WebSphere. software

IBM

IBM WebSphere Portal V5 A Guide for Portlet Application Development

- Learn about Portal Toolkit and portlet application development
- Actions, messaging, Credential Vault, cooperative portlets
- Access Web Services from portlet applications



Juan R. Rodriguez
Serena Chan
Belen Gonzalez
George Kroner
Monica Parlangelo
Sandro Schwedler
Andre Venancio

Redbooks

ibm.com/redbooks

Petitioner has added letters to pages (a)-(c) and numbers to pages 555-556. Otherwise, it leaves the original page numbering.

SAP AMERICA, INC.
EXHIBIT 1013

(a)



International Technical