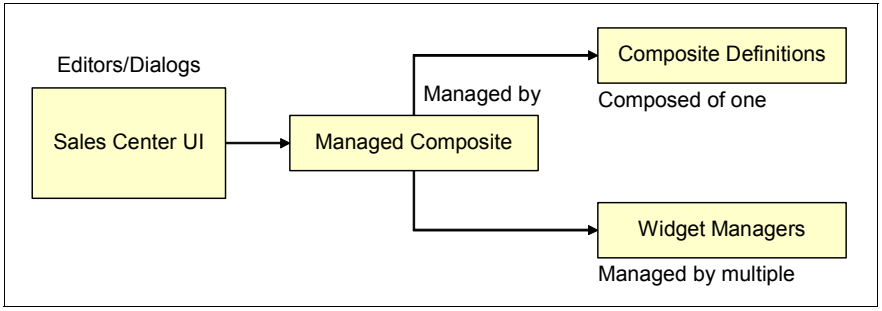


Figure 7-5 Relationship between the editor and the dialog and their managed composite

7.4.4 Service requests and Service request handlers

These classes can be extended to construct the BOD and interpret the response. An implementation must be provided for the interface ITelesalesRequestHandler by extending the TelesalesRequest.

Each type of service request in IBM Sales Center has its own service request handler. To customize the BOD format, create a subclass and customize this class. All the service request handlers are defined in the com.ibm.commerce.telesales.core.impl plug-in manifest file (plugin.xml). The plug-in manifest file has the list of the service requests and the corresponding request handler class, and the communication service ID. A service request handler definition looks as shown in Example 7-20.

Example 7-20 Service request handler definition

```
<extension
point="com.ibm.commerce.telesales.core.serviceRequests">
   <serviceRequest
      label="<Service request logon>"
      requestHandlerClass="<Request class>"
      id="<Service request identifier>"
      commServiceId="<Communication service identifier>">
   </serviceRequest>
</extension>
```

7.4.5 Model object

A model object can be extended by subclassing the ModelObject class. The model object must be registered with the model object extension point, and the instance created using the TelesalesModelObject factory.

Add instances of your new class to the model as properties of other model objects. If your model object is at the root level, add it as a property of ModelRoot. If your model objects are actually a list of model objects, use the ModelObjectList to group your model objects to a list. Add ModelObjectList as a property of the parent object, ModelObject.

An additional property can be added by the IBM Sales Center model object as follows:

1. Select a unique property name. To ensure that the property name does not clash with a default property name, prefix the property name with `ext_`.
2. Use the setData and getData methods in the ModelObject class to add, remove, and reference your new property.

If the new property is actually a list, use the ModelObjectList class to collect the list of values together. This ensures that the change notification is sent to the objects listening when changes are made to the list or to any of the model objects in the list.

Retrieve the IBM Sales Center client model objects by using TelesalesModelManager as follows:

```
TelesalesModelManager myModelManager =
TelesalesModelManager.getInstance();
```

Usually, data model implementation involves the following tasks:

1. To create an implementation of a model listener, register the listener to the model object that you want to listen to, as shown in Example 7-21.

Example 7-21 Registering the listener to the model object

```
public class <custom>ModelListener implements IModelListener
{
public void modelChanged(ModelEvent modelEvent)
{
// Do some thing with the property
}
}
```

2. Create an extension to the org.eclipse.ui.startup extension point in the extensions plug-in (plugin.xml) as shown in Example 7-22.

Example 7-22 Creating an extension to the org.eclipse.ui.startup extension point

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.0"?>
<plugin>
   <extension point="org.eclipse.ui.startup">
      <startup class="extensions.ExtendedStartup"/>
   </extension>
</plugin>
```

3. To hear all the changes to the model, add your listener to the ModelRoot object, as shown in Example 7-23.

Example 7-23 Adding the listener to the ModelRoot object

```
package extensions;
import org.eclipse.ui.IStartup;
import com.ibm.commerce.telesales.model.TelesalesModelManager;
public class ExtendedStartup implements IStartup {
```

```
public void earlyStartup() {

TelesalesModelManager.getInstance().getModelRoot().addModelListener(new
ExtendedModelListener());
 }
```

7.4.6 UserData property

Use the UserData element to pass the user-specific information to the server as name-value pairs and back to the client. The UserData properties can be added as parameters to the model object that is being passed to the service request handler through the TelesalesProperties. Any property added to the model object as a UserData property is automatically added as a UserData element in the BOD request:

```
<ModelObject >.setData(<property name>,<property value>);
<Model Object>.addUserDataProperty(<property name>);
```

If you have a model object instance for a customer and want to add a property called `place_of_birth`, perform the task as shown in Example 7-24.

Example 7-24 Adding a property called `place_of_birth`

```
public static final String PLACE_OF_BIRTH="place_of_birth";
customer.setData(PLACE_OF_BIRTH,"toronto");
customer.addUserDataProperty(PLACE_OF_BIRTH);
```

7.4.7 UserData support for the command extension

There are two types of extensions that are supported with BODs in IBM Sales Center, which do not require a change to the BOD schema definitions. For the create and sync BOD messages, you can add new name-value pairs to the UserData element. By default, the create and sync BODs map the name-value pairs in the UserData element to the request properties of the mapped controller command. For the get BOD messages, you can add new search criteria to the ReturnCriteria element.

7.4.8 Dynamic extension ID resolvers

The dynamic extension ID resolvers are responsible for resolving a dynamic ID into the ID of a valid extension declaration.

Following are the customization tasks involved:

1. In the plugin.xml file of your new plug-in, declare the dynamic IdResolver using the com.ibm.commerce.telesales.dynamicIdResolvers extension point as shown in Example 7-25.

Example 7-25 Declaring the dynamic IdResolver

```
<extension point="com.ibm.commerce.telesales.dynamicIdResolvers">
   <dynamicIdResolver
      id="<CustomIdResolver>"
      class="<extension Id resolver class>"/>
</extension>
```

2. After you define a dynamic ID resolver, define a new dynamic ID against the composite definition that you are going to use, as shown in Example 7-26.

Example 7-26 Defining a new dynamic ID

```
<extension point="com.ibm.commerce.telesales.dynamicIds">
<dynamicId
id="<CustomCompositeDefinition>"
resolverId="<CustomIdResolver>"/>
</extension>
```

The composite definitions appear as shown in Example 7-27.

Example 7-27 Composite definitions

```
<extension
point="com.ibm.commerce.telesales.widgets.compositeDefinitions">
<gridCompositeDefinition id="<CustomCompositeDefinition.type1>"
layoutId="defaultLayout">
<row id="orderGeneralPageCustomerInfoHeaderRow">
<!-- controls here--->
</row>
<!-- Additional rows here-->
</gridCompositeDefinition>
<gridCompositeDefinition id="<CustomCompositeDefinition.type2>"
layoutId="defaultLayout">
<row id="orderGeneralPageOrganizationInformationRow">
```

```
<!-- controls here--->
</row>
<!-- Additional rows here-->
</gridCompositeDefinition>
</extension>
```

3. The composite definitions are loaded based on the code in the ID resolver class, as shown in Example 7-28.

Example 7-28 Loading composite definitions

```
public String resolveId(String dynamicId, Set candidateIds,
ResolverContext context) {
String resolvedId = null;
if (TelesalesModelManager.getInstance().getActiveStore() != null) {
StringBuffer strBuf = new StringBuffer(dynamicId);
strBuf.append(".");
strBuf.append(TelesalesModelManager.getInstance().getActiveStore().getT
ype());
String id = strBuf.toString();
if (candidateIds.contains(id)) {
resolvedId = id;
}
}
if (resolvedId == null) {
/*
return either
StringBuffer(dynamicId).append(".type1").toString();
or
new StringBuffer(dynamicId).append(".type2").toString();
*/
```

7.4.9 System configurators

The IBM Sales Center platform provides the ability to specify a system configurator for each plug-in to identify which extensions will be used over the default IBM Sales Center extensions. The purpose of this extension point is to allow plug-ins to specify the location of their system configurator file. The file name must be config.ini. This file is a text file that allows you to provide a substitute extension ID that will be used to replace a standard IBM Sales Center extension ID.

The configurator file, config.ini, must be located in a directory called config, which is located in the root directory of the plug-in that is defining the extension. Example 7-29 shows a sample system configurator extension definition.

Example 7-29 Sample system configurator extension definition

```
<extension point="com.ibm.commerce.telesales.configurator">
 <configurator path="config"/>
</extension>
```

The following sample line in the config.ini file assumes that you have defined an alternative version of the login dialog box by using the dialogs extension point. It indicates that you want to replace the default login dialog box (com.ibm.commerce.telesales.logonDialog) with your own dialog box (com.mycompany.logonDialog):

```
com.ibm.commerce.telesales.logonDialog=com.mycompany.logonDialog
```

Note: The system configurator is an IBM Sales Center extension mechanism, and not an Eclipse extension mechanism.

7.4.10 Resources

A common customization scenario involves replacing the terminology used by the application with the terminology that is more appropriate to the organization where the application is deployed. You may also want to replace application images with your own images. To ensure that these customizations are possible, the IBM Sales Center application is designed to allow you to replace the text and the images that appear on the UI.

By default, the English language IBM Sales Center resources are located in the com.ibm.commerce.telesales.resources plug-in in the resources directory in the telesalesResources_en_US.properties file. Alternative locale files are located in the com.ibm.commerce.telesales.resources.nl1 plug-in in the resources directory.

However, not all the modifiable and translatable text and images are in the telesalesResources file. The following list contains the other possible changes that you may want to perform:

- Changing client branding (banner images and so on) involves creating a plug-in that defines a new extension to the org.eclipse.core.runtime.products extension point, a plugin_customization.ini file, and an about.ini file. For more information, refer to the WebSphere Everyplace Deployment Developer Guide.
- Changing the system level text such as menu names and entries, view or application names, or preference page names (any text that appears in a plugin.xml or its companion plugin.properties) involves recreating the extension declaration in your own plug-in and then supplying the text.
- When adding a new translation, define a fragment that extends the plug-in containing the original properties file. IBM Sales Center provides fragments with names containing nl1 that illustrate this. You can follow that pattern in these fragments to add fragments of your own, extending the same plug-ins.

7.5 Developing IBM Sales Center server components

This section discusses the development of various IBM Sales Center server components.

Refer to the information center topic “WebSphere Commerce integration”, which is available on the Web at:

http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.telesales.developer.doc/concepts/ctrcommerceintegration.htm

7.5.1 Message mappers

BOD messages sent from IBM Sales Center to the WebSphere Commerce server are mapped to the WebSphere Commerce Controller command through the WebSphere Commerce message mapper. A message mapper is a mechanism that takes XML messages and converts them into a command property object. A message mapper provides a common interface for messages to be converted to CommandProperty objects, and can be used by all the WebSphere Commerce commands.

The command property object is a representative of the controller command. The object contains the controller name to be run, the command properties when running the command, and the parameters of the command. The purpose of a message mapper is to convert inbound request messages to controller

commands to be run by an adapter. Although a message mapper can be used by all the components of WebSphere Commerce to map data into an extended TypedProperty object, the main purpose of a message mapper is to convert XML objects to common Java objects that represent the controller commands.

To map an extended BOD message to a new or modified command, perform the following tasks:

1. Identify the current mapping file and make a new copy of it. The IBM Sales Center mapping files are by default located in the XML configuration directory, <WCDE_installdir>\xml\messaging.

   The file naming convention used by IBM Sales Center is <*noun*><*verb*>BODMapping.xml, for example, *CreateCustomer*BODMapping.xml. The copy of the file must be in the same directory as the original, and must be uniquely identified as a file used for extensions, for example, ExtendedCreateCustomerBODMapping.xml.

2. Add the modified attributes of the BOD information to the copied version.

3. WebSphere Commerce uses !ENTITY declarations to include different files for each of the document command mappings, and has extended the message mapper declaration to include a file for extensions called webservice_SOABOD_template.extension.xml. You can update this file to include your own extensions. This allows you to easily create your own message mapper file and include those documents that you want to leave alone and any additional or changed mappings that you want to define. The message mapper files are located in the XML configuration directory WCDE_installdir\xml\messaging.

   The new extension file must be included in webservice_SOABOD_template.extension.xml (Example 7-30). Perform this task by using the !ENTITY directive, for example, after you modify the file to add the webservice_SOABOD_template.extension.xml definition.

Example 7-30 Including a new extension file

```
<!DOCTYPE ECTemplate SYSTEM 'ec_template.dtd' [ <!-- Source comment:
this [ is required, do not remove -->
<!ENTITY ExtendedCreateCustomerMappingDefinition SYSTEM
'ExtendedCreateCustomerBODMapping.xml'>
]> <!-- Source comment: this ]> is required, do not remove -->
<ECTemplate>
&ExtendedCreateCustomerMappingDefinition;
</ECTemplate>
```

7.5.2 Response builders

To create a BOD reply message, create a class that implements the com.ibm.commerce.telesales.messaging.bodreply.ITelesalesResponseBuilder interface. After that, add an entry to the custom registry to register the new response builder created.

To add a new BOD reply message, perform the following tasks:

1. Create a new class that implements the com.ibm.commerce.telesales.messaging.bodreply.ITelesalesResponseBuilde r interface.
2. Add an entry to the custom registry to register the new response builder. To do this, modify the existing TelesalesRegistry.xml file at location (WCDE_installdir/xml/messaging). However, this is not recommended because it may change during fix pack or migration.

   Therefore, in the same directory, create a new registry file, TelesalesRegistry-ext.xml, which overrides the existing TelesalesRegistry.xml file.
3. Enter the following (Example 7-31) in the new file, TelesalesRegistry-ext.xml.

Example 7-31 Adding an entry to the custom registry

```
<WCTBodResponseBuilderRegistry>
 <Noun name="<Noun Name>">
  <Verb name="<Action name eg. Get>">
   <ClassName><!-- The reply class name--></ClassName>
  </Verb>
 </Noun>
</WCTBodResponseBuilderRegistry>
```

4. Modify the wc-server.xml file in such a way that the extended file is also loaded with the existing entries in the TelesalesRegistry.xml (Example 7-32).

Example 7-32 Modifying the wc-server.xml file

```
<property
 baseRegistryFileName="TelesalesRegistry.xml"
 baseRegistryFilePath="messaging"
 customRegistryFileName="<!--The extended registry XML file name-->"
 customRegistryFilePath="messaging"
 display="false" enableBaseRegistryOverride="true" />
```

5. Restart the WebSphere Commerce Test Server for the server to reflect the change to the wc-server.xml file.

7.5.3 WebSphere Commerce server customizations

Perform WebSphere Commerce server customization in order to support most of the customizations performed in IBM Sales Center. WebSphere Commerce server customization includes creating new commands, Enterprise JavaBeans™ (EJBs), and associated database tables. You can invoke these through data beans, with proper access control from the reply builder or the response builder.

For details about WebSphere Commerce server customization, refer to the WebSphere Commerce Information Center at:

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

Also refer to the information center topic “Tutorials”, which is available on the Web at:

http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.base.doc/concepts/ctdtutorials.htm

8

Development tools and customization deployment

This chapter provides useful customization details and tools.

8.1 Development tools

This section discusses the development tools that can be used when creating IBM Sales Center customizations.

8.1.1 Deciding on the development environment to use

IBM Sales Center customization involves creating different assets that are a part of either the IBM Sales Center client or the WebSphere Commerce server. The development environment to be used to develop these assets depend on the type of asset being created.

All IBM Sales Center client assets are developed using the IBM Sales Center development environment using the plug-in development perspective. These assets include the plug-in projects that contain the customization extensions. The plug-ins contain the following:

- New and extended user interface (UI) definitions
- New and extended model objects
- New and extended request handlers
- New and extended resource bundles

All the WebSphere Commerce server code assets for the IBM Sales Center are developed using the WebSphere Commerce development environment. These assets include the following:

- New and extended response builders
- New and extended WebSphere Commerce commands, Enterprise JavaBeans (EJBs), and access beans

Update the following assets in the file system on the WebSphere Commerce development machine:

- Response builder registry
- Message mappers
- Commerce instance configuration file (WC_instancename.xml)

8.1.2 Widget hover logging

An important first step in most customization tasks is to determine the identifier of the object you want to customize. To facilitate the task of locating the identifiers of an UI element on an editor page or dialog box in the IBM Sales Center client, a debugging option called widget hover logging has been added to the com.ibm.commerce.telesales.widgets plug-in. This debugging option prints out the ID of the Sales Center UI widgets to the console when you hover over them, in the following format:

<namespace>.<element ID>

When you hover over an object in the IBM Sales Center client, for example, the output is written to the console prompt in Rational Application Developer as follows:

- Mouse hover control ID is:

  `com.ibm.commerce.telesales.ui.impl.`*`orderPaymentPageComposite`*

- The namespace for all default IBM Sales Center UI widgets is:

  `com.ibm.commerce.telesales.ui.impl`

  Note: The namespace printed out in the console is not necessarily the plug-in fragment that the element ID is defined in.

To enable widget hover logging, perform the following tasks:

1. In the IBM Sales Center development environment, select **Run ...** from the Run menu.
2. In the left-hand pane, expand **Run-time Workbench** and select **Sales Center**.
3. Select the **Tracing** tab.
4. Select the **Enable tracing for the selected plug-ins** check box.

   Note: Only when you select a plug-in in the left-hand pane will the tracing options be displayed in the right-hand pane, where it can be edited.

5. Select the **com.ibm.commerce.telesales.widgets** plug-in.

6. Select the **debug/widgetHoverLogging** check box and the **Debug** check box as shown in Figure 8-1.

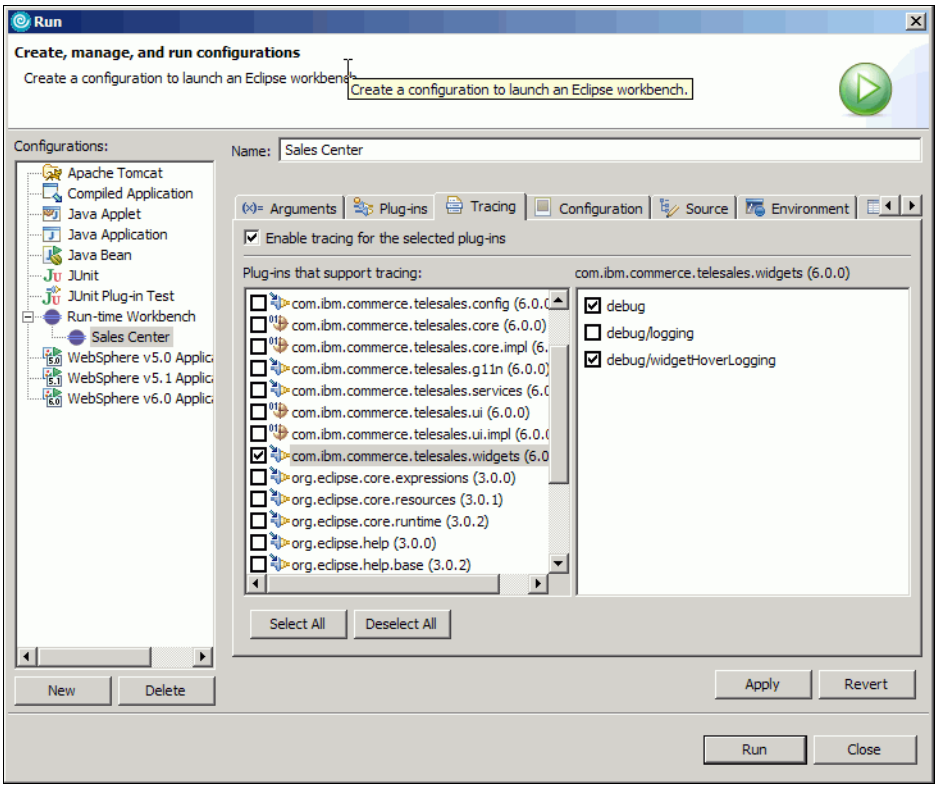


Figure 8-1 Enabling widget hover logging

7. Click **Apply**.
8. Click **Run**.

After you have found the element ID, search on the ID in the IBM Sales Center UI plug-ins.

Making plug-ins searchable

The IBM Sales Center UI plug-ins and plug-in fragments can be made searchable in the IBM Sales Center development environment by importing them as binary projects. This allows you to search on the ID of a UI element and find its fragment.xml file. To make the IBM Sales Center UI plug-ins searchable, perform the following tasks:

1. In the IBM Sales Center development environment, select the **Plugins view** tab.
2. Select all the plug-ins and fragments with names that begin with com.ibm.commerce.telesales.ui.impl.
3. After selecting the plug-ins, right-click and select **Import** → **As binary project**.

   These plug-ins will now be visible under the Package Explorer View tab.

Searching for the user element ID

To search for the definition of a UI element, perform the following tasks:

1. Using widget hover logging to determine the ID of the UI you want to search on. You can also search in the contents of the comments or an extension point type, for example, widget or managedComposite.
2. In the IBM Sales Center development environment, select **Search** → **File ...**
3. Enter your search term in the Containing text field.

4. Enter `*.xml` in the File name patterns field. This searches through all the plugin.xml and fragment.xml files in the workspace. Click **Search**. Figure 8-2 shows a file search dialog box.

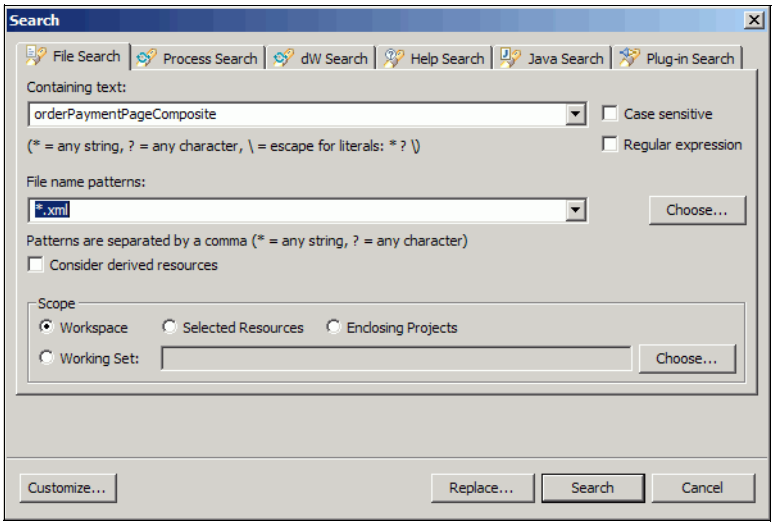


Figure 8-2 Searching for the ID using file search

5. In the page that appears on clicking the Search tab, expand the plug-ins where matches have been found and double-click a file to examine it.

Note: In most cases, when you search on a UI element ID such as a widget ID, you will be opening a file called fragment.xml. Do not modify the default IBM Sales Center fragment.xml files, because they will be overwritten when you migrate the IBM Sales Center development environment or when you install fix pack updates. Examine the file for customizing and extending the UI and then create your own new plug-in for packaging and deploying the customizations.

8.1.3 Enabling the task of showing the contents

When passing custom data between the WebSphere Commerce server and the IBM Sales Center client, sometimes, the data does not display as expected. Ensure that the Business Object Document (BOD) files really contain the custom data by viewing the contents of the BODs.

To view the contents of a BOD, perform the following tasks:

1. Within the IBM Sales Center development environment, select **Run** → **Run**.
2. Under the Arguments tab, make a note of the Workspace Data Location (by default, the location is *<WCDE_installdir>*/mscworkspace/runtime-workspace). This is the location in which the BOD files will be stored.
3. Under the Tracing tab, perform the following tasks:
   - Select **Enable tracing for the selected plug-ins**.
   - Select the **com.ibm.commerce.telesales.core.impl** plug-in.
   - Select all the debug options. The Tracing tab must look as shown in Figure 8-3.

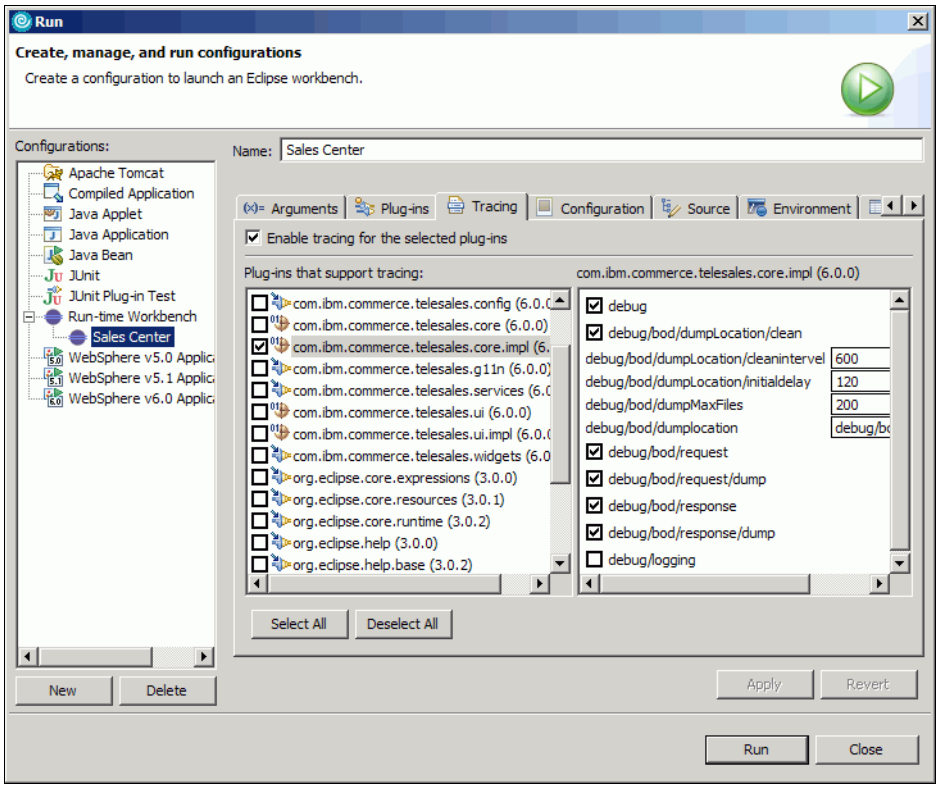


Figure 8-3 Enabling the options to show the contents of a BOD

   c. Click **Apply**.
   d. Click **Run**.

4. When the IBM Sales Center client opens, perform the action that generates the BOD, for example, open the IBM Sales Center - Order Management application and select **File** → **Logon** from the menu to log in to the WebSphere Commerce server.
5. In the file system, navigate to the Workspace Data Location you noted in step 2. Within this directory, navigate to the Workspace Data Location debug/bodmessages subdirectory (by default, the location is *<WCDE_installdir>*/mscworkspace/runtime-workspace). All the BODs exchanged between the client and the server are stored in this directory.
6. Locate the appropriate BOD file. Each file name contains the name of the command that generated the BOD and a time stamp. The ShowElectronicCatalog command's BOD, for example, is called wc.ShowElectronicCatalog_*timestamp*.xml, where timestamp is the time when the BOD was created.
7. In the file, find the <wc:UserData> tag. The manufacturer part number is, for example, passed from the server to the client (Example 8-1).

Example 8-1 Finding the <wc:UserData> tag

```
<wc:UserData>
<wc:UserDataField name="manufacturerPartNumber">
myMFPartNumber
</wc:UserDataField>
</wc:UserData>
```

8. Disable tracing under the Tracing tab when you no longer have to view the BODs.

8.1.4 Debugging in the IBM Sales Center development environment

To debug IBM Sales Center, a preconfigured Runtime Workbench configuration is available in the debug dialog box. To debug the IBM Sales Center, perform the following tasks:

1. Select **Debug...** from the Run menu.
2. In the left-hand pane, expand **Runtime Workbench** and select **Sales Center**.
3. Click **Debug** to debug the application.
4. Set the breakpoints as appropriate.

8.1.5 Tracing in the IBM Sales Center development environment

To enable tracing in IBM Sales Center, perform the following tasks:

1. In the IBM Sales Center development environment, select **Run ...** from the Run menu.
2. In the left-hand pane, expand **Runtime Workbench** and select **Sales Center**.
3. Click the **Tracing** tab.
4. Select the **Enable tracing for the selected plug-ins** check box.

   As a result of this change, when you select a plug-in in the left-hand pane, its tracing options will be displayed in the right-hand pane and can be edited there.
5. Select the plug-ins you want to trace.

8.1.6 Enabling tracing and debugging in the IBM Sales Center client

When working within the IBM Sales Center development environment, you can enable tracing and debugging. However, there may be situations in which you want to enable tracing and debugging in the IBM Sales Center client that is not running in the development environment.

To enable tracing and debugging in the IBM Sales Center client that is running outside the development environment, perform the following tasks:

1. Create an .options file. The .options file contains information about which plug-ins you want to debug and trace. It is possible to create an .options file manually, but it is difficult to know all the different plug-ins and what debug options are available for each one. This step uses the IBM Sales Center development environment to help you generate this file. This step is only required when you first enable tracing and debugging or when you want to add new plug-ins and have to regenerate the .options file.

   To create an options file, perform the following tasks:

   a. Enable tracing in the IBM Sales Center development environment using the instructions provided in 8.1.5, “Tracing in the IBM Sales Center development environment” on page 161.

   b. Select **Run** → **Sales Center** to launch the IBM Sales Center client and automatically generate the .options file with the tracing options as selected.

c. Locate the .options file in the *<WCDE_installdir>*\mscworkspace\plugins\.metadata\.plugins\org.eclipse.pde.core\Sales Center directory. Example 8-2 shows an .options file.

Example 8-2 An .options file

```
#Master Tracing Options
com.ibm.commerce.telesales.core.impl/debug/bod/dumpLocation/initialdelay=120
com.ibm.commerce.telesales.core.impl/debug/bod/request/dump=true
com.ibm.commerce.telesales.widgets/debug/logging=false
com.ibm.commerce.telesales.core.impl/debug=true
com.ibm.commerce.telesales.core.impl/debug/bod/response/dump=true
com.ibm.commerce.telesales.core.impl/debug/logging=true
com.ibm.commerce.telesales.core.impl/debug/bod/request=true
com.ibm.commerce.telesales.widgets/debug=true
com.ibm.commerce.telesales.widgets/debug/widgetHoverLogging=true
com.ibm.commerce.telesales.core.impl/debug/bod/dumpMaxFiles=200
com.ibm.commerce.telesales.core.impl/debug/bod/dumpLocation/cleanintervel=600
com.ibm.commerce.telesales.core.impl/debug/bod/dumpLocation/clean=true
com.ibm.commerce.telesales.core.impl/debug/bod/dumplocation=debug/bodmessages/
com.ibm.commerce.telesales.core.impl/debug/bod/response=true
```

d. Select the IBM Sales Center client and the development environment.

2. Place a copy of the .options file in *<SC_installdir>*. Open the *<SC_installdir>*/startup.bat file and add `-debug` at the end of the line. The resulting command line must be similar to the following:

   ```
   "%~dp0rcp\rcplauncher.exe" -product
   com.ibm.commerce.telesales.TelesalesWorkbenchProduct -debug
   ```

3. Save your changes.
4. Run the *<SC_installdir>*/startup.bat file to start the IBM Sales Center client.
5. The tracing and debugging output is created in the C:\Documents and Settings*<your_windows_logon_id>*\IBM\RCP*<random_numbers>**<your_windows_logon_id>*\.metadata\.log file. If you have enabled BOD requests and response dumping, then the BOD files are created in the directory you specified within the com.ibm.commerce.telesales.core.impl/debug/bod/dumplocation option. By default, this directory is C:\Documents and Settings*<your_windows_logon_id>*\IBM\RCP*<random_numbers>**<your_windows_logon_id>*\debug\bodmessages.

8.2 Deploying the customizations

This section describes the process involved in deploying the customizations from the IBM Sales Center development environment to the IBM Sales Center runtime client.

8.2.1 Exporting the client code from the development environment

Exporting the client code from the IBM Sales Center development environment includes the following steps:

1. Preparing the plug-in for packaging
2. Creating the feature project
3. Creating an update site project

Preparing the plug-in for packaging

To prepare the plug-in for packaging, perform the following tasks:

1. Open the IBM Sales Center development environment if it is not already open.
2. In the Package Explorer view, navigate to your plug-in project.
3. In your project, locate the plugin.xml file. Double-click the file to open it for editing.
4. Click the **Build** tab.

5. In the Binary Build section of the Build Configuration editor, keep all the default selections and select the config directory containing the configuration file to include it in your exported plug-in. The other assets are automatically included from the default build selections. The binary build selections must look as shown in Figure 8-4.

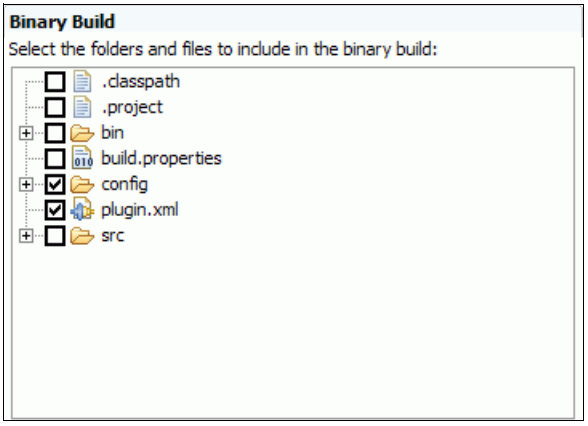


Figure 8-4 Binary build settings

6. Save your changes.

Creating the feature project

Before you create an Update Site project in Rational Application Developer, create one or more feature projects that will refer to all the plug-ins you want to deploy.

To create a feature project for your plug-ins in Rational Application Developer V6.0, perform the following tasks:

1. In the Package Explorer view, select **New** → **Feature Project**.
2. Enter a project name and click **Next**. (Optionally, you can also enter the Feature Provider's name after entering the Project Name, and then click **Next**.)
3. Select the desired plug-ins and fragments and click **Finish**.
4. The feature.xml editor opens. Right-click the **Update URLs** entry in the Feature URLs and select **New** → **Update URL**.

5. In the Properties view, change the Uniform Resource Locator (URL) to the Web server URL where you have an update site. If you do not know where the update site is, or if there are multiple update sites when the feature is deployed, enter an arbitrary, unique URL such as `http://<featureName>.URL`. As long as a nonempty update URL is supplied, it can be mapped to the correct URL later.
6. Optionally, provide other feature information and press Ctrl+S to save.

You must decide whether you want to create one feature or several features. The feature defines the granularity for installation and update. Populating an update site with several small features instead of one large feature allows for selective installation, for example, one base feature that is always installed can provide the functional plug-ins, and several optional features can each provide fragments containing message text translated into a different language. Different clients can be installed with lesser or more optional features, and new language features can be developed and deployed as necessary without changing the base feature.

Creating an update site project

After you create a feature project in Rational Application Developer, create an update site by performing the following tasks:

1. Select **New** → **Project** → **Update site** project from the list of plug-in development project types.
2. Enter a name for the site project and click **Finish**.
3. The site.xml editor opens. Click **Add** and select a desired feature.
4. Right-click the feature and select **Publish**.
5. Click the **Build All** button in the editor.

> **Note:** Unlike most projects in Rational Application Developer, update site projects are neither built automatically nor can they be built from build actions in the project menu. Only the corresponding button in the site.xml editor will build the site.

6. In the Package Explorer, right-click the newly created update site project and select **Export**.
7. Select **File system** as an export destination.
8. Click **Next**.
9. Enter a path to the directory.
10. Click **Finish**.

8.2.2 Exporting the server code from the development environment

After you create the customized code in WebSphere Commerce Developer and test it with the IBM Sales Center client and the WebSphere Commerce Test Server, deploy the code to a target WebSphere Commerce server running outside the WebSphere Commerce Developer. This target WebSphere Commerce server can run locally on your development machine, or it can be on another machine (using the same operating system or a different operating system).

Following are the types of WebSphere Commerce customized code you can create:

- Enterprise beans

  Enterprise beans are compressed and exported from Rational Application Developer as Enterprise JavaBeans (EJB™) JAR files, and then deployed to the IBM WebSphere Application Server.

- Commands and data beans

  Commands and data beans are compressed and exported from Rational Application Developer as a JAR file, and then deployed to the WebSphere Application Server.

- Store assets

  Store assets such as JavaServer Page (JSP) files and properties files are compressed and exported from Rational Application Developer as a JAR file, and then deployed to the WebSphere Application Server.

- Schema changes

  Updates to the database, such as new tables and inserted rows, must be made on the WebSphere Commerce server database.

- All the other changes

  All the other customized code changes are updates or new files that are stored outside the enterprise archive (EAR), such as XML input files for access control policies. These types of code changes must be transferred to the WebSphere Commerce server environment using a file transfer utility.

To export the customized WebSphere Commerce server assets from the Rational Application Developer, follow the high-level process:

- If the customized code is a command or data bean, export the code from Rational Application Developer as a JAR file.
- If the customized code is an enterprise bean, export the bean from Rational Application Developer as an EJB JAR file.

- If the customized code is a store asset, export the code as a single file for one file, and a compressed file or a JAR file for multiple files.

Ensure that you export all the assets relating to the IBM Sales Center customizations:

- Commands
  - WebSphere Commerce commands
  - Response builders
- Database objects
  - WebSphere Commerce EJBs and AccessBeans
- XML files
  - Instance configuration file
  - Response builder registry
  - Message mappers

8.2.3 Deploying the customizations

After the code is exported from the development environment, it is ready to be installed on the IBM Sales Center client and WebSphere Commerce server machines.

Refer to Chapter 4, “IBM Sales Center production environment installation” on page 45 for information about the manual and automatic installation of IBM Sales Center customizations.

To deploy WebSphere Commerce server customizations relating to the IBM Sales Center client customizations, follow the instructions provided in the topic “Customized code deployment” in the WebSphere Commerce V6 Information Center, which is available on the Web at:

http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.developer.doc/concepts/cdedeploycustomcode.htm

Part 4

Customization scenario examples

This part provides examples of customization scenarios.

9

User interface customization

This chapter demonstrates a complex user interface (UI) customization of IBM Sales Center, which involves customizing each of the Sales Center's customizable components. The example referred to in this chapter requires a modification of the Sales Center UI and the WebSphere Commerce Server.

9.1 Introduction

During the requirements phase, one of the business requirements requested by our clients was a new functionality that supports additional information about the customer's pets, including the following:

- The ability to display information pertaining to customers' pet, that is, name, type, and so on
- The ability to add, remove, and update information pertaining to customers
- The ability to find customers by the type of pets they own

> **Note:** Refer to Chapter 5, "Requirements and design" on page 95 for more information about gathering customization requirements.

To implement this customization, several UI modifications are required:

- Adding a new customer editor page to facilitate the inclusion of details pertaining to a customer's pets, as shown in Figure 9-1

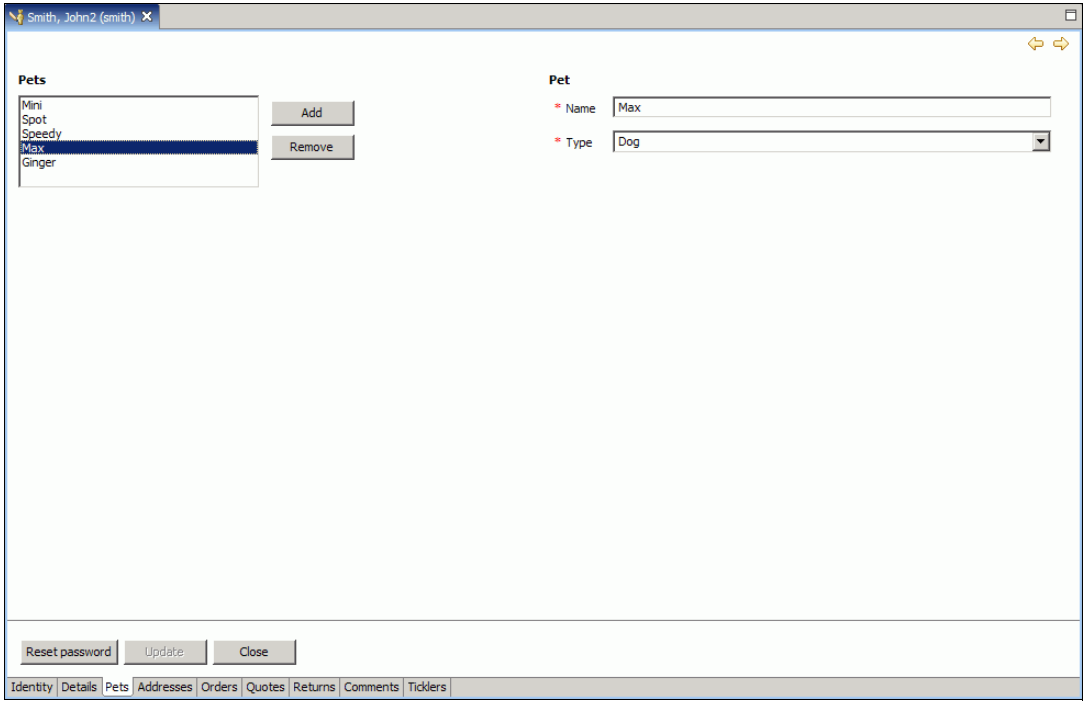


Figure 9-1 The customer pet editor page

- A dialog box to add a new pet (Figure 9-2) facilitating the addition of new pets to the customer editor pet page

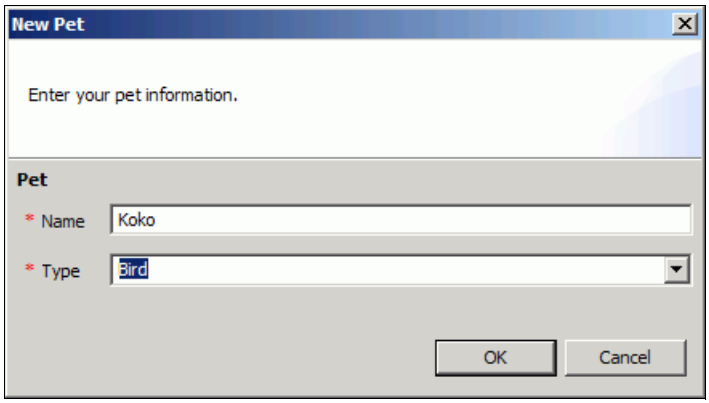


Figure 9-2 Adding a pet

- Modification of the Find Customer page (Figure 9-3) to include a new search field, Pet Type

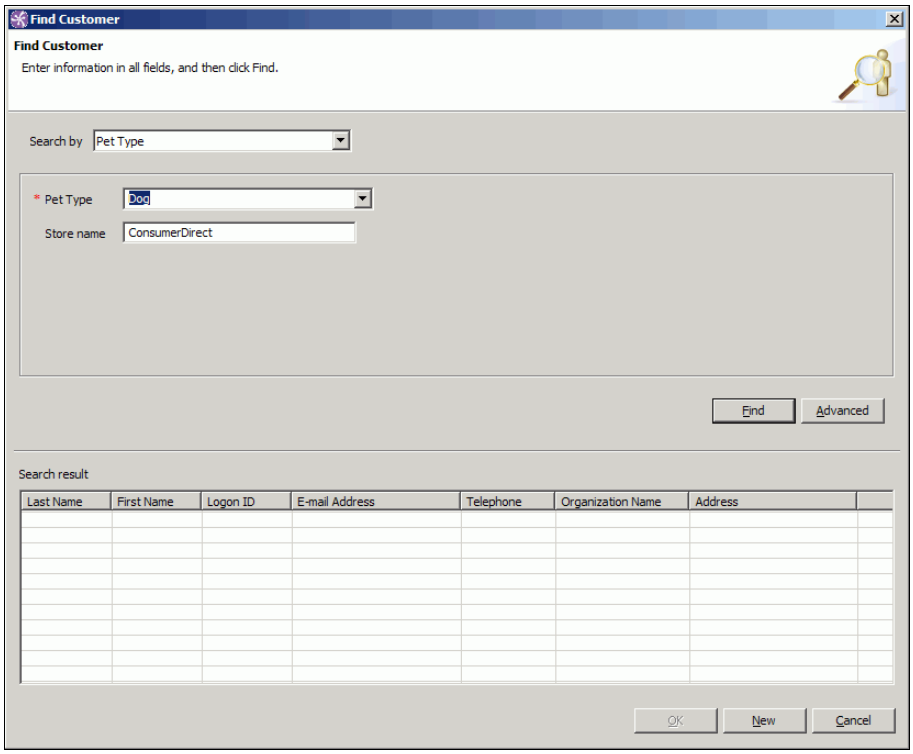


Figure 9-3 Finding customers by pet type

Because WebSphere Commerce does not support the base software install handling of information pertaining to customers' pets, several customizations must be performed on the server side:

- Creating a new table to store information pertaining to a customer's pets
- Writing new Enterprise JavaBeans (EJB) and an access bean to access information pertaining to a customer's pets from the new table
- Writing a new command to add and update information pertaining to a customer's pets
- Writing a new command to delete information pertaining to a customer's pets

To enable the passing of data between the IBM Sales Center client and the WebSphere Commerce server, implement the following components:

1. Create a new model to represent a pet.
2. Extend the customer model to store the list of a customer's pets.
3. Extend the GetCustomer request handler and its ShowCustomerRequest response builder to handle the finding and receiving of customer information by pet type.
4. Extend the SyncCustomerUpdate request handler and its ConfirmCustomer response builder to handle the process of sending information pertaining to pets, to the server.

9.2 Implementing the customization

The following sections explain how each of the customizations were developed. They contain snippets of the code along with explanations about the main parts of the customization. The customization is broken down as follows:

1. Developing the WebSphere Commerce server backend
2. Developing the Sales Center client customization base
3. Developing the new customer pet editor page
4. Developing the new add pet dialog box
5. Developing the find customer by pet dialog box

Note: The implementation of the code is provided as a sample that is not fully functional and is provided "as is" for demonstration purposes. Refer to 9.8, "Loading the customizations into WebSphere Commerce Developer" on page 244 at the end of this chapter for instructions about how to load the code into your development for analysis, and 9.9, "Testing the customized code" on page 251 for use case scenarios relating to the demonstration of the implemented functionalities.

9.3 Developing the WebSphere Commerce server backend

This section describes at a high level, the assets created for the WebSphere Commerce server backend to support the additional functionality pertaining to customers' pets.

9.3.1 Defining the new table

In our case, we defined a new table, XPET, in order to store information pertaining to the pets, with column definitions, as shown in Table 9-1.

Table 9-1 The XPET table

Column	Type	Restrictions
PET_ID	BIGINT	PRIMARY KEY
USERS_ID	BIGINT	NOT NULL FOREIGN KEY TO USERS_ID in USERS table
NAME	VARCHAR(32)	NOT NULL
TYPE	VARCHAR(32)	NOT NULL
OPTCOUNTER	SMALLINT	NOT NULL

The Structured Query Language (SQL) shown in Example 9-1 creates Table 9-1.

Example 9-1 SQL to create table

```
CREATE TABLE XPET
  (PET_ID BIGINT NOT NULL,
   USERS_ID BIGINT NOT NULL,
   NAME VARCHAR(32) NOT NULL,
   TYPE VARCHAR(32) NOT NULL,
   OPTCOUNTER SMALLINT NOT NULL);

ALTER TABLE XPET
  ADD CONSTRAINT xpet_p1 PRIMARY KEY (PET_ID);

ALTER TABLE XPET
  ADD CONSTRAINT xpet_f1 FOREIGN KEY (users_id) REFERENCES users
(users_id) ON DELETE CASCADE;
```

9.3.2 Implementing the new ExtPet EJB and ExtPetAccessBean

In our case, we defined a new EJB, ExtPet, and an AccessBean, ExtPetAccessBean, to provide access to the XPET table. Table 9-2 provides the ExtPet EJB definitions.

Table 9-2 ExtPet EJB definitions

Attribute	Type	
petId	java.lang.Long	key field
usersId	java.lang.Long	
name	java.lang.String	
type	java.lang.String	
optCounter	short	

Example 9-2 shows the ExtPetAccessBean methods.

Example 9-2 ExtPetAccessBean methods

```
ExtPetAccessBean()
   void ExtPetAccessBean(Long usersId, String name, String type)

   void setInitKey_petId(Long petId)
   Long getPetId()

   String getType()
   void setType(String type)

   Long getUserId()
   void setUserId(Long usersId)

   String getName()
   void setName(String name)
```

The ExtPet EJB was written to automatically generate a key using the ECKeyManager. To enable this feature, an entry is added to the KEYS table using an SQL, as follows:

```
insert into KEYS (TABLENAME, COLUMNNAME, COUNTER , KEYS_ID) values
('xpet', 'pet_id', 0, 1);
```

9.3.3 Implementing the new commands

In our case, we created two new commands to support the customers' pets functionality:

- The command ExtPetUpdateCmd to add pets and update information pertaining to pets, with the input parameters and behavior specified in Table 9-3.

Table 9-3 ExtPetUpdateCmd input parameters

Input parameter	Type	Description
petId	Long	The primary key of the pet. If it is provided, it is an update. If it is not provided, a new pet is created. The added or updated petId is returned in the response properties.
userId	Long	The pet owner ID. Mandatory for pet additions.
name	String	The name of the pet. Mandatory for pet additions.
type	String	The type of pet. Mandatory for pet additions.

- The command ExtPetDeleteCmd to delete pets, with the input parameters and behavior specified in Table 9-4.

Table 9-4 ExtPetDeleteCmd input parameters

Input parameter	Type	Description
petId	Long	The primary key of the pet to delete. Mandatory parameter. It is also returned in the response properties.

When writing WebSphere Commerce commands, it is important to think about their usage in the commerce solution. If it is to be used only by IBM Sales Center, it is not necessary to specify the Uniform Resource Locator (URL) to redirect to its response properties. However, if these commands are also going to be used by Web clients, for example, the storefront or IBM Gift Center, the URL must be specified. In our implementation, the URL is not specified because we only used it in IBM Sales Center.

9.4 Developing the Sales Center client customization base

Before starting to develop the specific UI parts, create the commonly used customization assets.

9.4.1 Defining the configurator and the properties

This section shows you how to define the system configurator and the property bundle.

The system configurator

To override any UI components, the system configurator file is used. To define the system configurator for this customization, perform the following tasks:

1. Create the configurator file config.ini in the config folder under the plug-in project.
2. Define the configurator location by specifying the extension shown in Example 9-3 in the plugin.xml file. The *path* parameter specifies the folder name of the configuration file.

Example 9-3 Defining the configurator location by specifying the extension

```
<extension point="com.ibm.commerce.telesales.configurator">
   <configurator path="config" />
</extension>
```

Property bundle

This customization requires the addition of new resource properties to define the labels, dialog names, titles, and so on. In order to enable the addition of new properties, define a new resource bundle as follows:

1. In the com.ext.commerce.telesales.resources package, create a new file, `extTelesales.properties`.
2. Register the new property bundle by specifying the extension shown in Example 9-5, in the plugin.xml file.

Example 9-4 Registering the new property bundle by specifying the extension

```
<extension point="com.ibm.commerce.telesales.resources.resources">
   <resourceBundle
    baseName="com.ext.commerce.telesales.resources.extTelesales" />
</extension>
```

9.4.2 Defining the new model objects

This section describes the process involved in defining a new model object and a new model object list to encapsulate the pet information.

ExtPet model object

To store the information pertaining to pets, define a new model object.

Define a new model object class, `ExtPet`, representing a pet. This object provides convenience methods to set and get the model object properties relating to the pet object. This class also provides an additional *marking* property that allows marking the object with the action taken on an object (existing, new, edited, and deleted) in such a way that the entire list of pets can be processed at one time and the appropriate action (add, update, or delete) be taken on the server side, depending on the object's marking. Example 9-5 shows the definition of the ExtPet model object.

Example 9-5 New model object: ExtPet

```
package com.ext.commerce.telesales.model;

/**
 * ExtPet is a ModelObject that represents a pet. The default
 * properties of the Pet object are described in this class.
 */
public class ExtPet extends ModelObject {

    /**
     * Constant for the pet name property. The data stored under the "name" property
      * is a String that contains the pet's name.
     */
    public static final String PROP_NAME = "name";

    /**
     * Constant for the pet id property. The data stored under the "petId" property
      * is a String that contains the pet's primary id.
     */
    public static final String PROP_PET_ID = "petId";

    /**
     * Constant for the pet type property. The data stored under the "type" property
      * is a String that contains the pet's type.
     */
    public static final String PROP_TYPE = "type";
```

```
/**
 * Constant for the marking property name.
 * The data stored under the "marking" property is a String that
 * is set to one of the following values:
 * <UL>
 * <LI>"MARKED_EXISTING"</LI>
 * <LI>"MARKED_NEW"</LI>
 * <LI>"MARKED_EDITED"</LI>
 * <LI>"MARKED_DELETED"</LI>
 * </UL>
 */
public static final String PROP_MARKING = "marking";

/**
 * Constant for the MARKED_DELETED value.This is one of the possible values
  * of the "marking" property.
 */
public static final String MARKED_DELETED = "MARKED_DELETED";

/**
 * Constant for the MARKED_EDITED value.
 * This is one of the possible values of the "marking" property.
 */
public static final String MARKED_EDITED = "MARKED_EDITED";

/**
 * Constant for the MARKED_EXISTING value.
 * This is one of the possible values of the "marking" property.
 */
public static final String MARKED_EXISTNG = "MARKED_EXISTNG";

/**
 * Constant for the MARKED_NEW value.
 * This is one of the possible values of the "marking" property.
 */
public static final String MARKED_NEW = "MARKED_NEW";

/**
 * Constructor for ExtPet New model objects are initialized with a marking of
 * MARKED_NEW value.
 */
public ExtPet() {
    super();
    setData(PROP_MARKING, MARKED_NEW);
}
```

```
/**
 * This method is a convenience method for retrieving the "petId" property
 * @return the primary ID of the pet
 */
public String getId() {
    return (String) getData(PROP_PET_ID, "");
}

/**
 * This method is a convenience method for retrieving the current setting of the
 * "marking" property. If the value has not been set, then this method will
 * return an empty string.
 * @return The marking of the pet.
 */
public String getMarking() {
    return (String) getData(PROP_MARKING, "");
}

/**
 * This method is a convenience method for retrieving the "name" property
 * @return
 */
public String getName() {
    return (String) getData(PROP_NAME, "");
}

/**
 * This method is a convenience method for retrieving the "type" property
 * @return
 */
public String getType() {
    return (String) getData(PROP_TYPE, "");
}

/**
 * This method is a convenience method for setting the "petId" property
 * @param petId
 */
public void setId(String petId) {
    if (petId != null) {
        setData(PROP_PET_ID, petId);
    }
}
```

```
    /**
     * This method is a convenience method for setting the "marking" property.
     * @param marking The new marking for the pet.
     */
    public void setMarking(String marking) {
        setData(PROP_MARKING, marking);
    }

    /**
     * This method is a convenience method for setting the "name" property
     * @param name The name of the pet
     */
    public void setName(String name) {
        if (name != null) {
            setData(PROP_NAME, name);
        }
    }

    /**
     * This method is a convenience method for setting the "type" property
     * @param type the type of the pet
     */
    public void setType(String type) {
        if (type != null) {
            setData(PROP_TYPE, type);
        }
    }
}
```

To use the new model object, declare it in the plugin.xml by inserting the extension shown in Example 9-6.

Example 9-6 Declaring the new model object in the plugin.xml

```
<extension point="com.ibm.commerce.telesales.core.modelObjects">
   <modelObject class="com.ext.commerce.telesales.model.ExtPet"
      id="com.ext.commerce.telesales.customer.extPetModel" />
</extension>
```

ExtPetList model object list

A customer might own several pets. Therefore, a list of pets has to be maintained for each customer. A convenient ExtPetList model object list is defined in order to work with a list of ExtPet objects. The model object list definition is specified in Example 9-7.

Example 9-7 New model object list: ExtPetList

```
package com.ext.commerce.telesales.model;

/**
 * ExtPetList is a ModelObjectList that represents a list of pets.
 */
public class ExtPetList extends ModelObjectList {
    /**
     * Constructor for ExtPetList
     */
    public ExtPetList() {
        super();
    }

    /**
     * This method is a convenience method for adding an ExtPet to the list. The
     * specified ExtPet object is added to the ModelObjectList.
     * @param pet The pet.
     */
    public void addPet(ExtPet pet) {
        addData(pet);
    }

    /**
     * This method retrieves the pet at specified index
     * @param index the index of the pet to be fetched
     * @return the pet at specified index
     */
    public ExtPet getPet(int index) {
        return (ExtPet) getData(index);
    }

    /**
     * This method is a convenience method for retrieving the pet with the specified
      * primary ID.
     * @return a pet with the specified primary ID
     * @param petId The pet id to get the pet for
     */
```

```
public ExtPet getPetForPetId(String petId) {

    for (int i = 0; i < size(); i++) {
        if (((ExtPet) getData(i)).getId().equals(petId)) {
            return (ExtPet) getData(i);
        }
    }

    return null;
}

/**
 * This method returns a vector of pets in the list.
 * @return A vector of all the customer's pets
 */
public Vector getPets() {
    Vector petsVector = new Vector();

    for (int i = 0; i < size(); i++) {
        petsVector.add(getData(i));
    }
    return petsVector;
}

/**
 * This method removes the ExtPet object with the specified primary ID from the
 * pet list.
 * @param petId The pet's primary ID
 */
public void removePetForPetId(String petId) {

    for (int i = 0; i < size(); i++) {
        ExtPet pet = (ExtPet) getData(i);
        if (pet.getId().equals(petId)) {
            removeData(pet);
        }
    }
}

/**
 * This method updates the list of pets.
 * @param pets an array of pet
 */
```

```
    public void setPets(ExtPet[] pets) {
        clear();
        for (int i = 0; i < pets.length; i++) {
            addData(pets[i]);
        }
    }
}
```

9.5 Developing the new customer pet editor page

This section demonstrates the steps involved in implementing the new customer pet editor page shown in Figure 9-4.

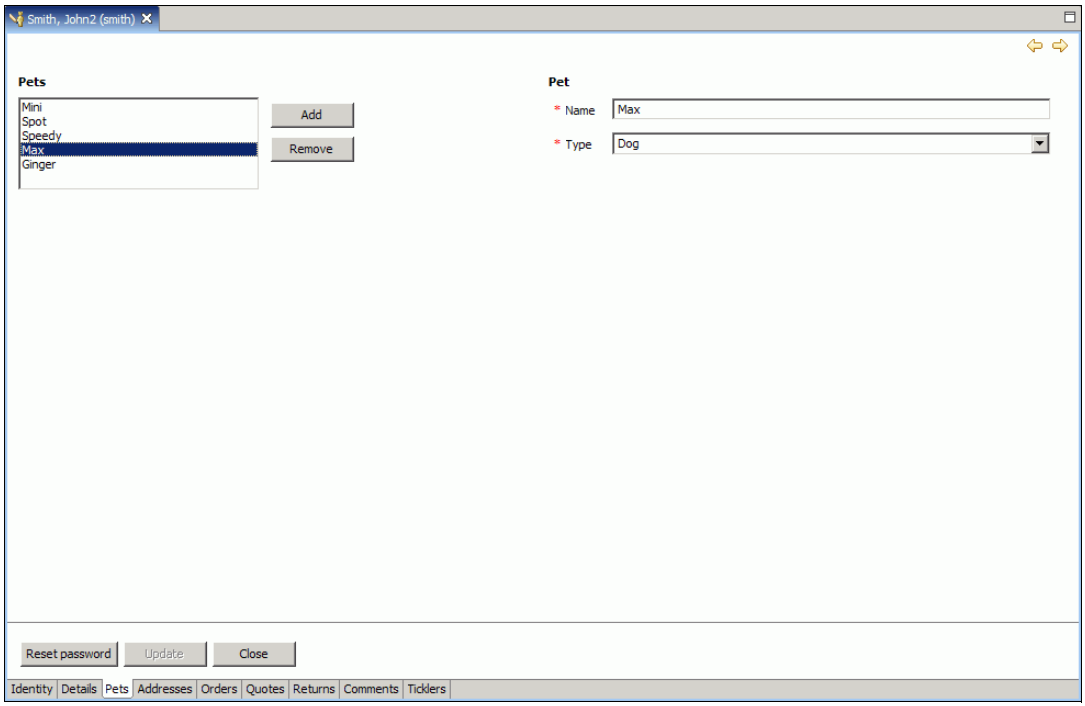


Figure 9-4 The customer pet editor page

9.5.1 Implementing the user interface components

This section shows you how to define the UI and the basic UI behavior for the customer pet editor page.

User interface definition

Create the UI elements by performing the following tasks:

1. Define the new customer pet editor page by creating a new editor page Java class, `ExtCustomerPetsPage.java`, as shown in Example 9-8.

   To inherit the existing customer editor page behavior such as the button bar, we extended from the TelesalesCustomerPage, which in turn extends from the base Sales Center UI framework class, TelesalesConfigurableEditorPage.

   Extending from the Sales Center UI framework class allows the definition of controls on the page using Extensible Markup Language (XML) by providing methods that specify which managed composite content area and which managed composite button bar that are defined in plugin.xml will be used to render the editor page. In our case, because we extended from the TelesalesCustomerPage, the button bar is already set. We only have to specify the content area by implementing the protected String getPageContentManagedCompositeId() method to the ID of our content area, which is defined in the plugin.xml. (The plugin.xml is defined later in this section.)

   The getHelpContextId() method is an abstract method that must be defined to return a non-null value. It returns the (F1) help context ID so that the EditorPart Page can have help information pop-up windows. In our case, for brevity, we set the value to a dummy value.

   The getTitle() method is an optional method to personalize the title of the editor page.

Example 9-8 ExtCustomerPetsPage.java implementation

```
package com.ext.commerce.telesales.ui.impl.editors.customer;

public class ExtCustomerPetsPage extends TelesalesCustomerPage {

protected String getPageContentManagedCompositeId() {
        return "com.ext.commerce.telesales.customer.customerPetsManagedComposite";
    }

public String getHelpContextId() {
        return "__DUMMY__";
    }
```

```
public String getTitle() {
        return Resources.getString("CustomerPetPage.title");
    }
}
```

Figure 9-5 shows the result of the new customer pet editor page, provided the specified composites are defined in the plugin.xml.

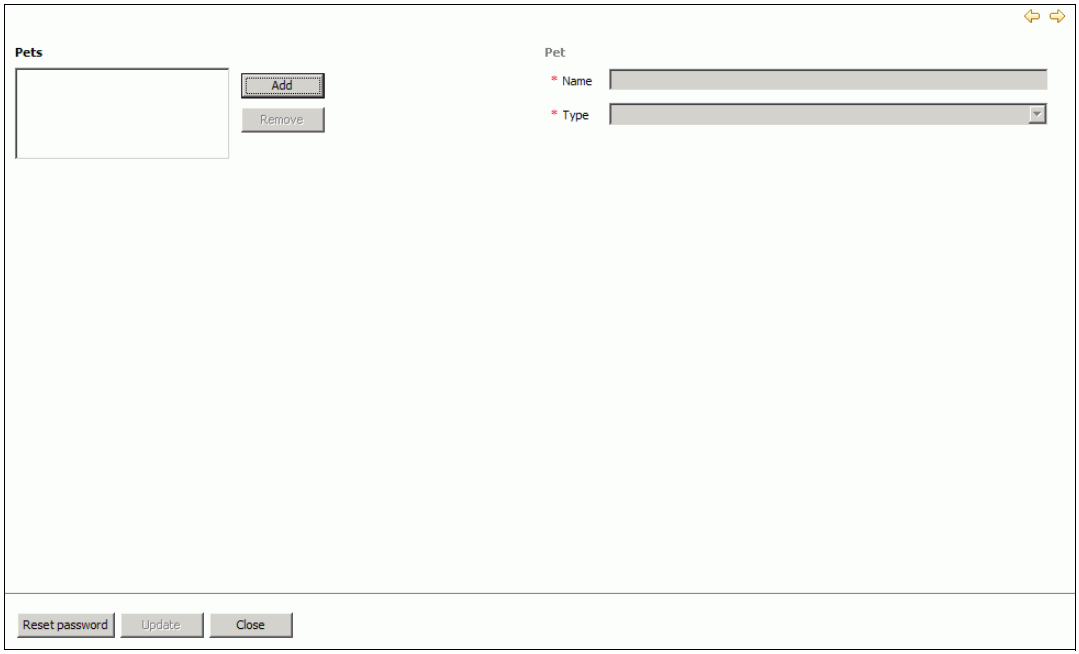


Figure 9-5 Customer pet editor page

2. Redefine the new customer pet editor page to include the new pet dialog box as part of the editor by adding the extension shown in Example 9-9 to the plugin.xml.

   This definition is an exact replica of the existing definition, with a new ID and with the addition of the “Pets” section (highlighted in bold in Example 9-9) specifying the new ExtCustomerPetsPage created earlier to be added after the Details page.

Example 9-9 Redefining the new customer pet editor page to include the new pet dialog box

```
<extension point="com.ibm.commerce.telesales.ui.editorPages">
   <editor editorId="com.ext.commerce.telesales.customer.customerEditor"
      id="com.ext.commerce.telesales.customer.customerEditorPages">
      <page name="%customerIdentityPageName"
```

```
   class="com.ibm.commerce.telesales.ui.impl.editors.customer.CustomerIdentityPage"
   id="com.ibm.commerce.telesales.customerIdentityPage" />

      <page name="%customerDetailsPageName"
      class="com.ibm.commerce.telesales.ui.impl.editors.customer.CustomerDetailsPage"
      id="com.ibm.commerce.telesales.customerDetailsPage" />

      <page name="Pets"
      class="com.ext.commerce.telesales.ui.impl.editors.customer.ExtCustomerPetsPage"
         id="com.ext.commerce.telesales.customerPetPage" />

      <page name="%customerShippingPageName"
   class="com.ibm.commerce.telesales.ui.impl.editors.customer.CustomerAddressesPage"
         id="com.ibm.commerce.telesales.customerAddressesPage" />

      <page name="%customerOrdersPageName"
      class="com.ibm.commerce.telesales.ui.impl.editors.customer.CustomerOrdersPage"
      id="com.ibm.commerce.telesales.customerOrdersPage" />

      <page name="%customerQuotesPageName"
      class="com.ibm.commerce.telesales.ui.impl.editors.customer.CustomerQuotesPage"
         id="com.ibm.commerce.telesales.customerQuotesPage" />

      <page name="%customerReturnsPageName"
      class="com.ibm.commerce.telesales.ui.impl.editors.customer.CustomerReturnsPage"
      id="com.ibm.commerce.telesales.customerReturnsPage" />

      <page name="%customerCommentsPageName"
   class="com.ibm.commerce.telesales.ui.impl.editors.customer.CustomerCommentsPage"
         id="com.ibm.commerce.telesales.customerCommentsPage" />

      <page name="%customerTicklersPageName"
   class="com.ibm.commerce.telesales.ui.impl.editors.customer.CustomerTicklersPage"
         id="com.ibm.commerce.telesales.customerTicklersPage" />
   </editor>
</extension>
```

3. Point to the redefinition of the customer pet editor page by creating the following entry in the config.ini file:

```
com.ibm.commerce.telesales.customerEditorPages=
            com.ext.commerce.telesales.customer.customerEditorPages
```

4. Define and lay out the controls for the customer pet form. Figure 9-6 displays the result of this task.

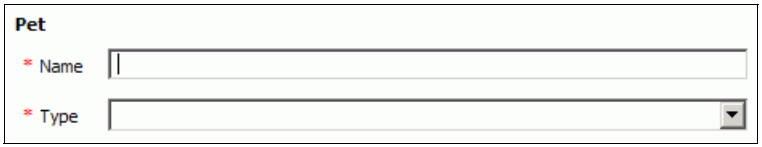


Figure 9-6 Layout defined by the customerPetCompositeDefinition

To define the controls, insert the extension shown in Example 9-11into the plugin.xml. This is the pet input form named customerPetComposite and its fields.

The Name and Type fields are required fields. The Sales Center UI framework automatically handles the processing of the required fields, enabling or disabling the Submit button depending on whether the required field is filled in. This is shown as a red asterisk (*) next to the label.

In order to enable the Sales Center UI framework to automatically handle the required fields, specify the following for the controls:

- The label and field must be connected with the use of the fieldId in the label control specifying the related input field ID
- The input field must have the required=”true” parameter specified
- The behavior of each of the fields is specified and controlled by a widget manager whose type is “pet”. (The process of defining this widget manager is described later in the section.)
- The composite specifies the “compositeType” property of the value “petForm”, which allows the widget manager to identify the control.
- The control’s text parameter specifies the resource property key that resolves to the resource property value at runtime.

Example 9-10 Defining the pet form controls

```
<extension point="com.ibm.commerce.telesales.widgets.controls">
   <!-- pet composite -->
   <control id="customerPetComposite" type="composite"
      compositeDefinitionId="customerPetCompositeDefinition"
      managerType="pet">
      <property name="compositeType" value="petForm" />
   </control>
   <!-- pet information composite components -->
   <control text="CustomerPetPage.petGroup" type="label"
      id="customerPetGroupLabel" font="org.eclipse.jface.bannerfont" />
   <control text="CustomerPetPage.petName" type="requiredLabel"
```

```
      id="customerPetNameLabel" fieldId="customerPetNameField" />
   <control required="true" type="text" id="customerPetNameField"
      modelPath="pet.name" managerType="pet"
      tooltip="AddPetDialog.tooltip.petName" />
   <control text="CustomerPetPage.petType" type="requiredLabel"
      id="customerPetTypeLabel" fieldId="customerPetTypeField" />
   <control required="true" type="combo" id="customerPetTypeField"
      modelPath="pet.type" managerType="pet"
      tooltip="AddPetDialog.tooltip.petType">
      <property name="list">
         <value>CustomerFindPetPage.petType1</value>
         <value>CustomerFindPetPage.petType2</value>
         <value>CustomerFindPetPage.petType3</value>
         <value>CustomerFindPetPage.petType4</value>
         <value>CustomerFindPetPage.petType5</value>
         <value>CustomerFindPetPage.petType6</value>
         <value>CustomerFindPetPage.petType7</value>
      </property>
   </control>
</extension>
```

Adding the extension shown in Example 9-11 into the plugin.xml defines the layout of the customerPetComposite pet form.

Example 9-11 Layout of the pet form

```
<extension
   point="com.ibm.commerce.telesales.widgets.compositeDefinitions">
   <!-- pet form content -->
   <gridCompositeDefinition id="customerPetCompositeDefinition"
      layoutId="com.ibm.commerce.telesales.ui.impl.standardGridLayout">
      <row id="customerPetGroupLabelRow">
         <control controlId="customerPetGroupLabel"
   dataId="com.ibm.commerce.telesales.ui.impl.headerLabelGridData" />
      </row>
      <row id="customerPetNameRow">
         <control controlId="customerPetNameLabel"
   dataId="com.ibm.commerce.telesales.ui.impl.requiredLabelGridData" />
         <control controlId="customerPetNameField"
   dataId="com.ibm.commerce.telesales.ui.impl.requiredFieldGridData" />
      </row>
      <row id="customerPetTypeRow">
         <control controlId="customerPetTypeLabel"
   dataId="com.ibm.commerce.telesales.ui.impl.requiredLabelGridData" />
         <control controlId="customerPetTypeField"
```

```
   dataId="com.ibm.commerce.telesales.ui.impl.requiredFieldGridData" />
      </row>
   </gridCompositeDefinition>
</extension>
```

5. Define and lay out the controls for the list buttons. Figure 9-7 displays the result of this task.

Figure 9-7 Layout of the customerPetListButtonCompositeDefinition list buttons

To define the controls, insert the extension shown in Example 9-13 into the plugin.xml, the list button composite named customerPetListButtonComposite and its buttons.

Example 9-12 Defining the list buttons

```
<!-- Pet editor pet list button components -->
<extension point="com.ibm.commerce.telesales.widgets.controls">
   <control id="customerPetListButtonComposite" type="composite"
      compositeDefinitionId="customerPetListButtonCompositeDefinition"
/>
   <control id="customerPetAddButton"
      text="CustomerPetPage.button.add" type="pushButton"
      managerType="pet">
      <property name="buttonType" value="addPet" />
   </control>
   <control id="customerPetDeleteButton"
      text="CustomerPetPage.button.delete" type="pushButton"
      managerType="pet">
      <property name="buttonType" value="deletePet" />
   </control>
</extension>
```

Adding the extension shown in Example 9-13 into the plugin.xml defines the layout of the list button composite customerPetListButtonComposite.

Example 9-13 Lay out the buttons by defining customerPetListButtonCompositeDefinition

```
<!-- Define the Pet List Buttons composite -->
<extension
   point="com.ibm.commerce.telesales.widgets.compositeDefinitions">
```

```
   <gridCompositeDefinition
      id="customerPetListButtonCompositeDefinition"
      layoutId="com.ibm.commerce.telesales.ui.impl.standardGridLayout">
      <row id="customerPetAddButtonRow">
         <control controlId="customerPetAddButton"
            dataId="com.ibm.commerce.telesales.ui.impl.buttonGridData"
/>
      </row>
      <row id="customerPetDeleteButtonRow">
         <control controlId="customerPetDeleteButton"
            dataId="com.ibm.commerce.telesales.ui.impl.buttonGridData"
/>
      </row>
   </gridCompositeDefinition>
</extension>
```

6. Define the new pet list controls and lay them out with the defined buttons composite. Figure 9-8 displays the result of this task.

Figure 9-8 Layout of the pet list composite as defined by customerPetListCompositeDefinition

To define new pet list controls, insert the extension from Example 9-14 into the plugin.xml. This consist of the list of pets composite named customerPetListComposite, the list field, and the group label.

Example 9-14 List of pet controls

```
<extension point="com.ibm.commerce.telesales.widgets.controls">
   <control id="customerPetListComposite" type="composite"
      compositeDefinitionId="customerPetListCompositeDefinition" />
   <control id="customerPetListGroupLabel" type="label"
      text="CustomerPetPage.customerPetListGroup"
      font="org.eclipse.jface.bannerfont" />
   <control id="customerPetList" type="list" managerType="pet">
      <property name="listType" value="pets" />
   </control>
</extension>
```

Adding the extension shown in Example 9-15 into the plugin.xml defines the layout of the customer pet list composite containing the pet list area and the action buttons.

Example 9-15 Laying out the pet list composite

```
<extension
   point="com.ibm.commerce.telesales.widgets.compositeDefinitions">
   <gridCompositeDefinition id="customerPetListCompositeDefinition"
      layoutId="com.ibm.commerce.telesales.ui.impl.standardGridLayout">
      <row id="customerPetListGroupLabelRow">
         <control controlId="customerPetListGroupLabel"
      dataId="com.ibm.commerce.telesales.ui.impl.headerLabelGridData"
/>
      </row>
      <row id="customerPetListRow">
         <control controlId="customerPetList"
         dataId="com.ibm.commerce.telesales.ui.impl.standardGridData"
/>
         <control controlId="customerPetListButtonComposite"
         dataId="com.ibm.commerce.telesales.ui.impl.standardGridData"
/>
      </row>
   </gridCompositeDefinition>
</extension>
```

7. Define and lay out the pet page content area. Figure 9-9 displays the result of this task.

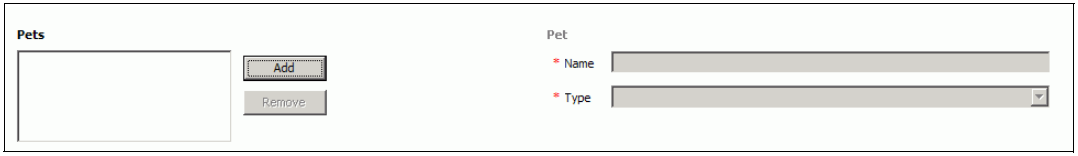


Figure 9-9 Lay out the pet page content area defined by the customerPetsCompositeDefinition

Insert the two extensions from Example 9-16 into the plugin.xml to define the new control, the pets composite named customerPetsComposite, and its layout, customerPetsCompositeDefinition, which lays out all the composites defined in the earlier steps.

Example 9-16 Laying out the pet page using all the defined composites

```
   <extension point="com.ibm.commerce.telesales.widgets.controls">
   <control id="customerPetsComposite" type="composite"
```

```
      compositeDefinitionId="customerPetsCompositeDefinition">
   </control>
</extension>

<!--  pet editor page layout of all top level components -->
<extension
   point="com.ibm.commerce.telesales.widgets.compositeDefinitions">
   <formCompositeDefinition id="customerPetsCompositeDefinition"
      layoutId="com.ibm.commerce.telesales.ui.impl.standardFormLayout">
      <control controlId="customerPetListComposite">
         <leftAttachment offset="0" numerator="0" />
         <rightAttachment offset="-15" numerator="50" />
      </control>
      <control controlId="customerPetComposite">
         <leftAttachment offset="15"
            relativeControlId="customerPetListComposite" />
         <rightAttachment numerator="100" offset="-15" />
      </control>
   </formCompositeDefinition>
</extension>
```

User interface behavior

Now that the interface elements are defined, define the basic UI behavior:

1. The behavior of the controls is managed by using widget managers. The Customer Pet Page content area controls are managed by the widget managers defined by the ExtPetWidgetManager class.

   Data sharing between the controls, request handlers, and the widget managers is performed with the property mechanism. The properties are saved and retrieved on an as-is required basis.

   The customer is a saved property with a name “customer” and contains the data of type Customer Model Object, which is used throughout the customer editor parts.

   In our extension, we defined the local “pets” property to maintain the current pet list of type ExtPetList, the “pet” property to maintain the currently selected item and displayed in the pet form. Also, all the pet-specific controls defined in the plugin.xml specify a property to identify the control during the custom widget manager initialization.

   Example 9-17 shows some of the methods dictating the behavior relating to the customer pet page.

- The constructor specifies the manager type to be “pet”. This ensures that only those controls that are defined as pet controls are to be managed by this widget manager.
- The initControl() method initializes all the pet controls managed by this widget manager calling the specific control intialization method, mainly specifying the listeners that will handle any events relating to the control (initControlPetList(), selectionListener_petList_).
- The refreshControl() method extends the default implementation to refresh the pet controls, enable and disable the buttons, coordinate the selected item in the list with the displayed information in the pet form, and so on.
- The saveControl() method extends the default implementation to save the pet controls to the customer model object. It saves any changes to the item displayed in the pet form to the “pet” property and saves the pet list changes into the customer object set in the “customer” property.

Example 9-17 ExtPetWidgetManager.java partial implementation

```
package com.ext.commerce.telesales.ui.impl.customer.pet;

public class ExtPetWidgetManager extends StandardWidgetManager {

    /**
     * Constructor for ExtPetWidgetManager. Initializes the manager of type "pet".
     */
    public ExtPetWidgetManager() {
        super();
        setManagerType(PROP_PET);
    }

public void initControl(ConfiguredControl configuredControl) {

        // initialize all the controls with the specified model path
        String modelPath = configuredControl.getModelPath();
        if (modelPath != null && modelPath.startsWith(PROP_PET_MODEL_PATH)) {
      modelPath = PROP_PET + modelPath.substring(PROP_PET_MODEL_PATH.length() - 1);
         configuredControl.setProperty(ControlDescriptor.ATT_MODEL_PATH, modelPath);
        }
        // initialize all the "pet" controls
        String compositeType = (String)
configuredControl.getProperty(PROP_COMPOSITE_TYPE);
        String buttonType = (String) configuredControl.getProperty(PROP_BUTTON_TYPE);
        String listType = (String) configuredControl.getProperty(PROP_LIST_TYPE);

        if (configuredControl != null
```

```
                && configuredControl.getManagerType().equals(getManagerType())) {

            // init the pet editor page controls
            if (BUTTON_TYPE_ADD_PET.equals(buttonType)) {
                initControl_AddPetButton(configuredControl);
            } else if (BUTTON_TYPE_DELETE_PET.equals(buttonType)) {
                initControl_DeletePetButton(configuredControl);
            } else if (LIST_TYPE_PETS.equals(listType)) {
                initControl_PetList(configuredControl);
            } else if (COMPOSITE_TYPE_PET_FORM.equals(compositeType)) {
              initControl_PetFormComposite((ConfiguredComposite) configuredControl);
            }
            // init the add pet dialog controls
            else if (BUTTON_TYPE_OK.equals(buttonType)) {
                initControl_OkButton(configuredControl);
            } else if (BUTTON_TYPE_CANCEL.equals(buttonType)) {
                initControl_CancelButton(configuredControl);
            }

        }
        super.initControl(configuredControl);
    }

/**
     * Initialize the pet list control
     * @param petsControl
     */
    protected void initControl_PetList(ConfiguredControl petsControl) {
        if (petsControl != null && petsControl.getControl() instanceof List) {
            control_PetList_ = petsControl;
            // listen in on selection events
            List list = (List) petsControl.getControl();
            list.addSelectionListener(selectionListener_petList_);
        }
    }
/**
     * Selection Listener handles the selection of pet list.
     */
    protected final SelectionListener
              selectionListener_petList_ = new SelectionAdapter() {
        public void widgetSelected(SelectionEvent event) {
            selected_PetList(event);
        }
    };
```

```
    /**
     * Refresh the specified control
     * @param configuredControl the configured control
     */
    public void refreshControl(ConfiguredControl configuredControl) {
        if (configuredControl != null && configuredControl.getControl() != null) {

            // refresh the pet specific controls

            // refresh the pet editor page controls
            if (configuredControl == control_PetList_) {
                refreshControl_PetList();
            } else if (configuredControl == control_PetFormComposite_) {
                refreshControl_PetFormComposite();
            } else if (configuredControl == control_deletePetButton_) {
                refreshControl_DeletePetButton();
            }
            // refresh the add dialog button controls
            else if (configuredControl == control_OKButton_) {
                refreshControl_OkButton();
            } else {
                super.refreshControl(configuredControl);
            }
        }
    }
/*
     * save the control data
     */
    public void saveControl(ConfiguredControl configuredControl) {
        if (configuredControl != null) {
            if (configuredControl.getManagerType().equals(getManagerType())) {
                // save existing objects
                ExtPet pet = null;
                Object object = getInputProperties().getData(PROP_PET);
                if (object instanceof ExtPet) {
                    pet = (ExtPet) object;
                    if (ExtPet.MARKED_EXISTNG.equals(pet.getMarking())
                            && configuredControl.isDirty()) {
                        pet.setMarking(ExtPet.MARKED_EDITED);
                    }
                } else {
                    // first selection or when returning from add_pet dialog
                    pet = (ExtPet) TelesalesModelObjectFactory
                            .createModelObject(ExtCustomerContants.MODEL_OBJECT_PET);
                    getInputProperties().setData(PROP_PET, pet);
```

```
            }
        }
        // Save the Pet List
        if (configuredControl == control_PetList_) {
            // save the list into the customer object
            Customer customer = getCustomerObject();
            if (customer != null) {
                // Pets
                ExtPetList pets = getPetsObject();
                ExtPetList mpets = new ExtPetList();
                mpets.setPets((ExtPet[]) pets.toArray(new ExtPet[pets.size()]));
                customer.setData(ExtCustomerContants.PROP_CUSTOMER_PETS, mpets);
            }
        } else {
            super.saveControl(configuredControl);
        }
    }
  }
}
```

2. Register the pet widget manager by adding the extension shown in Example 9-18 into the plugin.xml.

Example 9-18 Defining the pet widget manager

```
<!-- add pet editor page widget manager -->
<extension
   point="com.ibm.commerce.telesales.widgets.widgetManagers">
   <widgetManager
managerClass="com.ext.commerce.telesales.ui.impl.customer.pet.ExtPetWid
getManager"
id="customerPetWidgetManager" />
</extension>
```

3. Specify the customer pets composite to be managed by the pet widget manager by adding the extension shown in Example 9-19 into the plugin.xml.

Example 9-19 Defining the customer pets managed composite

```
<extension
   point="com.ibm.commerce.telesales.widgets.managedComposites">
   <managedComposite id="customerPetsManagedComposite"
      compositeId="customerPetsComposite">
      <widgetManager id="customerPetWidgetManager" />
      <!--  are these below really needed? -->
      <widgetManager
```

```
id="com.ibm.commerce.telesales.ui.impl.customerEditorWidgetManager" />
      <widgetManager
         id="com.ibm.commerce.telesales.widgets.standardWidgetManager"
/>
   </managedComposite>
</extension>
```

4. Extend the existing customer editor, CustomerEditor class, to handle the additional customer pet data by creating the ExtCustomerEditor class:
   a. Extend the initialization, and save, update, and refresh the methods to save and refresh the pet lists into the customer object as defined in Example 9-20.

      intializePets() reads the pet information from the customer object and saves the read list into the “pets” property.

      initializePets() is called on the customer editor save with the doSave() method and any customer model change with modelChanged() method (using the refreshPetWidgetManagerInputProperties() method).

Example 9-20 ExtCustomerEditor.java: Handling data implementation

```
package com.ext.commerce.telesales.ui.impl.editors.customer;

public class ExtCustomerEditor extends CustomerEditor {

    /**
     * Save the "pets" property with the pets stored in the customer object.
     * @param customer
     * @param widgetManagerInputProperties
     */
    protected void initializePets(Customer customer,
            WidgetManagerInputProperties widgetManagerInputProperties) {
        ExtPetList customerPets = ((ExtPetList) customer
                .getData(ExtCustomerContants.PROP_CUSTOMER_PETS));
        ExtPetList pets = new ExtPetList();
        for (int i = 0; customerPets != null && i < customerPets.size(); i++) {
            ExtPet pet = customerPets.getPet(i);
            try {
                pets.addPet((ExtPet) pet.clone());
            } catch (CloneNotSupportedException e) {
                UIImplPlugin.log(e);
            }
        }
        widgetManagerInputProperties.setData(ExtPetWidgetManager.PROP_PETS, pets);
```

```
    }

    /**
     * Refresh the "pets" property with the customer object.
     */
    protected void refreshPetWidgetManagerInputProperties() {
        WidgetManagerInputProperties widgetManagerInputProperties =
getWidgetManagerInputProperties();
        Customer customer = getCustomer(widgetManagerInputProperties);
        if (customer != null) {

            ExtPetList pets = (ExtPetList) widgetManagerInputProperties
                    .getData(ExtPetWidgetManager.PROP_PETS);
            if (pets != null) {
                boolean petsChanged = false;
                for (int i = 0; i < pets.size(); i++) {
                    ExtPet pet = pets.getPet(i);
                    // changed after fetch from database
                    if (!ExtPet.MARKED_EXISTNG.equals(pet.getMarking())) {
                        petsChanged = true;
                        break;
                    }
                }

                if (!petsChanged) {
                    initializePets(customer, widgetManagerInputProperties);
                }
            }
        }
    }

public void modelChanged(ModelObjectChangedEvent event) {

        super.modelChanged(event);
        refreshPetWidgetManagerInputProperties();
    }

    // ****** the two methods below should have only required update of
    // initWidgetManagerInputProperties() but that causes an infinite loop
    // **********

public WidgetManagerInputProperties getWidgetManagerInputProperties() {
        WidgetManagerInputProperties widgetManagerInputProperties = super
                .getWidgetManagerInputProperties();
```

```
        // initialize the "pets" property with customer object "pets"
        // NOTE: this step would have not been necessary if extending
        // initWidgetManagerInputProperty did not result in an unavoidable
        // infinite loop
        if (widgetManagerInputProperties != null
                &&
         widgetManagerInputProperties.getData(ExtPetWidgetManager.PROP_PETS) == null)
         {
            Customer customer = getCustomer(widgetManagerInputProperties);
            initializePets(customer, widgetManagerInputProperties);
        }
        return widgetManagerInputProperties;
    }

public boolean doSave() {
        // save the "pets" property with customer object "pets" upon save
        // NOTE: this step would have not been necessary if extending
        // initWidgetManagerInputProperty did not result in an unavoidable
        // infinite loop
        boolean savedResult = super.doSave();

        Customer customer = (Customer) getEditorInput().getAdapter(Customer.class);
        customer.suspendListenerNotification();

        if (savedResult) {
            Object[] pages = getPages();
            WidgetManagerInputProperties widgetManagerInputProperties
                 = getWidgetManagerInputProperties();
            Customer customerProp = getCustomer(widgetManagerInputProperties);
            //
            initializePets(customerProp, widgetManagerInputProperties);
            // unmark as dirty
            for (int i = 0; i < pages.length; i++) {
                ISaveablePart page = (ISaveablePart) pages[i];
                page.doSave(null);
            }
        }

        customer.resumeListenerNotification();
        return savedResult;
    }
```

b. Implement the ExtCustomerEditor methods that send the updated pet changes to the WebSphere Commerce Suite server (Example 9-21), initiated by clicking the **Update** button in the customer editor:

- ExtCustomerEditor.java only shows the add task snippets. The update and delete methods are implemented in a similar manner.
- The update() method is extended to process the pet list using the updatePetChanges() method.
- The updatePetChanges() method calls the appropriate task method depending on the marking on the pet object (addPet method, updatePet method, and deletePet method).
- The addPet() method calls the service request (with the ID com.ext.commerce.telesales.customer.addPet) to execute the addPet task and send the data to the server.

Example 9-21 ExtCustomerEditor.java: Send to server implementation

```
/*
     * Extend the customer object to update Pet changes.
     */
    public TelesalesRequestStatus update(Customer customer) {

        TelesalesRequestStatus status = super.update(customer);
        // save pet changes if customer object saved without problems
        if (status != null && status.isOK()) {
            ExtPetList petList = (ExtPetList) customer
                    .getData(ExtCustomerContants.PROP_CUSTOMER_PETS);
            if (petList != null) {
                Vector pets = petList.getPets();
                updatePetChanges(pets);
            }
        }
        return status;
    }

    /**
     * This method processes the pet changes, adds, updates or deletes the marked pet
objects.
     * @param pets list of pets to be checked if need to be updated
     */
    protected void updatePetChanges(Vector pets) {

        for (int i = 0; i < pets.size(); i++) {
            // ExtPet
            ExtPet pet = (ExtPet) pets.get(i);
            if (pet.getMarking().equals(ExtPet.MARKED_NEW)) {
                addPet(pet);
            } else if (pet.getMarking().equals(ExtPet.MARKED_EDITED)) {
```

```
            updatePet(pet);
        } else if (pet.getMarking().equals(ExtPet.MARKED_DELETED)) {
            deletePet(pet);
        }
    }
}

/**
 * Issues an Add action.
 *
 * The TelesalesCustomerPage implementation of this framework method sends a new
 * pet request to WebSphere Commerce issuing the action ID of
 * ExtCustomerContants.SERVICE_REQUEST_ADD_PET to the TelesalesJobScheduler.
 */
public void addPet(ExtPet pet) {
    TelesalesProperties parms = getParameters_AddPet(pet);

    try {
        TelesalesRequestStatus status = TelesalesJobScheduler.getInstance().run(
                ExtCustomerContants.SERVICE_REQUEST_ADD_PET, parms, true);
        TelesalesJobScheduler.handleErrors(status, this, true);
    } catch (InterruptedException ie) {
        UIImplPlugin.log(ie);
    } catch (Exception e) {
        UIImplPlugin.log(e);
    }
}

/**
 * Constructs the parameters for the ExtCustomerContants.SERVICE_REQUEST_ADD_PET
 * telesales action.
 * @param pet the pet
 * @return parameters for the add pet action
 */
protected TelesalesProperties getParameters_AddPet(ExtPet pet) {
    TelesalesProperties parms = new TelesalesProperties();
    parms.put("username",TelesalesModelManager.getInstance().getActiveOperator().
           getUID());
    parms.put("store", TelesalesModelManager.getInstance().getActiveStore());

    Customer customer = (Customer) getEditorInput().getAdapter(Customer.class);
    parms.put("customer", customer);
    parms.put("pet", pet);

    return parms;
```

```
}
```

5. Redefine the customer editor to use the extended ExtCustomerEditor.java class by adding the extension shown in Example 9-22 into the plugin.xml.

Example 9-22 Redefining the customer editor

```
<extension point="org.eclipse.ui.editors">
   <editor name="%customerEditorName"
      icon="icons/ctool16/edit_customer.gif"
      class="com.ext.commerce.telesales.ui.impl.editors
               .customer.ExtCustomerEditor"
      contributorClass="com.ibm.commerce.telesales.ui.impl.editors
               .customer.CustomerEditorActionBarContributor"
      id="com.ext.commerce.telesales.customer.customerEditor">
   </editor>
</extension>
```

6. Point to the new customer editor definition by inserting the following entryinto the config.ini:

```
com.ibm.commerce.telesales.customerEditor =
                  com.ext.commerce.telesales.customer.customerEditor
```

9.5.2 Implementing the integration code on the client side (part 1)

Now that the UI and its basic behavior (add, update, and remove) are created, you must be able to send the updated pet information to the server.

Request handler

Request handlers are responsible for creating the Business Object Document (BOD) messages with the information to be sent to the WebSphere Commerce server. They also handle the response BODs that are sent as response to the request from the server.

When you click the **Update** button in the customer editor, if customer information changes have occurred, the main request handler handling the sending of data to the WebSphere Commerce server is the SyncCustomerUpdateRequest that creates the SyncCustomer BOD message. Extend this request handler to add the pet information into the BOD message.

Note: The customer editor requires two integrations with the server:

- Send pet updates to the server
- Receive the existing customers' pet lists from the server

The first integration is documented in this section when discussing the development of the customer editor. The second integration is explained in 9.7, "Developing the find customer by pet dialog box" on page 230.

Request handler: Business Object Document message format

The passing of data between the server and the client is through the use of BOD messages. Extend the SyncCustomerUpdateRequest to generate the pet-specific components identified by the bold text in Example 9-23 for the Add Pet BOD message, Example 9-24 on page 207 for the Update Pet BOD message, and Example 9-25 on page 208 for the Delete Pet BOD message to be sent to the WebSphere Commerce server.

Note: To enable the showing of the generated BOD messages being passed between the client and the server, refer to the instructions provided in 8.1.3, "Enabling the task of showing the contents" on page 158.

Business Object Document message: Adding a customer's pet

Example 9-23 shows the SyncCustomer BOD message for the Add Customer's Pet task.

Example 9-23 SyncCustomer BOD message for the Add Customer's Pet task

```
<?xml version="1.0" encoding="UTF-8"?>
<wc:SyncCustomer .. ">
   <oa:ApplicationArea>
      ..
   </oa:ApplicationArea>
   <wc:DataArea>
      <oa:Sync confirm="Always">
         <oa:SyncCriteria expressionLanguage="XPath">
            <oa:SyncExpression action="Add">
               CustomerPet
         </oa:SyncExpression>
         </oa:SyncCriteria>
      </oa:Sync>
      <wc:Customer>
         <oa:CustomerParty>
            ..
         </oa:CustomerParty>
```

```
         <wc:CommerceArea>
            ..
         </wc:CommerceArea>
         <wc:UserAccount>
            ..
         </wc:UserAccount>
         <wc:CustomerDemographics>
            ...
         </wc:CustomerDemographics>
         <wc:UserData />
         <Pets>
            <Pet>
               <petName>Gigi</petName>
               <petType>Dog</petType>
            </Pet>
         </Pets>
      </wc:Customer>
   </wc:DataArea>
</wc:SyncCustomer>
```

Business Object Document message: Changing a customer's pet

Example 9-24 shows the SyncCustomer BOD for the Change Customer Pet task.

Example 9-24 SyncCustomer BOD for the Change Customer Pet task

```
<?xml version="1.0" encoding="UTF-8"?>
<wc:SyncCustomer ...>
   <oa:ApplicationArea>
      ...
   </oa:ApplicationArea>
   <wc:DataArea>
      <oa:Sync confirm="Always">
         <oa:SyncCriteria expressionLanguage="XPath">
            <oa:SyncExpression action="Change">
               CustomerPet
            </oa:SyncExpression>
         </oa:SyncCriteria>
      </oa:Sync>
      <wc:Customer>
         <oa:CustomerParty>
            ...
         </oa:CustomerParty>
         <wc:CommerceArea>
            ...
         </wc:CommerceArea>
```

```
         <wc:UserAccount>
            ...
         </wc:UserAccount>
         <wc:CustomerDemographics>
            ...
         </wc:CustomerDemographics>
         <wc:UserData />
         <Pets>
            <Pet>
               <petId>183</petId>
               <petName>Ginger</petName>
               <petType>Dog</petType>
            </Pet>
         </Pets>
      </wc:Customer>
   </wc:DataArea>
</wc:SyncCustomer>
```

Business Object Document message: Deleting a customer's pet

Example 9-25 shows the SyncCustomer BOD message for the Delete Customer Pet task.

Example 9-25 SyncCustomer BOD message for Delete Customer Pet task

```
<?xml version="1.0" encoding="UTF-8"?>
<wc:SyncCustomer ... ">
   <oa:ApplicationArea>
      ...
   </oa:ApplicationArea>
   <wc:DataArea>
      <oa:Sync confirm="Always">
         <oa:SyncCriteria expressionLanguage="XPath">
            <oa:SyncExpression action="Delete">
               CustomerPet
            </oa:SyncExpression>
         </oa:SyncCriteria>
      </oa:Sync>
      <wc:Customer>
         <oa:CustomerParty>
            ...
         </oa:CustomerParty>
         <wc:CommerceArea>
            ...
         </wc:CommerceArea>
         <wc:UserAccount>
```

```
            ,,,
         </wc:UserAccount>
         <wc:CustomerDemographics>
            ...
         </wc:CustomerDemographics>
         <wc:UserData />
         <Pets>
            <Pet>
               <petId>164</petId>
               <petName>Spot</petName>
               <petType>Small Animal</petType>
            </Pet>
         </Pets>
      </wc:Customer>
   </wc:DataArea>
</wc:SyncCustomer>
```

Request handler implementation: Sending the request

Implement a code to send the update request to the server by performing the following tasks:

1. To handle the task of sending the additional pet update data, extend SyncCustomerUpdateRequest by creating a new ExtSyncCustomerUpdateRequest request handler to generate the additional BOD message content.

   Example 9-28 shows the ExtSyncCustomerUpdateRequest request handler implementation. This request handler is only run to update the information pertaining to pets. The customer update information continues to be handled by the out-of-the-box SyncCustomreUpdateRequest request handler.

   The createSyncCriteriaElement() method overwrites the parent implementation to create the pet-specific sync criteria element (with the help of the getSyncExpressionAction() method) that will be read on the server side by the message mapper to determine which WebSphere Commerce command is to be executed to process the pet data, as shown in Example 9-26.

Example 9-26 The getSyncExpressionAction() method

```
<oa:SyncCriteria expressionLanguage="XPath">
   <oa:SyncExpression action="Delete/Add/Change">
      CustomerPet
   </oa:SyncExpression>
</oa:SyncCriteria>
```

The overridden createCustomerElement() adds the pet data section after the existing elements within the Customer elements, with the use of the createPetsElement() and createPetElement() methods, as shown in Example 9-27.

Example 9-27 The createPetsElement() and createPetElement() methods

```
<Pets>
   <Pet>
      <petId>164</petId>
      <petName>Spot</petName>
      <petType>Small Animal</petType>
   </Pet>
</Pets>
```

Example 9-28 shows the ExtSyncCustomerUpdateRequest: Send the request implementation.

Example 9-28 ExtSyncCustomerUpdateRequest: Send the request implementation

```
package com.ext.commerce.telesales.core.impl.request;

/**
 *  Extend to send customer pet add/delete/update requests and response.
 */
public class ExtSyncCustomerUpdateRequest extends SyncCustomerUpdateRequest {

    /**
     * petsElement_ contains the request document's Pets element. It
     * is initialized by the createPetsElement method.
     */
    protected Element petsElement_;

    /**
     * Builds the SyncCriteria element and adds it as a child of
     * syncElement_. The element is stored in syncCriteriaElement_.
     *
     * The following sample shows the structure of the SyncCriteria element and its
     * construction:
     *        <oa:SyncCriteria expressionLanguage="XPath">
     *           <oa:SyncExpression action="Change">CustomerPet</oa:SyncCriteria>
     *        </oa:SyncCriteria>
     * @return The SyncCriteria element.
     */
    protected Element createSyncCriteriaElement() {
        syncCriteriaElement_ = createOADocumentElement(syncElement_,
```

```
                IRequestConstants.BOD_TAG_OA_SYNC_CRITERIA);
            syncCriteriaElement_.setAttribute(
            IRequestConstants.BOD_ATT_EXPRESSION_LANGUAGE,
                IRequestConstants.BOD_VALUE_XPATH);
            Element syncExpressionElement = createOADocumentElement(syncCriteriaElement_,
                IRequestConstants.BOD_TAG_OA_SYNC_EXPRESSION, "CustomerPet");
            syncExpressionElement.setAttribute(IRequestConstants.BOD_ATT_ACTION,
                getSyncExpressionAction());
            return syncCriteriaElement_;
        }

        /**
         * Returns the action attribute for the SyncExpression element. This is
         * determined by the service request ID.
         * <UL>
         * <LI>If the service request ID=ExtCustomerContants.SERVICE_REQUEST_ADD_PET,
         * it returns the value "Add".</LI>
         * <LI>If the service request ID=ExtCustomerContants.SERVICE_REQUEST_DELETE_PET,
         * it returns the value "Delete".</LI>
         * <LI>If the service request ID=ExtCustomerContants.SERVICE_REQUEST_CHANGE_PET,
         * it returns the value "Change".</LI>
         * </UL>
    */
        protected String getSyncExpressionAction() {
            if
    (ExtCustomerContants.SERVICE_REQUEST_ADD_PET.equals(serviceRequest_.getId())) {
                return IRequestConstants.BOD_VALUE_ADD;
            } else if
    (ExtCustomerContants.SERVICE_REQUEST_DELETE_PET.equals(serviceRequest_.getId())) {
                return IRequestConstants.BOD_VALUE_DELETE;
            } else if
    (ExtCustomerContants.SERVICE_REQUEST_CHANGE_PET.equals(serviceRequest_.getId())) {
                return IRequestConstants.BOD_VALUE_CHANGE;
            }
            return null;
        }

        /**
         * Builds the Customer element and adds it as a child of
         * dataAreaElement_. The element is stored in customerElement_.
         *
         * Extended to create the pets element The following outline shows the structure
          * of the Customer element and its construction:
         *    <Customer>
         *       <oa:CustomerParty> <!-- constructed by createCustomerPartyElement() -->
```

```
     *          .
     *       </oa:CustomerParty>
     *       <CommerceArea> <!-- constructed by createCommerceAreaElement() -->
     *          .
     *       </CommerceArea>
     *       <UserAccount> <!-- constructed by createUserAccountElement() -->
     *          .
     *       </UserAccount>
     *       <CustomerDemographics> <!-- constructed by
createCustomerDemographicsElement() -->
     *          .
     *          .
     *       </CustomerDemographics>
     *       <UserData> <!-- constructed by createUserDataElement() -->
     *          .
     *          .
     *       </UserData>
     *   *     <Pets> <!-- constructed by createPetsElement() -->
     *          .
     *          .
     *       </Pets>
     *    </Customer>
     *
     * @return The Customer element.
     */
    protected Element createCustomerElement() {

        Element element = super.createCustomerElement();
        element = createPetsElement(element);
        return element;

    }

    /**
     * Builds the Pets element and adds it as a child of
     * customerElement_. The element is stored in petsElement_.
     *
     * The following outline shows the structure of the Pets element and its
     * construction:
     *          <Pets>
     *             <Pet> <!-- constructed by createPetElement(). -->
     *                .
     *                .
     *             </Pet>
     *          </Pets>
```

```
     * The pet passed to the createPetElement() method is determined by the
     * "pet" TelesalesProperties property.
     * @return The Pets element.
     */
    protected Element createPetsElement(Element customerElement) {
        petsElement_ = createOADocumentElement(customerElement,
ExtRequestConstants.BOD_TAG_PETS);
        createPetElement((ExtPet) getTelesalesProperties().get("pet"));
        return petsElement_;
    }

    /**
     * Builds the ExtPet element and adds it as a child of petsElement_.
     * The following sample shows the structure of the ExtPet element and its
     * construction:
     *          <Pet>
     *             <PetId>4052</PetId> - excluded if adding a new pet
     *              <Name>Rock</Name>
     *             <Type>Dog</Type>
     *          </Pet>
     * @param pet The com.ext.commerce.telesales.model.ExtPet data bean.
     * @return The ExtPet element.
     */
    protected Element createPetElement(ExtPet pet) {
        Element petElement = createWCDocumentElement(petsElement_,
ExtRequestConstants.BOD_TAG_PET);

        //only use pet id when updating or deleting a pet
        String petId = pet.getId();
        if (petId != null && petId.trim().length() > 0) {
            createOADocumentElement(petElement,
                    ExtRequestConstants.BOD_TAG_PET_ID, petId);
        }

        createOADocumentElement(petElement,
                    ExtRequestConstants.BOD_TAG_PET_NAME, pet.getName());
        createOADocumentElement(petElement,
                    ExtRequestConstants.BOD_TAG_PET_TYPE, pet.getType());
        return petElement;
    }
}
```

2. Register the service requests, specifying which request handler is to be called to run the service request task, as shown in Example 9-29. The service request, com.ext.commerce.telesales.customer.addPet, which is called from

within the customerEditor.java class to run pet updates, must be processed by the ExtSyncCustomerUpdateRequest request handler.

Example 9-29 Defining the pet command service requests

```
   <!-- Add Pet -->
   <serviceRequest label="Add Pet"
      requestHandlerClass=
         "com.ext.commerce.telesales.core.impl.request.ExtSyncCustomerUpdateRequest"
      id="com.ext.commerce.telesales.customer.addPet"
      commServiceId="com.ibm.commerce.telesales.services.TsCommunication">
   </serviceRequest>
   <!-- Delete Pet -->
   <serviceRequest label="Delete Pet"
      requestHandlerClass=
         "com.ext.commerce.telesales.core.impl.request.ExtSyncCustomerUpdateRequest"
      id="com.ext.commerce.telesales.customer.deletePet"
      commServiceId="com.ibm.commerce.telesales.services.TsCommunication">
   </serviceRequest>
   <!-- Update Pet -->
   <serviceRequest label="Update Pet"
      requestHandlerClass=
         "com.ext.commerce.telesales.core.impl.request.ExtSyncCustomerUpdateRequest"
      id="com.ext.commerce.telesales.customer.updatePet"
      commServiceId="com.ibm.commerce.telesales.services.TsCommunication">
   </serviceRequest>
</extension>
```

9.5.3 Implementing the integration code on the server side

The pet update information has been sent by the client and is now ready to be received by the server. Update the server side to recognize the new message that is being received.

Business Object Document message mapping

When the BOD message is received on the server side, it passes through the message mapper that determines the command that is to be run. The command is defined in the BOD message mapping files.

Provide the pet update BOD message mapping by extending the SyncCustomerBODMapping.xml as follows:

1. Copy SyncCustomerBODMapping.xml into a new file, ExtSyncCustomerBODMapping.xml.

2. Add the three new pet actions as defined in Example 9-31 within the <CommandMapping> tag of the XML definition shown in Example 9-30.

Example 9-30 Adding the three new pet actions

```
<TemplateDocument>
      <DocumentType version="8.1">SyncCustomer</DocumentType>
      <StartElement>SyncCustomer</StartElement>
      <TemplateTagName>SyncCustomer81Map</TemplateTagName>
<CommandMapping>To add here</CommandMapping>
```

In Example 9-31, the TemplateTagName defines the new parameter map for this command mapping, the CommandName defines the WC command that will handle this request, and the Condition specifies that the BOD message's syncExpression action and commerceObject identifies this as a customer pet action and must match what we have specified in our BOD Message format.

Example 9-31 Command mapping

```
<Command
CommandName="com.ext.commerce.usermanagement.commands.ExtPetUpdateCmd"
TemplateTagName="SyncCustomerPet81"
Condition='action="Change" AND commerceObject="CustomerPet"'>
</Command>

<Command
CommandName="com.ext.commerce.usermanagement.commands.ExtPetUpdateCmd"
TemplateTagName="SyncCustomerPet81"
Condition='action="Add" AND commerceObject="CustomerPet"'>
</Command>

<Command
CommandName="com.ext.commerce.usermanagement.commands.ExtPetDeleteCmd"
TemplateTagName="SyncCustomerPet81" Condition='action="Delete" AND
commerceObject="CustomerPet"'>
</Command>
```

3. Add the <TemplateTag> definition from Example 9-32 into the ExtSyncCustomerBODMapping.xml file to define the parameter mapping for the required parameters of the WebSphere Commerce pet update commands.

Example 9-32 ExtSyncCustomerBODMapping: Command parameters

```
<TemplateTag name="SyncCustomerPet81">
   <!-- Command Parameters -->
   <Tag XPath="DataArea/Sync/SyncCriteria/SyncExpression@action"
```

```
        Field="action" FieldInfo="COMMAND"/>
    <Tag XPath="DataArea/Sync/SyncCriteria/SyncExpression"
        Field="commerceObject" FieldInfo="COMMAND"/>
    <!--Command Parameters for pet info-->
    <Tag XPath="DataArea/Customer/Pets/Pet/petName" Field="name"/>
    <Tag XPath="DataArea/Customer/Pets/Pet/petType" Field="type"/>
    <Tag XPath="DataArea/Customer/Pets/Pet/petId" Field="petId"/>
    <Tag XPath="DataArea/Customer/CustomerParty/PartyId/Id"
        Field="userId"/>
    <Tag XPath="DataArea/Customer/UserData/UserDataField"
        XPathType="USERDATA"/>
</TemplateTag>
```

4. Add the extended mapping file ExtSyncCustomerBODMapping.xml into the webservice_SOABOD_template.extension.xml to register the new definiton, as shown in Example 9-33, with the newly added items displayed in **bold** text.

Example 9-33 webservice_SOABOD_template.extension.xml

```
<!-- If you are viewing with browser (IE for example), select View Source to see
actual source of file -->
<!DOCTYPE ECTemplate SYSTEM 'ec_template.dtd' [     <!-- Source comment: this [ is
required, do not remove -->

<!ENTITY ExtSyncCustomerMappingDefinition SYSTEM 'ExtSyncCustomerBODMapping.xml'>

]>   <!-- Source comment: this ]> is required, do not remove -->

<ECTemplate>
&ExtSyncCustomerMappingDefinition;
</ECTemplate>
```

Response builder

Response builders are responsible for creating BOD messages with the information to be sent to the Sales Center client. The response to the SyncCustomer BOD message is the ConfirmCustomer BOD message generated by the ConfirmCustomer response builder.

Response Business Object Document message

Extend the ConfirmCustomer response builder to generate the pet-specific response identified by the **bold** text in Example 9-34, for the confirm pet action message to be sent to the Sales Center client. The message contains the petId of the added, updated, or deleted pet.

Example 9-34 Response BOD: ConfirmCustomer

```
<?xml version="1.0" encoding="UTF-8"?>
<wc:ConfirmBOD .. >
   <oa:ApplicationArea>
      ..
   </oa:ApplicationArea>
   <wc:DataArea>
      <wc:Confirm />
      <wc:BOD>
         <wc:BODHeader>
            ..
         </wc:BODHeader>
         <wc:NounOutcome>
            <oa:DocumentIds>
               <oa:DocumentId>
                  <oa:Id />
               </oa:DocumentId>
               <wc:AddressId>
                  <oa:Id>NOT FOUND</oa:Id>
               </wc:AddressId>
               <petId>
                  <oa:Id>183</oa:Id>
               </petId>
            </oa:DocumentIds>
            <oa:NounSuccess />
            <oa:UserArea />
         </wc:NounOutcome>
      </wc:BOD>
   </wc:DataArea>
</wc:ConfirmBOD>
```

Response builder implementation

Modify the response builder to create a response to the pet update. In our case, we extended the ConfirmCustomer BOD response builder to add the <petId> tag by creating the ExtConfirmCustomer response builder, as defined in Example 9-35.

The createDocumentIDsElement() method adds the new <petId> element section into the <DocmentIds> element. The ConfirmCustomer response builder generates the confirmation for all the SyncCustomer BOD messages, including the CustomerPet and Customer and CustomerAddress object updates. Therefore, care must be taken not to break the original response.

determinePetId() fetches the petId passed in the response properties by the execute WebSphere Commerce pet commands.

Example 9-35 ExtConfirmCustomer.java implementation

```
package com.ext.commerce.telesales.messaging.bodreply;

/**
 * Extended ConfirmCustomer to return the customerPet object petId that was added,
deleted or
 * updated.
 */
public class ExtConfirmCustomer extends ConfirmCustomer {

    /**
     * Creates the DocumentIds element for the given parent element and document ID.
     *
     * Calls the createDocumentIdsElement method to create the
     * DocumentIds element.
     *
     * This extension will include the PetId element into the
     * DocumentIds element when the syncObject is "customerPet".
      */
    protected Element createDocumentIdsElement(Element aParentElement, String
                          astrDocumentId) throws ECException {

        Element documentIdsElement = super.createDocumentIdsElement(
               aParentElement, astrDocumentId);

        String syncObject = null;
        String taskName = getTaskName();

        syncObject = getElementValueByLocalName(getRequestBod(),
                 BodConstants.TAG_LOCAL_SYNC_CRITERIA,
                 BodConstants.TAG_LOCAL_SYNC_EXPRESSION);

        // add this section if customerPet was updated
        if (null != syncObject
            && syncObject.equalsIgnoreCase(ExtBODConstants.WC_OBJECT_CUSTOMER_PET)){

            Element petIdElement = createWCDocumentElement(documentIdsElement,
                    ExtBODConstants.BOD_TAG_PET_ID);
            String astrPetId = determinePetId();
            createOADocumentElement(petIdElement, BodConstants.TAG_OA_ID, astrPetId);
        }
```

```
        return documentIdsElement;
    }

    /**
     * Gets the pet ID to be used for the response Business Object Document.
     * @return the pet ID.
     */
    protected String determinePetId() {
        String strPetId = "";

        /* If a CustomerId after successful add of a customer, return that value */
        if (getRequestProperties().containsKey(ExtBODConstants.KEY_PET_ID)) {
            try {
               strPetId = getResponseProperties().getString(
                     ExtBODConstants.KEY_PET_ID);
            } catch (ParameterNotFoundException e) /* Should not occur catch */
            { strPetId = BodConstants.TAG_NOT_FOUND; }
        } else {
            strPetId = BodConstants.TAG_NOT_FOUND;
        }
        return strPetId;
    }
}
```

Response builder registration

The response builder must be registered for it to be recognized.

Note: For instructions about registering the new response builders, refer to the information center topic “Modifying an existing Business Object Document reply message”, which is available on the Web at:

http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.telesales.developer.doc/tasks/ttrmodifybodreply.htm

To register the response builder, perform the following tasks:

1. Create a new TelesalesRegistry-ext.xml file in the <Toolkit>/xml/messaging folder.
2. Add the text shown in Example 9-39.

Example 9-36 Adding a new TelesalesRegistry-ext.xml file

```
<WCTBodResponseBuilderRegistry>
   <Noun name="Customer">
      <Verb name="Sync">
```

```
        <ClassName>
    com.ext.commerce.telesales.messaging.bodreply.ExtConfirmCustomer
        </ClassName>
      </Verb>
   </Noun>
</WCTBodResponseBuilderRegistry>
```

3. Update wc-server.xml in the <Toolkit>/xml/config folder by modifying the entry shown in Example 9-36 to match the definition shown in Example 9-37. Note the modified text identified by the **bold** text.

Example 9-37 Updating the wc-server.xml

```
<property
 baseRegistryFileName="TelesalesRegistry.xml"
 baseRegistryFilePath="messaging"
 customRegistryFileName="TelesalesRegistry-ext.xml"
 customRegistryFilePath="messaging"
display="false" enableBaseRegistryOverride="true" />
```

9.5.4 Implementing the integration code on the client side (part 2)

The server response has been sent back to the client. The client must be modified to react to the response from the server, reporting on the success of the pet updates.

Request handler implementation: Unmarshalling the response

Request handlers not only handle the data to be sent to the server but also the data in the response BODs sent as a response to the request from the server.

In our scenario, the existing SyncCustomerUpdateRequest also handles the response (Example 9-38) from the ConfirmCustomer response builder (ConfirmCustomer BOD message), specifying the success or failure messages and the passed data. To process the pet-specific confirmation information returned in the response BOD, extend the ExtSyncCustomerUpdateRequest.

To handle the additional pet-specific confirmation information, implement the additional methods in the ExtSyncCustomerUpdateRequest class created in “Request handler implementation: Sending the request” on page 209:

1. Read the response BOD petId by overriding the parent method unmarshallDocumentIds() to unmarshall the pet ID section, and unmarshallPetId() to fetch the petId and update the petId if the request is an add request. Because the ExtSyncCustomerUpdateRequest will only be

called for pet-specific service requests, you can disregard the other document IDs being passed in the message.

2. Extend the updateModel() method to update the model on successful syncCustomer execution, by removing the deleted pet from the model and changing the marking to existing for the added or updated pets.

Example 9-38 ExtSyncCustomerUpdateRequest: Unmarshalling the response to the request

```
/**
     * Unmarshalls the DocumentIds element. The following sample indiates the
     * structure of the DocumentIds element and how it is unmarshalled. Extended to
     * unmarshall the tag "petId".
     *
     *       <oa:DocumentIds>
     *          <oa:DocumentId> <!-- unmarshalled by unmarshallDocumentId() -->
     *            .
     *            .
     *          </oa:DocumentId>
     *       </oa:DocumentIds>
     *
     * @param documentIdsElement The element that is unmarshalled.
     */
      protected void unmarshallDocumentIds(Element documentIdsElement) {
          if (documentIdsElement != null) {
            if (ExtCustomerContants.SERVICE_REQUEST_ADD_PET.equals(
                    serviceRequest_.getId())) {
                unmarshallPetId(getChildElement(
                    documentIdsElement, ExtRequestConstants.KEY_PET_ID));
            }
        }
    }
    /**
     * Unmarshall the response BOD document ID
     * @param petIdElement
     */
    protected void unmarshallPetId(Element petIdElement) {
        if (petIdElement != null) {
            String petId = getChildElementValue(petIdElement,
IRequestConstants.BOD_TAG_GEN_ID);
            ExtPet pet = (ExtPet) getTelesalesProperties().get("pet");
            pet.setId(petId);
        }
    }
    /**
     * Updates the model after the action has completed. The Customer object that was
```

```
     * passed in as a the "customer" TelesalesProperties parameter is
     * updated to reflect the change in the pet list.
     */
    protected void updateModel(Object databean) {
        TelesalesRequestStatus status = getResponseStatus();
        if (status == null || status.getSeverity() != IStatus.ERROR) {
            ExtPet pet = (ExtPet) getTelesalesProperties().get("pet");
            // if item was to be deleted - remove it from the list
            if
(ExtCustomerContants.SERVICE_REQUEST_DELETE_PET.equals(serviceRequest_.getId())) {
                ExtPetList pets = (ExtPetList) customer_
                        .getData(ExtCustomerContants.PROP_CUSTOMER_PETS);
                pets.removePetForPetId(pet.getId());
                customer_.setData(ExtCustomerContants.PROP_CUSTOMER_PETS, pets);
            }
            // add/update requests change it to existing
            else {
                pet.setMarking(ExtPet.MARKED_EXISTNG);
            }
        }
    }
```

9.6 Developing the new add pet dialog box

This section describes the actions required to implement the new Add Pet dialog box shown in Figure 9-10.

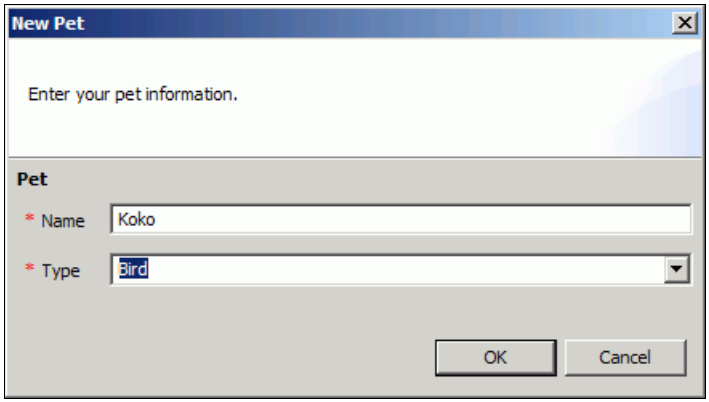


Figure 9-10 Add pet dialog box

9.6.1 Implementing the user interface components

This section defines the UI and the basic UI behavior for the add pet dialog box.

User interface definition

To create the UI elements, perform the following tasks:

1. Create the dialog box by performing these tasks:
   a. Define the new Add Pet dialog box by creating a new dialog Java class, `ExtAddPetDialog.java`, which must be similar to that shown in Example 9-39.

      In our case, to reuse the Sales Center UI framework, we extended from the ConfiguredTitleAreaDialog that provides support for the managed composites and contains a title area. Most Sales Center dialog boxes extend from this class.

      Extending from the Sales Center UI framework classes allows the defining of controls on the page using XML, by providing the methods that specify which managed composite content area and which managed composite button bar defined in plugin.xml will be used to render the dialog box. The content area is specified by implementing the protected String getDialogAreaManagedCompositeId() method to the ID of the content area, and the protected String getButtonBarManagedCompositeId() method to the ID of the button bar, which is defined later in this section.

      The configureShell() and getDefaultMessage() methods have been implemented to personalize the dialog box with a custom title and a message informing the user through a default message about what to enter in the dialog box.

      cancelPressed() and okPressed() are extended methods to provide add pet dialog box-specific actions to the buttons, cancelling the window without providing a result and returning the pet model object.

Example 9-39 ExtAddPetDialog.java

```
package com.ext.commerce.telesales.ui.impl.dialogs;

public class ExtAddPetDialog extends ConfiguredTitleAreaDialog {

    /**
     * Field to store the dialog result - the new pet
     */
    private Object data_ = null;

    /**
     * Constructor for ExtAddPetDialog
```

```
     */
    public ExtAddPetDialog() {
        super();
        getWidgetManagerInputProperties().
               setData(ExtPetWidgetManager.PROP_ADD_PET_DIALOG, this);
    }

protected String getDialogAreaManagedCompositeId() {
      return "com.ext.commerce.telesales.customer.addPetDialogAreaManagedComposite";
    }

protected String getButtonBarManagedCompositeId() {
        return "com.ext.commerce.telesales.customer.addPetButtonBarManagedComposite";
    }

    /*
     * Return the newly created pet
     */
    public Object getResult() {
        return this.data_;
    }

    /**
     * Set the newly created pet
     * @param data
     */
    public void setResult(Object data) {
        this.data_ = data;
    }

    // ********************************************************************** //
    /**
     * Configures the given shell in preparation for opening this window in it.
     * @param newShell shell
     */
    public void configureShell(Shell newShell) {
        newShell.setText(Resources.getString("AddPetDialog.shellTitle"));
    }

    /**
     * Returns the default message for this dialog.
     * @return the default message
     */
    protected String getDefaultMessage() {
        return Resources.getString("AddPetDialog.petDefaultMessage");
```

```
    }

    // ********************************************************************//
    /**
     * Notifies that the Cancel button of this dialog has been pressed.
     */
    public void cancelPressed() {
        setResult(null);
        super.cancelPressed();
    }
public void okPressed() {
        // save the data
        getDialogAreaManagedComposite().save();
        ExtPet newPet = (ExtPet) getWidgetManagerInputProperties().getData(
                ExtPetWidgetManager.PROP_PET);
        setResult(newPet);
        super.okPressed();
    }
}
```

b. Declare the new dialog box by adding the extension defined in Example 9-40 to the plugin.xml.

Example 9-40 Declaring the dialog box

```
<extension point="com.ibm.commerce.telesales.ui.dialogs">
   <dialog
    class="com.ext.commerce.telesales.ui.impl.dialogs.ExtAddPetDialog"
      id="com.ext.commerce.telesales.customer.extAddPetDialog" />
</extension>
```

2. Define and lay out the controls for the dialog box buttons by performing the following tasks (Figure 9-11 displays the result of this step):

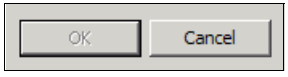


Figure 9-11 addPetDialogButtonsCompositeDefinition

a. Insert the extension shown in Example 9-41 into the plugin.xml to define the controls, the buttons composite (named addPetdialogButtonsComposite), and the buttons.

Example 9-41 Defining the dialog box button controls

```
<!-- Control and composite definitions for the add pet dialog buttons
-->
```

```
<extension point="com.ibm.commerce.telesales.widgets.controls">
   <!-- add pet dialog button composite -->
   <control id="addPetDialogButtonsComposite" type="composite"
      compositeDefinitionId="addPetDialogButtonsCompositeDefinition" />
   <!-- add pet dialog button composite components -->
   <control id="addPetOkButton" type="pushButton"
      text="AddPetDialog.button.ok" managerType="pet">\-
      <property name="buttonType" value="ok" />
   </control>
   <control id="addPetCancelButton" type="pushButton"
      text="AddPetDialog.button.cancel" managerType="pet">
      <property name="buttonType" value="cancel" />
   </control>
</extension>
```

b. Add the extension shown in Example 9-42 to the plugin.xml to define the layout of the addPetDialogButtonComposite button bar.

Example 9-42 Laying out the add pet dialog box button bar

```
<!-- add pet dialog buttons composite layout definition -->
<extension
   point="com.ibm.commerce.telesales.widgets.compositeDefinitions">
   <!-- add pet dialog buttons composite layout definition -->
   <gridCompositeDefinition
      id="addPetDialogButtonsCompositeDefinition"
    layoutId="com.ibm.commerce.telesales.ui.impl.buttonBarGridLayout">
      <row>
         <control controlId="addPetOkButton"
            dataId="com.ibm.commerce.telesales.ui.impl.buttonGridData"
/>
         <control controlId="addPetCancelButton"
            dataId="com.ibm.commerce.telesales.ui.impl.buttonGridData"
/>
      </row>
   </gridCompositeDefinition>
   </extension>
```

3. Define and lay out the content area for the Add Pet dialog box. This content area is the same area that was defined in Figure 9-6 on page 190, which will be reused for rendering the dialog box content area (described later in this section).

User interface behavior

Now that the interface elements are defined, define the basic UI behavior.

1. The behavior of the Add Pet dialog box is defined with the use of a widget manager. In our scenario, the widget manager ExtPetWidgetManager class is the same as that for the customer pet page.

   Example 9-43 shows some of the methods that dictate the behavior relating to the Add Pet dialog box:

   – The Add Pet button in the Customer Pet page is initialized to listen to the selectionListener_AddPetButton listener, which launches the add pet dialog box on clicking it (first through the pressedButton_AddPet method and then the openDialog_AddPet method).

   – The openDialog_AddPet() method instantiates and then opens the dialog box. It also processes the returned dialog box result, adding the new pet to the maintained pet list (“pets” property).

   – All the Add Pet dialog box buttons are initialized using the widget manager that specifies a listener for each button and an appropriate action to be taken on clicking the button. The actual action that is called is the one defined in the dialog class (see the selectionListener_OKButton selection listener and the pressedButtonOK method in Example 9-43).

Example 9-43 ExtPetWidgetManager.java: Add pet dialog methods

```
/**
     * Selection Listener handles the button click of the Add Pet Button.
     */
    protected SelectionListener
         selectionListener_AddPetButton_ = new SelectionAdapter() {
            public void widgetSelected(SelectionEvent arg0) {
               pressedButton_AddPet();
            }
         };

/**
     * This method handles pressing the Add Pet Dialog button. It opens the dialog.
     */
    protected void pressedButton_AddPet() {
        openDialog_AddPet();
    }

/**
     * Opens the add pet dialog and processes its result.
     */
    protected void openDialog_AddPet() {
```

```
        IDialog myDialog = DialogFactory
                .getDialog("com.ext.commerce.telesales.customer.extAddPetDialog");
        myDialog.open(); //show the dialog

        if (myDialog.getResult() != null) {
            // save the added pet
            ExtPet pet = (ExtPet) myDialog.getResult();
            pet.setMarking(ExtPet.MARKED_NEW);
            getPetsObject().addPet(pet);
            getInputProperties().setData(PROP_PET, pet);
        }
    }

    /**
     * Opens the add pet dialog and processes its result.
     */
    protected void openDialog_AddPet() {
        IDialog myDialog = DialogFactory
                .getDialog("com.ext.commerce.telesales.customer.extAddPetDialog");
        myDialog.open(); //show the dialog

        if (myDialog.getResult() != null) {
            // save the added pet
            ExtPet pet = (ExtPet) myDialog.getResult();
            pet.setMarking(ExtPet.MARKED_NEW);
            getPetsObject().addPet(pet);
            getInputProperties().setData(PROP_PET, pet);
        }
    }

/**
     * Selection Listener handles the button click of add pet dialog OK button.
     */
    protected final SelectionListener
         selectionListener_OKButton_ = new SelectionAdapter() {
            public void widgetSelected(SelectionEvent arg0) {
               pressedButton_OK();
            }
      };

    /**
     * this method is called when the ok button is pressed on the add pet dialog.
     */
    protected void pressedButton_OK() {
        if (control_AddPetDialog_ != null) {
```

```
            control_AddPetDialog_.okPressed();
        }
    }
}
```

2. Register the Add Pet dialog box widget manager by adding the extension shown in Example 9-44 into the plugin.xml.

Example 9-44 Defining the add pet dialog box widget manager

```
<!-- add pet dialog widget manager -->
<extension
   point="com.ibm.commerce.telesales.widgets.widgetManagers">
   <widgetManager
managerClass="com.ext.commerce.telesales.ui.impl.customer.pet.ExtPetWid
getManager"
      id="addPetWidgetManager" />
</extension>
```

3. Specify the button bar composite to be managed by the add pet widget manager by adding the extension shown in Example 9-45 into the plugin.xml. This managed composite, addPetButtonBarManagedComposite, is the one that is specified in the class ExtAddPetDialog.java as the button bar to be rendered.

Example 9-45 Defining the add pet dialog box button bar

```
<!-- Add pet dialog button bar managed composite -->
<extension
   point="com.ibm.commerce.telesales.widgets.managedComposites">
      <managedComposite id="addPetButtonBarManagedComposite"
         compositeId="addPetDialogButtonsComposite">
      <widgetManager id="addPetWidgetManager" />
   </managedComposite>
```

4. Specify the dialog box content area composite to be managed by the add pet widget manager by adding the extension shown in Example 9-46 into the plugin.xml. Note that the content area customerPetComposite defined in Figure 9-6 on page 190 is being reused in this definition. This managed composite, addPetDialogAreaManagedComposite, is the one that is specified in the class ExtAddPetDialog.java as the content area to be rendered.

Example 9-46 Defining the dialog box content managed composite

```
<!-- Add pet dialog managed composite -->
<extension
   point="com.ibm.commerce.telesales.widgets.managedComposites">
```

```
   <managedComposite id="addPetDialogAreaManagedComposite"
      compositeId="customerPetComposite">
      <widgetManager id="addPetWidgetManager" />
      <widgetManager
         id="com.ibm.commerce.telesales.widgets.standardWidgetManager"
/>
   </managedComposite>
</extension>
```

9.7 Developing the find customer by pet dialog box

This section demonstrates the steps involved in implementing the find customer by pet dialog box that is shown in Figure 9-12.

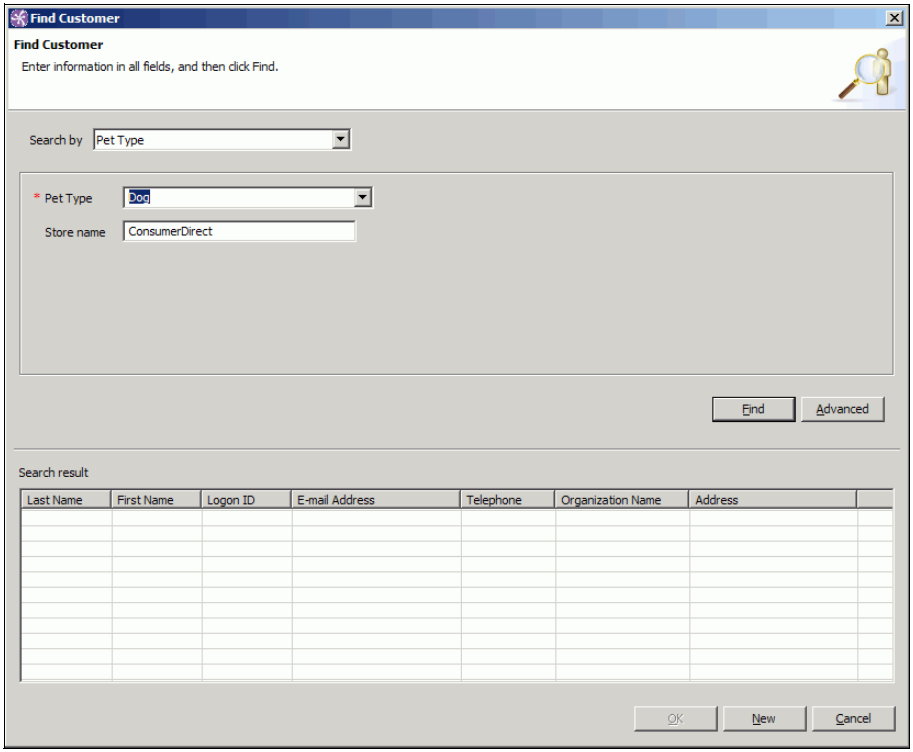


Figure 9-12 Find customer by pet dialog box

9.7.1 Implementing the user interface components

This section defines the UI and the basic UI behavior for the Find Customer by Pet page.

User interface definition

Create the UI elements by performing the following tasks:

1. Define and lay out the controls for the Find Customer by Pet dialog box (Figure 9-13 displays the result of this step) by performing the following tasks:

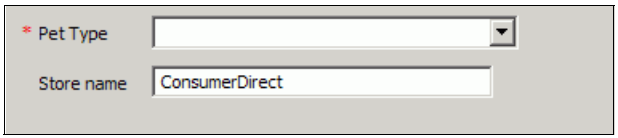


Figure 9-13 Layout of the find customer by pet form

   a. Insert the extension shown in Example 9-47 into the plugin.xml to define the controls, the form composites (customerWithStoreFindPetPageCompositeDefinition and customerFindPetPageComposite), labels, and fields. (In our case, we followed the way the existing pages are defined in the plugin.xml to define the new search type form.)

      Specifying the managerType=”find” on the customerFindPetPageComposite helps reuse the existing widget manager to handle the new search type that processes the composite properties such as the title property, which is added to the drop-down search type list.

Example 9-47 Defining the controls for the find customer by pet dialog box

```
<!-- Control and composite definitions for quick page pet of the find customer dialog
-->
<extension point="com.ibm.commerce.telesales.widgets.controls">
   <control id="customerFindPetPageComposite" type="composite"
      compositeDefinitionId="customerFindPetPageCompositeDefinition"
      managerType="find">
      <property name="compositeType" value="findCriteriaPage" />
      <property name="title" value="CustomerFindPetPage.title" />
   </control>
   <control id="customerWithStoreFindPetPageComposite"
      referenceId="customerFindPetPageComposite"
      compositeDefinitionId="customerWithStoreFindPetPageCompositeDefinition" />
   <control id="findCustomerPetTypeLabel" type="requiredLabel"
      text="CustomerFindPetPage.label.petType"
      fieldId="findCustomerPetTypeField" />
```

```
   <control textLimit="256" required="true" type="combo"
      managerType="find" tooltip="CustomerFindPetPage.tooltip.petType"
      id="findCustomerPetTypeField">
      <property name="findCriteriaFieldName" value="petType" />
      <property name="list">
         <value>CustomerFindPetPage.petType1</value>
         <value>CustomerFindPetPage.petType2</value>
         <value>CustomerFindPetPage.petType3</value>
         <value>CustomerFindPetPage.petType4</value>
         <value>CustomerFindPetPage.petType5</value>
         <value>CustomerFindPetPage.petType6</value>
         <value>CustomerFindPetPage.petType7</value>
      </property>
   </control>
</extension>
```

b. Adding the extension elements shown in Example 9-48 into the plugin.xml defines the layout of the new search type form.

   Specifying the referenceId of an existing control helps reuse the previous definition and add the new declaration to what is already defined. The referenceId="com.ibm.commerce.telesales.ui.impl.findStoreOptionalNameRow" declaration in the layout definition adds the Store field defined out-of-the-box and easily adds it to the extension without redeclaring everything from the scratch.

Example 9-48 Defining the layout of the find customer by pet composite definition

```
<extension
   point="com.ibm.commerce.telesales.widgets.compositeDefinitions">
   <gridCompositeDefinition
      id="customerFindPetPageCompositeDefinition"
      layoutId="com.ibm.commerce.telesales.ui.impl.standardGridLayout">
      <row id="findCustomerPetTypeRow">
         <control controlId="findCustomerPetTypeLabel"
            dataId="com.ibm.commerce.telesales.ui.impl.requiredLabelGridData" />
         <control controlId="findCustomerPetTypeField"
            dataId="com.ibm.commerce.telesales.ui.impl.findTextFieldGridData" />
      </row>
   </gridCompositeDefinition>

   <gridCompositeDefinition
      id="customerWithStoreFindPetPageCompositeDefinition"
      referenceId="customerFindPetPageCompositeDefinition">
      <row
         referenceId="com.ibm.commerce.telesales.ui.impl.findStoreOptionalNameRow" />
```

```
   </gridCompositeDefinition>
</extension>
```

c. Adding the extension shown in Example 9-49 into the plugin.xml defines the quick stack definition of the search-type forms. The find customer by pet dialog box is added as the last item in the search-type list.

Example 9-49 Defining the quick stack composite definition

```
<!-- Find customer dialog quick stack composite definitions -->
<extension
   point="com.ibm.commerce.telesales.widgets.compositeDefinitions">
   <!-- B2C Find customer dialog quick stack definition -->
   <stackCompositeDefinition

referenceId="com.ibm.commerce.telesales.ui.impl.findCustomerQuickStackDefinition.defa
ult"
      id="findCustomerQuickStackDefinition.default"
      layoutId="com.ibm.commerce.telesales.ui.impl.standardStackLayout">
      <control controlId="customerFindPetPageComposite" />
   </stackCompositeDefinition>
   <!-- B2B Find customer dialog quick stack definition -->
   <stackCompositeDefinition

referenceId="com.ibm.commerce.telesales.ui.impl.findCustomerQuickStackDefinition.B2B"
      id="findCustomerQuickStackDefinition.B2B"
      layoutId="com.ibm.commerce.telesales.ui.impl.standardStackLayout">
      <control controlId="customerFindPetPageComposite" />
   </stackCompositeDefinition>

   <!-- Find customer with store dialog quick stack definition -->
   <stackCompositeDefinition

referenceId="com.ibm.commerce.telesales.ui.impl.findCustomerWithStoreQuickStackDefini
tion"
      id="findCustomerWithStoreQuickStackDefinition"
      layoutId="com.ibm.commerce.telesales.ui.impl.standardStackLayout">
      <control controlId="customerWithStoreFindPetPageComposite" />
   </stackCompositeDefinition>
</extension>
```

d. Point to the redefinition of the stack definitions by creating the entries displayed in Example 9-50 in the config.ini file.

Example 9-50 Updating config.ini with the new stack definitions

```
com.ibm.commerce.telesales.ui.impl.findCustomerQuickStackDefinition.default =
        com.ext.commerce.telesales.customer.findCustomerQuickStackDefinition.default

com.ibm.commerce.telesales.ui.impl.findCustomerQuickStackDefinition.B2B =
        com.ext.commerce.telesales.customer.findCustomerQuickStackDefinition.B2B

com.ibm.commerce.telesales.ui.impl.findCustomerWithStoreQuickStackDefinition =
      com.ext.commerce.telesales.customer.findCustomerWithStoreQuickStackDefinition
```

9.7.2 Implementing the integration code on the server side

The server receives a request to find customers with a specific type of pet. Modify the server side to limit the result to only that list of customers to be returned. Also, the server side must return the customer information, including additional pet information.

Response builder

Response builders are responsible for creating BOD messages with information to be sent to the Sales Center client. The response to the GetCustomer BOD message initiated by the Find Customer action is the ShowCustomer BOD message generated by the ShowCustomer response builder.

Response Business Object Document message

The Find dialog box does not require the extension of the BOD because no pet-specific information is displayed in the Find dialog box. For the Find dialog box, the BOD message stays as the default message.

However, in order to display the customer information in the customer editor, extend the ShowCustomer BOD message to contain the response BOD tags. These are identified by the **bold** text in Example 9-51 that is to be sent to the Sales Center client. The new tags contain the list of the customer's pets.

Example 9-51 ShowCustomer BOD message

```
<?xml version="1.0" encoding="UTF-8"?>
<wc:ShowCustomer ..>
   <oa:ApplicationArea>
      ..
   </oa:ApplicationArea>
   <wc:DataArea>
```

```
      <wc:Show confirm="Always" numSearchResults="1"
         resultSetSize="1" />
      <wc:Customer>
         <oa:CustomerParty active="false">
            ..
         </oa:CustomerParty>
         <wc:CommerceArea>
            ..
         </wc:CommerceArea>
         <wc:CustomerDemographics />
         <Pets>
            <Pet>
               <petId>166</petId>
               <petName>Max</petName>
               <petType>Dog</petType>
            </Pet>
            <Pet>
               <petId>183</petId>
               <petName>Ginger</petName>
               <petType>Dog</petType>
            </Pet>
         </Pets>
         <wc:UserData />
      </wc:Customer>
   </wc:DataArea>
</wc:ShowCustomer>
```

Response builder implementation

Extend the response builder to limit the result in such a way that it contains only customers with a specific pet type, and to return the additional pet information.

We extended the ShowCustomer response builder to support the find dialog search by pet type and the customer editor to show the customer's pets, by creating the ExtShowCustomer response builder, as defined in Example 9-52:

- Find Dialog search by pet requires limiting the search to customers with a specific pet type.

  The initializeCustomerQuery() method limits the search by defining the where clause and the join clause on the XPET table to return only those users who have the requested pet type. The existing methods handle the result.

- Customer Editor requires showing the pets. Therefore, the ShowCustomer reply builder must be extended to add the <Pets> tag to the BOD message.

  The populateSearchResult() method returns the list of pets found of the ExtPetAccessBean type.

  The buildCustomerElementDetailInfo() method extends the existing method to add the <Pets> section using the createPetsElement() and the createPetElement() methods, with data found by the populateSearchResult() method. This buildCustomerElementDetailInfo() method is called to fetch the complete customer information to be displayed in the customer editor.

  Another method, buildCustomerElementBaseInfo(), is called to fetch the base customer information to be displayed in the Find Dialog table that lists the found customers. (Our scenario did not require displaying pet information in that table.)

Example 9-52 ExtShowCustomer.java: Response builder

```
package com.ext.commerce.telesales.messaging.bodreply;

/**
 * Extended ShowCustomer reply BOD to add search by Pet Type
 */
public class ExtShowCustomer extends ShowCustomer {

    /**
     * The property storing the current pet result
     */
    public final static String EXT_SEARCH_RESULT_PROPERTY_NAME_PETS =
"EXT.WC.CUSTOMER.PETS";

/*
     * Add the search condition to search by pet type.
     */
    protected WhereClauseCondition initializeCustomerQuery(
                                        SearchCriteria aSearchCriteria) {

        SelectExpression selectExpressionPetType = aSearchCriteria
                .getSelectExpression(ExtBODConstants.BOD_TAG_PET_TYPE);
        // create OTB clause
        WhereClauseCondition whereClause =
super.initializeCustomerQuery(aSearchCriteria);

        // add the pet type to the search criteria
        if (selectExpressionPetType != null) {
            int petTypeCriteriaOpDefault =
```

```
                WhereClauseSearchCondition.SEARCHTYPE_CASESENSITIVE_EXACTMATCH;
        Integer petTypeCriteriaOp =
                convertStringToInt(getSearchTypeValue(selectExpressionPetType
                .getSearchType()));
        String petTypeCriteria = selectExpressionPetType.getValue();

        // search the pet table
        WhereClauseSearchCondition petCondition = new WhereClauseSearchCondition(
                new TableField("XPET", "TYPE"),
                (petTypeCriteriaOp != null) ? petTypeCriteriaOp.intValue() :
                petTypeCriteriaOpDefault, petTypeCriteria);

        whereClause.appendANDCondition(petCondition);

        // join the pet and users table by usersId
        WhereClauseJoinCondition petJoinCondition =
          new WhereClauseJoinCondition("USERS.USERS_ID = XPET.USERS_ID");

        whereClause.appendANDCondition(petJoinCondition);
    }
    return whereClause;
}

/*
 * Extended to return an arrayList of ExtPetAccessBeans with the found pets when
 * aboolGetDetails=true and is stored in the
  * "EXT_SEARCH_RESULT_PROPERTY_NAME_PETS" property
 */
protected CustomerSearchResultBean populateSearchResult(
        String astrUserId, CustomerSearchResultBean abnSearchResult,
        boolean aboolGetDetails)
              throws ECException {

    CustomerSearchResultBean newAbnSearchResult = null;
    newAbnSearchResult = super.populateSearchResult(astrUserId, abnSearchResult,
            aboolGetDetails);

    if (aboolGetDetails) {
        try {
            Enumeration pets = new ExtPetAccessBean().findByUserId(
                    new Long(astrUserId));
            ArrayList petArrayList = new ArrayList();
            while (pets.hasMoreElements()) {
                ExtPetAccessBean elem = (ExtPetAccessBean) pets.nextElement();
                petArrayList.add(elem);
```

```
                }
                newAbnSearchResult.setData(
                       EXT_SEARCH_RESULT_PROPERTY_NAME_PETS, petArrayList);

            } catch (Exception e) {
                // TODO: handle exception
                System.out.println(e);
            }
        }
        return newAbnSearchResult;
    }

    /**
     * Builds the Customer element which cotains the detailed information. This
     * information can be used to display data in an editor. This method extends the
      * BOD with additional Pets element by calling the following methods to create
      * the child elements:
     * <UL>
     * <LI>the createPetsElement method to create the Pets element
     * </LI>
     * </UL>
     *    <Customer>
     *       <CustomerParty/>
     *       <CommerceArea/>
     *       <CustomerDemographics/>
     *       <AssignedTeam/>
     *       <AssignedRepresentative/>
     *       <Pets/>
     *    </Customer>
      */
    protected Element buildCustomerElementDetailInfo(
                       CustomerSearchResultBean abnCustomerSearchResult,
                       Element aParentElement)
                          throws ECException {

        Element element = super.buildCustomerElementDetailInfo(
                             abnCustomerSearchResult, aParentElement);
         createPetsElement(abnCustomerSearchResult, element);
        return element;
    }

/**
     * Builds the Pets element. This method calls the
     * createPetElement(ExtPetAccessBean, Element) to create the Pet
     * element for each Pet found in the customer's Pet list.
```

```
 */
protected Element createPetsElement(
                      CustomerSearchResultBean abnCustomerSearchResult,
                      Element aParentElement)
                            throws ECException {

    // create the tops Pets element
    Element PetsElement = createWCDocumentElement(
                               aParentElement, ExtBODConstants.BOD_TAG_PETS);
    // When there is no Pets, just return
    if (null ==
           abnCustomerSearchResult.getData(EXT_SEARCH_RESULT_PROPERTY_NAME_PETS))
     {
        return PetsElement;
    }

    // create pet element details for each pet in list
    Iterator customerPets = ((ArrayList) abnCustomerSearchResult
            .getData(EXT_SEARCH_RESULT_PROPERTY_NAME_PETS)).iterator();

    while (customerPets.hasNext()) {
        ExtPetAccessBean bnCurrentPet = (ExtPetAccessBean) customerPets.next();
        createPetElement(bnCurrentPet, PetsElement);
    }

    return PetsElement;
}

/**
 * Builds the Pet element. This method uses the values found in
 * abnPet to create the Pet element. The new element is appended
 * to the specified parent element.
 *
 *   <Pet primary="true" type="SB">
 *      <PetId/>   <!-- abnPet.getPetId() -->
 *      <Name/> <!-- abnPet.getName() -->
 *      <Type/> <!-- abnPet.getType() -->
 *   </Pet>
 */
public Element createPetElement(ExtPetAccessBean abnPet, Element aParentElement)
        throws ECException {

    Element petElement = null;
    if (abnPet != null) {
        try {
```

```
                Long petId = abnPet.getPetId();
                String petName = abnPet.getName();
                String petType = abnPet.getType();

                petElement = createWCDocumentElement(
                       aParentElement, ExtBODConstants.BOD_TAG_PET);
                createWCDocumentElement(
                       petElement, ExtBODConstants.BOD_TAG_PET_ID, petId);

                if (petName != null && petName.length() > 0) {
                    createWCDocumentElement(
                          petElement, ExtBODConstants.BOD_TAG_PET_NAME, petName);
                }

                if (petType != null && petType.length() > 0) {
                    createWCDocumentElement(
                          petElement, ExtBODConstants.BOD_TAG_PET_TYPE, petType);
                }
            } catch (Exception e) {
                // TODO: handle exception
            }
        }
        return petElement;
    }
}
```

Response builder registration

The response builder must be registered for it to be recognized.

Note: For instructions about registering new response builders, refer to the information center topic “Modifying an existing Business Object Document reply message”, which is available on the Web at:

http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.telesales.developer.doc/tasks/ttrmodifybodreply.htm

To register the response builder, perform the following tasks:

1. Open the TelesalesRegistry-ext.xml file in the <Toolkit>/xml/messaging folder, the creation of which is described in “Response builder registration” on page 219. It was created for the ExtConfirmCustomer response builder.

2. Add to it the text shown in Example 9-53 in order to register the new ExtShowCustomer response builder.

Example 9-53 Registering the new ExtShowCustomer response builder

```
<WCTBodResponseBuilderRegistry>
   <Noun name="Customer">
      <Verb name="Get">
         <ClassName>
       com.ext.commerce.telesales.messaging.bodreply.ExtShowCustomer
         </ClassName>
      </Verb>
   </Noun>
</WCTBodResponseBuilderRegistry>
```

9.7.3 Implementing the integration code on the client side

The server response has been sent back to the client in the form of a BOD message. The client must be modified to react to the response from the server containing the customer list and the added pet list.

Request handler

Request handlers are responsible for creating BOD messages with the information to be sent to the WebSphere Commerce server. They also handle the response BODs that are sent as the response to the request from the server.

Clicking the **Find Customer** button in the Find dialog box results in a call to the GetCustomerRequest request handler that generates the request to fetch the customer information. The returned response is ShowCustomer BOD containing the found customers. The Find dialog box receives the default BOD, ShowCustomer, and displays the found customers as a list in the GUI that looks like a table.

The modified ShowCustomer BOD containing the customer's pet list, as shown in Example 9-51 on page 234, is received when the customer editor requests the detailed customer information.

Request handler implementation: Unmarshalling the response

Extend GetCustomerRequest to process the received pet list returned in the ShowCustomer response BOD.

To handle the received pet list, implement the methods in the ExtSGetCustomerRequest class, as shown in Example 9-54:

- The unmarshallCustomer method is extended to process the response BOD pet information using the unmarshallCustomerPets method.
- The unmarshallCustomerPets method reads the list of pets, creates the ExtPetList ModelObjectList, and adds it to the customer Model Object to be used by the client.

Example 9-54 ExtGetCustomerRequest.java implementation

```
package com.ext.commerce.telesales.core.impl.request;

/**
 * ExtGetCustomerRequest is the request handler implementation that handles the get
 * customer service request. It is extended to handle unmarshalling retrieved
customer pets.
 */
public class ExtGetCustomerRequest extends GetCustomerRequest {

    /**
     *  add the following tag to the customer element as the last element
     *        <Pets> <!-- unmarshalled by unmarshallCustomerPets() -->
     *          .
     *          .
     *        </Pets>
*/
    protected void unmarshallCustomer(Customer customer, Element customerElement) {

        super.unmarshallCustomer(customer, customerElement);
        unmarshallCustomerPets(customer, getChildElement(customerElement,
                ExtRequestConstants.BOD_TAG_PETS));
    }

    /**
     * Unmarshalls the Pets element. The following sample shows the structure of the
     * Pets element and how it is unmarshalled.
     *     <Pets>
     *    <Pet>
     *        <PetId>21</PetId>
     *        <Name>F</Name>
     *        <Type>51000</Type>
```

```
     *     </Pet>
     *     </Pets>
     *
     * @param customer The customer data bean to populate.
     * @param customerDemographicsElement The element that is unmarshalled.
     */
    protected void unmarshallCustomerPets(Customer customer, Element petsElement) {

        Customer cust = (Customer) customer;
        ExtPetList extPetListModel = new ExtPetList();
        if (petsElement != null) {

            ArrayList extPetArrayList = getChildElements(petsElement,
                    ExtRequestConstants.BOD_TAG_PET);

            if (extPetArrayList.size() > 0) {
                for (int i = 0; i < extPetArrayList.size(); i++) {
                    Element petElement = (Element) extPetArrayList.get(i);
                    if (petElement != null) {
                        ExtPet extPet = (ExtPet) TelesalesModelObjectFactory
                          .createModelObject(ExtCustomerContants.MODEL_OBJECT_PET);
                        extPet.setName(getChildElementValue(petElement,
                                ExtRequestConstants.BOD_TAG_PET_NAME));
                        extPet.setId(getChildElementValue(petElement,
                                ExtRequestConstants.BOD_TAG_PET_ID));
                        extPet.setType(getChildElementValue(petElement,
                                ExtRequestConstants.BOD_TAG_PET_TYPE));
                        extPet.setMarking(ExtPet.MARKED_EXISTNG);
                        extPetListModel.addPet(extPet);
                    }
                }
            }
            customer.setData(
                 ExtCustomerContants.PROP_CUSTOMER_PETS, extPetListModel);
        }
    }
}
```

Redefine the service requests (Example 9-55), specifying which request handler is to be called to run the service request task. The service request, com.ibm.commerce.telesales.findCustomer, which is called by the Find Customer dialog box is processed by the ExtGetCustomerRequest.

Example 9-55 Defining the redefined findCustomer service request

```
<extension
   point="com.ibm.commerce.telesales.core.serviceRequests">
   <serviceRequest label="Find Customer"
      requestHandlerClass=
               "com.ext.commerce.telesales.core.impl.request.ExtGetCustomerRequest"
      id="com.ibm.commerce.telesales.findCustomer"
      commServiceId="com.ibm.commerce.telesales.services.TsCommunication">
   </serviceRequest>
```

9.8 Loading the customizations into WebSphere Commerce Developer

All the code that has been provided for this customization is available for download in CustomerPet_SalesCenterCode.zip, the details about which are available in Appendix A, “Additional material” on page 375.

This section outlines how to load the sample code into your WebSphere Commerce Developer (into both the toolkits, the WebSphere Commerce toolkit and the IBM Sales Center toolkit).

9.8.1 Installing the WebSphere Commerce Developer 6.0.0.1 Fix Pack

The prerequisite to run and test the customized code on your development environment is to install the WebSphere Commerce Developer 6.0.0.1 Fix Pack.

To install this fix pack, download and follow the instructions provided in the WebSphere Commerce Developer 6.0.0.1 Update Guide, which can be accessed from the Technote *WebSphere Commerce 6.0.0.1 Fix Pack*, which is available on the Web at:

http://www-1.ibm.com/support/docview.wss?uid=swg24013056

To ensure that the fix pack is installed in your WebSphere Commerce toolkit, run the following command from a command prompt:

<*WCDE_installdir*>/bin/versionInfo.bat

Note: This command is only available for WebSphere Commerce V6.0 with Fix Pack 1.

Running this command produces an output that includes the product information. You must, for example, see the output of the versionInfo command, as shown in Example 9-56.

Example 9-56 Output of the versionInfo command

```
Installed Product
------------------------
Name IBM WebSphere Commerce
Version 6.0.0.1
ID wc.toolkit.be
Build Level xxxxxxxxx
Build Date xx/xx/xx"
```

9.8.2 Creating the XPET table on the WebSphere Commerce toolkit

This section shows you how to create the XPET table in the IBM Cloudscape™ database used by your WebSphere Commerce toolkit. If your WebSphere Commerce development environment is set to use DB2 or Oracle®, you can perform similar steps that are suitable for your database in order to create the table, the constraints, and the key values, as required. Note that the SQL statements are also provided within the CustomerPet_SalesCenterCode.zip in the CustomerPetExtension.sql file (refer to Appendix A, “Additional material” on page 375).

To create the XPET table on the WebSphere Commerce toolkit, perform the following tasks:

1. Ensure that the test environment is started.

   Note: For intsructions, refer to the information center topic “Starting and stopping WebSphere Commerce Test Server within the WebSphere Commerce Developer”, which is available on the Web at:

   `http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.developer.doc/tasks/tsrwcsstudio.htm`

2. Open a browser and type the following URL:

   `http://localhost/webapp/wcs/admin/servlet/db.jsp`

3. In the input box, enter the SQL statement shown in Example 9-57.

Example 9-57 Entering the SQL statement

```
CREATE TABLE XPET (PET_ID BIGINT NOT NULL, USERS_ID BIGINT NOT NULL,
NAME VARCHAR(32) NOT NULL, TYPE VARCHAR(32) NOT NULL, OPTCOUNTER
SMALLINT NOT NULL);
ALTER TABLE XPET ADD CONSTRAINT xpet_p1 PRIMARY KEY (PET_ID);
ALTER TABLE XPET ADD CONSTRAINT xpet_f1 FOREIGN KEY (users_id)
REFERENCES users  (users_id) ON DELETE CASCADE;
INSERT INTO KEYS (TABLENAME, COLUMNNAME, COUNTER , KEYS_ID) values
('xpet', 'pet_id', 0, 1);
```

4. Click **Submit Query**. You must see a message that states that the statement resulted in 0 updates. For the last SQL statement, you must see a message that states that the statement resulted in 1 update.

9.8.3 Loading the access control policies

This section shows you how to load the access control policies for your new resources into your WebSphere Commerce development environment. Load the ACPs to your WebSphere Commerce development environment database by performing the following tasks for the Cloudscape database:

1. Extract the XML file ExtPetACPolicy.xml from CustomerPet_SalesCenterCode.zip to <*WCDE_installdir*>\xml\policies\xml.
2. Load the access control policies by performing the following tasks:
   a. Ensure that the WebSphere Commerce test environment is stopped.
   b. At a command prompt, navigate to the directory <*WCDE_installdir*>\bin.
   c. Issue the **acpload** command, which has the following form for Cloudscape:

      `acpload` *`inputXMLFile`*

      Here, *inputXMLFile* is the XML file containing the ACP specification. In this case, specify `ExtPetACPolicy.xml`. The following is an example of the command with variables to load the ACP to Cloudscape:

      ```
      acpload ExtPetACPolicy.xml
      ```

3. Check for errors in the log files. Note that errors might not appear on the command line.
   – Check the acpload.log and messages.txt files in the <*WCDE_installdir*>/logs directory
   – Any error files generated in the <*WCDE_installdir*>/xml/policies/xml directory

Note: To run the ACP load on other databases, refer to the information center topic “acpload utility”, which is available on the Web at:

http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.admin.doc/refs/raxacpload.htm

9.8.4 Mapping a modified Business Object Document message

This section shows you how to include a modified BOD that is passing additional data between the client and the server into your WebSphere Commerce development environment.

To include a modified BOD message for the new command in your WebSphere Commerce development environment, perform the following tasks:

1. Extract the following XML files from CustomerPet_SalesCenterCode.zip to <WCDE_installdir\>xml\messaging:
   - TelesalesRegistry-ext.xml
   - webservice_SOABOD_template.extension.xml
   - ExtSyncCustomerBODMapping.xml

   Note: All the custom response builders for this IBM Redbook are registered in the TelesalesRegistry-ext.xml file. When installing more than one customization sample, make sure that you manually merge this file in order to avoid overwriting the registration data.

   It is recommended that you refer to the following topics in the information center:

   - “Mapping a new Business Object Document message to a new command”

     http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.telesales.developer.doc/tasks/ttrmapbod.htm

   - “WebSphere Commerce integration”

     http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.telesales.developer.doc/concepts/ctrcommerceintegration.htm

   - “Register the WebSphere Commerce Server Extension”

     http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.telesales.developer.doc/tutorial/ttravaildate11.htm

   It is possible to edit the TelesalesRegistry.xml file to point to your new ExtShowCustomer and ExtConfirmCustomer classes. However, it is not recommended because the TelesalesRegistry.xml file might be altered by a

fix pack installation or during migration. Instead, in our case, we created a new custom registry file, TelesalesRegistry-ext.xml, which overrides or merges with the TelesalesRegistry.xml file. Entries in the custom registry are combined with the entries in the base registry file. In the event that the same verb-noun pair is registered in both the registry files, the custom entry is the one that is used.

WebSphere Commerce uses the !ENTITY declarations to include different files for each of the document command mappings, and has extended the message mapper declaration to include a file for extensions called webservice_SOABOD_template.extension.xml. You can update this file to include your own extensions and any additional or changed mappings that you want to define.

2. Navigate to <*WCDE_installdir*>/xml/config.
3. Open the **wc-server.xml** file for editing.
4. Locate the text shown in Example 9-58.

Example 9-58 Locating the text

```
<component
compClassName="com.ibm.commerce.telesales.configuration.TelesalesRegist
ryComponentConfiguration" enable="true" name="Telesales Response
Builder Registry Configuration"> <property
baseRegistryFilePath="messaging" customRegistryFileName=""
customRegistryFilePath="" display="false"
enableBaseRegistryOverride="false"/> </component>
```

5. Modify the customRegistryFileName, customRegistryFilePath, and enableBaseRegistryOverride values, as shown in Example 9-59.

Example 9-59 Modifying the customRegistryFileName, customRegistryFilePath, and enableBaseRegistryOverride values

```
<component
compClassName="com.ibm.commerce.telesales.configuration.TelesalesRegist
ryComponentConfiguration" enable="true" name="Telesales Response
Builder Registry Configuration"> <property
baseRegistryFileName="TelesalesRegistry.xml"
```

```
baseRegistryFilePath="messaging"
customRegistryFileName="TelesalesRegistry-ext.xml"
customRegistryFilePath="messaging" display="false"
enableBaseRegistryOverride="true"/> </component>
```

6. Start, or alternatively, restart the WebSphere Commerce Test server if it is running. Note that a server restart is required to pick up the changes made to wc-server.xml.

9.8.5 Importing the EJB JAR file

This section shows you how to import and deploy the new EJB into your WebSphere Commerce development environment. To import the new ExtPet EJB into your WebSphere Commerce development workspace and deploy to the WebSphere Commerce test server, perform the following tasks:

1. In the file system, navigate to CustomerPet_SalesCenterCode.zip and extract it to a temporary directory.
2. In the J2EE perspective Project Navigator view, navigate to **EJB Projects** → **WebSphereCommerceServerExtensionsData** project.
3. Right-click the Project and select **Import** → **EJB JAR File**. Click **Next**.
4. In the EJB Jar file field, enter or browse to the location of the temporary directory, WebSphereCommerceServerExtensionsData.jar.
5. Ensure that WebSphereCommerceServerExtensionsData is entered in the EJB project field. Back up your customizations if you have created them. Select **Overwrite existing resources without warning**. Click **Finish**. If your Rational Application Developer is set to automatic compilation, wait for the building workspace to complete. There should be no problems.
6. In the Mapping Editor, open the Map.mapxmi file to verify the XPET table mapping. To verify that this entity bean uses optimistic locking and has a field called optCounter of the type short, which has marked the OptimisticPredicate property as true in the Properties view, select the **optCounter** field in the Outline view.

   Refer to the information center topic “Creating new entity beans”, which is available on the Web at:

   http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.developer.doc/tasks/tdecreateentitybean.htm
7. In the J2EE perspective Project Navigator view, navigate to **EJB Projects** → **WebSphereCommerceServerExtensionsData** project.
8. Right-click the SRC directory and select **Deploy**. Wait for the deployment of the selected modules to complete.

9.8.6 Importing the commands and the new bodreply messages

To import the new commands (ExtPetUpdate, ExtPetDelete) and the new bodreply messages (ExtConfirmCustomer, ExtShowCustomer) into your WebSphere Commerce development workspace, perform the following tasks:

1. In the file system, navigate to CustomerPet_SalesCenterCode.zip and extract it to a temporary directory.
2. In the J2EE perspective Project Navigator view, navigate to **Other Projects → WebSphereCommerceServerExtensionsLogic → src directory**.
3. Right-click **src directory** and select **Import → Zip File**. Click **Next**.
4. In the From zip file field, enter or browse to the location of WebSphereCommerceServerExtensionsLogic.jar.
5. Back up your customizations if you have created them. Verify the files to be imported. All the files are selected by default. In the right pane, ensure that the resources that begin with a period (.) are not selected, for example, deselect the following files in the right pane to ensure that these are not imported:

   – .classpath
   – .classpath.template
   – .project
   – .serverPreference

   Click **Finish**. If your Rational Application Developer is set to automatic compilation, wait for the building workspace to complete. There should be no problems. The resulting project must be similar to that shown in Figure 9-14.

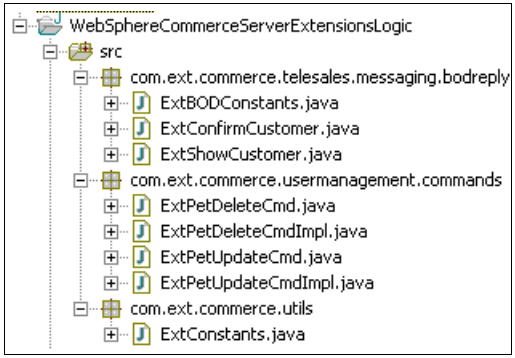


Figure 9-14 Imported new commands and new bodreply messages

9.8.7 Loading the client code into the IBM Sales Center toolkit

To load the client code into your IBM Sales Center development environment, perform the following steps:

1. In the file system, navigate to CustomerPet_SalesCenterCode.zip and extract it to a temporary directory.
2. In the plug-in development perspective Package Explorer view, right-click the pop-up menu, and select **Import** → **Existing Project into Workspace**. Click **Next**.
3. In Project contents field, enter or browse to the location of the plug-in, com.ext.commerce.telesales.customer directory. The Project name field is automatically filled in as com.ext.commerce.telesales.customer.

   Click **Finish**. If your Rational Application Developer is set to automatic compilation, wait for the building workspace to complete. Some error messages relating to org.eclipse.* plug-ins might appear. However, you can ignore them.

9.9 Testing the customized code

After importing the customized code into your WebSphere Commerce Developer (into both the WebSphere Commerce development environment and the IBM Sales Center development environment), test the customizations.

Notes:

- Our assumptions for testing the customized code are as follows:
  - No validation has been implemented in this sample. Assume that valid data is always provided, and that a unique name will be provided for a pet.
  - The code provided is a sample that might not be fully functional, and is provided on an “as is” basis for demonstration purposes only.
- When you click **Update** to send the customer pet information to the server, if you see the error message “`_ERR_MESSAGING_BOD_MESSAGE_MAPPER`”, check whether you have installed the WebSphere Commerce Developer 6.0.0.1 Fix Pack.

To test the customizations, perform the following tasks:

1. Start the WebSphere Commerce Test Server in your WebSphere Commerce development environment. At the command prompt, for example, navigate to <*WCDE_installdir*>\bin and run the script startwcserver.bat. The server is started when you see the message "`Server server1 open for e-business`".

   Note: Running the startwcserver.bat script is equivalent to starting the WebSphere Commerce Test Server using the Rational Application Developer graphical interface, but without the memory overhead required by that graphical interface. For now, you do not have to use the graphical interface for the WebSphere Commerce server side. If you launch the Rational Application Developer graphical interface, you will see that the WebSphere Commerce Test Server is already running.

   In order to log in from the IBM Sales Center client, which is a part of the IBM Sales Center development environment, to the server, you must have the WebSphere Commerce server running. This can either be the WebSphere Commerce Test Server within the WebSphere Commerce toolkit running on the same or another system as IBM Sales Center toolkit, or it can also be a runtime WebSphere Commerce server, running on the same or another system as WebSphere Commerce Developer.

   For the purpose of this book, we have only demonstrated importing the customized code into the WebSphere Commerce development environment. To create and deploy the customized code to the runtime server environment, refer to the information center tutorial titled "Deploying your customization to the WebSphere Commerce server", which is available on the Web at:

   `http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.giftcenter.refapp.doc/tutorial/tgcibmgiftcentercustomization44.htm`

2. Start the IBM Sales Center development environment by selecting **Start** → **IBM WebSphere Commerce Developer** → **IBM Sales Center development environment**.
3. From the Run menu, click **Run**.
4. In the Configurations list, select **Sales Center**, and click **Run**.
5. When the IBM Sales Center opens, click **Open** in the left pane and select **IBM Sales Center - Order Management** to open the Order Management application main window.

6. From the main menu, select **File** → **Logon** to display the login panel. Click the **Connectivity** button to display the Preferences panel. Under Hypertext Transport Protocol Secure (HTTPS) settings, for example, for Server, enter `localhost` and for Hypertext Transfer Protocol Secure (HTTPS) port enter `443`, and click **OK**.
7. In the login dialog box, enter the user name and the password of a user with site administrator privileges, for example, `wcsadmin`. Note that by default, the password for the user name wcsadmin on WebSphere Commerce toolkit is wcsadmin. You are required to change it the first time you log in.
8. From the main menu, select **Store** → **Select**. Click the **Find** button to search for all the stores.
9. From the Search Result list, select a store, for example, **ConsumerDirect**, and click **OK**.
10. Create a customer profile. Click **Create** to create a customer profile.

    For more information, launch the IBM Sales Center Information Center by selecting **Help** → **Help Contents** in the IBM Sales Center client, and search for the topic “Creating a customer profile”. Alternatively, you can refer to the online instructions in the topic “Creating a customer profile”, which is available in the information center at:

    `http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.telesales.tsr.doc/tasks/ttrcreateb2bcust.htm`
11. Open the Pets page and click **Add**.
12. Enter the information pertaining to a pet such as the name of the pet, and select the pet type from the drop-down menu. Click **OK**.

13. Click **Update** to send the updates to the server (Figure 9-15).

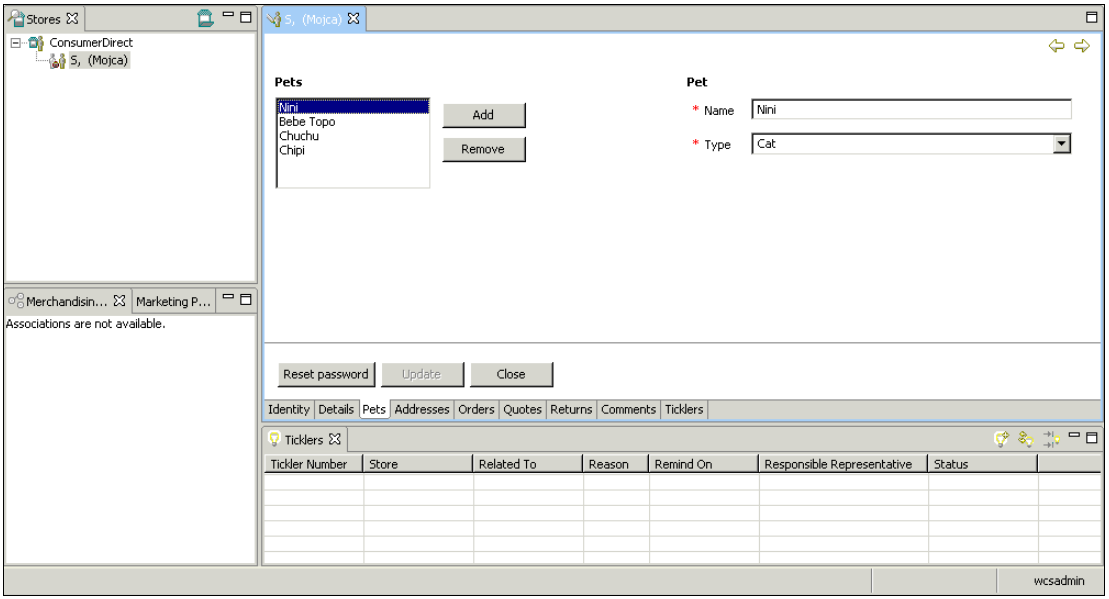


Figure 9-15 Testing pet customization

14. In the pet page, select the pet to update. In the fields, change the information pertaining to the pet. Click **Update** to send the updates to the server and to observe the change of the local pet list on the client side.
15. In the pet page, select the pet to be removed. Click **Remove** to observe the update of the local list on the client side. Click **Update** to send the updates to the server, or **Close** to choose whether you want to save or discard the changes.
16. Close the customer profile and clear the customer from view.
17. Right-click the store in the Stores view, and select **Find** → **Customer**.

18. From the Search Type drop-down menu, select **Pet Type** and select the pet type you have created for your customer in the previous steps. Click **Find** to open the customer profile, if only one customer exists with this type of the pet, or to get the list of customers with the selected pet type if multiple customers exist with the same type of the pet (Figure 9-16).

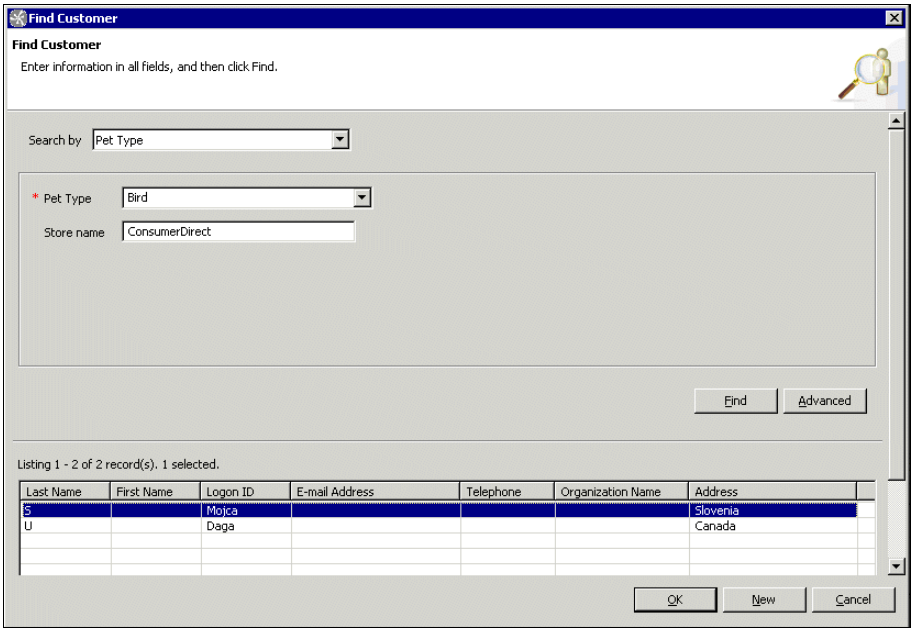


Figure 9-16 Testing Find Customer

10

Role-based customizations

In IBM Sales Center, the following user interface (UI) elements can be hidden and displayed based on a user's role:

- Views and editors
- Perspectives
- Preference and property pages
- New project wizard
- Menus and toolbars

This chapter demonstrates how to perform the following tasks:

- Create a new role that can access IBM Sales Center
- Display different menu items based on roles

The bulk of what is explained in this chapter takes place within the WebSphere Commerce and IBM Sales Center development environments. Only section 10.3.5, "Deploying to production for both the server and the client" on page 283 references the production environment.

The prerequisites for this chapter are:

- The WebSphere Commerce server development environment is installed
- The IBM Sales Center development environment is installed
- There is at least one published starter store

10.1 Duplicating an existing role

This section describes how to create a new customer service representative (CSR) role called NewCSR. Users assigned to this role initially have access to the same activities as users with the CSR role. However, the latter part of this chapter shows you how to modify the user authority.

10.1.1 Creating a new role and a user in the Organization Administration console

Perform the tasks described in this section in the WebSphere Commerce development environment.

To create a new role, perform the following tasks:

1. Open WebSphere Commerce Developer and start the WebSphere Commerce Test Server.
2. Open the WebSphere Commerce Organization Administration console in an Internet Explorer browser. By default, the URL for this tool is https://*yourHostname*:8004/orgadminconsole.
3. Log in as a user with site administrator authority.
4. Select **Access Management** → **Roles**.
5. Click **New**.
6. In the Name field, enter `NewCSR`, and click **OK**.

To create a new user under this role, perform the following tasks:

1. Select **Access Management** → **Create user**.
2. Provide the necessary details in the required fields, ensuring that the account policy is `Administrators`.
3. Click **OK**.
4. Select your new user and click **Roles**.
5. Select the appropriate organization, which will most likely be **Root Organization**.
6. From the Role list, select **NewCSR**.
7. Click **Add.**
8. Click **OK**.

9. If you do not already have a user with a CSR role, create one now by performing steps 1 - 7. You require this user later in this chapter to compare the menu items that are displayed for the CSR role and the NewCSR role.

Determine the role_id value of your new role by performing the following tasks:

1. On the machine that runs the WebSphere Commerce development environment, open the following URL:

   `http://localhost/webapp/wcs/admin/servlet/db.jsp`

2. Enter the following SQL statement and click **Submit Query**:

   `select * from role where name='NewCSR';`

3. Record the value in the role_id column for use later in this chapter.

10.1.2 Revising and loading the access control policies

By default, the new role is not associated with any access control policies. In this example, you duplicate the same access control policies as a CSR, modify two access control policy files, and add the new role whenever the CSR role is mentioned.

defaultAccessControlPolicies

Modify defaultAccessControlPolicies.xml by performing the following tasks:

1. Navigate to the *WCDE_installdir*\xml\policies\xml directory.
2. Make a copy of the defaultAccessControlPolicies.xml file called defaultAccessControlPolicies_NewCSR.xml.
3. Edit the defaultAccessControlPolicies_NewCSR.xml file so that any stanza that contains CSR is followed by the same stanza, but for the role NewCSR. Example 10-1 shows the added stanza in bold type.

Example 10-1 Adding a stanza for NewCSR for every CSR stanza

```
[...]
<RelationGroup
Name="CustomerOrderManagers->RegisteredOrganizationalEntity"
OwnerID="RootOrganization">
   <RelationCondition><![CDATA[
      <profile>
         <orListCondition>
            <openCondition name="RELATIONSHIP_CHAIN">
               <parameter name="ROLE" value="customer service
                  representative"/>
```

```
            <parameter name="RELATIONSHIP"
               value="RegisteredOrganizationalEntity"/>
         </openCondition>
         <openCondition name="RELATIONSHIP_CHAIN">
            <parameter name="ROLE" value="NewCSR"/>
            <parameter name="RELATIONSHIP"
               value="RegisteredOrganizationalEntity"/>
         </openCondition>
         <openCondition name="RELATIONSHIP_CHAIN">
[...]
```

4. Save the changes.

ACUserGroups_en_US.xml

Modify ACUserGroups_en_US.xml by performing the following tasks:

1. Navigate to the *WCDE_installdir*\xml\policies\xml directory.
2. Make a copy of the ACUserGroups_en_US.xml file called ACUserGroups_NewCSR_en_US.xml.
3. Edit the ACUserGroups_NewCSR_en_US.xml file so that any stanza that contains CSR is followed by the same stanza for NewCSR. (Multiple <simpleCondition> tags must be surrounded by <orListCondition> tags. If, by default, there exists only one <simpleCondition> tag, add a <orListCondition> tag around the <simpleCondition> tags). Example 10-2 shows the added stanza in bold type.

Example 10-2 Adding a stanza for NewCSR: Adding <orListCondition> tags when necessary

```
[...]
<UserGroup Name="CustomerServiceRepresentatives"
   OwnerID="RootOrganization" Description="Users with role of customer
   service representative" MemberGroupID="-3">
   <UserCondition><![CDATA[
      <profile>
         <orListCondition>
            <simpleCondition>
               <variable name="role"/>
               <operator name="="/>
               <value data="customer service representative"/>
            </simpleCondition>
            <simpleCondition>
               <variable name="role"/>
               <operator name="="/>
               <value data="NewCSR"/>
            </simpleCondition>
```

```
        </orListCondition>
      </profile>
   ]]></UserCondition>
</UserGroup>
[...]
```

4. Save your changes.

Loading the policies

Load the revised access control policies. The commands that are run in the following tasks assume that you are using a Cloudscape database. If you are using DB2, Oracle, or IBM i5/OS®, you may require more parameters.

Load the access control policies by performing the following tasks:

1. Stop the WebSphere Commerce Test Server. Because there can be only one connection to the Cloudscape database, the following scripts cannot access the database if the WebSphere Commerce Test Server is running.
2. From the command prompt, navigate to the *WCDE_installdir*\bin directory.
3. Run the following command:

   ```
   acpload defaultAccessControlPolicies_NewCSR.xml
   ```

4. Navigate to the *WCDE_installdir*\logs directory. Inspect the acpload.log and the messages.txt files in order to ensure that the access control policy has loaded successfully. (The messages.txt file may not exist if the load has completed successfully.) Also check if the policy files, defaultAccessControlPolicies_NewCSR_idres.xml and defaultAccessControlPolicies_NewCSR_xmltrans.xml, are created successfully in the *WCDE_installdir*\xml\policies\xml directory. These two files are created as part of a successful idresgen utility process. If any other error files are generated in this directory, this indicates that there is a problem. If there is a problem, fix the problem, and rerun the command specified in the previous step.
5. In the *WCDE_installdir*\bin directory, run the following command:

   ```
   acugload ACUserGroups_NewCSR_en_US.xml
   ```

6. Navigate to the *WCDE_installdir*\logs directory and check the acpload.log and the messages.txt files to ensure that the access control policy loaded successfully. (The messages.txt file may not exist if the load completed successfully.) Also check if the policy files ACUserGroups_NewCSR_idres.xml and ACUserGroups_NsewCSR_xmltrans.xml are created successfully in the *WCDE_installdir*\xml\policies\xml directory. These two files are created as

part of a successful idresgen utility process. If any other error files are generated in this directory, this indicates that there is a problem. If there is a problem, fix the problem and rerun the command specified in the previous step.

10.1.3 Extending the server code for ShowStore

This section describes how to extend the code on the WebSphere Commerce server side in order to allow the NewCSR role to have access to the ShowStore class. Without access to this class, users with the NewCSR role will not be able to see any stores from IBM Sales Center.

Installing APAR IY88078

APAR IY88078 allows you to add the new role, NewCSR, to the list of roles that can see stores from IBM Sales Center. Without this APAR, users with the role NewCSR will not be able to see the stores in IBM Sales Center, and will not be able to perform any functions other than the login function.

When you install this APAR over WebSphere Commerce Developer, specify `WCDE_installdir` when asked for the installation location of the product you want to update. Also ensure that the Rational Application Developer and the WebSphere Commerce Test Server are stopped before installation of any APARs. Contact WebSphere Commerce support to obtain this APAR.

Extending ShowStore

Write code to add the new role to the list of roles that can see the stores, by performing the following tasks:

1. In the WebSphere Commerce development environment, in the Project Explorer view, expand **Other Projects** → **WebSphereCommerceServerExtensionsLogic**.
2. Right-click the **src** directory and select **New** → **Package**.
3. In the Name field, enter `com.ext.commerce.telesales.messaging.bodreply` and click **Finish**.
4. Right-click **com.ext.commerce.telesales.messaging.bodreply** package and select **New** → **Class**.
   - In the Name field, enter `ExtendedShowStore`.
   - In the Superclass field, enter or browse to **com.ibm.commerce.telesales.messaging.bodreply.ShowStore**.
5. Click **Finish**.

Figure 10-1 shows the New Java Class window with the name and the superclass specified.

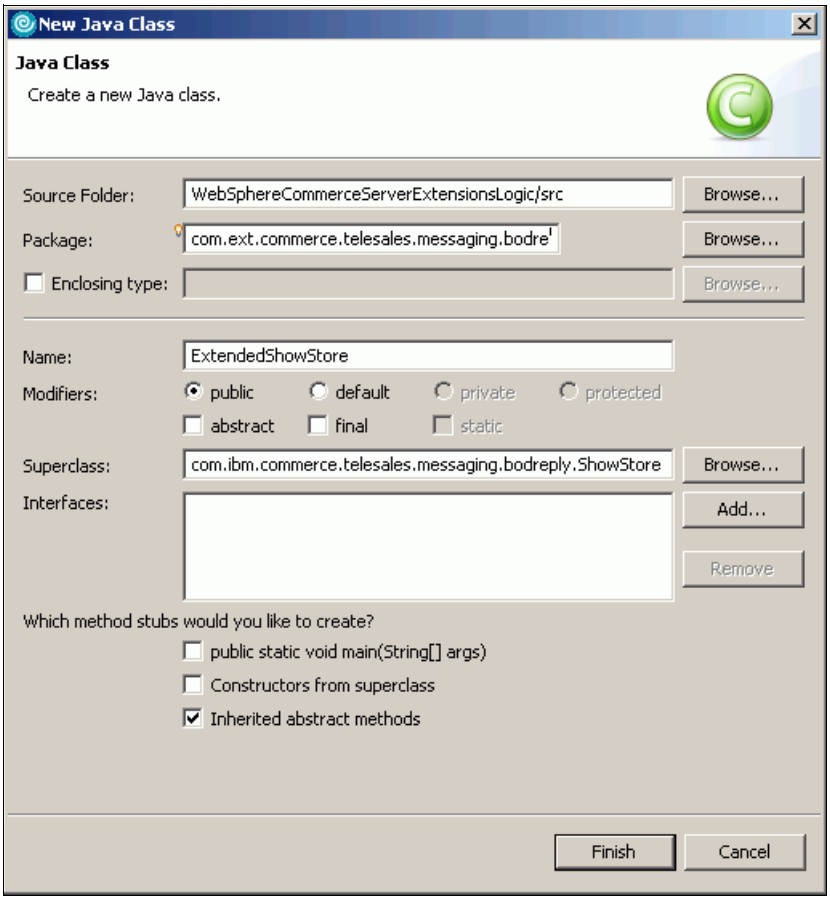


Figure 10-1 The New Java Class window

6. The ExtendedShowStore class opens for editing. Enter the code shown in Example 10-3 into the class, substituting the role_id you found in “Creating a new role and a user in the Organization Administration console” for *yourRoleId*.

Example 10-3 ExtendedShowStore class opens for editing

```
/**
* Override the createStoreLanguageBean()method of the ShowStore
* class to create the instance and add the supported roles before
* returning the instance to the base class.
* This method creates an instance of the StoreLanguageBean to do the
```

```
* store search.
*
* @return the <code>StoreLanguageBean</code> created.
*/
protected StoreLanguageBean createStoreLanguageBean() {
StoreLanguageBean slb = new StoreLanguageBean();
slb=super.createStoreLanguageBean();
slb.addSupportedRoleId("yourRoleId");
return slb;
}
```

7. Right-click anywhere in the class and select **Source** → **Organize Imports** to automatically add any required import statements.
8. Save your changes.

Updating the IBM Sales Center registry

The IBM Sales Center registry file called TelesalesRegistry.xml contains a mapping between the commands that must run based on different noun-verb combinations. Example 10-4 shows the mapping between the ElectronicCatalog-Get noun-verb pair and the com.ibm.commerce.telesales.messaging.bodreply.ShowElectronicCatalog class. Whenever the IBM Sales Center client tries to get an ElectronicCatalog object, the class com.ibm.commerce.telesales.messaging.bodreply.ShowElectronicCatalog will run.

Example 10-4 Noun-verb to Java class mapping in the IBM Sales Center registry

```
<Noun name="ElectronicCatalog">
   <Verb name="Get">
      <ClassName>
      com.ibm.commerce.telesales.messaging.bodreply.ShowElectronicCatal
      og</ClassName>
   </Verb>
</Noun>
```

In this example, to make the new com.ext.commerce.telesales.messaging.bodreply.ExtendedShowStore class to run instead of the com.ibm.commerce.telesales.messaging.bodreply.ShowStore class, create your own custom registry file and add a reference to the new file in the wc-server.xml file. You must not make changes directly to the TelesalesRegistry.xml file because the changes may be overwritten in a fix pack or migration.

To create a new custom registry file, perform the following tasks:

1. Navigate to the *WCDE_installdir*\xml\messaging folder.
2. Make a copy of the TelesalesRegistry.xml file called TelesalesRegistry-ext.xml. Edit the TelesalesRegistry-ext.xml file so that only the lines shown in Example 10-5 remain.

Example 10-5 Editing the TelesalesRegistry-ext.xml file

```
<WCTBodResponseBuilderRegistry>
      <Noun name="Store">
         <Verb name="Get">
            <ClassName>
               com.ibm.commerce.telesales.messaging.bodreply.ShowStore
            </ClassName>
         </Verb>
      </Noun>
</WCTBodResponseBuilderRegistry>
```

3. Change the following:

   ```
   <ClassName>com.ibm.commerce.telesales.messaging.bodreply.ShowStore
   </ClassName>
   ```

   to

   ```
   <ClassName>com.ext.telesales.messaging.bodreply.ExtendedShowStore
   </ClassName>
   ```

4. Save your changes.
5. Update the wc-server.xml file with a reference to your new registry file by performing the following tasks:
   a. Navigate to the *WCDE_installdir*\xml\config folder.
   b. Open the wc-server.xml file for editing.
   c. Find the text shown in Example 10-6.

Example 10-6 Finding text

```
<component
compClassName="com.ibm.commerce.telesales.configuration.TelesalesRegist
ryComponentConfiguration"
      enable="true" name="Telesales Response Builder Registry
      Configuration">
      <property
         baseRegistryFileName="TelesalesRegistry.xml"
```

```
        baseRegistryFilePath="messaging"
        customRegistryFileName=""
        customRegistryFilePath="" display="false"
        enableBaseRegistryOverride="false"/>
</component>
```

d. Modify the text by adding the text in **bold** as shown in Example 10-7.

Example 10-7 Modifying text

```
<component
compClassName="com.ibm.commerce.telesales.configuration.TelesalesRegist
ryComponentConfiguration"
      enable="true" name="Telesales Response Builder Registry
      Configuration">
      <property
         baseRegistryFileName="TelesalesRegistry.xml"
         baseRegistryFilePath="messaging"
         customRegistryFileName="TelesalesRegistry-ext.xml"
         customRegistryFilePath="messaging" display="false"
         enableBaseRegistryOverride="true"/>
</component>
```

e. Restart the WebSphere Commerce Test Server to pick up the change to wc-server.xml.

10.1.4 Extending the client side for the new role

Add code in the IBM Sales Center client side to recognize the new role and ensure that the role is provided with access to the proper activities.

Creating a new plug-in

When customizing IBM Sales Center, place all the extensions in one or more plug-ins that you have created specifically for your customizations. To create a new plug-in for the IBM Sales Center workspace, perform the following tasks:

1. Ensure that the IBM Sales Center development environment is open.
2. Ensure that the current perspective is the Plug-in Development perspective. If it is not, select it from the menu **Window** → **Open Perspective**.
3. Create a new plug-in project by clicking the **New** button in the toolbar and selecting **Plug-in Development** → **Plug-in Project**. This launches the Plug-in Project wizard.

   Alternately, from the Eclipse File menu, launch the Plug-in Project wizard in the Eclipse environment.

4. Enter the required information. The Project name for your plug-in must follow Java package conventions, for example:

   `com.ext.commerce.telesales.sample.extensions`

5. Ensure that you select the check box against **Create an OSGi bundle manifest for the plug-in**.
6. Click **Next**.
7. After validating the information, click **Finish**.

The plugin.xml file opens for editing.

Adding code to the new plug-in

This section provides information about how to add code to the new plug-in so that it defines what the NewCSR role can see and perform in IBM Sales Center.

To add code to the new plug-in, perform the following tasks:

1. The com.ibm.commerce.telesales.roles extention point defines which activitySets a user can access. You can see the CSR code in the com.ibm.commerce.telesales.ui.impl.roles plug-in, in the fragment.xml file.

   This step provides the NewCSR role with access to the same group of activitySets as the CSR role. (The latter part of this chapter describes how to change this definition to show the role-based context menu.)

   In the com.ext.commerce.telesales.sample.extensions plug-in's plugin.xml file, enter the code shown in Example 10-8 within the <plugin> tag.

Example 10-8 Entering code in the com.ext.commerce.telesales.sample.extensions plug-in's plugin.xml file

```
<!-- NewCSR-->
<extension point="com.ibm.commerce.telesales.roles">
   <role id="newCSR" roleId="NewCSR" label="NewCSRLabel">
      <activitySet
      id="com.ibm.commerce.telesales.activitySet.applications">
      </activitySet>
      <activitySet
      id="com.ibm.commerce.telesales.activitySet.baseStoreActions">
      </activitySet>
      <activitySet
      id="com.ibm.commerce.telesales.activitySet.baseCustomerActions">
      </activitySet>
      <activitySet
      id="com.ibm.commerce.telesales.activitySet.baseCommentsActions">
      </activitySet>
```

```
      <activitySet
      id="com.ibm.commerce.telesales.activitySet.baseOrderActions">
      </activitySet>
      <activitySet
      id="com.ibm.commerce.telesales.activitySet.baseQuoteActions">
      </activitySet>
      <activitySet
      id="com.ibm.commerce.telesales.activitySet.baseReturnActions">
      </activitySet>
      <activitySet
      id="com.ibm.commerce.telesales.activitySet.baseProductActions">
      </activitySet>
      <activitySet
      id="com.ibm.commerce.telesales.activitySet.baseTicklerActions">
      </activitySet>
      <activitySet
      id="com.ibm.commerce.telesales.activitySet.basePreferencePages">
      </activitySet>
   </role>
</extension>
```

10.1.5 Testing the new role

To ensure that the new role is working well, open IBM Sales Center and select a store:

1. In the IBM Sales Center development environment, select **Run** → **Sales Center**.
2. When the IBM Sales Center client opens, log in using the user you created in 10.1.1, “Creating a new role and a user in the Organization Administration console” on page 258.
3. Select **Store** → **Select**.
4. In the Find Store window, leave an asterisk in the Store name field and click **OK**.

   The stores that are returned must be the same list of stores that a CSR can see.

10.2 Chapter checkpoint

At this point in the chapter, a new user is assigned to a new role called NewCSR. This user has the same user authority as a CSR. This was accomplished by completing the following tasks:

- On the WebSphere Commerce server side, you performed the following tasks:
  a. Created a new role called NewCSR and assigned a user to this role on the WebSphere Commerce server side.
  b. Loaded access control policies to provide the NewCSR role with access to the same functions as the CSR role.
  c. Extended the ShowStore class to provide the NewCSR role with access to see the stores through IBM Sales Center.
- On the IBM Sales Center client side, you assigned activitySets to the NewCSR role. At this point, these activitySets are the same as the CSR. (However, this changes subsequently, and this process is described later in this chapter.)

10.3 Displaying the menu items based on the roles

The following sections describe the process involved in completing the customization by writing code to display different menu items depending on the user's role.

This section describes the process involved in using the samples provided with IBM Sales Center. The IBM Sales Center samples are used here only to demonstrate the role-based functions. The samples use the Eclipse extensions, but the customizations explained in this chapter use the IBM Sales Center framework.

10.3.1 Installing the samples

To import the samples into your workspace, perform the following tasks:

1. Open the IBM Sales Center development environment.
2. Select **File** → **Import**.
3. In the Select an Import Source box, click **Existing Project into Workspace**.
4. Click **Next**.
5. Click **Browse**.

6. Navigate to the WCDE_installdir\samples\SalesCenter\com.ibm.commerce.telesales.samples directory and click **OK**.
7. Click **Finish**.

In order to test whether the samples are imported properly, perform the following tasks:

1. Open the IBM Sales Center client within the development environment.
2. Log in to IBM Sales Center with a site administrator user name and password. The default user name and password is wcsadmin.
3. Select the **Customer** menu (Figure 10-2). If you see the following items in the Customer menu, it means that the samples are imported properly:
   - Sample Create Customer Dialog
   - Sample Edit Customer
   - Sample Create Customer

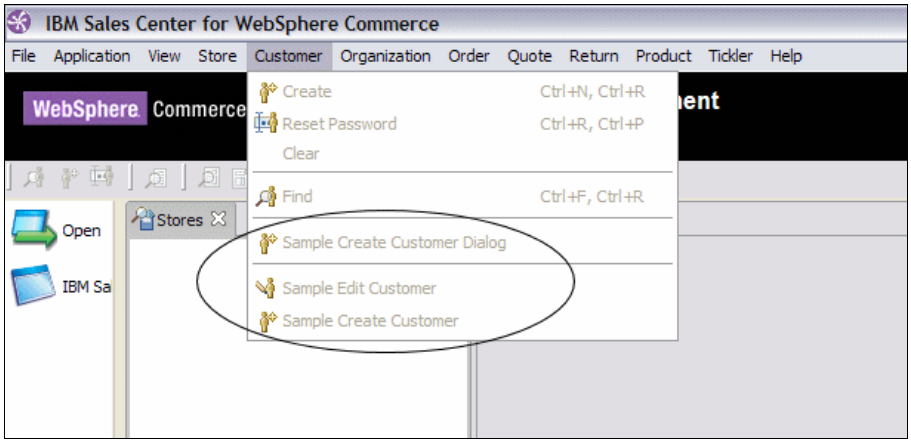


Figure 10-2 Samples are installed successfully

10.3.2 Extending the samples to display the context menu

The default samples installation creates the menu items under the Customer menu, but does not create the context menu that appears when you right-click the store in the Stores view. You must now extend the samples to display the context menu.

Examining the default context menu

To take a look at the default context menu before changing it, perform the following tasks:

1. Select **Store** → **Select**.
2. In the Find Store window, click **Find**.
3. A list of stores is displayed. Select the **ConsumerDirect** store and click **OK**. (If you have published just one store, the store is selected automatically when you click **Find**.)
4. Right-click **ConsumerDirect** store in the Stores view (Figure 10-3). The Sample Create Customer window does not appear in the context menu.

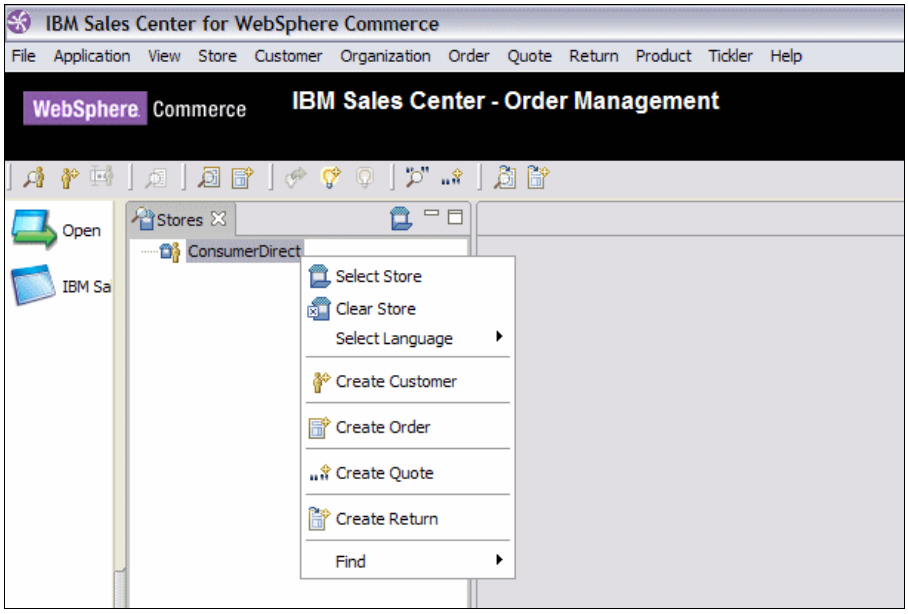


Figure 10-3 Sample plug-in does not create the context menu

Exporting the code from the sample plug-in

By default, the com.ibm.commerce.telesales.samples plug-in does not export its libraries for use by other plug-ins. To continue with this customization, export the libraries so that you can access them from your own custom plug-in. To export the libraries, perform the following tasks:

1. Open the com.ibm.commerce.telesales.samples plug-in for editing.
2. In the Runtime tab, in the Library exporting section, click **Add**.
3. Select all the libraries from the list and click **OK**.
4. Save your changes.

Figure 10-4 shows the result of adding all the libraries.

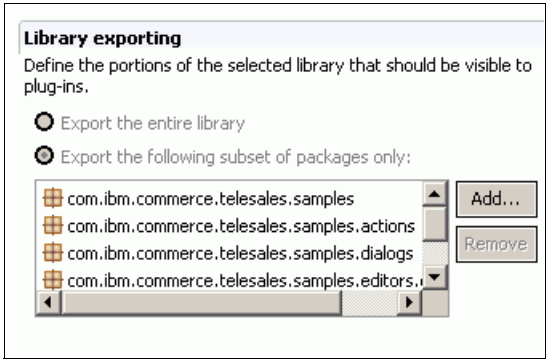


Figure 10-4 The Library exporting section of the Runtime tab

Modifying the context menu for the Sample Create Customer

To extend the samples in order to show the context menu for the Sample Create Customer, perform the following tasks:

1. Open the plug-in manifest file (plugin.xml) for the custom plug-in you created earlier, com.ext.commerce.telesales.sample.extension.
2. Add a dependency in the com.ibm.commerce.telesales.samples plug-in so that you can access its code, which is similar to an import statement in a Java class, by performing the following tasks:
   a. In the Dependency tab, click **Add**.
   b. Select **com.ibm.commerce.telesales.samples** and click **OK**.
3. Create an extension to org.eclipse.ui.popupMenus by adding the code in Example 10-9 to the plugin.xml file.

Example 10-9 Code for plugin.xml file

```
<extension point="org.eclipse.ui.popupMenus">
   <objectContribution
   objectClass="com.ibm.commerce.telesales.model.Store"
   adaptable="true">
      <action
      label="%CreateCustomerDialogWorkbenchActionDelegate.WorkbenchActi
      onDelegate.SampleCreateDialogCustomer"
         class="com.ibm.commerce.telesales.samples.actions.CreateCustom
         erWorkbenchActionDelegate"
         menubarPath="store.ext"
```

```
        id="com.ibm.commerce.telesales.samples.actions.CreateCustomerA
        ction">
      </action>
   </objectContribution>
</extension>
```

4. The label value %CreateCustomerDialogWorkbenchActionDelegate.WorkbenchActionDelegate.SampleCreateDialogCustomer is found in the resources.properties file of the com.ibm.commerce.telesales.samples plug-in. Add the code shown in Example 10-10 into your plugin.xml file so that it can find the properties file.

Example 10-10 Adding code into the plugin.xml file for it to find the properties file

```
<extension point="com.ibm.commerce.telesales.resources.resources">
      <resourceBundle
      baseName="com.ibm.commerce.telesales.samples.resources"/>
</extension>
```

5. Close and open the IBM Sales Center client and open the store after logging in.
6. Right-click **ConsumerDirect** store to see a new menu item for Sample Create Customer (Figure 10-5).

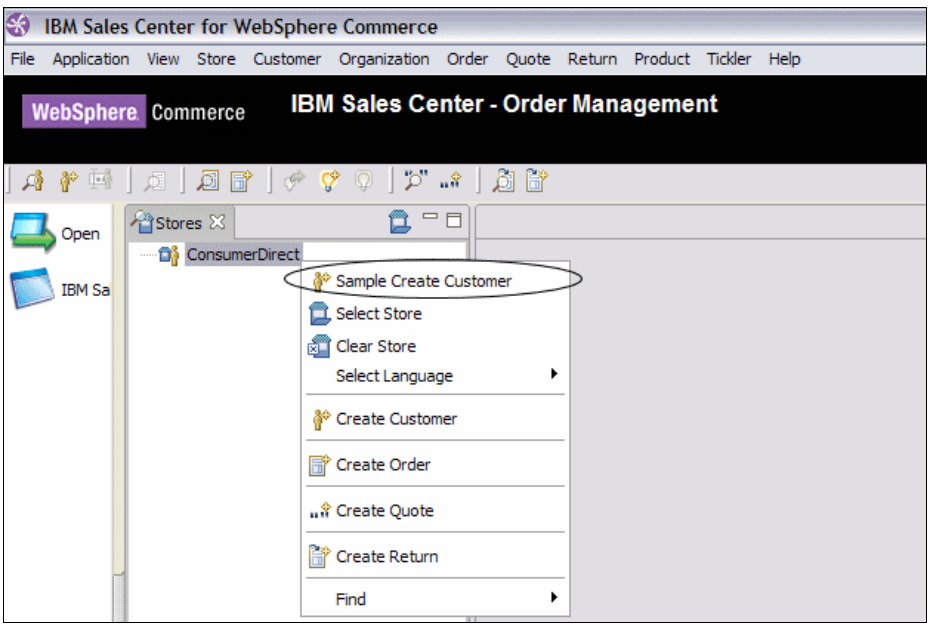


Figure 10-5 Context menu is created for the Sample Create Customer

10.3.3 Creating the activities and activity sets and mapping them to roles

Menus and Editors can be hidden from all the users or only from users with specified roles. This is accomplished with Eclipse activities, which are groupings of Eclipse plug-in contributions that can be enabled or disabled in plug-in manifests or by programming.

Plug-ins define activities by extending the org.eclipse.ui.activities extension point. The activity is given a name and a description, and is bound to plug-in contributions by a regular expression pattern.

If the activity is disabled, any extensions whose ID matches the pattern will be suppressed.

IBM Sales Center defines an activity to individually match each menu item the IBM Sales Center provides. These activity definitions can be found in the fragment.xml manifest of the com.ibm.commerce.telesales.ui.impl.activities plug-in fragment.

Although it is possible to work directly with Eclipse activities, IBM Sales Center extends the activities mechanism with two additional extension points, com.ibm.commerce.telesales.activitySet, which defines groups (possibly overlapping) of activities, and com.ibm.commerce.telesales.roles, which defines the activitySets to be enabled for each of the user roles defined by the WebSphere Commerce server. WebSphere Commerce defines three roles, CustomerServiceRepresentatives, CustomerServiceSupervisors, and Site Administrator. For each of these roles, IBM Sales Center specifies a collection of predefined activitySets that are enabled.

Hiding the Create Customer menu item

This section shows you how to hide the Create Customer menu item from the Customer menu for all the roles, by creating new definitions for the base customer actions activity set.

The default definition of the base customer actions activity set includes all the actions that can be performed on the customers by all the roles. This activity set is enabled for all the roles.

To prevent a Create Customer activity from being enabled, remove the activity from all the activity sets that include it, for example, to remove the create customer activity com.ibm.commerce.telesales.createCustomerActivity, extend the com.ibm.commerce.telesales.activitySet.baseCustomerActions activity set.

Add the extension definitions shown in Example 10-11 to your plug-in's plugin.xml file. Note that the com.ibm.commerce.telesales.createCustomerActivity has been commented out of the activitySet.

Example 10-11 Adding extension definitions to plug-in's plugin.xml file

```
<extension point="com.ibm.commerce.telesales.activitySets">
<!-- Base Customer Actions Activity Set -->
   <activitySet
   id="com.ext.commerce.telesales.sample.extensions.activitySet.baseCus
   tomerActions" label="%baseCustomerActionsActivitySetName">
      <!-- <activity
      activityId="com.ibm.commerce.telesales.createCustomerActivity">
      </activity>
      -->
      <activity
      activityId="com.ibm.commerce.telesales.findCustomerActivity">
      </activity>
      <activity
      activityId="com.ibm.commerce.telesales.editCustomerActivity">
      </activity>
      <activity
      activityId="com.ibm.commerce.telesales.clearCustomerActivity">
      </activity>
      <activity
      activityId="com.ibm.commerce.telesales.enableCustomerActivity">
      </activity>
      <activity
      activityId="com.ibm.commerce.telesales.findOrganizationActivity">
      </activity>
      <activity
      activityId="com.ibm.commerce.telesales.showContactHistoryActivity
      ">
      </activity>
      <activity
      activityId="com.ibm.commerce.telesales.resetPasswordActivity">
      </activity>
   </activitySet>
</extension>
```

Using the system configurator

Use the system configurator to indicate that your new activity set definition must be used in place of the default. To use the system configurator, perform the following steps:

1. In the Package Explorer, navigate to the com.ext.commerce.telesales.sample.extensions plug-in.
2. Right-click the plug-in folder and select **New** → **Folder**.
3. In the Folder Name field, enter `config` and click **Finish**.
4. Right-click the config folder and select **New** → **File**.
5. In the File name field, enter `config.ini` and click **Finish**.
6. Open the config.ini file for editing, and insert the following text:

```
com.ibm.commerce.telesales.activitySet.baseCustomerActions=com.ext.c
ommerce.telesales.sample.extensions.activitySet.baseCustomerActions
```

7. Save your changes.
8. In the com.ext.commerce.telesales.sample.extensions plugin.xml file, add the following text to indicate the location of your config.ini file:

```
<extension point="com.ibm.commerce.telesales.configurator">
   <configurator path="config"/>
</extension>
```

9. Save your changes.
10. Rerun the IBM Sales Center client and open the store after login. Select the **Customer** menu (Figure 10-6). The Create Customer link does not display.

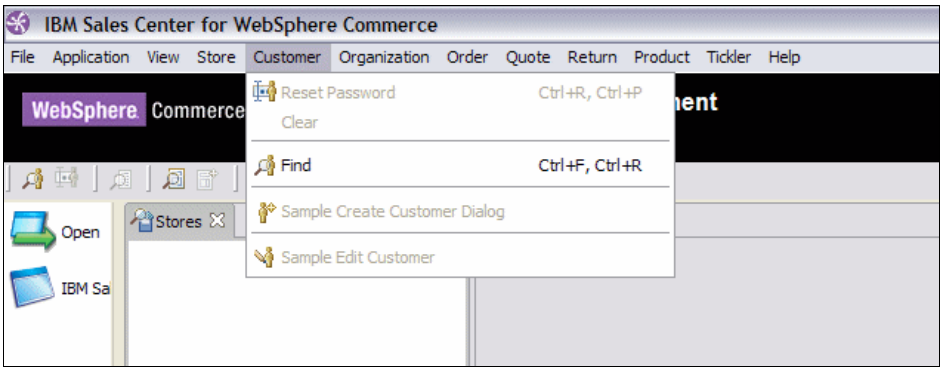


Figure 10-6 Create Customer menu item is disabled for all the users

Creating the activity and the activity sets

This section describes the processes involved in creating an activity and activitySets that will later be assigned to the CSR role and the NewCSR role. To create the activity and the activity sets, perform the following tasks:

1. Open the plug-in manifest file (plugin.xml) for the com.ext.commerce.telesales.sample.extensions plug-in.
2. Add the extension definitions shown in Example 10-12 to your plugin.xml file. These definitions create the activity that will map to the Sample Create Customer action.

Example 10-12 Adding extension definitions to create the activity to map to Sample Create Customer action

```
<extension point="org.eclipse.ui.activities">
   <!-- Sample Create Customer activity -->
   <activity name="%sampleCreateCustomerActivityName"
      id="com.ibm.commerce.telesales.samples.actions.CreateCustomerActi
      vity">
   </activity>
   <activityPatternBinding
      activityId="com.ibm.commerce.telesales.samples.actions.CreateCust
      omerActivity"
      pattern="com\.ibm\.commerce\.telesales\.samples/com\.ibm\.commerc
      e\.telesales\.samples\.actions\.CreateCustomerAction">
   </activityPatternBinding>
</extension>
```

3. Add the extension definitions shown in Example 10-13 to your plugin.xml file. These extension definitions create the activity sets for the sample create customer and create customer dialogs.

Example 10-13 Adding extensions to create activity sets for Sample Create Customer and Create Customer dialogs

```
<extension point="com.ibm.commerce.telesales.activitySets">
   <activitySet
   id="com.ext.commerce.telesales.sample.extensions.activitySet.sampleC
   reateCustomerActivitySet"
   label="%sampleCreateCustomerActivitySetName">
      <activity
      activityId="com.ibm.commerce.telesales.samples.actions.CreateCust
      omerActivity">
      </activity>
   </activitySet>
```

```
   <activitySet
   id="com.ext.commerce.telesales.sample.extensions.activitySet.createC
   ustomerActivitySet" label="%createCustomerActionsActivitySetName">
      <activity
      activityId="com.ibm.commerce.telesales.createCustomerActivity">
      </activity>
   </activitySet>
</extension>
```

At this point, the activity sets can be assigned to different roles.

Assigning activity sets to roles

This section describes the process involved in creating the extensions for the roles to enable the Create Customer and Sample Create Customer for different roles.

Provide Sample Create Customer and Create Customer access to the CSR, but restrict the NewCSR to view only the Sample Create Customer. To achieve that, create another extension in the plugin.xml file by using the code provided in Example 10-14.

Example 10-14 Providing Sample Create Customer and Create Customer access to CSR

```
<extension id="com.ibm.commerce.telesales.roles" name="%rolesName"
point="com.ibm.commerce.telesales.roles">
   <!-- customer service representative -->
   <role id="com.ext.commerce.telesales.sample.extensions.csrRole"
   roleId="customer service representative" label="%CSRRoleName">
      <activitySet
      id="com.ext.commerce.telesales.sample.extensions.activitySet.samp
      leCreateCustomerActivitySet">
      </activitySet>
      <activitySet
      id="com.ext.commerce.telesales.sample.extensions.activitySet.crea
      teCustomerActivitySet">
      </activitySet>
      <activitySet
      id="com.ibm.commerce.telesales.activitySet.applications">
      </activitySet>
      <activitySet
      id="com.ibm.commerce.telesales.activitySet.baseStoreActions">
      </activitySet>
      <activitySet
      id="com.ibm.commerce.telesales.activitySet.baseCustomerActions">
```

```
   </activitySet>
   <activitySet
   id="com.ibm.commerce.telesales.activitySet.baseCommentsActions">
   </activitySet>
   <activitySet
   id="com.ibm.commerce.telesales.activitySet.baseOrderActions">
   </activitySet>
   <activitySet
   id="com.ibm.commerce.telesales.activitySet.baseQuoteActions">
   </activitySet>
   <activitySet
   id="com.ibm.commerce.telesales.activitySet.baseReturnActions">
   </activitySet>
   <activitySet
   id="com.ibm.commerce.telesales.activitySet.baseProductActions">
   </activitySet>
   <activitySet
   id="com.ibm.commerce.telesales.activitySet.baseTicklerActions">
   </activitySet>
   <activitySet
   id="com.ibm.commerce.telesales.activitySet.basePreferencePages">
   </activitySet>
</role>
<!--NewCSR -->
<role id="com.ext.commerce.telesales.sample.extensions.NewCSRRole"
roleId="NewCSR" label="%CSRRoleName">
   <activitySet
   id="com.ext.commerce.telesales.sample.extensions.activitySet.samp
   leCreateCustomerActivitySet">
   </activitySet>
   <activitySet
   id="com.ibm.commerce.telesales.activitySet.applications">
   </activitySet>
   <activitySet
   id="com.ibm.commerce.telesales.activitySet.baseStoreActions">
   </activitySet>
   <activitySet
   id="com.ibm.commerce.telesales.activitySet.baseCustomerActions">
   </activitySet>
   <activitySet
   id="com.ibm.commerce.telesales.activitySet.baseCommentsActions">
   </activitySet>
   <activitySet
   id="com.ibm.commerce.telesales.activitySet.baseOrderActions">
   </activitySet>
```

```
        <activitySet
        id="com.ibm.commerce.telesales.activitySet.baseQuoteActions">
        </activitySet>
        <activitySet
        id="com.ibm.commerce.telesales.activitySet.baseReturnActions">
        </activitySet>
        <activitySet
        id="com.ibm.commerce.telesales.activitySet.baseProductActions">
        </activitySet>
        <activitySet
        id="com.ibm.commerce.telesales.activitySet.baseTicklerActions">
        </activitySet>
        <activitySet
        id="com.ibm.commerce.telesales.activitySet.basePreferencePages">
        </activitySet>
</role>
```

10.3.4 Testing your changes

To test your changes, log in using two different users with two different roles. If customization is successful, the user with the CSR role must see both the Sample Create Customer and the Create Customer menu items, and the NewCSR role must see only the Sample Create Customer menu item.

Figure 10-7 and Figure 10-8 show what a user will see when the user belongs to the CSR role. This user sees both the Sample Create Customer and the Create Customer menu items.

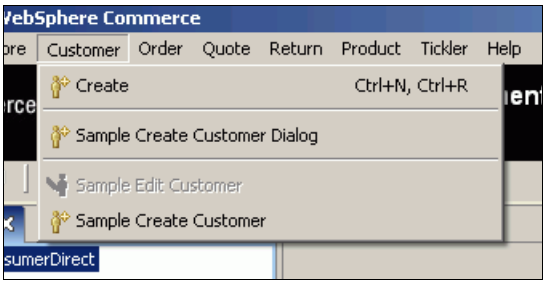


Figure 10-7 The customer menu for a user with the CSR role

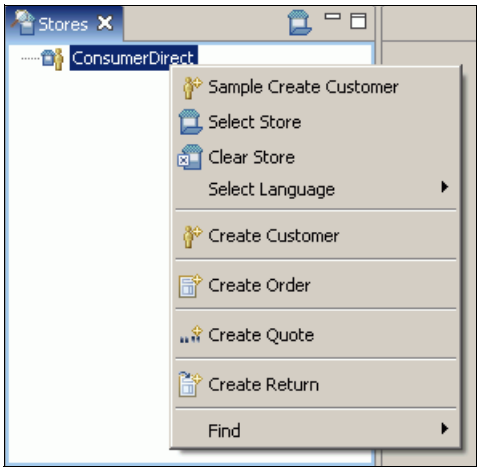


Figure 10-8 The stores context menu for a user with the CSR role

Figure 10-9 and Figure 10-10 show what a user will see when the user belongs to the NewCSR role. This user sees only the Sample Create Customer menu item and not the Create Customer menu item.

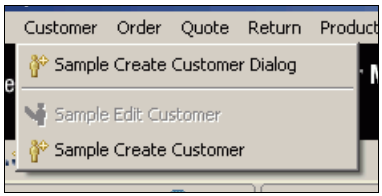


Figure 10-9 The customer menu for a user with the NewCSR role

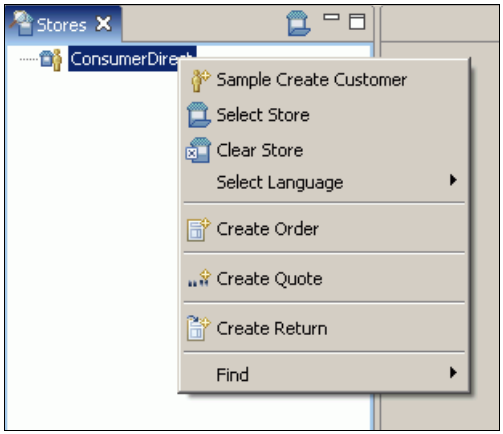


Figure 10-10 The stores context menu for a user with the NewCSR role

Figure 10-11 and Figure 10-12 show what a user will see when the user has neither the CSR role nor the NewCSR role.

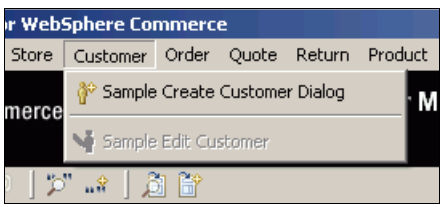


Figure 10-11 Customer menu when user has neither a CSR role nor a NewCSR role

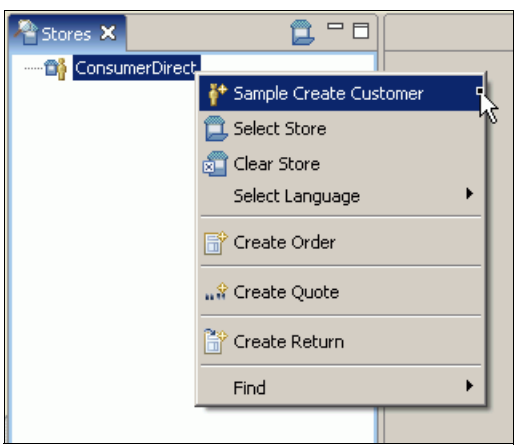


Figure 10-12 Stores context menu when user has neither a CSR role nor a NewCSR role

10.3.5 Deploying to production for both the server and the client

You can deploy the customizations in this chapter from your development environment to a production environment in the same way as any other customization. For more details, refer to 8.2, “Deploying the customizations” on page 163. Following are some hints and tips pertaining to this partciular customization:

- Deploy changes to both the WebSphere Commerce server side and the IBM Sales Center side.
  - On the WebSphere Commerce server side, deploy the following files:
    - WebSphereCommerceServerExtensionsLogic.jar, which contains the ExtendedShowStore class. Use the WebSphere Application Server administration console to deploy this file. You can combine this file with TelesalesRegistry-ext.xml as a partial application update if desired.
    - TelesalesRegistry-ext.xml, which contains the modified IBM Sales Center registry that points to the new ExtendedShowStore class. Use the IBM WebSphere Application Server administration console to deploy this file. You can combine this file with WebSphereCommerceServerExtensionsLogic.jar as a partial application update if desired.
    - *WC_installdir*\instances*instance_name*\xml*instance_name*.xml, which is the production equivalent of wc-server.xml in the development environment. Redo any changes made to wc-server.xml to the appropriate *instance_name*.xml, and then run the following ANT task with the target UpdateEAR to propagate changes to WebSphere Application Server:

      ```
      WC_installdir\bin\config_ant.bat -DinstanceName=instance_name
      UpdateEAR
      ```

      On non-Windows platforms, the file is called config_ant.sh.
  - On the IBM Sales Center side, package and deploy the com.ext.commerce.telesales.sample.extensions plug-in.
- APAR IY88078 must be installed on your WebSphere Commerce server machine. When running outside the development environment, APAR IY88078 requires the 6000 enterprise archive-enablement (EAR-enablement) APAR as a prerequisite. Install the 6000 enterprise archive-enablement (EAR-enablement) APAR on the product, but not the instance.

Part 5

Integration customization scenario examples

This part describes scenarios based on our work with Customer Care integration.

11

Customer Care integration with Lotus Sametime

e-business collaboration involves the interaction between people and information to support critical business functions. e-business collaboration capabilities add value to e-commerce by strengthening the entire value chain.

WebSphere Commerce provides a powerful solution to sell products and services through the Web. Supporting thousands of users, WebSphere Commerce enables organizations to optimize marketing, business relationships, and channel management to maximize e-commerce revenue. WebSphere Commerce is currently capable of providing real-time, online customer service and technical support. In addition, we have created a sample IBM Sales Center customization has been created to demonstrate the IBM Sales Center integration with Lotus Sametime.

Business-building benefits such as Web-based team-building among colleagues, customers, suppliers, and partners are possible through two efficient online solutions:

- Customer Care (available for WebSphere Commerce 6.0 Professional or Enterprise)
- Collaborative Workspace (available for WebSphere Commerce 6.0 Enterprise)

11.1 Introduction to Customer Care

WebSphere Commerce provides real-time customer care support through synchronous text interface (instant messaging) using Lotus Sametime server.

A customer can enter the site and click **Live Chat with Customer Representative** on the Store page to connect to a customer service representative (CSR) so that the two parties can communicate or chat over the Internet. A CSR accesses the Customer Care interface through the WebSphere Commerce Accelerator and through the IBM Sales Center client. A CSR can also view the Store page where the customer requires assistance, and retrieve the shopping cart and the profile information.

To facilitate a more efficient communication with customers, Customer Care supports the use of predefined lists of Uniform Resource Locator (URLs) and topics. After these lists are created, they are available during conversations so that CSRs have accessible lists of the most commonly referenced store pages and answers to the most commonly asked questions.

Customer Care also supports multiple queues so that customers looking for help can be routed to the most appropriate queue, monitored by CSRs who are qualified to answer their questions. Custom messages for the waiting customers can also be configured. This interface also allows a CSR to chat with other CSRs.

11.2 Installation and configuration

This section describes how to install and enable the Customer Care component in WebSphere Commerce V6.0.

11.2.1 Software prerequisites

Following are the software prerequisites:

- WebSphere Commerce 6.0, Professional or Enterprise
- IBM Lotus® Domino® 7.0.2
- IBM Lotus Notes®, IBM Lotus Domino Designer®, and the IBM Lotus Domino Administrator Client
- IBM Lotus Sametime 7.5
- Customer Care component

Refer to *Integration Guide for WebSphere Commerce with Sametime and Quickplace*, which is available on the Web at:

http://www-1.ibm.com/support/docview.wss?uid=swg24012535

11.2.2 Installing IBM Lotus Sametime

Install IBM Lotus Sametime 7.5 according to the instructions provided in the Lotus Developer Domain Documentation library, which is available on the Web at:

http://www-1.lotus.com/ldd/doc

11.2.3 Changing the default Hypertext Transfer Protocol port for the Sametime server

This is required if you have installed the Web server, which is used for WebSphere Commerce, and the Lotus Sametime server, on the same machine. If you are installing them on separate machines, skip this task.

To change the default Hypertext Transfer Protocol (HTTP) port for the Sametime® server, perform the following tasks:

1. Start the Domino server.
2. Launch the Lotus Administration client.
3. Log in using the server administrator user ID and password.
4. Select **File** → **Open Server**. Select the Domino server where Lotus Sametime is present.
5. Click the **Configuration** tab.
6. Edit the server document for the Domino server where Lotus Sametime is present.
7. Click the **Ports** tab.
8. Click the **Internet Ports** tab.

9. Click the **Web** tab and change the TCP/IP port number from 80 to 88 (or to any value which is not used) (Figure 11-1).
10. Click **Save** and close the server document.

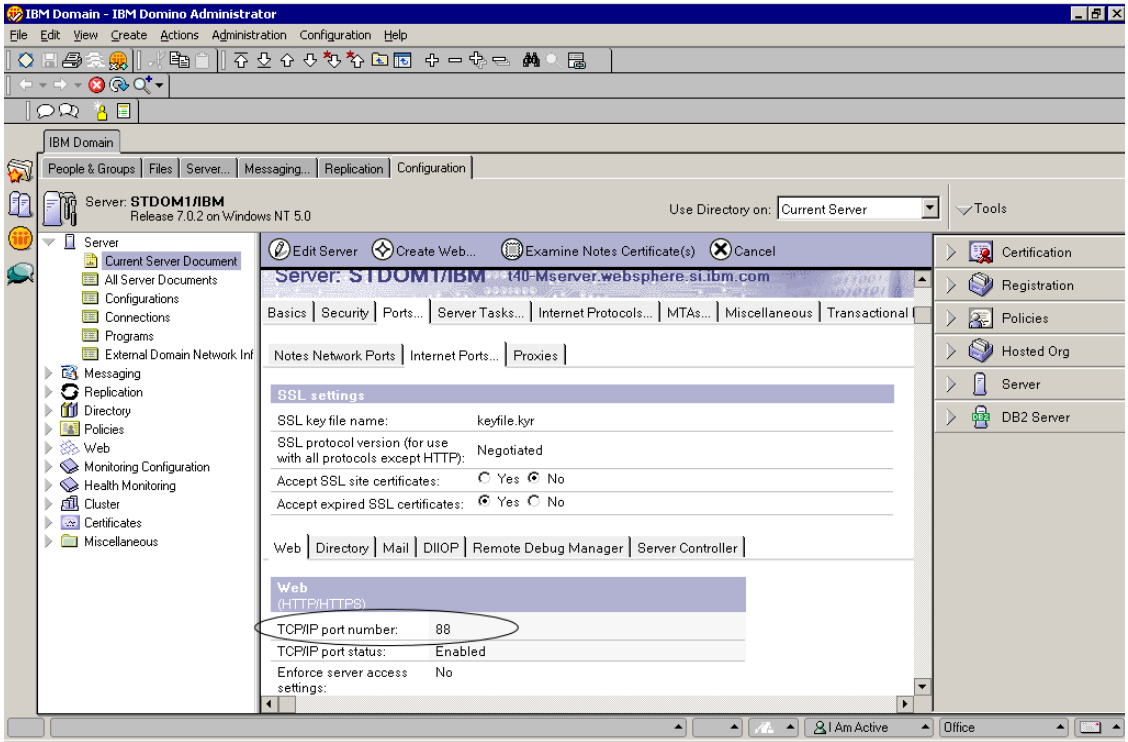


Figure 11-1 Changing the default HTTP port

11.2.4 Installing the Customer Care component

Install the Customer Care component on the Lotus Sametime server. Download the collaborations reference application CustomerCare.zip and install it manually.

For details, refer to *Integration Guide for WebSphere Commerce with Sametime and Quickplace*, which is available on the Web at:

`http://www-1.ibm.com/support/docview.wss?uid=swg24012535`

To install the Customer Care component, perform the following tasks:

1. Extract the WC60CollaborationRefApp.zip file that you have downloaded.
2. Extract the CustomerCare.zip from the WC60CollaborationRefApp.zip that you extracted in the previous step.
3. Copy the contents of the Customer Care folder from the CustomerCare.zip file to the <Domino_installdir>\data\Domino\html\wc folder.

Note: If you plan to configure Lotus Sametime to use the same Lightweight Directory Access Protocol (LDAP), ensure that you configure WebSphere Commerce to use the same LDAP server as Lotus Sametime.

11.2.5 Enabling Customer Care in WebSphere Commerce

This section discusses how to enable Customer Care in WebSphere Commerce.

Manual configuration

To enable Customer Care manually in WebSphere Commerce, complete the following tasks in your WebSphere Commerce machine:

1. Open the file <*WC_installdir*>/instances/<*instance_name*>/xml/<*instance_name*>.xml. Change the display flag from false to `true` in the sections collaboration and Sametime, for example, the <instance_name>.xml must be similar to the following:

   ```
   <Collaboration display="true">
      <Sametime... ... ... ... ... ...display="true" ... />
   ```

2. Open the file wc-server.xml located at <*WAS_installdir*>/profiles/<*instance_name*>/installedApps/<WC_instance_name_cell>/<*WC_instance_name*>.ear/xml/config. Change the display flag from false to `true` in the collaboration and Sametime nodes as follows:

   ```
   <Collaboration display="true">
      <Sametime... ... ... ... ... ...display="true" ... />
   ```

3. Open the file wc-server.xml and search for the ToolsGeneralConfig node, and search for the following under the ToolsGeneralConfig node:

   ```
   <component enabled="false" name="Sametime"/>
   ```

   Change the enabled flag to `true`:

   ```
   <component enabled="true" name="Sametime" />
   ```

4. The wc-server.xml is located at <*WAS_installdir*>/profiles/<*instance_name*>/installedApps/<*WC_instance_name_cell*>/<*WC_instance_name*>.ear/xml/config. Open the <instance_name>.xml and search for the ToolsGeneralConfig node, and search for the following under the ToolsGeneralConfig node:

   ```
   <component enabled="false" name="Sametime"/>
   ```

   Change the enabled flag to `true`:

   ```
   <component enabled="true" name="Sametime"/>
   ```

Configuration using the Configuration Manager

To enable Customer Care in WebSphere Commerce using the Configuration Manager, perform the following tasks on your WebSphere Commerce machine:

1. Stop the WebSphere Commerce server instance.
2. Launch the WebSphere Commerce Configuration Manager.
3. Type your Configuration Manager user ID and password.

4. Select <***host_name***> → **Commerce** → **Instance List** → <***instance_name***> → **Instance Properties** → **Collaboration** → **Sametime** (Figure 11-2).

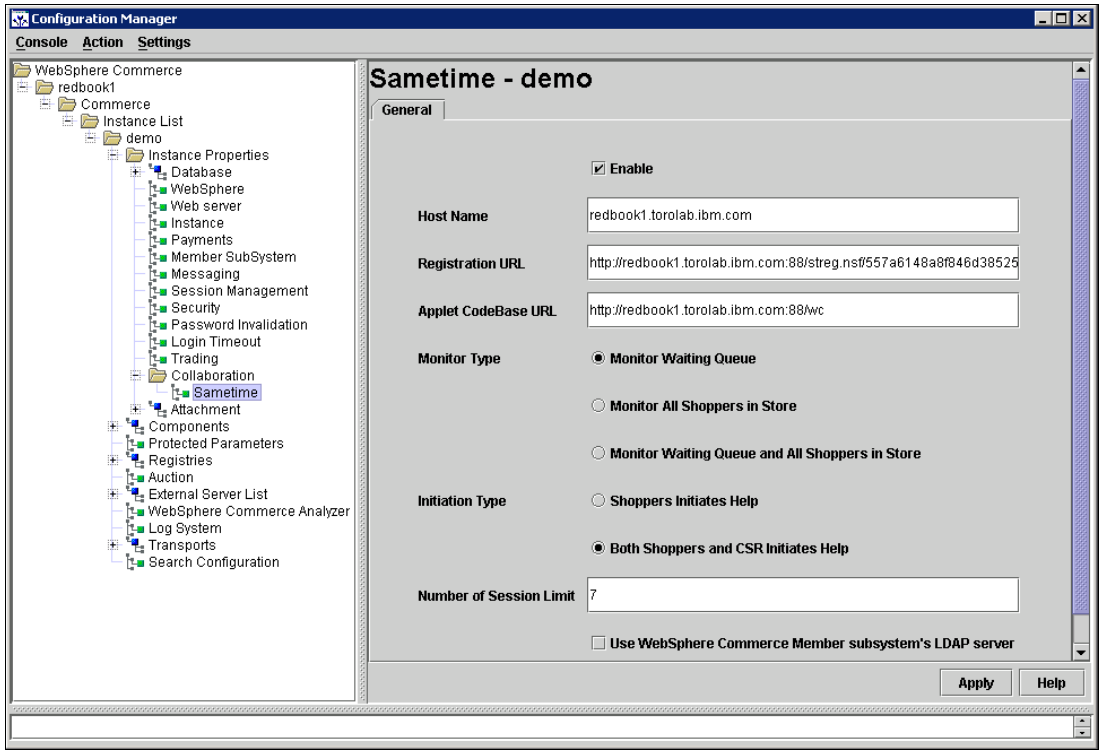


Figure 11-2 Configuring the WebSphere Commerce server for Sametime

5. In the Sametime - demo panel (Figure 11-2), complete the following tasks:
   - Select the **Enable** check box.
   - Type the Host Name. This is the fully qualified host name of your Lotus Sametime server.
   - Type the Registration URL. This is the host name of your Lotus Sametime server.

> **Note:** You must only change the <*host_name*> of the Web address, for example, http://<*host_name*>/wc. If you have changed the default port of the Lotus Sametime server, mention it here, for example, http://<*host_name*>:88/wc.

– Type the Applet CodeBase URL. This is the location of the applet code created by the WebSphere Commerce Customer Care installation program. Ensure that the applet code is installed on the Lotus Sametime server machine.

> **Note:** You must only change the <host_name> of the Web address, for example, http://<*host_name*>/streg.nsf/557a6148a8f846d3852563e10000ca95?CreateDocument. If you have changed the default port of the Sametime server, mention it here, for example, http://<*host_name*>:88/streg.nsf/557a6148a8f846d3852563e10000ca95?CreateDocument.

– Change the Monitor Type, Initiation Type, and the Number of Session Limit to suit your testing or production environment.

– If Customer Care uses WebSphere Commerce as the LDAP server, select **Use WC Member subsystem's LDAP server**.

– Click **Apply**.

– A message indicating that Lotus Sametime is configured successfully for WebSphere Commerce appears. Click **OK** to continue.

6. Close the WebSphere Commerce Configuration Manager.
7. Start the WebSphere Commerce server instance.

11.2.6 Configuring the Lotus Sametime self-registration feature

If you configure Lotus Sametime to not use an LDAP server, configure the Lotus Sametime self-registration feature by performing the following tasks:

1. Set STCENTER.NSF as your default home page by performing the following tasks:
   a. Start the Domino server.
   b. Launch the Lotus Administration client.
   c. Log in using the server administrator user ID and password.
   d. Select **File** → **Open Server**.
   e. Select the Domino server where Lotus Sametime is present.
   f. Click the **Configuration** tab.
   g. Edit the server document for the Domino server where Lotus Sametime is present.
   h. Click the **Internet Protocols** tab.

i. Click the **HTTP** tab.

j. In the mapping section, type `STCENTER.NSF` as the Home URL.

k. Save and close the server document.

2. Set user access rights for the Domino directory as follows:

   a. Start the Domino server.

   b. Launch the Lotus Administration client.

   c. Log in using the server administrator user ID and password.

   d. Select **File** → **Open Server**.

   e. Select the Domino server where Lotus Sametime is present.

   f. Click the **Files** tab.

   g. In the Show Me field, select **Database Only**.

   h. Select the directory document for the Domino server where Lotus Sametime is present, for example, names.nsf.

   i. Right-click the directory document and select **Access Control** → **Manage to launch Access Control List**.

   j. Select **Sametime Development/Lotus Notes Companion Products** user ID from the People, Server, and Group lists. If this ID does not exist, click **Add** to add the corresponding ID.

   k. Select **Editor** from the Access field.

   l. Click **OK**.

   m. Save and close the server document.

3. Enable the self-registration feature as follows:

   a. Launch the Lotus Administration client.

   b. Log in using the server administrator user ID and password.

   c. Select **File** → **Open Server**. Select the Domino server where Lotus Sametime is present.

   d. Click the **Files** tab.

   e. In the Show Me field, select **Database Only**.

   f. Select and double-click the stconfig.nsf document from the Domino server where Lotus Sametime is present.

   g. Click **By Form** in the open document.

   h. Select the **AnonymousAccess** form and double-click it to edit it.

   i. Change the Anonymous Users can register themselves setting to `true`.

j. Select **File** → **Save**.

k. Restart your Domino server.

4. Test the self-registration feature as follows.

   a. Type the URL `http://Lotus Sametime_server/stcenter.nsf` into a Web browser.

   b. Click **Register**.

   c. Select **Register to use Lotus Sametime**.

   d. Type your user information in the Register to use Lotus Sametime page and click **Submit Request**. A confirmation page must display a message confirming the user registration.

11.2.7 Enabling the flex flow for the Customer Care feature

To enable the Customer Care feature in WebSphere Commerce Accelerator, perform the following steps:

1. Open WebSphere Commerce Accelerator using an appropriate user ID and password.
2. Select **AdvancedB2B** store.
3. Select **Stores** → **Change flow**.
4. Select **Customer Care** in the left navigation pane.
5. Select **Enable the Customer Care**.
6. Click **Apply**.
7. After the changes take effect, launch the Web site of the seller organization and ensure that the Live chat with customer assistance link is displayed on the sidebar.

Note: If Customer Care flex flow is not available for your store, refer to the information center topic “Adding the option to enable or disable the feature to the Change Flow notebook”, which is available on the Web at:

`http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.developer.tutorial.doc/tutorial/ttdsfcflow2.htm`

11.3 Adding Customer Care to your store

To add Customer Care to a store where it does not exist, perform the following tasks:

1. Copy the Customer Care integration files from the starter store, which is already Customer Care-enabled, for example, Advanced business-to-business starter store, to your store or another starter store, which is not Customer Care-enabled. To do this, perform the following tasks:

   a. Locate the store archive file for the Advanced business-to-business starter store. The store archive files are located in the <*WC_installdir*>/starterstores directory.

   b. Open the ConsumerDirect folder, and select **ConsumerDirect.sar**. For our example, we added Customer Care to the Consumer Direct starter store. To add Customer Care to other stores, for example, the Business-to-business Direct starter store or your own store, select **B2Bdirect.sar** in the B2BDirect folder or your own sar file.

   c. Open the Advanced business-to-business store archive file using WinZip or a similar tool.

   d. Extract the files from the Advanced business-to-business store archive to a temporary directory or to the directory that contains the Web assets for your store.

   > **Note:** You can add the Customer Care-enabled JavaServer Page (JSP) files to your store sar file and republish the store archive, or copy these JSPs to the already published starter store, for example, <*WAS_HOME*>\profiles\<*instance*>\installedApps\WC_<*instance*>_cell\WC_<*instance*>.ear\Stores.war\ConsumerDirect directory.

   e. To maintain the same directory structure as the samples stores, create subdirectories for the following files:

      - ../CustomerServiceArea/CollaborationSection
        - CustomerCareAppletReadySetup.jsp
        - CustomerCareBlankSetup.jsp
        - CustomerCareBlankSetup.jsp
        - CustomerCareFrameSetup.jsp
        - CustomerCareInformationSetup.jsp
        - CustomerCareMonitorList.jsp
        - CustomerCareStoreQuestionList.jsp
        - CustomerCareStoreURLList.jsp
        - CustomerCareChatSetup.jsp

- /include
  - CustomerCareHeaderSetup.jsp
  - EnvironmentSetup.jsp
- ../
  - index.jsp
  - Sametime.js
  - StoreFramesetPage.jsp

2. Add code to the JSPs to determine which page the customer is browsing, as follows:
   a. Include the CustomerCareHeaderSetup.jsp file to the store's header file, for example, <%@ includes file="include\CustomerCareHeaderSetup.jsp" %>.
   b. Add the following code to any page that must be marked personal, and is therefore not available for access by the CSR. Ensure that the code shown in Example 11-1 is added before you include the CustomerCareHeaderSetup.jsp file.

Example 11-1 Adding code to a page that must be marked personal

```
<flow:ifEnabled feature="customerCare">
<%
request.setAttribute("liveHelpPageType", "personal");
%>
</flow:ifEnabled>
```

> **Note:** For more details, refer to the information center topic "Determining which page the customer is browsing", which is available on the Web at:
>
> http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.customercare.doc/refs/rlhwhichpage.htm

3. Add a link to allow customers to access Customer Care from your store by performing the following tasks:
   a. Determine where you want to place the link to Customer Care. You may, for example, want to place the link in a navigation bar so that it is always available to customers, or in certain pages in the store.

b. Copy the code shown in Example 11-2 into the pages that contain the link.

Example 11-2 Copying code into pages that contain the link

```
<a href="javascript:if((parent.sametime != null))
top.interact();"><fmt:message key="LiveHelp" bundle="${storeText}"
/></a>
```

4. Create an entry page or modify your index.jsp, which will redirect to the Customer Care frameset page. Because the frameset is required for most Customer Care features to function properly, the customer must call the StoreFramesetView command to activate the frameset. For an example, see index.jsp of the Advanced business-to-business starter store, which can be accessed in the <*WC_installdir*>/starterstores directory.

Note: For details, refer to the information center topic “Using the frameset”, which is available on the Web at:

http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.customercare.doc/tasks/tlhframeset.htm

11.4 Integrating Customer Care with IBM Sales Center

This section discusses how to integrate Customer Care with IBM Sales Center to leverage some of the functionalities provided by Customer Care within the WebSphere Commerce server. The scenario presented in this section demonstrates the capability of IBM Sales Center to extend itself to support a full-scale integration.

Figure 11-3 illustrates a simple scenario where the store is handled by a single CSR who can interact with several customers visiting the store. Queue support has not been implemented. Therefore, the CSR is allowed to view all the users logged into the site.

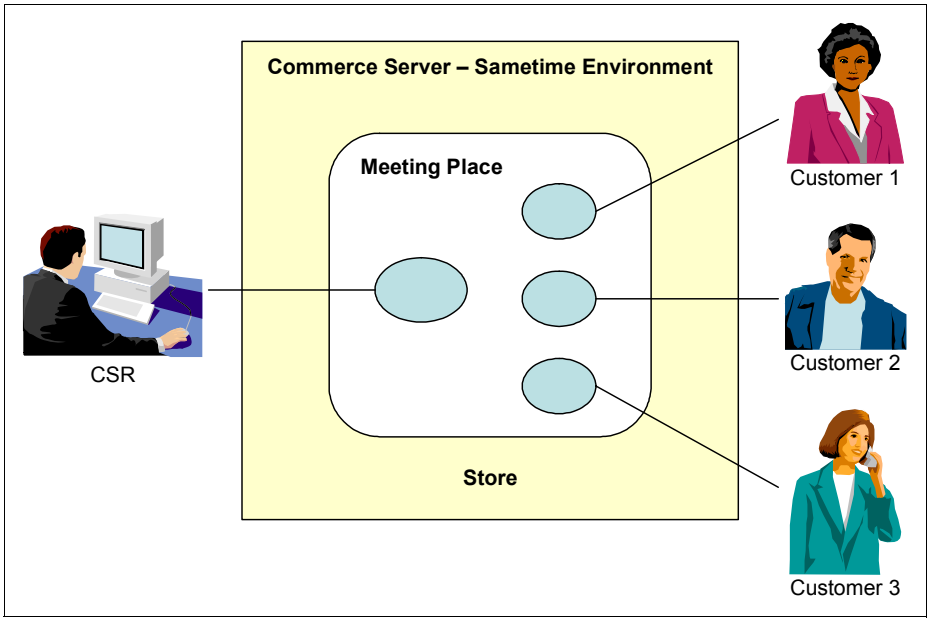


Figure 11-3 Sametime environment

Note: To demonstrate the integration and use of Lotus Sametime in IBM Sales Center, we created a sample integration application. This sample integration application does not follow any standard coding practice. However, the sample integration application can be used as a reference to achieve results that are similar to those documented in this chapter.

11.4.1 Use case example

A simple use case example for the implementation of our sample integration application is described here:

1. A customer visiting the storefront initiates a chat from the storefront.
2. A chat window opens at the storefront and waits for acceptance by a CSR.
3. The CSR who is logged in to the store is able to view the customer as a guest user who is logged in. The CSR can accept the user to initiate a chat.

4. If a CSR is logged in, the customer chat window shows the appropriate information.
5. After the CSR accepts the conversation, a new window appears in the IBM Sales Center. This enables the CSR to communicate with the customer.
6. If the customer goes offline, the entry is automatically removed from the view.

11.4.2 Prerequisites

This section lists the prerequisites for Sametime to run within IBM Sales Center:

- The Lotus Sametime server must be set up and configured with the WebSphere Commerce server.
- A user name must exist in the Sametime server for the CSRs.
- The sample integration application plug-in for Sametime integration must be downloaded and deployed in the IBM Sales Center client environment.
- For further development, you must have the Sametime libraries in the CLASSPATH of the development environment you are using. In our example, we used Lotus Sametime 7.5 Java SDK, libraries, and samples that are downloadable from the IBM developerWorks Web site:

  http://www-128.ibm.com/developerworks/lotus/downloads/toolkits.html

11.4.3 Sample integration application implementation

The sample integration application plug-in can be deployed in the IBM Sales Center client environment like any other Eclipse plug-ins. Based on the simple scenario shown in Figure 11-3, complete the following steps to integrate the sample integration application:

1. Start the IBM Sales Center client.
2. Download and deploy the downloaded plug-in to the existing installation by using any of the Eclipse update mechanisms. The plug-in file can be retrieved from the com.ibm.commerce.telesales.sametime.zip file. For details about this, refer to Appendix A, “Additional material” on page 375.

3. A new preference page is available to set the Sametime server location, user name, and password of the CSR account. To perform this task, select **File** → **Preferences**. The information shown in Figure 11-4 is used by the CSR to log in to the Sametime environment. Enter the necessary information in the fields and click **Apply** to update this information.

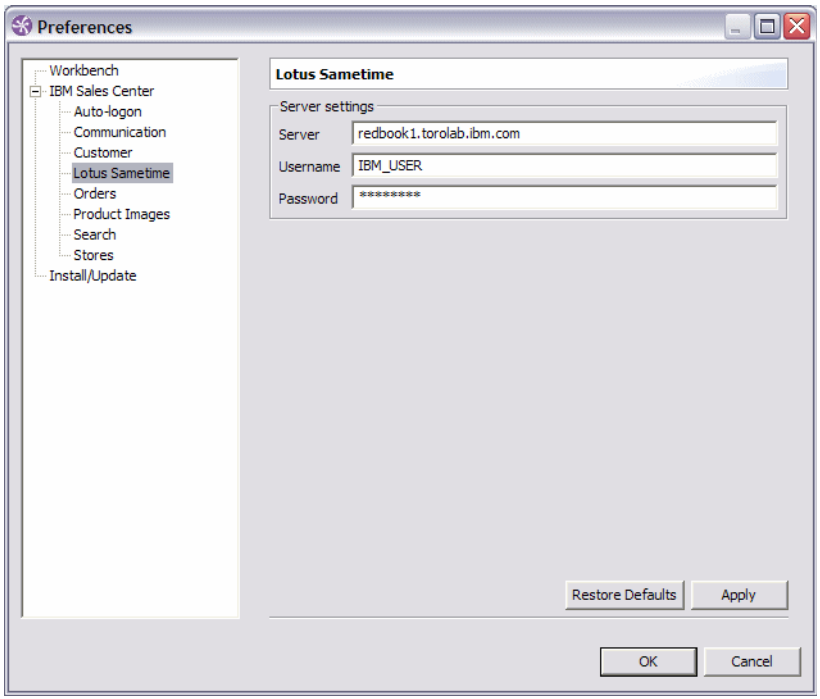


Figure 11-4 Setting Sametime preference

Attention: Refer to 7.3, “Steps to develop customizations” on page 126 for details about creating a new customized plug-in.

A new preference window is added into IBM Sales Center by extending the org.eclipse.ui.preferencePages extension point, as shown in Example 11-3.

Example 11-3 Extending the org.eclipse.ui.preferencePages extension point

```
<extension point="org.eclipse.ui.preferencePages">
<page
name="Lotus Sametime"
category="com.ibm.commerce.telesales.preferences.TelesalesPreferencePag
e"
```

```
class="com.ibm.commerce.telesales.sametime.preference.SametimePreferenc
ePage"
id="com.ibm.commerce.telesales.preferences.SametimePreferencePage">
</page>
</extension>
```

The SametimePreferencePage class shown in Example 11-4 defines the layout of the preference page being displayed.

Example 11-4 SametimePreferencePage class

```
public class SametimePreferencePage extends FieldEditorPreferencePage
implements IWorkbenchPreferencePage {
/*
Implementation here
*/
}
```

4. Open the IBM Sales Center application, and then the Sametime View, and select **View** → **Show View** → **Lotus Sametime**. Double-click **Lotus Sametime** (Figure 11-5).

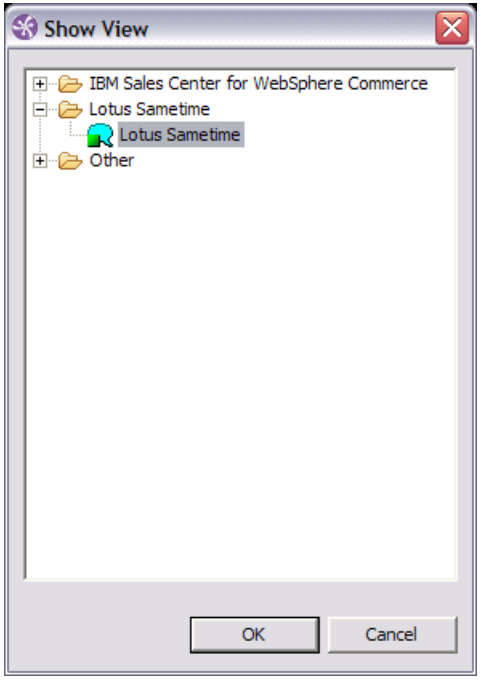


Figure 11-5 Selecting the Sametime view

A new view is added to the plugin.xml for this support. The code additions in the XML fragment will look as shown in Example 11-5.

Example 11-5 Code additions in the XML fragment

```
<extension
point="org.eclipse.ui.views">
<category
name="Lotus Sametime"
id="com.ibm.commerce.telesales.sametime">
</category>
<view
name="Lotus Sametime"
icon="icons/sample.gif"
category="com.ibm.commerce.telesales.sametime"
class="com.ibm.commerce.telesales.sametime.views.SametimeView"
id="com.ibm.commerce.telesales.sametime.views.SametimeView">
</view>
</extension>
```

Also, a fragment part is added in the plugin.xml file to include this newly created view as a part of the IBM Sales Center default perspective (Example 11-6).

Example 11-6 Adding a fragment part in the plugin.xml file

```
<extension point="org.eclipse.ui.perspectiveExtensions">
<perspectiveExtension
targetID="com.ibm.commerce.telesales.ordersPerspective">
<view id="com.ibm.commerce.telesales.sametime.views.SametimeView"
relative="com.ibm.commerce.telesales.sametime.views.SametimeView"
relationship="bottom"
ratio="0.8"/>
</perspectiveExtension>
</extension>
```

5. After the Sametime view is selected, a new view appears next to the Ticklers view (Figure 11-6).

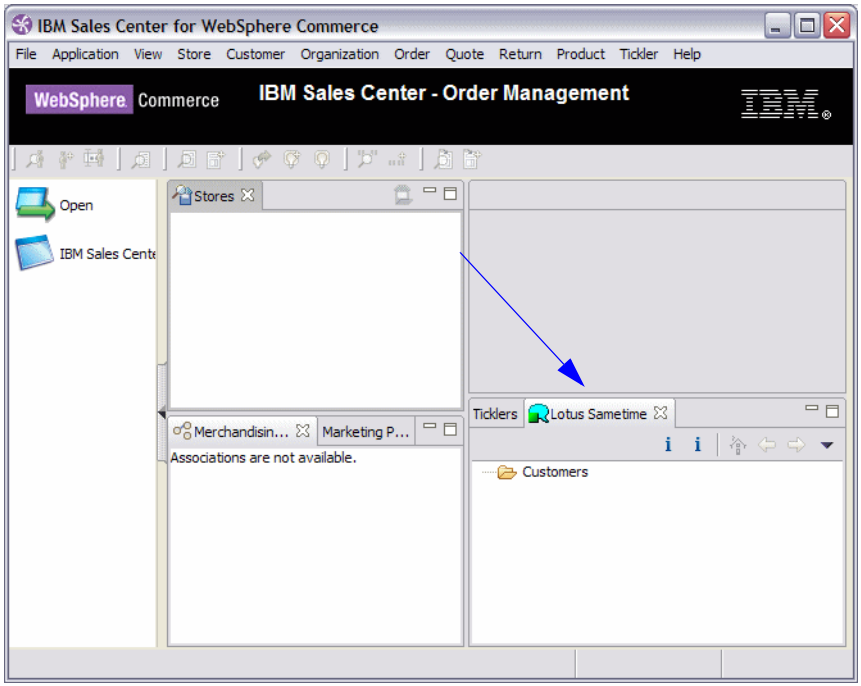


Figure 11-6 Sametime view

6. Log in to WebSphere Commerce server. Open the store that you wish to assist the customers to enter. In our example, the store is AdvancedB2BDirect. Right-click the store and select **Customer care** as shown in Figure 11-7.

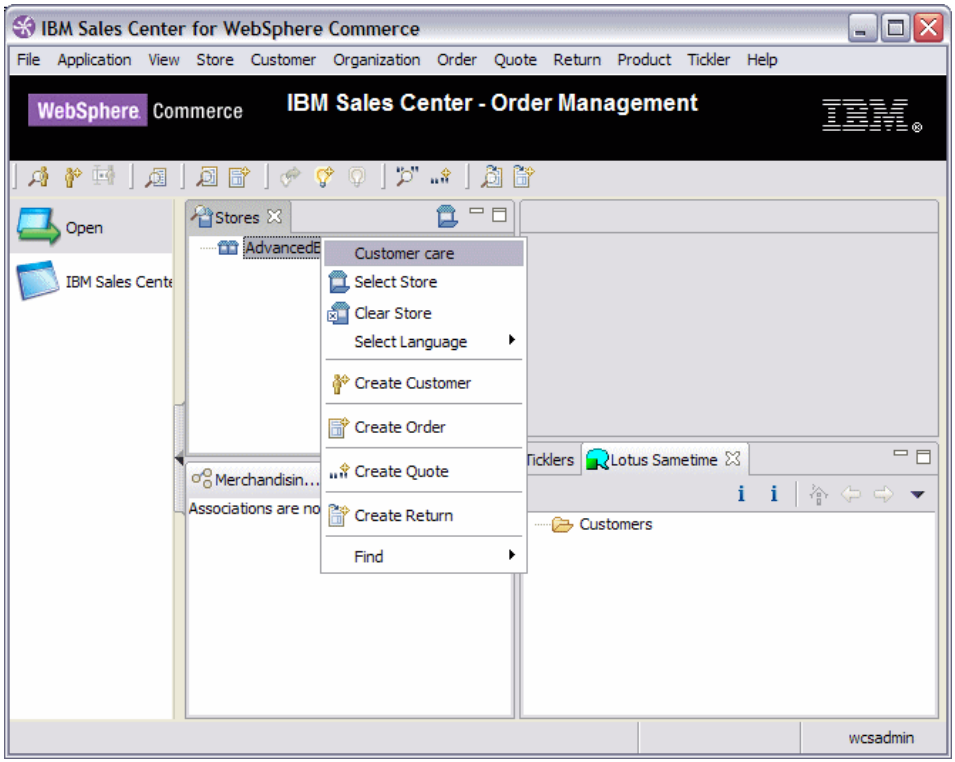


Figure 11-7 Selecting the Customer Care menu

This action internally initiates a connection to the Sametime server by creating an STSession (for more information about this, refer to the Sametime software development kit (SDK) documentation downloaded with the toolkit) and perform user authentication. The code is as shown in Example 11-7.

Example 11-7 Performing user authentication

```
try
{
m_session = new STSession("Login " + storeId);
m_session.loadAllComponents();
_comm = (CommunityService) m_session
.getCompApi(CommunityService.COMP_NAME);
m_comm.addLoginListener(this);
```

```
m_comm.loginByPassword(serverName, username, password);
m_session.start();
}
catch (DuplicateObjectException doe)
{
doe.printStackTrace();
}
```

The LoginListener listens to the login event (Example 11-8).

Example 11-8 LoginListener listening to the login event

```
public void loggedIn(LoginEvent arg0)
{
String placeName = placeStr;
String placeDisplayName = placeStr;
placesService = (PlacesService) m_session
.getCompApi(PlacesService.COMP_NAME);
place =
placesService.createPlace(placeName,placeDisplayName,EncLevel.ENC_LEVEL
_DONT_CARE, 0);
place.enter();
place.addPlaceListener(this);
}
```

Note: The Sametime Place is a place for people to meet no matter where they are located physically. You can use the place to see who else is there, exchange text and data messages, and initiate one-on-one sessions with other place members, or simply monitor them.

After successful authentication, a place name is created in a standard format with the store ID, Commerce Server instance name, and server name as a part of it, for example, for a store with store ID 15001, the place name will be 10051_store@demo@redbook1.torolab.ibm.com, where demo is the instance name at redbook1.torolab.ibm.com.

A Place Listener is attached to the newly created place. It fires an event when a new section is added to it. In the Sametime Place model, a virtual place comprises sections. This makes the Place similar to a real place with different rooms inside it. Every user who enters a place enters a specific section in that place in the same way it occurs in an actual place.

Add a new SectionListener to each Place to listen to the customers coming in and going out of the section, as shown in Example 11-9.

Example 11-9 Adding a new SectionListener to each Place

```
public void sectionAdded(PlaceEvent event)
{
event.getSection().addSectionListener(this);
}
```

SectionListener has implementation for the following methods (Example 11-10).

Example 11-10 Methods for which SectionListener has implementation

```
public void usersEntered(SectionEvent event)
{
System.out.println("Agent users Entered");
UserInPlace[] users = event.getUsers();
for (int i = 0; i < users.length; i++)
{
String dispName = users[i].getDisplayName().trim();
// Checking whether user is a CSR.
int idx = dispName.indexOf("/");
if(idx!=-1)
dispName = dispName.substring(0,idx).trim();
if (!dispName.equalsIgnoreCase(csrUser))
{
custList.put(users[i].getDisplayName(), users[i]);
//update customer in the view
updateCustomer(scRoot, users[i].getDisplayName());
}
}
public void userLeft(SectionEvent event)
{

System.out.println("Agent users left "+
event.getUser().getDisplayName());
UserInPlace user = event.getUser();
//remove customer in the view
removeCustomer(scRoot, user.getDisplayName());
}
```

7. The CSR logs in to the Sametime server and starts *waiting* to receive a chat ticket from the customers who are requesting for sales support.

   Let us now see what happens when a customer requests help using the Customer Care window from the storefront.

8. A customer opens a browser and accesses the storefront in which the CSR is waiting for chat tickets.

9. After logging in successfully, the customer sees an HTTP link from the store, and is ready to connect with the CSR. The customer sees a window similar to that shown in Figure 11-8 if connected successfully.

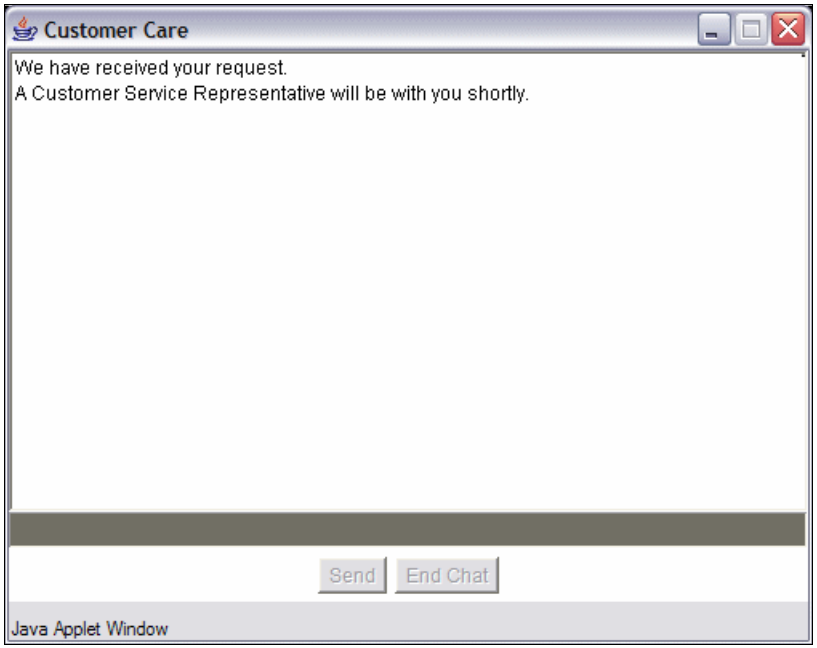


Figure 11-8 Chat initialization window

10. The customer action in the storefront adds a new customer in the view (Figure 11-9). Now it is up to the CSR to decide whether to accept or reject the chat. In our example, only the functionality to accept is implemented. After the customer chat window is opened, the CSR can view the specific user available for the CSR in the IBM Sales Center. To establish a chat session,

the CSR right-clicks the user and selects **Accept**. In this example, the customer object information is not fetched for each of the users listed in WebSphere Commerce. However, this can be achieved by performing additional coding.

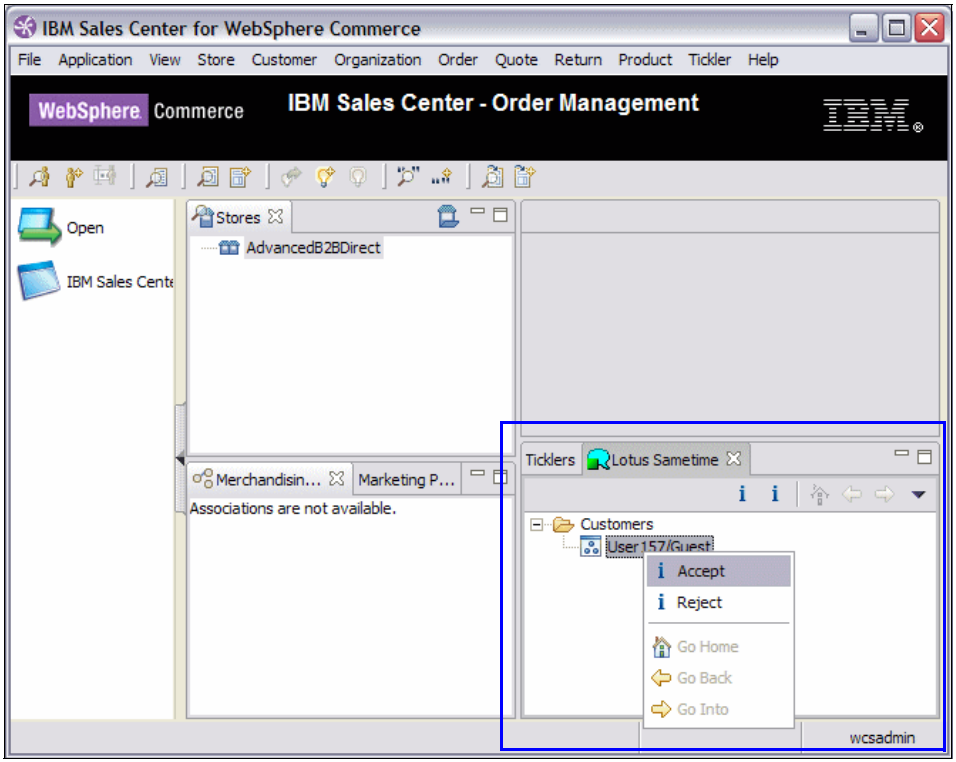


Figure 11-9 Customer list

11. After being accepted, a new chat window (Figure 11-10) appears in the IBM Sales Center client to communicate with the customer.

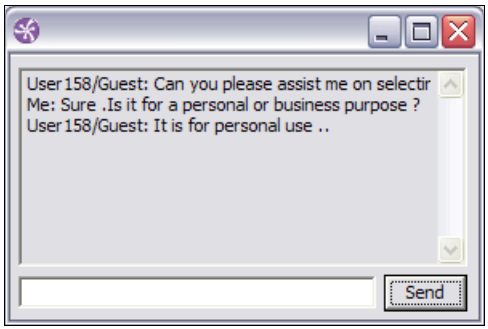


Figure 11-10 Chat window

The chat window on the customer side is similar to that shown in Figure 11-11.

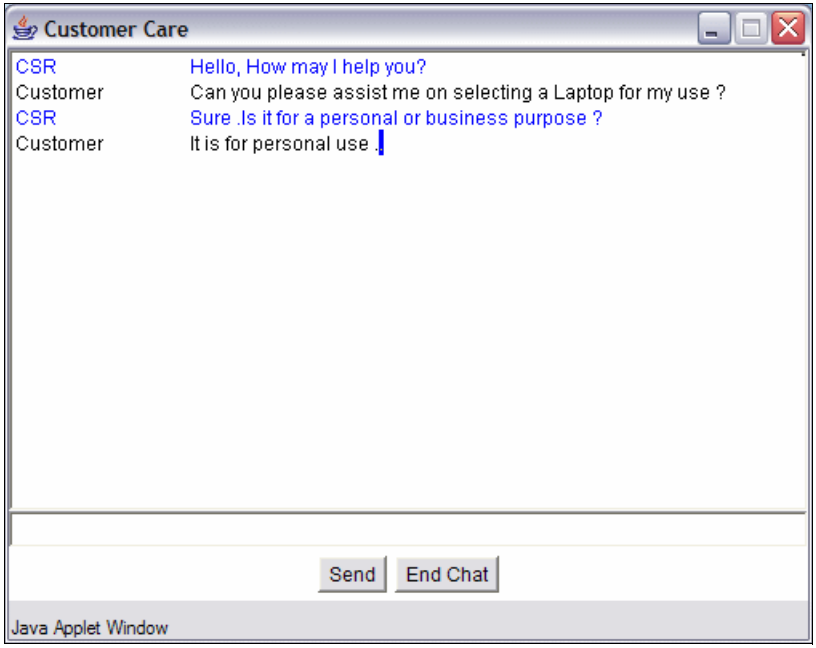


Figure 11-11 Customer chat window

After the customer quits the chat window, the user disappears from the customer list.

11.4.4 Scope for further expansion

A full-scale integration of Customer Care functionalities into the IBM Sales Center client adds a lot of value to the product, and is what most customers seek. With the support of integrated audio and video in the latest release of Sametime 7.5, you can integrate data, voice, and video into a single environment.

The sample implementation demonstrated in 11.4.3, “Sample integration application implementation” on page 301 displayed only online customers as guest users. None of the WebSphere Commerce attributes of these users are available for the CSR to view. In a full-scale implementation, customer information can be viewed and the customers who are coming online and offline can be identified. This can be implemented by adding more listeners in the client side.

When customer information is available in the IBM Sales Center for any user, the Find Customer action can be invoked internally with the customer number and name, and a host of related actions can be performed.

Part 6

Reports

This part discusses the implementation of the WebSphere Commerce Analyzer and provides details about how to generate customized reports.

12

Installing, configuring, and running the WebSphere Commerce Analyzer

This chapter provides details about how to prepare the WebSphere Commerce Analyzer (WCA) and integrate it with WebSphere Commerce in order to view analytical data for customer service reports. This chapter also describes the installation, configuration, and running of the WCA.

12.1 Introduction to WebSphere Commerce Analyzer

WebSphere Commerce is shipped with the WCA, which is a powerful tool that you can use to perform business intelligence and analytics.

WebSphere Commerce analytics refers to the consolidation and analysis of data collected in the day-to-day operations of a business, which is then used as a basis for taking better business decisions and for competitive advantage. You can, for example, use business intelligence and analytics to analyze customer information and order information to determine the relationship between the customers and the average order value. If you determine that the returning customers have higher order values, you can implement a marketing campaign to target these customers.

There are two major reasons why you must generate and analyze customer service reports using WCA:

- WCA involves looking at historical performance. It stores large amounts of historical data that IBM WebSphere Commerce does not. The WebSphere Commerce server overwrites data in its database, where WCA retains it.
- Running complex Structured Query Language (SQL) queries against WebSphere Commerce database can cause performance issues on the Commerce server. Running the report SQL scripts against the WCA data mart database offloads the Commerce server.

Three types of users use the WCA:

- System administrators

  The system administrator installs, configures, and maintains the WCA.

- Business analysts

  The business analyst has data analysis expertise. For WCA, the business analyst works with the system administrator to define how the business reports are customized.

- Business managers

  The business manager is concerned with the store's business operations. Using the results of the WCA reports, the business manager develops the marketing strategy, determines the types of customers the store's marketing activities target, and plans promotional events and their associated advertising.

The WCA is an optional application included with WebSphere Commerce. When installed, the WCA provides a robust business intelligence solution designed to analyze and report your customers' activities.

12.2 Installing the WebSphere Commerce Analyzer

This section discusses the hardware and software requirements, and a step-by-step installation of the WCA.

Important: For performance reasons, ensure that the WCA and WebSphere Commerce are installed on two different machines.

12.2.1 WebSphere Commerce databases supported by WebSphere Commerce Analyzer

The WCA data mart can only exist on the DB2 Universal Database V8.2.3, but the WCA can connect to any of the following WebSphere Commerce database servers:

- IBM DB2 Universal Database, Enterprise Edition V8.2.3
- IBM DB2 Universal Database, Express Edition V8.2.3
- Oracle 10g Release 1, Enterprise Edition
- Oracle 10g Release 1, Standard Edition
- Oracle 9i Release 2, Enterprise Edition with Fix Pack 1
- Oracle 9i Release 2, Standard Edition with Fix Pack 1

12.2.2 Hardware and software prerequisites

Ensure that your system meets the following minimum hardware and software requirements before installing the WCA.

A Pentium 4 (3 GHz or higher) IBM-compatible personal computer. The computer must have the following:

- A minimum of 2 GB random access memory (RAM)
- A CD-ROM drive
- A graphics-capable monitor with a color depth of at least 256 colors and screen resolution of at least 1024 x 768 pixels
- A minimum triple disk space of WebSphere Commerce database size. A 100 GB free disk space is recommended for WCA operations
- A local area network (LAN) adapter that is supported by the TCP/IP
- Microsoft Windows Server® 2003, Enterprise Edition or Windows 2000 Professional Service Pack 4 or later, or Windows 2000 Server with Service Pack 4 or later

Ensure that the Windows locale is set to one of the following languages:

- de_DE - German
- en_US - English (United States)
- es_ES - Spanish
- fr_FR - French
- it_IT - Italian
- ja_JP - Japanese
- ko_KR - Korean
- pt_BR - Brazilian Portuguese
- zh_CN - Simplified Chinese
- zh_TW - Traditional Chinese

12.2.3 The WebSphere Commerce Analyzer installation program

The WCA installation program installs WCA and the DB2 Universal Database, Enterprise Server Edition V8.2.3, unless DB2 is already installed.

Perform the following tasks to install the WCA:

1. Close all the programs that are running.
2. Ensure that you are logged in as a Windows administrator.
3. Insert the WCA CD into the CD-ROM drive and double-click **setup.exe**.
4. Click **Next** in the Welcome window.
5. Review and accept the licence agreement and click **Next**.

6. If DB2 is not installed, DB2 install summary is displayed (Figure 12-1). Click **Next**.

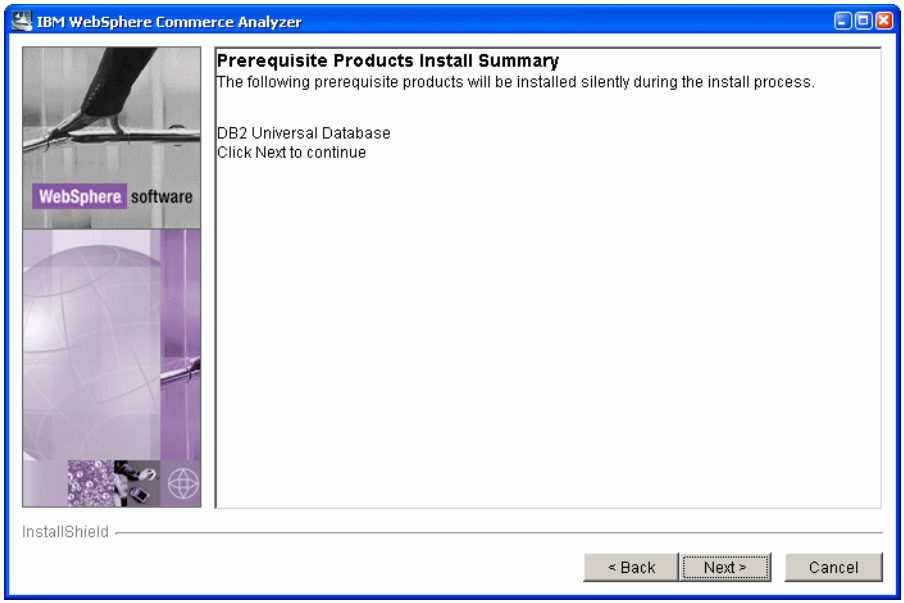


Figure 12-1 Prerequisite Products Install Summary

7. Select the location for the DB2 Universal Database CD (Figure 12-2). Click **Next**.

.

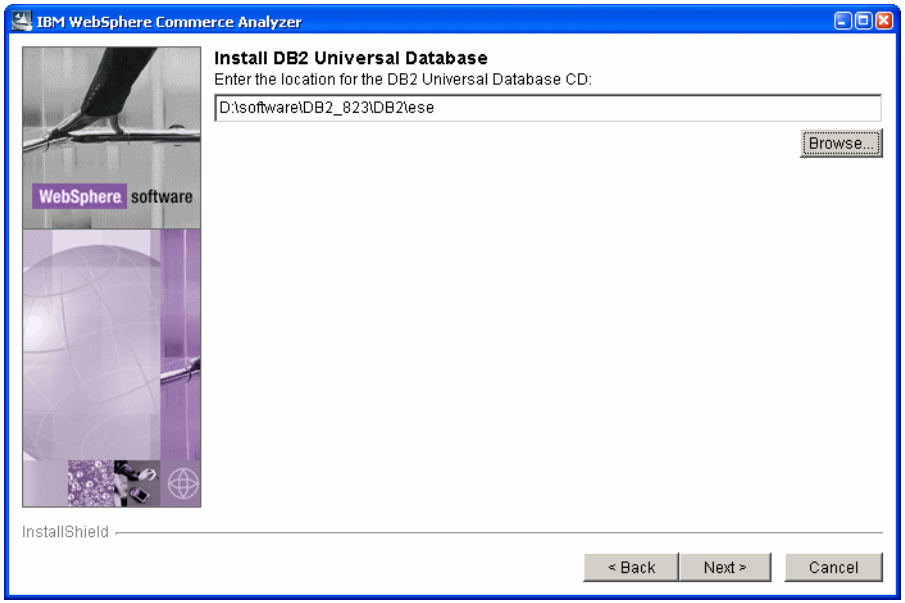


Figure 12-2 Entering the location for the DB2 Universal Database installation setup.exe

8. Select the directory (Figure 12-3) in which to install the DB2 Universal Database, or accept the default. Click **Next**.

.

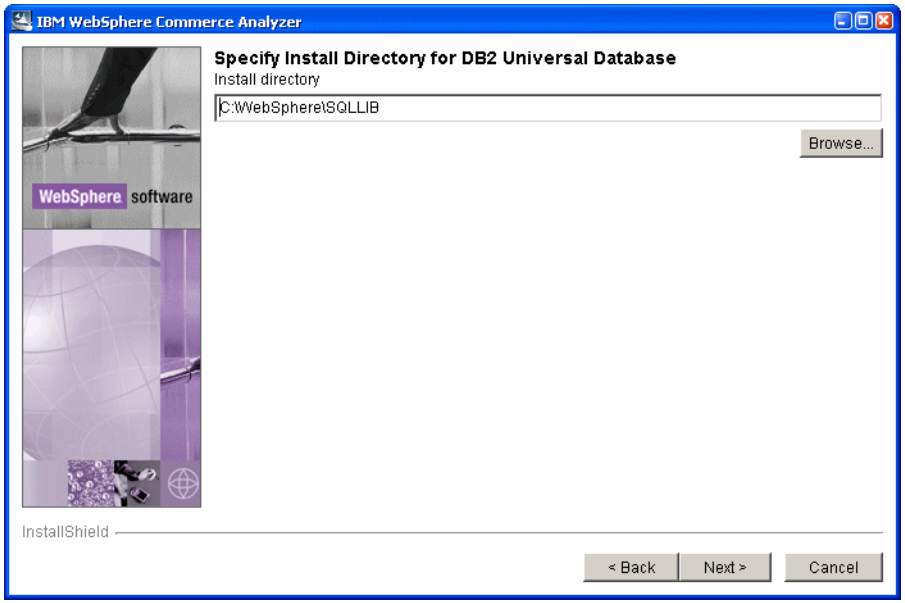


Figure 12-3 Specifying the install directory for the DB2 Universal Database

9. Enter the DB2 instance owner user ID and password (Figure 12-4) and click **Next**.

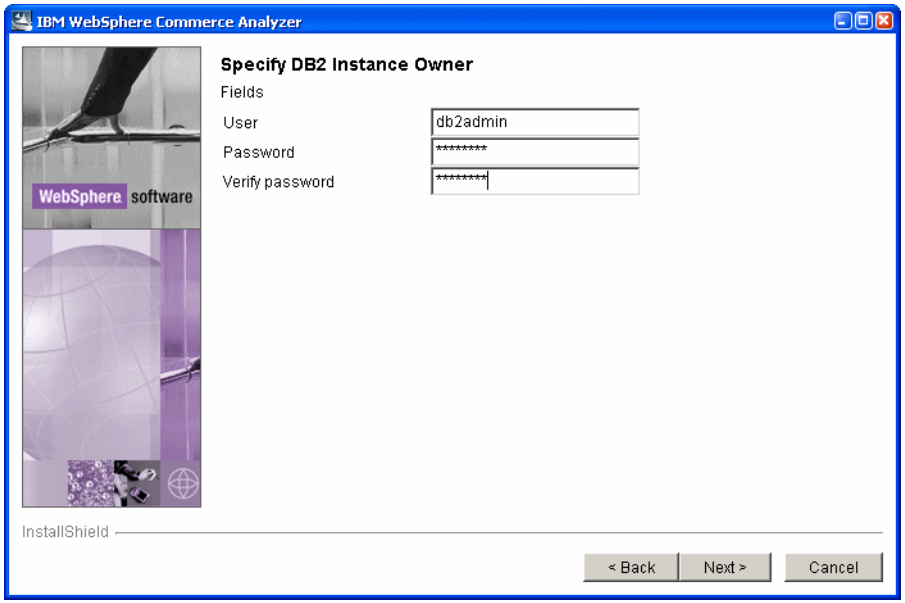


Figure 12-4 Specifying the DB2 instance owner

10. After the DB2 installation is complete, click **OK**. Click **Next** to continue.

11. A message indicating "`If necessary, insert WCA install CD`" is displayed. Insert the WCA install CD if it is not already inserted into the CD-ROM drive. Click OK.

12. Enter a directory name (Figure 12-5) in which to install the WCA. Specify a fully qualified host name and a port number (the default port is 8001) of the WebSphere Commerce machine that has the Information Center installed. Click **Next**.

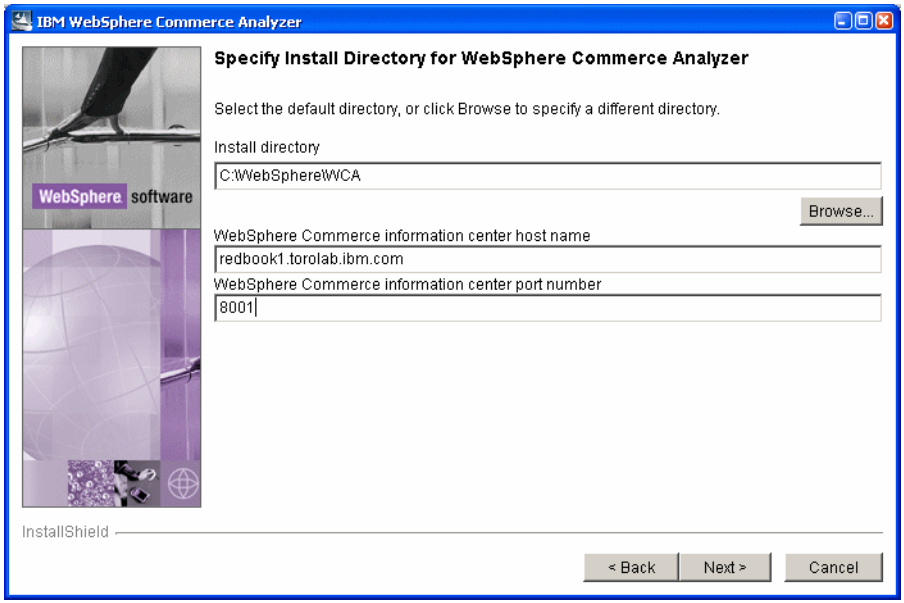


Figure 12-5 Entering the WCA install directory

13. In the review summary panel (Figure 12-6), click **Next** to start installing the WCA.

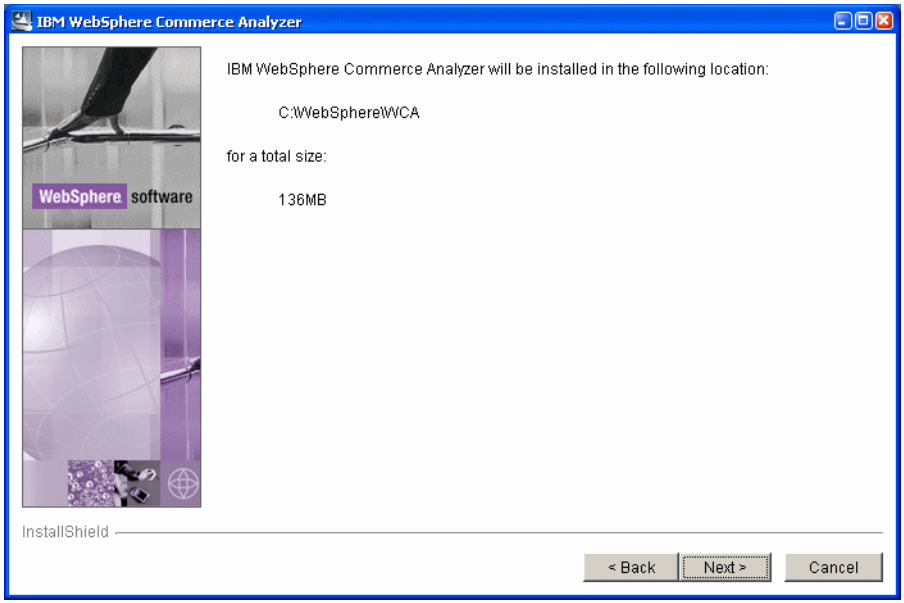


Figure 12-6 Reviewing the WCA installation summary panel

14. After the WCA installation is complete, click **Finish** to exit the wizard.

12.3 Preparing WebSphere Commerce for analytics

Prepare and integrate the WebSphere Commerce server to record the data for analytics before starting the WCA configuration. You must also collect information such as the database name, the fully qualified host name, the database user ID, the password, and so on from the WebSphere Commerce server because they are required during WCA configuration.

12.3.1 Configuring WebSphere Commerce to record analytics data

To configure the WebSphere Commerce system for analytics data, specific component listeners and events must be enabled using WebSphere Commerce Configuration Manager. If these components are not enabled, customer service reports will not contain any data.

Perform the following tasks to integrate the WebSphere Commerce system for analytics:

1. Launch the WebSphere Commerce Configuration Manager.

   Note: Before launching the WebSphere Commerce Configuration Manager, ensure that the WebSphere Commerce Configuration Manager service is started in Windows services.

2. Enter the Configuration Manager user ID (default user is configadmin) and password.
3. Expand **Host name** → **Commerce** → **Instance List** → **Instance_name** → **Components**.
4. Enable and start the listeners by performing the following tasks:
   a. Navigate to the OrderSubmit Event Listener, select **Enable Component** under the General tab and select **Start** under the Advanced tab. Click **Apply**.

b. Navigate to the Order Cost Calculation Event Listener (Figure 12-7), select **Enable Component** under the General tab and select **Start** under the Advanced tab. Click **Apply.**

Note: Changes in the instance configuration settings are published to the WebSphere Commerce application on exiting from the Configuration Manager.

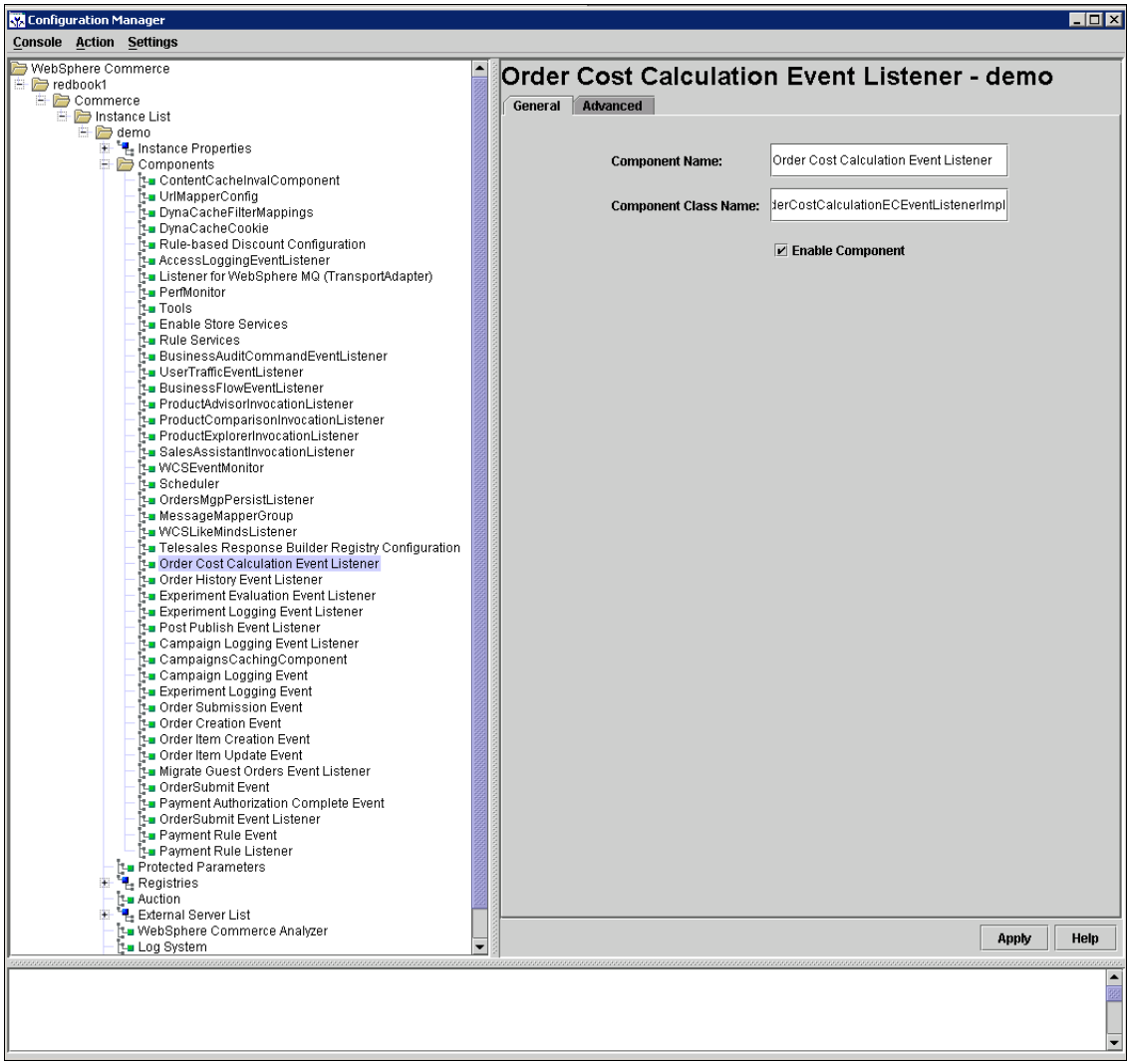


Figure 12-7 Enabling Order Cost Calculation Event Listener and other listeners for the Demo instance

5. Enable the following events:
   a. Navigate to the Order Submission Event and select **Enable Component** under the General tab. Click **Apply**.
   b. Navigate to the Order Creation Event (Figure 12-8) and select **Enable Component** under the General tab. Click **Apply**.
   c. Navigate to the Order Item Creation Event and select **Enable Component** under the General tab. Click **Apply**.

d. Navigate to the Order Item Update Event and select **Enable Component** under the General tab. Click **Apply**.

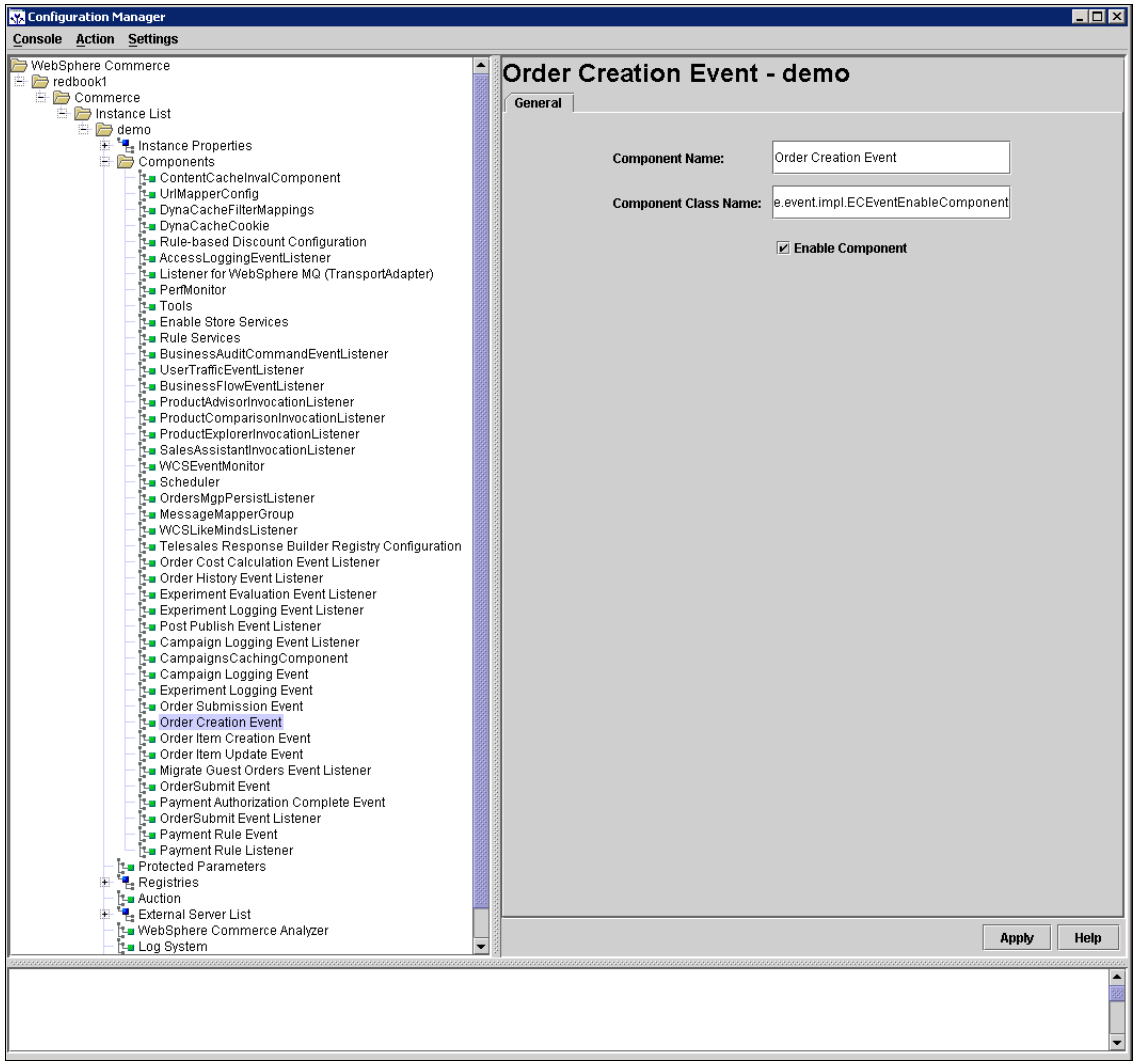


Figure 12-8 Enabling the Order Creation Event and other events for the Demo instance

Note: These component listeners and events are enabled to capture data relating to IBM Sales Center. Each action taken by IBM Sales Center is notified by these enabled component listeners and events. Ensure that these listeners and events are enabled.

6. Collapse **Components**.
7. Exit the WCA.
8. Click **OK** to publish the changes to the WebSphere Commerce application. This may take a couple of minutes.
9. For security reasons, the Configuration Manager Server is stopped. Click **OK**.
10. Restart the WebSphere Commerce application from the WebSphere Application Server Admin console.

12.3.2 Verifying the currency conversions setup in WebSphere Commerce

The WCA uses currency conversions to compare the sales value of orders within and between the stores. The CURCONVERT table in the WebSphere Commerce database contains information that the WCA uses to perform currency conversions.

Verify that the WebSphere Commerce CURCONVERT table is populated with the correct currency conversion before you configure the WCA. In the DB2 Universal Database that is used, this verification can be carried out by running the following SQL query with the currency conversion shown in Figure 12-9.

```
select storeent_id, fromcurr, tocurr from curconvert where storeent_id
= published_store_id
```

STOREENT_ID	FROMCURR	TOCURR
10001	USD	BRL
10001	USD	CAD
10001	USD	CNY
10001	USD	EUR
10001	USD	JPY
10001	USD	KRW
10001	USD	TWD

Figure 12-9 Default populated currency conversion in WebSphere Commerce database

12.3.3 Collecting the information required for WebSphere Commerce Analyzer configuration

In order to configure the WCA, the following information is required from the WebSphere Commerce server machine:

- The fully qualified host name of the machine where WebSphere Commerce is installed
- The database name of the WebSphere Commerce server database

- The currency that will be used for reports (ensure that proper currency conversions exist)
- The user ID and the password used to access the WebSphere Commerce server database

12.4 WebSphere Commerce Analyzer configuration

The configuration of the WCA is, in fact, the configuration of the WebSphere Commerce database with the WCA database on the basis of specific business logic. Basically, this configuration includes the following tasks:

1. Creating a data source connection to the WebSphere Commerce database
2. Creating a WCA database (data mart)
3. Setting up a replication
4. Selecting a store, report language, and currency
5. Selecting the store catalogs
6. Loading the language and financial periods

To start the WCA configuration, perform the following tasks:

1. Gather information from the WebSphere Commerce machine (refer to 12.3.3, “Collecting the information required for WebSphere Commerce Analyzer configuration” on page 329).
2. Ensure that the WebSphere Commerce database server is started.
3. Create a Windows administrator user on the WCA machine in order to own the data mart. Ensure that this user is a member of the Administrators group.
4. Log in to the WCA machine as a Windows administrator.

   Note: You can log in as any Windows administrator to run the configuration. You do not have to log in as the owner of the data mart.

5. Ensure that all the automatic DB2 services on the WCA machine are started.
6. To start the WCA configuration, select **Start** → **Programs** → **IBM WCA** → **Configuration Manager**. The IBM WCA Configuration Manager opens.

Note: If you encounter the error "`Can't find bundle for base name nls.stepmgr, locale en_CA()`" when starting the WCA configuration manager, change the language of the machine to one of the WCA-supported locales listed in 12.2.2, "Hardware and software prerequisites" on page 317, using the Regional Option and the Language Option in the Control Panel.

7. In the IBM WebSphere Commerce Database Access panel, input the following required fields (our example is shown in Figure 12-10):
   - Select **WebSphere Commerce** database type. For our example, we selected DB2.
   - In the Database Name field, type the WebSphere Commerce database name.
   - In the User Name field, type the user ID of the WebSphere Commerce database administrator.
   - In the Password field, type the password of the WebSphere Commerce database administrator.
   - From the Platform list, select the platform on which the WebSphere Commerce database resides.
   - In the Encryption key field, type a 16-digit hexadecimal number that will be used to encrypt the passwords for the WebSphere Commerce database and the WebSphere Commerce Analyzer data mart.
   - In the Key file full name field, type the name and the location of the file on your local system where the encryption key will be stored. This key file is required to run the WCA.
   - In the Host name field, type the fully qualified host name of the machine where the WebSphere Commerce database resides.
   - In the Port number field, type the port number for the WebSphere Commerce database. By default, this is 50000.
8. Click **Connect**.
9. Verify whether your connection to the WebSphere Commerce database is successfully created. Click **OK**.
10. If your connection is not successful, click **View Log** to view the problem. Fix the problem before continuing with the configuration.

11. After you are connected successfully, from the Schema list, select the correct WebSphere Commerce database schema if it has not been selected already.
12. Click **Next**.

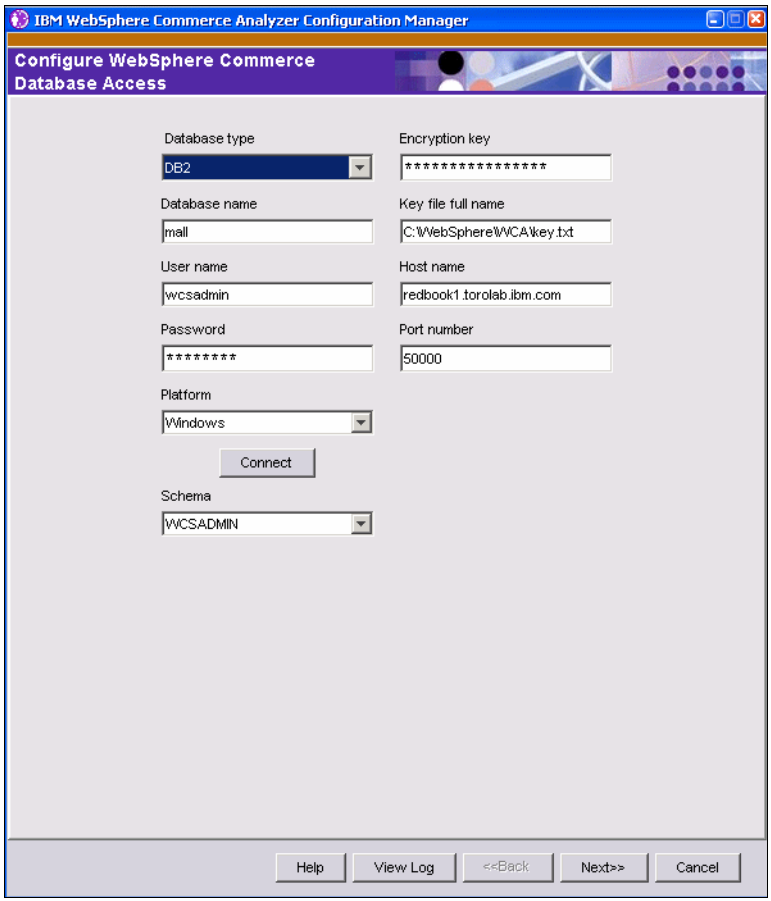


Figure 12-10 Configuring the WebSphere Commerce database access

13. In the Create WCA data mart panel, input the following required fields (our example is shown in Figure 12-11):
    - In the Data mart name field, type the name of the data mart (`wcamart`). This name must be eight characters or less and must contain only characters in the DB2 character set.
    - In the Data mart user field, type the name of the existing Windows administrator who will be the data mart user.

- In the Data mart password field, type the corresponding password for the user.
- From the Data mart location list, select the drive on which the data mart will reside.
- From the Table Space list, select the type of data storage for DB2 to use when it creates table spaces. You have two options:
  - Database-managed Storage
  - System-managed storage, which is the recommended and the default option

14. Click **Configure**.
15. Verify that the data mart is created successfully, and click **OK**.
16. If the data mart creation is not successful, click **View Log** to view the problem. Fix the problem before continuing with the configuration.
17. Click **Next**.

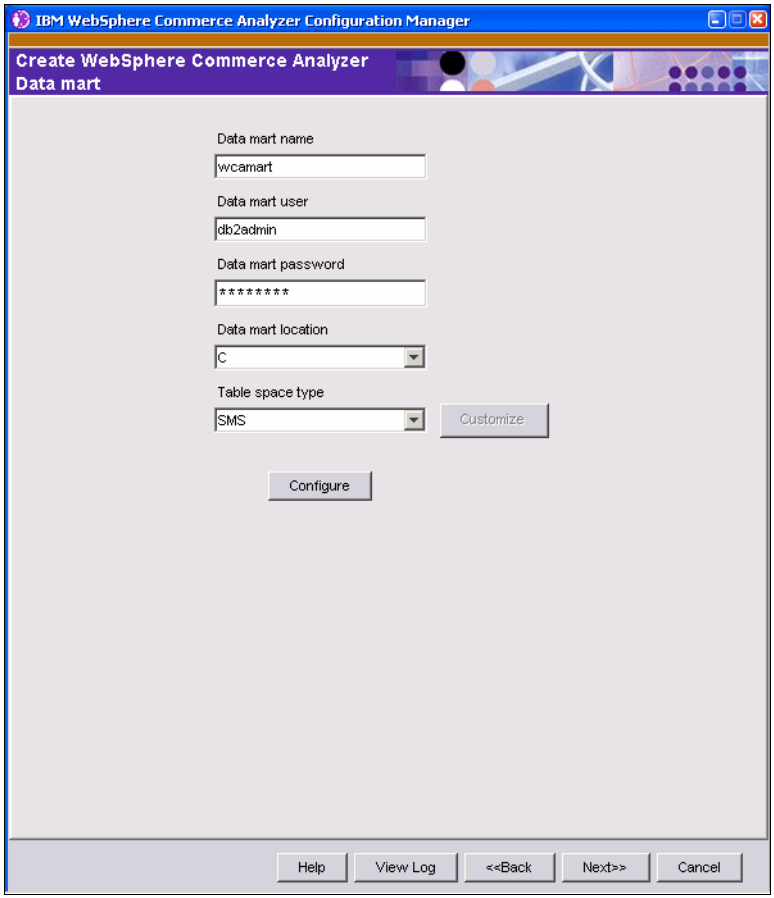


Figure 12-11 Creating the WCA data mart database

18. In the Setup Replication for Source Databases panel (Figure 12-12), click **Apply**.

> **Note:** This step may take a couple of minutes to be completed.

19. Verify that the replication setup is successful, and click **OK**.
20. If the replication setup is not successful, click **View Log** to view the problem. Fix the problem before continuing with the configuration, and click **Next**.

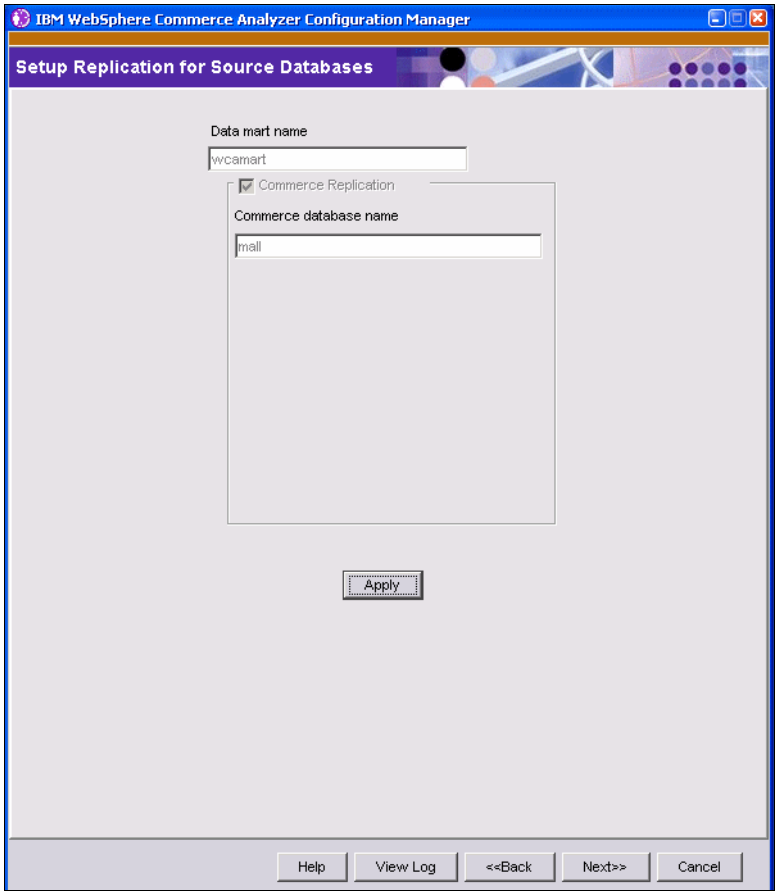


Figure 12-12 Setup replication for the source databases

21. In the Select Stores, Report Language, and Currency panel (see Figure 12-13), input the following required fields:
    - Select the store or stores for which you want to generate the reports by selecting the check box that corresponds to the store listed in the Store Name column.
    - In the Report Language field, the language of the locale is displayed. This is the language in which the reports will be generated.
    - From the Report currency list, select the currency in which the reports will be generated.
22. Click **Apply**.
23. Verify that the WCA store parameters have been successfully updated, and click **OK**.

24. If the update is not successful, click **View Log** to view the problem. Fix the problem before continuing with the configuration.

25. Click **Next**.

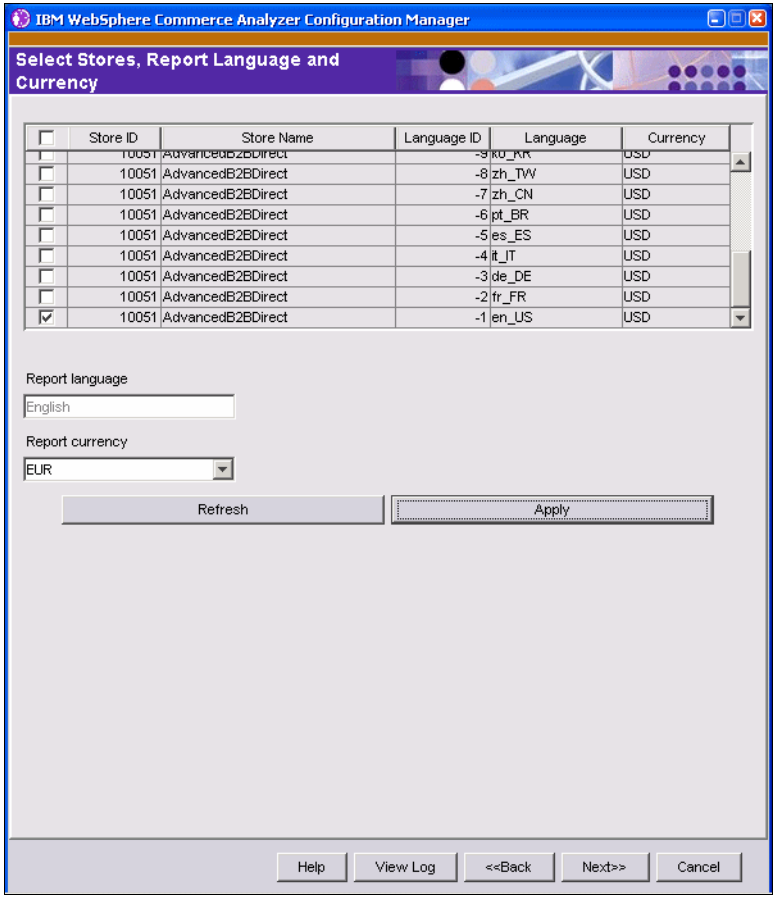


Figure 12-13 Selecting the stores and the report currency

26. In the Select Catalog panel (Figure 12-14), select the catalog or catalogs for which you want to capture the data for reporting.

27. Click **Apply**.

28. Verify that the selected catalog is saved successfully, and click **OK**.

29. If the catalog selection is not successful, click **View Log** to view the problem. Fix the problem before continuing with the configuration.

30. Click **Next**.

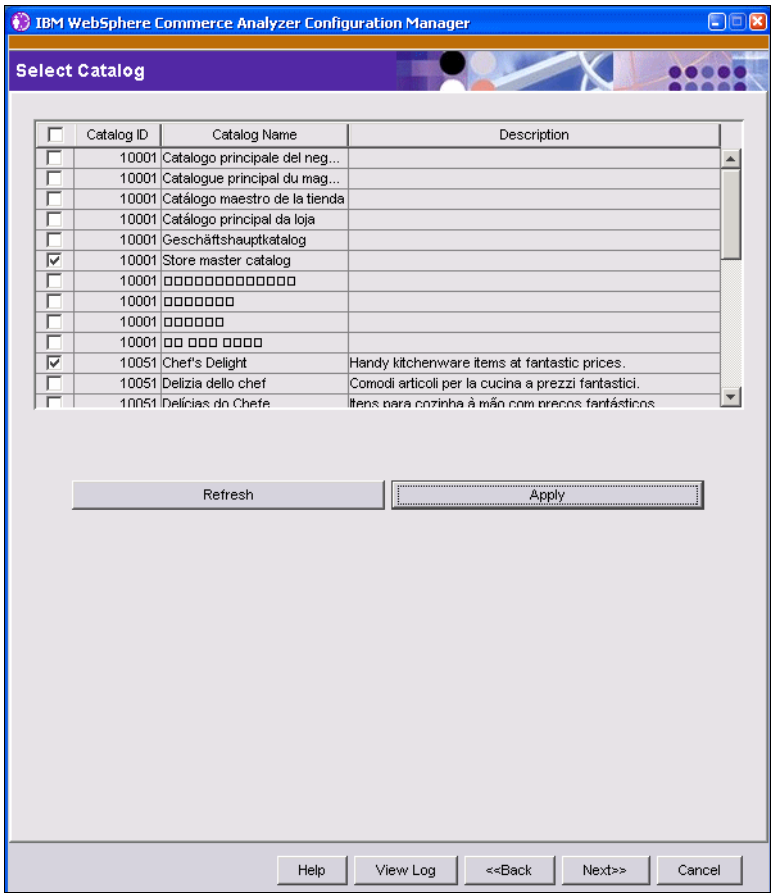


Figure 12-14 Selecting the catalog for which the report data will be captured

31. In the Load Language and Financial Periods panel (Figure 12-15), input the following required fields:

   – From the list, select the language in which you will be loading the reference texts relating to the financial periods. If you want to select multiple languages, complete all the fields, and click **Load**. Repeat for each language.

   – Select **Reference texts** to load the language reference texts. Ensure that the Reference text check box is selected.

– Select the day of the month on which the fiscal year starts.
– Select the month in which the fiscal year starts.

> **Note:** You cannot change the day and the month on which the fiscal year starts after you have selected them.

– If your fiscal year started in the previous year, select the **Fiscal year backshift** check box if it is not disabled.
– From the available list, select a year from which to start loading the financial periods.

> **Note:** You cannot change the year from which you are loading after you have selected it.

– From the available list, select the number of years to load. The maximum number of years to load is nine.

32. Click **Load** to load the financial periods and texts that you have selected.

> **Note:** If you want to change the start of the financial period after loading, change it in the Parameter Manager by selecting **Start** → **Programs** → **IBM WCA** → **Parameter Manager**.

33. Verify that the periods and texts have been successfully updated, and click **OK**.
34. If the load is not successful, click **View Log** to view the problem. Fix the problem before continuing with the configuration.
35. Click **Finish**.

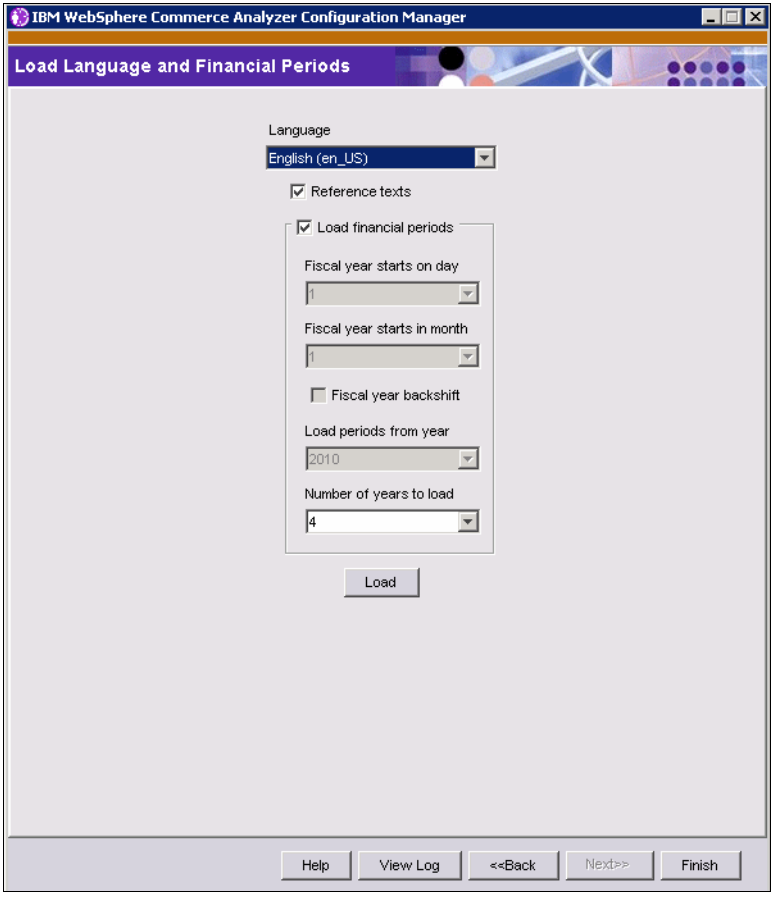


Figure 12-15 Loading the language and the financial periods

The WCA Configuration Manager Window closes.

12.5 Integrating the WebSphere Commerce Analyzer with WebSphere Commerce

To integrate the WCA with WebSphere Commerce, use the WebSphere Commerce Configuration Manager. Perform the following tasks:

1. Copy the following files from the /DB2_installdir/SQLLIB/java in the WCA machine to the /WC_installdir/instances/instance_name/lib folder in the WebSphere Commerce machine:
   - db2jcc.jar
   - db2jcc_license_cisuz.jar
   - db2jcc_license_cu.jar

   If the lib folder does not exist, create it.

   Note: The WCA runs with the DB2 Universal Database (DB2 UDB). Even if your WebSphere Commerce database is Oracle and not DB2, you must copy the DB2 Java Database Connectivity (JDBC™) drivers to the WebSphere Commerce machine to configure the WebSphere Commerce data source for the WCA data mart database access. This data source is used by the WCA to retrieve data and view business intelligence reports.

2. Ensure that the lib folder and its contents have the same access settings as other folders in the WebSphere Commerce instance that are accessible by the WebSphere Application Server user group.
3. Launch the WebSphere Commerce Configuration Manager.
4. In the WebSphere Commerce Configuration Manager, select the WebSphere Commerce *instance_name*, and then the WCA.
5. At this point, perform the following tasks (our example is shown in Figure 12-16):
   - Select **Yes** against the Is WebSphere Commerce Analyzer installed field.
   - In the Data mart name field, type the name of the WCA data mart database name.
   - In the Data mart user field, type the name of the Windows administrator who is the data mart user.
   - In the Data mart user password field, type the corresponding password for the user.
   - In the Confirm data mart user password field, type the corresponding password for the user again.

– In the JDBC Driver Location field, enter the path of the folder containing the JDBC driver for the WebSphere Commerce server database, for example, the same one specified in step 1, which is `/WC_installdir/instances/instance_name/lib`.

– In the Data mart host name field, type the fully qualified host name of the machine where the WCA data mart resides.

– In the Data mart port field, enter the port number on which the WCA is listening. For DB2, the default port number is 50000.

Ensure that the WebSphere Application Server is running on the WebSphere Commerce server, and click **Apply**.

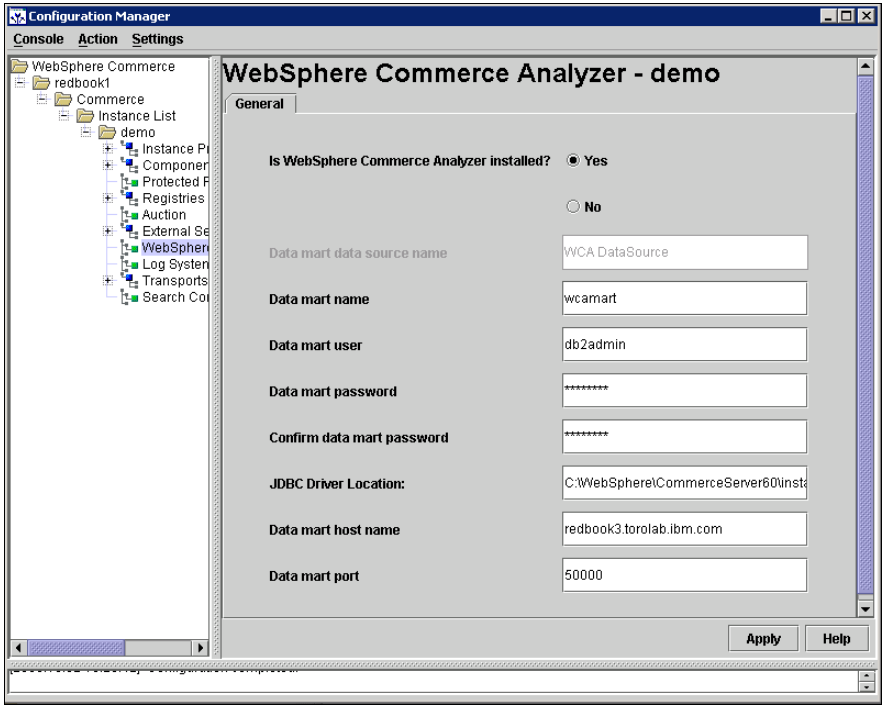


Figure 12-16 Information to create WCA DataSource using the Configuration Manager

– A window similar to that shown in Figure 12-17 is displayed. Click **OK**. A data source is created in WebSphere Application Server for the data mart.

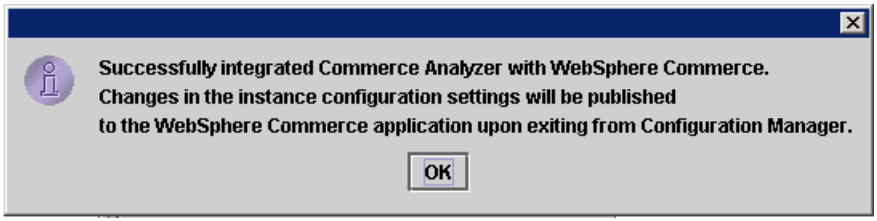


Figure 12-17 Successful integration of the WCA with WebSphere Commerce

6. Exit the WebSphere Commerce Configuration Manager. Click **OK** to publish the changes to the WebSphere Commerce application. This may take a couple of minutes.
7. For security reasons, the Configuration Manager Server is stopped. Click **OK**.
8. Restart the WebSphere Commerce server.
9. (Optional task) Using the IBM WebSphere Application Server Admin console, verify whether the WCA data source is connecting successfully to the WCA data mart. If the connection is not successful, view the Java virtual machine (JVM™) logs for troubleshooting to resolve the problem. Otherwise, the WCA will not be able to retrieve the data from the data mart to view the reports.

12.6 Running the WebSphere Commerce Analyzer

To run the WCA, it is important to ensure that the capture program (ASNCAP) is running successfully on the WebSphere Commerce machine and that the DB2 database service is running on the Windows services of the WebSphere Commerce machine and on the WCA machine.

12.6.1 Running the capture program on the WebSphere Commerce database

To run the capture program on the WebSphere Commerce database, perform the following tasks:

1. Before starting the capture program, update the configuration in the WebSphere Commerce database by performing the following tasks:
   a. Stop the WebSphere Commerce application from the WebSphere Application Server Admin console.

b. Run the DB2 command-line processor.

c. Run the following command to see the configuration of the WebSphere Commerce database:

```
get db config for wc_database_name
```

d. Verify the LOGRETAIN setting:

```
Log retain for recovery enabled              (LOGRETAIN) = OFF
```

e. If it is not set to RECOVERY, modify the configuration by running the following commands:

```
connect to wc_database_name
update db config using LOGRETAIN RECOVERY
```

f. Take a backup of the WebSphere Commerce database at this point:

```
backup db wc_database_name
```

g. Verify whether you are able to connect to the WebSphere Commerce database.

2. Open the Windows command prompt and start the capture program:

```
asncap wc_database_name startmode=cold
```

3. Start the WebSphere Commerce application from the WebSphere Application Server Admin console.

12.6.2 Running the replication and the extract, transform, and load processes

To start the WCA for the replication and the extract, transform, and load (ETL) processes in the WCA machine, select **Start** → **Programs** → **IBM WCA** → **Run WebSphere Commerce Analyzer** → **Start WCA**. Leave the window open. You can view the progress of the tasks (Figure 12-18).

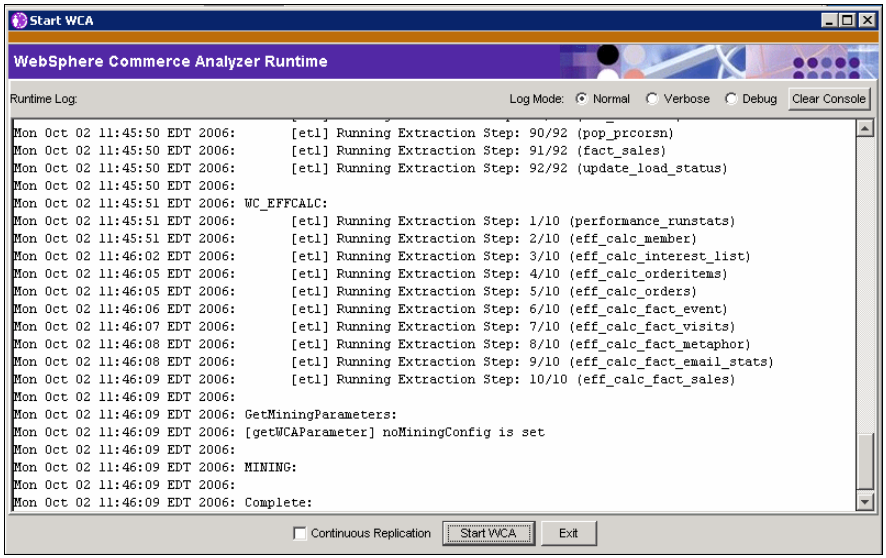


Figure 12-18 WCA run completed

Note: The WCA must run successfully at least twice for it to see any data in the customer service reports.

13

Developing and customizing customer service reports

This chapter explains in detail about how to develop and customize customer service reports that are displayed in the WebSphere Commerce Accelerator (WCA). This chapter also provides an overview of all the customer service reports that are available out-of-the-box in WebSphere Commerce.

13.1 WebSphere Commerce customer service reports

The WCA provides default reports on customer service representatives' (CSRs) performance and activities, revenue, sales, orders, price overrides, profit, quotes, and so on. These reports display information that is either based on an individual CSR or on an entire CSR team, and can be viewed using the WCA either under Operations (business-to-consumer) → Customer Service Reports or under Sales (business-to-business) → Customer Service Reports menu item.

By default, these reports can be accessed by the following WebSphere Commerce user roles:

- Site Administrator (siteAdmin role)
- customer service representative (CSR) (cusRep role)
- Seller (seller role)
- Sales Manager (salesMgr role)
- Customer Service Supervisor (cusSup role)

> **Note:** The CSR role will only have access to the Personal Revenue, Profit, and Ranking report that displays the revenue and profit generated by the CSR.

Two types of reports are available in the WCA, one that reads data from the WCA data mart database and is therefore called a business intelligence report, and another that retrieves and displays data from the WebSphere Commerce operational (production) database and is therefore called the operational report.

The following individual and team business intelligence customer service reports are available in WCA:

- Personal Revenue, Profit, and Ranking (individual CSR report)

  This report allows CSRs to view the revenue and profit they have generated.

  > **Note:** By default, only users with Site Administrator, Seller, or CSR role can access this report.

- Revenue, Profit, and Ranking (individual CSR or CSR team report)

  This report allows CSR supervisors to view the revenue and profit generated by a CSR or by a CSR team.

- Revenue, Profit, and Ranking (CSR team report)

  This report provides data about the revenue and the profit generated by a CSR team.

- Price Quotes (individual CSR report)

  This report lists the prices a CSR has quoted to customers.

- Quotes to Order Conversion Rate (individual CSR or CSR team report)

  This report lists the number of price quotes generated by an individual CSR or by a CSR team. These are converted to orders and the corresponding conversion rate.

- Price Override (individual CSR report)

  This report details the price override made to items by a particular CSR.

- Price Override (CSR team report)

  This report lists the total amount of price override made by a particular CSR team.

- Price Override Summary (individual CSR or CSR team report)

  This report lists the total amount of price override made by a particular CSR or by a CSR team.

- Revenue by Product Category (individual CSR or CSR team report)

  This report details the sales revenue by product category by a particular CSR or by a CSR team.

- Shipped Orders (individual CSR report)

  This report details the total shipped orders sold by a particular CSR.

- Orders Pending Fulfillment (individual CSR report)

  This report details the pending orders sold by a particular CSR.

  Note: Pending orders are orders that have not been shipped yet.

The following operational customer service reports are available in WCA:

- (Business-to-business only) Organization Contact (individual CSR report)

  This report lists the organization contacts assigned to a particular CSR.

- Customer Territories (individual CSR report)

  This report lists the customer territories assigned to a particular CSR.

- Customer Territories (CSR team report)

  This report lists the customer territories assigned to a particular CSR team.

- Daily Sales (CSR team report)

  This report details the daily sales generated by a particular CSR team.

13.2 Developing customer service reports

This chapter describes the step-by-step development of one of the default business intelligence customer service reports, the Personal Revenue, Profit, and Ranking report.

Following are the tasks that must be performed to develop a new customer service report:

- Write two new JavaServer Page (JSP) files (report Input jsp and report Output jsp)
- Write three new Extensible Markup Language (XML) files (InputDialog xml, OutputDialog xml, and Report xml)
- Update the WCA common files
- Write and load the access control for the new report views

13.2.1 Writing the JavaServer Page files

You must have two JSP files for each report. These files must be located in the *WC_profiledir/*installedApps*/cell_name/*WC*_instance_name.ear/*CommerceAcce lerator.war/tools/bi directory. This directory already contains report JSP files for the rest of the WebSphere Commerce default reports. The existing JSP files can be used as examples for developing new reports.

Perform the following tasks:

1. Create the JSP file from which the report is requested (input jsp file). This JSP file collects the input data from the customer. The report is then generated based on this input data. The file name for this input JSP file must follow the naming convention, *reportName*ReportInputView.jsp.

   Example 13-1 shows the sample code explaining how to create this JSP file, which, in our case was biCSRIndividualRevenueReportInputView.jsp.

Example 13-1 Writing the new report input view JSP file

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">

//Importing the required packages
<%@page import="java.util.*" %>
<%@page import="com.ibm.commerce.tools.util.*" %>
<%@page import="com.ibm.commerce.tools.xml.*" %>
<%@include file="/tools/reporting/common.jsp" %>
<%@include file="/tools/reporting/ReportStartDateEndDateHelper.jspf" %>
<%@include file="/tools/reporting/ReportFrameworkHelper.jsp" %>
```

```
<%
//Instance of Command Context which provides the user ID of the viewing
//user and locale where this report is executed.
   CommandContext biCommandContext = (CommandContext)
   request.getAttribute(ECConstants.EC_COMMANDCONTEXT);
   Locale biLocale = biCommandContext.getLocale();
   Long userId = biCommandContext.getUserId();

//biNLS is the locale specific hashtable which is supposed to have all
//locale specific text display informations.
   Hashtable biNLS =
   (Hashtable)com.ibm.commerce.tools.util.ResourceDirectory.lookup("bi.
   biNLS", biLocale);
%>

//Body of the input page starts here
<HTML>
<HEAD>

//Using the defined style sheet in WebSphere Commerce
   <link rel=stylesheet href="<%= UIUtil.getCSSFile(biLocale) %>"
   type="text/css">
   <%=fHeader%>
   <TITLE><%=biNLS.get("CSRIndividualRevenueReportWindowTitle")%></TITL
   E>

//CSRIndividualRevenueReportWindowTitle is the text key which should be
//defined in the BINLS locale specific properties file. Same for the
//other text keys also.

//Importing the java scripts defined for reporting.
   <SCRIPT SRC="/wcs/javascript/tools/common/Util.js"></SCRIPT>
   <SCRIPT SRC="/wcs/javascript/tools/common/DateUtil.js"></SCRIPT>
   <SCRIPT SRC="/wcs/javascript/tools/common/SwapList.js"></SCRIPT>
   <SCRIPT
SRC="/wcs/javascript/tools/reporting/ReportHelpers.js"></SCRIPT>

   <SCRIPT>

//Intializing local variables. Called every time the page is loaded.
function initializeValues() {
onLoadStartDateEndDate("enquiryPeriod");

//Loads the start date and end date time period for user on input page.
   ResetValues();
```

```
   onLoadOrderByOption("myHelperIndividualCSRRevenue");

//Generating order by option on the input page.
   if (parent.setContentFrameLoaded)
      parent.setContentFrameLoaded(true);
    }

//Called every time a user exits from this page.
function savePanelData() {
   var selectedSort = 'CSRNAME';

////////////////////////////////////////////////
// Specify the report framework particulars
////////////////////////////////////////////////

setReportFrameworkOutputView("DialogView");
//Specifying the report output dialog xml which is supposed to call
//output jsp to load the report data.
setReportFrameworkParameter("XMLFile","bi.biCSRIndividualRevenueReportO
utputDialog");

//Saving the report XML name which is executed once the input page gets
//data.
setReportFrameworkReportXML("bi.biCSRIndividualRevenueReport");

//Storing the selected input data in time period
saveStartDateEndDate("enquiryPeriod");

////////////////////////////////////////////////////
// Specify the report framework particulars
////////////////////////////////////////////////////

   if(returnOrderByDesc("myHelperIndividualCSRRevenue"))
      selectedOrder = 'DESC';

//Setting the report sql lid name which will be executed in report xml.
   setReportFrameworkReportName("biCSRIndividualRevenueReport");

////////////////////////////////////////////////////
// Specify the report specific parameters and save
////////////////////////////////////////////////////

setEndDate();
setReportFrameworkParameter("StartDate",
returnStartDateAsTimestamp("enquiryPeriod"));
```

```
setReportFrameworkParameter("EndDate",
returnEndDateAsTimestamp("enquiryPeriod"));

//Assigning the values to variables which are being used in report xml.
setReportFrameworkParameter("Order", selectedOrder);
setReportFrameworkParameter("user_id", <%=userId%>);
saveReportFramework();
top.saveModel(parent.model);
return true;
}

//Validating input data before going to the next page.
function validatePanelData(){
   if (validateStartDateEndDate("enquiryPeriod") == false)
      return false;
   return true;
}

//Resetting the variables of time period.
   function ResetValues(){
      document.enquiryPeriod.StartDateEndDateHelperYearSD.value = "";
      document.enquiryPeriod.StartDateEndDateHelperMonthSD.value = "";
      document.enquiryPeriod.StartDateEndDateHelperDaySD.value = "";
      document.enquiryPeriod.StartDateEndDateHelperYearED.value = "";
      document.enquiryPeriod.StartDateEndDateHelperMonthED.value = "";
      document.enquiryPeriod.StartDateEndDateHelperDayED.value = "";
   }

function setEndDate(){
      if((document.enquiryPeriod.StartDateEndDateHelperYearSD.value !=
"" &&
document.enquiryPeriod.StartDateEndDateHelperMonthSD.value != "" &&
document.enquiryPeriod.StartDateEndDateHelperDaySD.value != "" ) &&
          (document.enquiryPeriod.StartDateEndDateHelperYearED.value
== "" &&
         document.enquiryPeriod.StartDateEndDateHelperMonthED.value ==
"" && document.enquiryPeriod.StartDateEndDateHelperDayED.value == "" ))
      {

parent.alertDialog("<%=UIUtil.toJavaScript(biNLS.get("EndDateIsCurrentD
ate"))%>");
         document.enquiryPeriod.StartDateEndDateHelperYearED.value =
getCurrentYear();
         document.enquiryPeriod.StartDateEndDateHelperMonthED.value =
getCurrentMonth();
```

```
        document.enquiryPeriod.StartDateEndDateHelperDayED.value =
getCurrentDay();
}
   }



////////////////////////////////////////////////////
// Validate is done by the HTML radio button
////////////////////////////////////////////////////

function validateOrderByOption(container)
   {
      return true;
   }

////////////////////////////////////////////////
// initialize function for the status dates
////////////////////////////////////////////////

function onLoadOrderByOption(container)
   {
     var myContainer = parent.get(container, null);

// If this is the first time set it to the default.
     myContainer = new Object();

     myContainer.StatusChosen = 1;

     with (document.forms[container]) {
      orderBy[0].checked = true;
     }
     parent.put(container, myContainer);
     return;
   }
//////////////////////////////////////////
// Return the Orderby Status Chosen
//////////////////////////////////////////

function returnOrderByDesc(container) {
      return document.forms[container].orderBy[0].checked;
   }
</SCRIPT>
</HEAD>
<BODY ONLOAD="initializeValues()" CLASS=content>
```

```
//Displaying the report window title by reading hashtable. These text
//keys must be defined in locale specific properties file.
   <H1><%=biNLS.get("CSRIndividualRevenueReportWindowTitle") %></H1>
   <%=biNLS.get("CSRIndividualRevenueReportInputDescription")%>
   <br> <br>
   <b><%=biNLS.get("timePeriod")%></b>
   <p></p>
   </DIV>
   <table border="0" bordercolor="black" CELLPADDING="0"
CELLSPACING="0" width="470">
      <tr>
         <td>
            <br>
            <DIV ID=pageBody STYLE="display: block; margin-left: 0">
               <%=generateStartDateEndDate("enquiryPeriod", biNLS,
null)%>
            </DIV>
         </td>
      </tr>
   </table>
   <br>
   <table border="0" bordercolor="black" CELLPADDING="0"
CELLSPACING="0" width="210">
      <tr height=25>
         <td align="left">

//Some common text keys are already defined in properties file like
//orderby, descend, ascend etc. Verify them before writing in
//properties file.
         <B><%=biNLS.get("orderby")%></B>
         </td>
      </tr>
   </table>
   <BR>
   <DIV ID=pageBody STYLE="display: block; margin-left:0">
      <FORM NAME=myHelperIndividualCSRRevenue>
         <TABLE border=0 bordercolor=black CELLPADDING=0 CELLSPACING=0
width=200>
            <TR HEIGHT=5>
               <TD ALIGN=left VALIGN=TOP>
                  <INPUT TYPE=RADIO  NAME=orderBy VALUE=All id=ord1>
                  <label for= ord1>
                  <%=biNLS.get("descend")%> </label>
                   </INPUT>
```

```
                <BR>
                <BR>
                <INPUT TYPE=RADIO NAME=orderBy VALUE=PayA id=ord2>
                <label for= ord2>
                <%=biNLS.get("ascend")%></label>
                </INPUT>
                <BR>
              </TD>
            </TR>
          </TABLE>
        </FORM>
      </DIV>
</BODY>
</HTML>
```

2. Create the JSP file to display the report (output jsp file). This JSP file is used to visualize the report to the customer. The file name of this output JSP file must follow the naming convention *reportName*OutputView.jsp.

   This output JSP file can import the helper JSP files. You can customize the helper files as necessary. Example 13-2 shows the sample code that explains how to create this JSP file, which, in our case was biCSRIndividualRevenueReportOutputView.jsp.

Example 13-2 Writing the new report output view JSP file

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<%@page import="java.util.*" %>

//Importing required files.
<%@include file="/tools/common/common.jsp" %>
//Here we are importing the Helper JSP which helps to display the
//report on output page.
<%@include file="/tools/reporting/ReportHeaderSummaryHelper.jsp" %>
<%!
   private String generateSpecificOutputInputCriteria2(String
reportPrefix, Hashtable reportsRB, Locale locale)
   {
      StringBuffer buff = new StringBuffer("");
      Hashtable aReportDataBeanEnv = aReportDataBean.getEnv();
//An instance of data bean helps to collect the input data from input
//page.
      String StartDate                    = (String)
aReportDataBeanEnv.get("StartDate");
      String EndDate                      = (String)
aReportDataBeanEnv.get("EndDate");
```

```
      Timestamp currentTime = TimestampHelper.getCurrentTime();

      buff.append("   <DIV ID=pageBody STYLE=\"display: block;
margin-left: 0\">");
      buff.append("<b>" +
reportsRB.get("ReportOutputViewReturnSelectedDateRange") + "</b> ");
      buff.append(getFormattedDate(StartDate,locale) + " ");
      buff.append("<b>" +
reportsRB.get("ReportOutputViewReturnSelectedDateRangeTo") + "</b> ");
//These are the text keys above which are common for each report.

      buff.append(getFormattedDate(EndDate,locale) + "<BR>");
      buff.append("   </DIV>");
      buff.append("   <DIV ID=pageBody STYLE=\"display: block;
margin-left: 0\">");

//Here we generate text keys like reportPrefix +
//ReportOutputViewRunDateTitle into properties file. report prefix
//is a common key which is supposed to be prefixed before each text
//keys in report output jsp.
      buff.append("<b>" + reportsRB.get(reportPrefix +
"ReportOutputViewRunDateTitle") + "</b> ");
      buff.append((String)
TimestampHelper.getDateFromTimestamp(currentTime, locale) + " ");
      buff.append((String)
TimestampHelper.getTimeFromTimestamp(currentTime) + "<BR>");
      buff.append("   </DIV>   <BR><BR>");

      return buff.toString();
   }
%>
<%

//Generating a report prefix which is used to find out the text
//displays in hashtable according to report.
String reportPrefix = "CSRIndividualRevenue";
   CommandContext biCommandContext = (CommandContext)
request.getAttribute(ECConstants.EC_COMMANDCONTEXT);
   Locale biLocale = biCommandContext.getLocale();
   Hashtable biNLS =
(Hashtable)com.ibm.commerce.tools.util.ResourceDirectory.lookup("bi.biN
LS", biLocale);
%>

<HTML>
```

```
   <HEAD>
   <link rel=stylesheet href="<%= UIUtil.getCSSFile(biLocale) %>"
type="text/css">

//Generating report header informations here to display in header.
   <%=generateHeaderInformation(reportPrefix, biNLS, request)%>
   </HEAD>

   <body class="content"
onload="javascript:parent.setContentFrameLoaded(true)">
   <%=generateOutputHeading(reportPrefix, biNLS)%>
   <%=generateSpecificOutputInputCriteria2(reportPrefix, biNLS,
biCommandContext.getLocale())%>

//Generating the data table which is displayed on output page.
   <%=generateOutputTable(reportPrefix, biNLS,
biCommandContext.getLocale())%>
</BODY>
</HTML>
```

13.2.2 Writing the Extensible Markup Language files

You must have three XML files for each report. These files must be located in the *WC_profiledir/*installedApps*/cell_name/*WC*_instance_name.ear/*xml/tools/bi directory. This directory already contains report XML files for the rest of the WebSphere Commerce default reports. The existing XML files can be used as examples for developing new reports.

Perform the following tasks to write the XML files:

1. Create the XML file to display the text for the input JSP file. Example 13-3 shows the sample code that explains how to create this XML file, which, in our case was biCSRIndividualRevenueReportInputDialog.xml.

Example 13-3 XML file to display text for the input JSP file

```
//Defining the hashtable instance.
<dialog resourceBundle="bi.biNLS"
         windowTitle="CSRRevenue"
         finishURL="GenericReportController" >
  <panel name="report"
//The url which is supposed to be called to load the input page.
         url="biCSRIndividualRevenueReportInputView"
//Here report framework is getting data whether to load buttons
         hasFinish="NO"
```

```
      hasCancel="YES"
//Defining the help URL for the input page.
         helpKey="MC.bi.IndividualRevenueInput.Help" />
  <button name="viewReport" action="finish();"/>
</dialog>
```

2. Create the XML file to display text for the output JSP file. Example 13-4 shows the sample code that explains how to create this XML file, which, in our case was biCSRIndividualRevenueReportOutputDialog.xml.

Example 13-4 XML file to display text for the output JSP file

```
<dialog resourceBundle="bi.biNLS"
          windowTitle="CSRRevenue"
          finishURL="" >
  <panel name="report"
//The url is passed to load the report output page to show data.
url="biCSRIndividualRevenueReportOutputView"
         passAllParameters="true"
//Parameters which decide to show buttons on output page.
hasFinish="NO"
hasCancel="NO"
//Help file for output page.
helpKey="MC.bi.IndividualRevenueOutput.Help" />
  <button name="ReportOutputViewPrintTitle"
          action="CONTENTS.printButton()" />
  <button name="ReportOutputViewOkTitle"
          action="CONTENTS.okButton()" />
</dialog>
```

3. Create the XML file for the SQL used to retrieve the report data. Ensure that the SQL statement runs successfully before writing it into the report XML file. Example 13-5 shows the sample code that explains how to create this XML file, which, in our case was biCSRIndividualRevenueReport.xml.

Example 13-5 XML file for the SQL used to retrieve the report data

```
<Reporting>
  <Report reportName="biCSRIndividualRevenueReport" online="true"
dataSourceName="WCA DataSource">
      <comment></comment>
      <SQLvalue>

//In between these tags, we place the report SQL which is used
//to retrieve report data from database.
```

```
      </SQLvalue>

//Here we define the customized display as per need.
   <display>
      <standardInfo>

//Defining the hashtable instance which is used for text display.
         <resourceBundle>bi.biNLS</resourceBundle>

//Title of report display which should be defined in properties file.
         <title>biCSRRevenueReport</title>
         <message>biCSRRevenueReport</message>
      </standardInfo>

//In the below tag, we can define the style of report layout with
//options of choosing different colors, text formats etc.
      <userDefinedParameters>
         <THStyle>TH { font-family: Arial, Helvetica, Sans-serif;
font-size: 9pt; line-height: 9pt; color : white; background-color :
darkblue; font-weight : bold; }</THStyle>
         <TDStyle>TD { font-family : Arial, Helvetica, Sans-serif;
font-size : 9pt; line-height: 9pt; color : Black; }</TDStyle>
         <spaceBetweenColumns>25</spaceBetweenColumns>
         <columnDefaultAttributes>
            <displayInReport>true</displayInReport>
            <columnWidth>0</columnWidth>
            <maxEntryLength>999</maxEntryLength>
            <columnType>string</columnType>
            <columnOptions>ALIGN=LEFT HEIGHT=20 NOWRAP</columnOptions>
             <displayInHeader>False</displayInHeader>
         </columnDefaultAttributes>

//Here we can define the report column names with their types and
//alignment. We Can have as many number of columns defined here as
//report SQL query is returning. All these column name keys should
//be defined into properties file.
         <columns>
            <columnKey>C0</columnKey>
            <columnName>CSRIndividualRevenueReportCSRID</columnName>
            <columnOptions>ALIGN=LEFT HEIGHT=20 NOWRAP</columnOptions>
            <columnType>string</columnType>
         </columns>
         <columns>
            <columnKey>C1</columnKey>
<columnName>CSRIndividualRevenueReportCSRName</columnName>
```

```
            <columnOptions>ALIGN=LEFT HEIGHT=20 NOWRAP</columnOptions>
            <columnType>string</columnType>
          </columns>
        </userDefinedParameters>
    </display>
   </Report>
</Reporting>
```

13.2.3 Updating the common files to reflect the new report

There are a few common files that are used by all the reports. Update these common files with the new report to reflect it in the Accelerator:

- Update the resources.xml file with the new report XML file. This is the XML file that contains information about all the report XML files.

  Update the resources.xml file located in the *WC_profiledir/*installedApps*/cell_name/*WC*_instance_name.ear*/xml/tools/bi directory with the three XML files, the creation of which is described in Chapter 12, “Installing, configuring, and running the WebSphere Commerce Analyzer” on page 315.

  Ensure that the added lines are similar to that shown in Example 13-6.

Example 13-6 Updating the resources.xml file

```
<resourceConfig>
<resource>
.....................................................................
.....................................................................
   <resourceXML name="biCSRIndividualRevenueReport"
   file="bi/biCSRIndividualRevenueReport.xml" />
   <resourceXML name="biCSRIndividualRevenueReportInputDialog"
   file="bi/biCSRIndividualRevenueReportInputDialog.xml" />
   <resourceXML name="biCSRIndividualRevenueReportOutputDialog"
   file="bi/biCSRIndividualRevenueReportOutputDialog.xml" />
.....................................................................
.....................................................................
</resource>
</resourceConfig>
```

- Update the BINLS.properties and BINLS_<*locale*>.properties files with locale-specific text display information. These are the properties files that are used to define all the locale-specific text display information used by the new report.

Update the BINLS.properties and BINLS_<*locale*>.properties file located in the *WC_profiledir/i*nstalledApps*/cell_name/*WC*_instance_name.ear*/properties/com/ibm/commerce/tools/bi/properties directory with any new report text keys.

Example 13-7 shows the updating of properties files with the keys, as required.

Example 13-7 Updating the BINLS.properties and BINLS_<locale>.properties files

```
CSRIndividualRevenue=Individual: Personal Revenue, Profit, and Ranking
CSRIndividualRevenueReportWindowTitle=Individual: Personal Revenue,
Profit, and Ranking
CSRIndividualRevenueReportInputDescription=To view a report on the
revenue and profit you have generated, complete the following fields
and click View Report.
CSRIndividualRevenueReportDescription=This report provides data about
the revenue and profit generated by a specific customer service
representative (CSR).
CSRIndividualRevenueReportOutputViewTitle=Individual: Personal Revenue,
Profit, and Ranking
CSRIndividualRevenueReportOutputViewRunDateTitle=Report generated:
CSRIndividualRevenueReportCSRID=CSR logon ID
CSRIndividualRevenueReportCSRName=CSR Name
CSRIndividualRevenueReportCSRname=CSR name
CSRIndividualRevenueReportRankBy=Ranking Criteria
CSRIndividualRevenueReportRevenue=Revenue
CSRIndividualRevenueReportProfit=Profit
CSRIndividualRevenueReportRanking=Ranking
```

- Update csrReportsContextB2C.xml and csrReportsContextB2B.xml to add a new report to the customer service reports list.

  To add this new report to the Customer Service Reports list, edit *WC_profiledir/i*nstalledApps*/cell_name/*WC*_instance_name.ear*/xml/tools/reporting/csrReportsContextB2C.xml for business-to-business store and *WC_profiledir/i*nstalledApps*/cell_name/*WC*_instance_name.ear*/xml/tools/reporting/csrReportsContextB2B.xml for a business-to-business store.

  This XML file contains the <context> and <entry> tags. The <context> XML tags define a group of reports that are displayed on one page in the WCA. The <entry> XML tags exist between the <context> and the </context> tags, and represent each report that is displayed on the customer service reports page.

Example 13-8 shows the XML file updated with the <context> and <entry> tags, as required.

Example 13-8 Updating the csrReportsContextB2C.xml and csrReportsContextB2B.xml files

```
<context name = "csrReportsContextB2C" displayKey = "csrReportContext"
autoLaunch="true">
//In this context tag, a group of reports can be defined.

<!-- ====== INDIVIDUAL CSR REPORTS====== -->

//Here entry tag is having one CSR report defined. All the text keys
//should be defined in properties file.
<entry nameKey="csrPersonalRevenue"
descriptionKey="csrPersonalRevenueDescription"
breadCrumbTrailTextKey="csrPersonalRevenue"
toolsComponent="CommerceAnalyzer">
//If the Commerce Analyzer is not enabled, then Accelerator would not
//be able to showing this report.

//Here we are defining the access roles who can view this report.
<roles>
        <role>siteAdmin</role>
        <role>cusRep</role>
        <role>seller</role>
</roles>
<command name = "DialogView"> <parameter name="XMLFile"
value="bi.biCSRIndividualRevenueReportInputDialog" />
//The report input dialog xml file is being called which is calling
//the input jsp to load the input page.
</command>
</entry>
.....................................
.....................................
//Other CSR reports can be defined here
.....................................
.....................................
</context>
```

- Update the OperationalReportsNLS.properties and OperationalReportsNLS_<*locale*>.properties files with locale-specific text display information.

  When updating the csrReportsContextB2C.xml file, all the text key information must be defined in the locale-specific OperationalReportsNLS.properties file under the

*WC_profiledir/*installedApps*/cell_name/*WC*_instance_name.ear*/properties/com/ibm/commerce/tools/reporting/properties/ directory.

Example 13-9 shows updating of the OperationalReportsNLS.properties file and the OperationalReportsNLS_en_US.properties file.

Example 13-9 Updating the files

```
csrPersonalRevenueDescription=Individual:This report allows a customer
service representative to view the revenue and profit that they have
generated. A CSR can only view their own information using this report.
csrRevenueProfitDescription=Individual: This report allows you to view
the revenue and profit generated by a customer service representative.
csrPriceQuotesDescription=Individual: This report lists the prices a
customer service representative has quoted to customers.
csrQuotesConversionDescription=Individual: This report lists the number
of price quotes per customer service representative that are converted
to orders, and the corresponding conversion rate.
csrPriceOverridesDescription=Individual: This report details price
override made to items by a particular customer service representative.
csrPriceOverridesSummaryDescription=Individual: This report lists the
total amount of price override made by a particular customer service
representative.
csrRevenueCategoryDescription=Individual: This report details the sales
revenue by product category by a particular customer service
representative.
csrShippedOrdersDescription=Individual: This report details the total
shipped orders sold by a particular customer service representative.
csrOpenOrdersDescription=Individual: This report details the pending
orders sold by a particular customer service representative. Pending
orders are orders that have not been shipped yet.
csrActiveAccountsB2BDescription=Individual: This report lists the
customer organizations assigned to a particular customer service
representative.
csrActiveAccountsB2CDescription=Individual: This report lists the
customer territories assigned to a particular customer service
representative.
csrTeamRevenueProfitDescription=Team: This report provides data about
the revenue and profit generated by a team of customer service
representatives.
csrTeamPriceOverridesDescription=Team: This report lists the total
amount of price override made by a particular team of customer service
representatives.
csrTeamActiveAccountsB2BDescription=Team: This report lists the
customer organizations assigned to a particular team of customer
service representatives.
```

```
csrTeamActiveAccountsB2CDescription=Team: This report lists the
customer territories assigned to a particular team of customer service
representatives.
csrDailyRevenueDescription=This report details the daily sales
generated by customer service representatives in a given team.
```

- Update the struts-config.xml file with the newly created views. Register the newly created views in the Struts Configuration file, struts-config.xml.

  The new views using reportNameIReportInputView.jsp and the reportNameReportOutputView.jsp must be added to the configuration file.

  To register the new views, add the entries into the struts-config.xml file under the *WC_profiledir/*installedApps*/cell_name/*WC*_instance_name*.ear/CommerceAccelerator.war/WEB-INF directory.

  Example 13-10 shows the struts-config.xml file being updated with the new views, biCSRIndividualRevenueReportInputView.jsp and biCSRIndividualRevenueReportOutputView.jsp, as required.

Example 13-10 Updating the struts-config.xml file

```
<global-forwards>
.....................................................................
.....................................................................
.....................................................................
.....................
//Adding our report views inside the <global-forwards> tag
<forward className="com.ibm.commerce.struts.ECActionForward"
            name="biCSRIndividualRevenueReportInputView"
path="/tools/bi/biCSRIndividualRevenueReportInputView.jsp">
            <set-property property="resourceClassName"
value="com.ibm.commerce.tools.command.ToolsForwardViewCommandImpl"/>
</forward>
<forward className="com.ibm.commerce.struts.ECActionForward"
            name="biCSRIndividualRevenueReportOutputView"
path="/tools/bi/biCSRIndividualRevenueReportOutputView.jsp">
            <set-property property="resourceClassName"
value="com.ibm.commerce.tools.command.ToolsForwardViewCommandImpl"/>
</forward>
.....................................................................
.....................................................................
.....................................................................
.....................
<!-- Action Mappings -->
<action-mappings type="com.ibm.commerce.struts.ECActionMapping">
```

```
.....................................................................
.....................................................................
.....................................................................
.....................
//Adding action mappings for our report views

<action path="/biCSRIndividualRevenueReportInputView"
type="com.ibm.commerce.struts.BaseAction">
<set-property property="https" value="0:1"/>
</action>
<action path="/biCSRIndividualRevenueReportOutputView"
type="com.ibm.commerce.struts.BaseAction">
<set-property property="https" value="0:1"/>
</action>
.....................................................................
.....................................................................
.....................................................................
.....................
</action-mappings>
```

13.2.4 Loading the access control policies for new reports

The default access control policies are already available in the WebSphere Commerce database as loaded from defaultAccessControlPolicies.xml. Create new access control policy entries for the report views under the default entries, which can access the CSR report views, and load them into the WebSphere Commerce database.

To create and load the new access control policy entries for the reports, perform the following tasks:

1. Navigate to the *WC_profiledir/*installedApps*/cell_name/*WC*_instance_name.ear*/xml/policies/xml folder.
2. Create a new xml file, defaultAccessControlPolicies_delta.xml.
3. Define the action names for a new report in the newly created XML file. The file content is similar to that shown in Example 13-11.

Example 13-11 Access control policy entries in defaultAccessControlPolicies_delta.xml

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<!DOCTYPE Policies SYSTEM "../dtd/accesscontrolpolicies.dtd">
<Policies>
```

```
<Action Name="reportNameReportInputView"
CommandName="reportNameReportInputView">
</Action>

<Action Name="reportNameReportOutputView"
CommandName="reportNameReportOutputView">
</Action>

//Define the new reports access control entries under pre-defined
//action control entry ActionGroup of the CSR reports views.

<ActionGroup Name="CustomerServiceRepresentativeViews"
OwnerID="RootOrganization">

<ActionGroupAction Name="reportNameReportInputView"/>
<ActionGroupAction Name="reportNameReportOutputView"/>

</ActionGroup>
</Policies>
```

4. Close the file.
5. Stop the WebSphere Commerce server if it is running.
6. Open a Windows command prompt.
7. Navigate to the WC_installdir/bin directory using the Windows command prompt.
8. Run the following command to load the access control policy into the WebSphere Commerce database:

   ```
   acpload wc_database_name wc_database_user user_password
   xml_file_name wc_database_schema
   ```

   In our example, we executed the following command:

   ```
   acpload mall wcsadmin <password>
   defaultAccessControlPolicies_delta.xml wcsadmin
   ```

9. Verify that no *.error.xml is generated in the WC_installdir/xml/policies/xml directory to ensure that the access control policies file is loaded without any errors into the WebSphere Commerce database.
10. Start the WebSphere Commerce server.

13.3 Displaying the customer service reports in the WebSphere Commerce Accelerator

Before the customer service reports display in the WCA, the business intelligence component must be enabled as described in Chapter 12, “Installing, configuring, and running the WebSphere Commerce Analyzer” on page 315, and the replication and ETL processes must be run on the database at least twice. Otherwise, there is no data for the reports to display.

Important: Customer service reports only display data for the CSRs working with IBM Sales Center.

To view the CSRs, perform the following tasks:

1. Access the following URL in your browser:

   https://<*host_name*>:8000/accelerator

   In this URL, <*host_name*> is the fully-qualified WebSphere Commerce Web host name.

2. You will see a window similar to that shown in Figure 13-1. Click **Yes**.

Figure 13-1 Security alert for Secure Sockets Layer certificate

3. From the WebSphere Commerce Logon page (Figure 13-2), enter the CSR login ID and password. Click **Log On**.

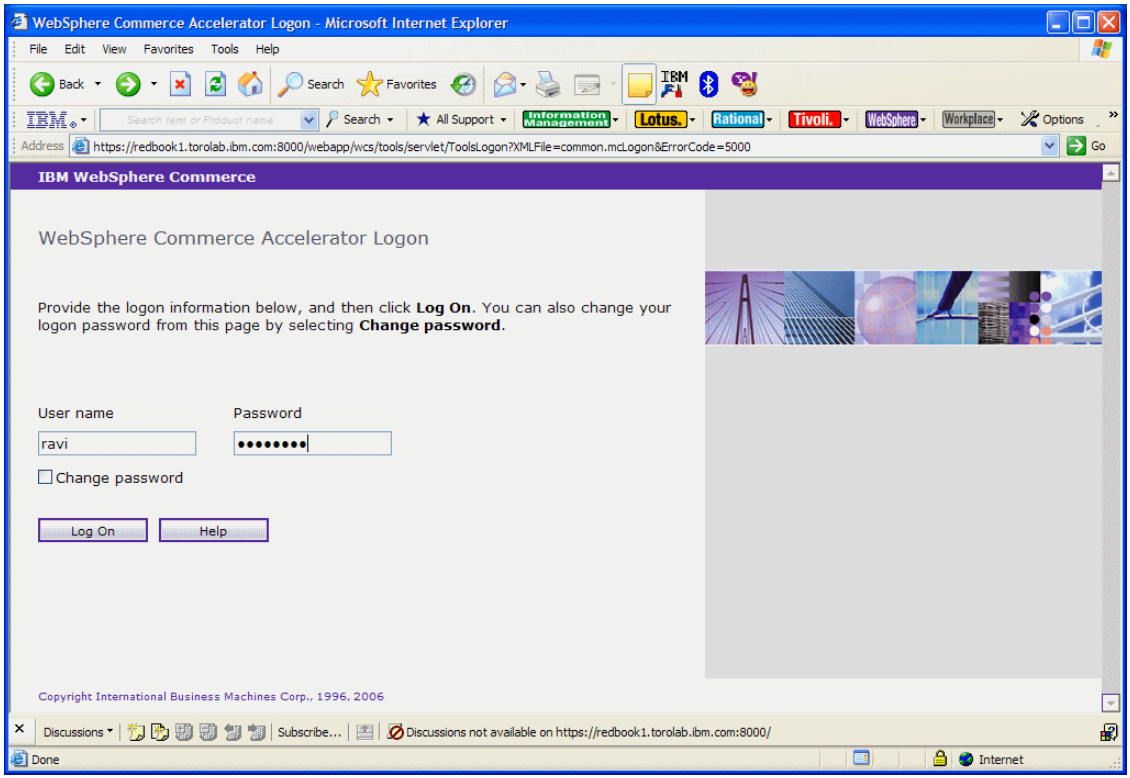


Figure 13-2 WCA logon page

4. Specify the name of the store for which you want to display customer service reports, as shown in Figure 13-3.

   Under Fulfillment centers, select a center from the list, if available, and under the Language to work in, select a language. (In our example, we selected **United States English**.)

   Click **OK**.

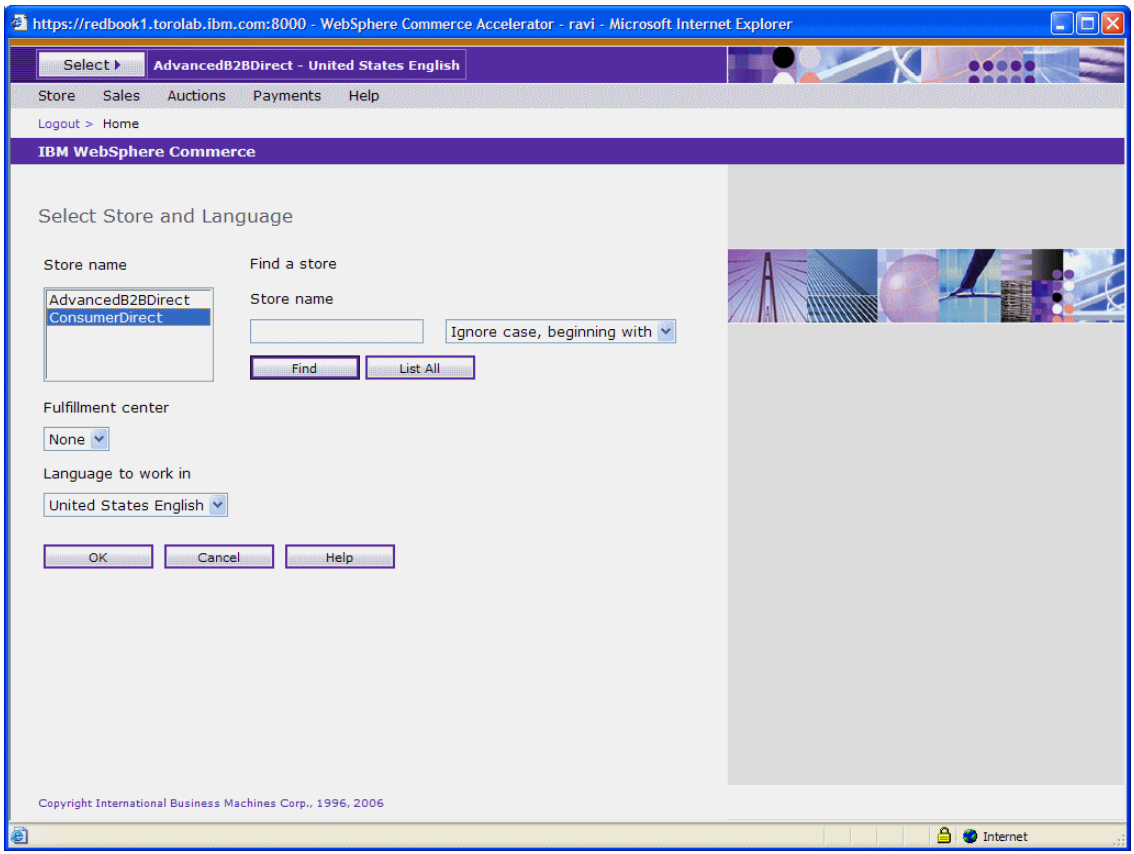


Figure 13-3 Selecting a store, fulfillment, and language for which a report is to be displayed

5. The WCA page (Figure 13-4) is launched. From the Operations menu, select **Customer Service Reports**.

> **Note:** If you do not see this menu, it means that your login ID does not have the appropriate authority to perform this task.

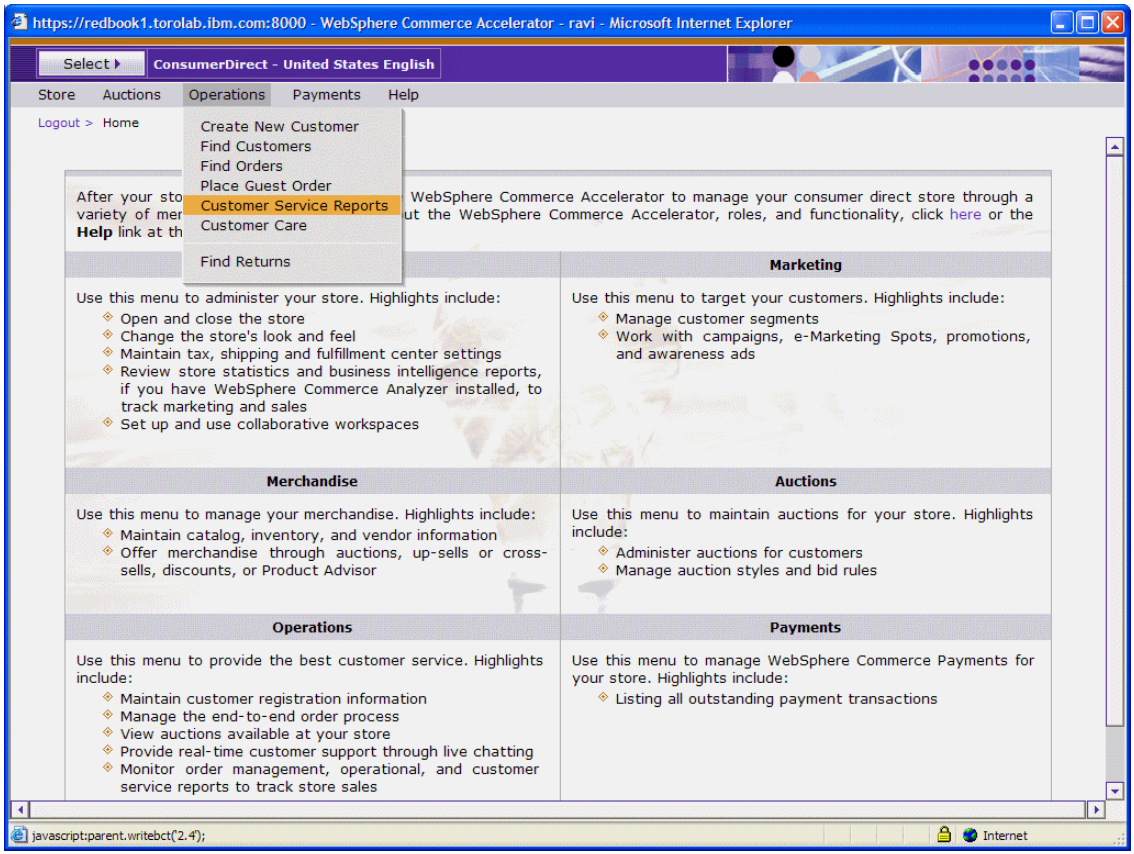


Figure 13-4 The customer service reports menu option in the WCA

6. A list of available customer service reports is displayed (Figure 13-5). Select the report that you want to view, for example, the **Personal Revenue, Profit, and Ranking** report.

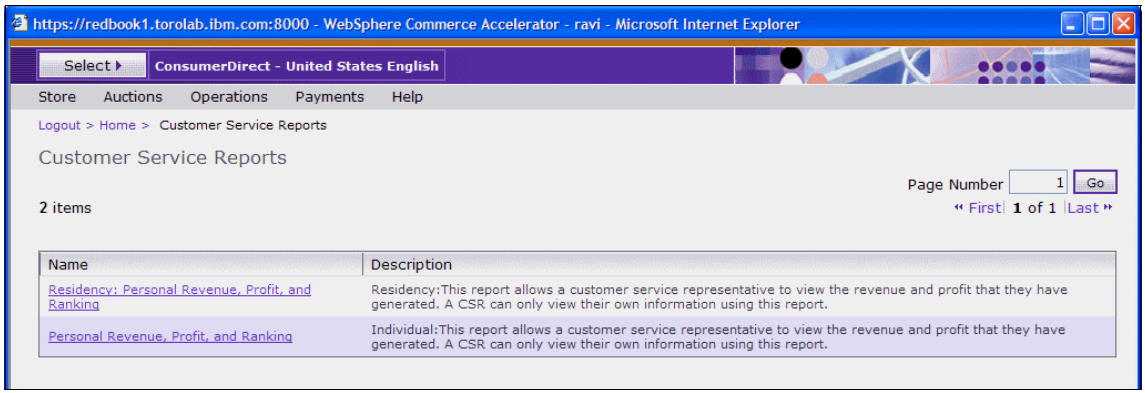


Figure 13-5 List of customer service reports

7. In the report input page, select the start date and the end date under the Time Period field, and the sorting option under the Order field (Figure 13-6). Click **View Report**.

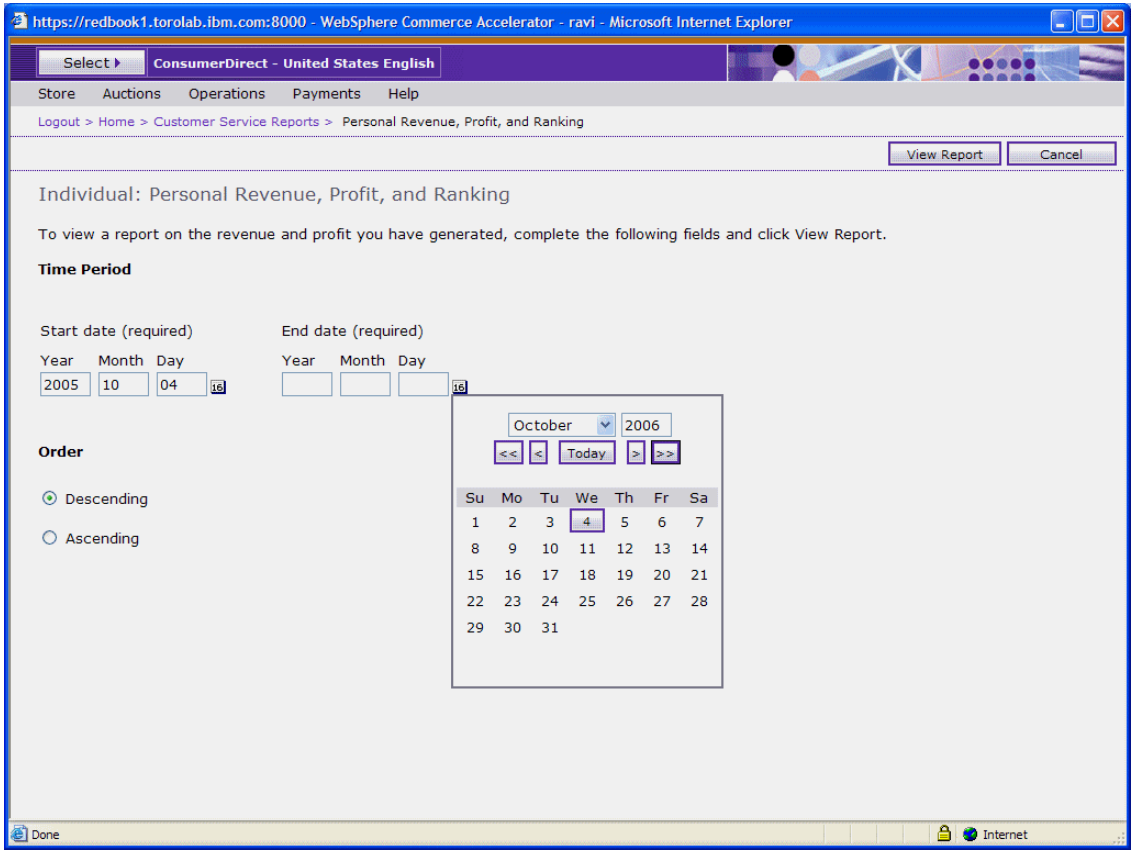


Figure 13-6 Input page to enter inputs for generating report data display

8. The report output page is displayed (Figure 13-7) with the report data. Click **OK** to return to the input page if required. To print the report data, click **Print**.

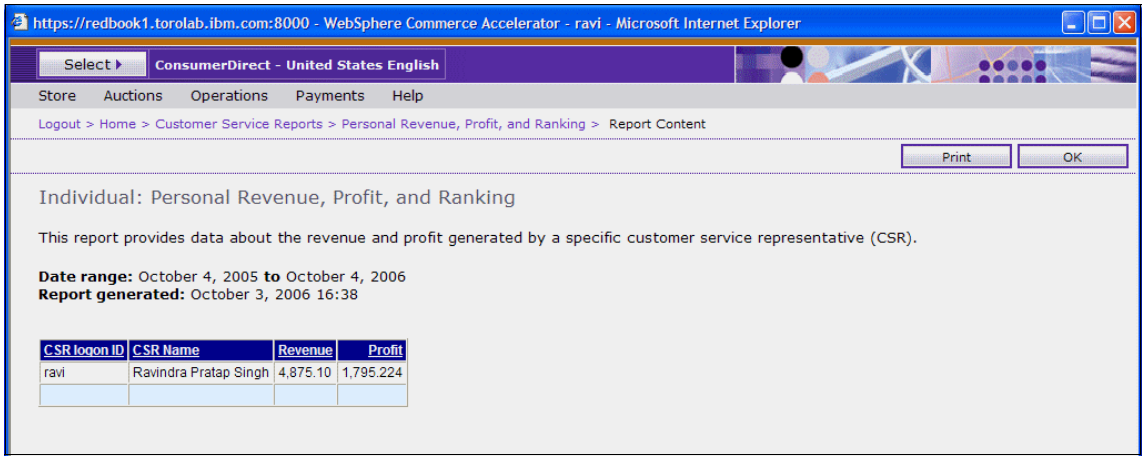


Figure 13-7 The report output page displaying the report data

A

Additional material

This IBM Redbook makes references to additional material that can be downloaded from the Internet, as described here.

Locating the Web material

The Web material associated with this IBM Redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

`ftp://www.redbooks.ibm.com/redbooks/SG247249`

Alternatively, you can go to the IBM Redbooks Web site at:

`ibm.com/redbooks`

Select **Additional materials** and open the directory that corresponds to the IBM Redbook form number SG24-7249-00.

Using the Web material

The additional Web material that accompanies this IBM Redbook includes the following files:

File name	*Description*
CustomerPet_SalesCenterCode.zip	Chapter 9, “User interface customization” on page 171
RoleBaseCustomization.zip	Chapter 10, “Role-based customizations” on page 257
com.ibm.commerce.telesales.sametime.zip	Chapter 11, “Customer Care integration with Lotus Sametime” on page 287

How to use the Web material

Create a subdirectory (folder) on your workstation, and extract the contents of the Web material compressed 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 IBM Redbook.

IBM Redbooks

For information about ordering these publications, see “How to get IBM Redbooks” on page 382. Note that some of the documents referenced here may 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
- *Extended Sites in WebSphere Commerce Business Edition V5.6.1*, SG24-6683
- *Keeping Commerce Applications Updated: WebSphere Commerce 5.1 to 5.6 Migration Guide*, SG24-6320
- *Accounts and Contracts in WebSphere Commerce V5.6.1,* REDP-4077-00
- *Remodeling a Standard Store into the Extended Sites Model With WebSphere Commerce Business Edition V5.6.1,* REDP-4091-00
- *Shipping Simplified: Integrating WebSphere Commerce with Third-party Shipping Providers,* REDP-3910-00
- *IBM WebSphere Commerce V6.0 Enterprise and Professional Installation Guide for Windows* (GC10-4261-01)
- *WebSphere Commerce Developer Enterprise and Professional Version 6.0 Installation Guide* (GC10-4255-03)

Online resources

These Web sites are also relevant as further information sources:

- acpload utility

  http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.admin.doc/refs/raxacpload.htm

- Change Flow notebook instructions

  http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.developer.tutorial.doc/tutorial/ttdsfcflow2.htm

- Creating a customer profile

  http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.telesales.tsr.doc/tasks/ttrcreateb2bcust.htm

- Creating new entity beans

  http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.developer.doc/tasks/tdecreateentitybean.htm

- Customized code deployment

  http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.developer.doc/concepts/cdedeploycustomcode.htm

- Deploying your customization to the WebSphere Commerce server

  http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.giftcenter.refapp.doc/tutorial/tgcibmgiftcentercustomization44.htm

- Determining which page the customer is browsing

  http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.customercare.doc/refs/rlhwhichpage.htm

- Eclipse

  http://www.eclipse.org

- Fix Pack 1 for WebSphere Everyplace Deployment for Windows and Linux 6.0

  http://www-1.ibm.com/support/docview.wss?rs=2314&context=SSNLT6&dc=D400&q1=wedfpintfx&uid=swg24012062&loc=en_US&cs=utf-8&lang=en

- Globalization in the IBM Sales Center

  http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.telesales.developer.doc/concepts/ctrglobalization.htm

- IBM developerWorks Web site:

  http://www-128.ibm.com/developerworks/lotus/downloads/toolkits.html

- IBM Sales Center - Adding a column to the order items table

  http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.telesales.developer.doc/tutorial/ttravaildate.htm

- IBM Sales Center: Shortcut keys

  http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.telesales.tsr.doc/concepts/ctrhotkeys.htm

- IBM WebSphere Commerce Developer V6.0, hardware prerequisites

  http://www-1.ibm.com/support/docview.wss?uid=swg27007490

- IBM WebSphere Commerce Developer Version 6.0 operating system prerequisites

  http://www-1.ibm.com/support/docview.wss?uid=swg27007488

- IBM WebSphere Everyplace Deployment V6.0 Interim Fix 2

  http://www-1.ibm.com/support/docview.wss?rs=2314&context=SSNLT6&dc=D400&q1=wedfpintfx&uid=swg24013683&loc=en_US&cs=utf-8&lang=en

- Installing IBM Sales Center for WebSphere Commerce silently from a CD

  http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.telesales.admin.doc/tasks/ttrin_silent.htm

- Integrating with back-end systems and external applications

  http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.integration.doc/concepts/ccvcapabilities.htm

- Integration Guide for WebSphere Commerce with Sametime and Quickplace

  http://www-1.ibm.com/support/docview.wss?uid=swg24012535

- Launching the IBM Sales Center development environment

  http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.telesales.developer.doc/tasks/ttrdevlaunch.htm

- Lotus Documentation

  http://www-1.lotus.com/ldd/doc

- Mapping a new Business Object Document message to a new command

  http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.telesales.developer.doc/tasks/ttrmapbod.htm

- Microsoft Windows Update Web site:

  http://windowsupdate.microsoft.com

- Modifying an existing Business Object Document reply message

  http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.telesales.developer.doc/tasks/ttrmodifybodreply.htm

- Open Application Group

  http://www.oagi.org/

- Process: Sales Center

  http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.business_process.doc/concepts/processSales_Center.htm

- Register the WebSphere Commerce Server Extension

  http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.telesales.developer.doc/tutorial/ttravaildate11.htm

- Resources in the IBM Sales Center

  http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.telesales.developer.doc/concepts/ctrresources.htm

- Starting and stopping the WebSphere Commerce Test Server via command line

  http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.developer.doc/tasks/tsrwcdevnogui.htm

- Starting and stopping WebSphere Commerce Test Server within the WebSphere Commerce Developer

  http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.developer.doc/tasks/tsrwcsstudio.htm

- Technote: IBM Rational Application Developer Fix Pack 6.0.1.1

  http://www-1.ibm.com/support/docview.wss?uid=swg24010926

- Technote: IBM WebSphere Commerce V6.0 hardware prerequisites

  http://www-1.ibm.com/support/docview.wss?uid=swg27007428

- Technote: IBM WebSphere Commerce V6.0 operating system prerequisites

  http://www-1.ibm.com/support/docview.wss?uid=swg27007429

- Technote: Recommended fixes for WebSphere Application Server

  http://www-1.ibm.com/support/docview.wss?uid=swg27004980#ver60

- Technote: Required Maintenance Scenario: WebSphere Application Server was manually installed at the V6.0.2.5 level

  http://www-1.ibm.com/support/docview.wss?uid=swg21237197

- Technote: Install of Rational Application Developer 6.x WebSphere Test Environment 6.0 server failed on Windows Operating System

  http://www-1.ibm.com/support/docview.wss?uid=swg21209120

- Technote: Installation of WebSphere Commerce Developer with WebSphere Application Server Fix Pack 6.0.2.11 fails

  http://www-1.ibm.com/support/docview.wss?uid=swg21243206

- Technote: WebSphere Commerce Developer, V6.0 required maintenance

  http://www-1.ibm.com/support/docview.wss?uid=swg21236356
- Technote: WebSphere Commerce 6.0.0.1 Fix Pack

  http://www-1.ibm.com/support/docview.wss?uid=swg24013056
- Technote: WebSphere Commerce required maintenance

  http://www-1.ibm.com/support/docview.wss?uid=swg21232042
- Tivoli Enterprise Installation Guide

  http://publib.boulder.ibm.com/infocenter/tivihelp/v3r1/topic/com.ibm.tivoli.frmwrk.doc/instguid.htm
- Tutorial: Adding a new search option in the IBM Sales Center

  http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.telesales.developer.doc/tutorial/ttrfinddialog.htm
- Tutorial: Conducting an e-mail campaign

  http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.samples.doc/tutorial/tcpemail1.htm
- Tutorial: Creating a multicultural store

  http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.samples.doc/tutorial/tgbglobilization1.htm
- Tutorial: Creating new business logic

  http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.developer.tutorial.doc/tutorial/ttd09.htm
- Tutorial: Customizing the appearance of the IBM Sales Center

  http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.telesales.developer.doc/tutorial/ttrsalescentercustomization_1.htm
- Tutorial: Importing and exporting contracts

  http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.samples.doc/tutorial/tctcontractimportexport_1.htm
- Tutorial: Modifying a page in an editor

  http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.telesales.developer.doc/tutorial/ttradvanced_1.htm
- Tutorials: WebSphere Commerce

  http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.base.doc/concepts/ctdtutorials.htm

- WebSphere Commerce Developer Editions

  http://www-306.ibm.com/software/genservers/commerce/commercestudio/lit-tech-general-be-en.html#v60

- WebSphere Commerce Developer Version 6.0 Networking Prerequisites

  http://www-1.ibm.com/support/docview.wss?uid=swg27007489

- WebSphere Commerce Version 6 Information Center

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

- WebSphere Commerce integration

  http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.telesales.developer.doc/concepts/ctrcommerceintegration.htm

- WebSphere Everyplace Deployment for Windows and Linux 6.0 Interim Fix 3

  http://www-1.ibm.com/support/docview.wss?rs=2314&context=SSNLT6&dc=D400&q1=wedfpintfx&uid=swg24013807&loc=en_US&cs=utf-8&lang=en

- WebSphere Everyplace Deployment System Administrator's Guide

  http://www-1.ibm.com/support/docview.wss?uid=swg27006861

- Using the frameset

  http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.customercare.doc/tasks/tlhframeset.htm

How to get IBM Redbooks

You can search for, view, or download IBM Redbooks, IBM Redpapers, Hints and Tips, draft publications, and Additional materials, and order hardcopy IBM 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

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

IBM Sales Center for WebSphere Commerce V6

(0.5” spine)
0.475”<->0.875”
250 <-> 459 pages

Deploying and Customizing IBM Sales Center for WebSphere Commerce V6

Automated deployment with IBM Tivoli Configuration Manager and IBM WebSphere Everyplace Deployment

User interface and role-based customization

Customer Care integration with Sametime

The IBM Sales Center for WebSphere Commerce V6 is an application for customer service representatives to capture and manage customer orders. This IBM® Redbook helps you understand IBM Sales Center for WebSphere Commerce and provides you with how-to instructions to deploy the business solution, customize it, and integrate the Sales Center with other applications.

This IBM Redbook helps you install, tailor, and configure the Sales Center development environment and production environment for creating and deploying the Sales Center customizations. In addition, this book discusses the use of IBM Tivoli Configuration Manager and IBM WebSphere Everyplace Deployment, to perform automated deployment.

This book discusses how to plan and design Sales Center customizations. Examples are provided to help you through this process. The customization scenarios that include the integration of additional IBM software and original equipment manufacturer (OEM) software are described.

This book provides user interface and role-based customization examples to demonstrate customization within the user interface framework and the role-based tools. This book also provides code sample that you can use to integrate IBM Lotus Sametime V7.5 into Sales Center, where live help and customer care functionality are achieved.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-7249-00 ISBN 0738489778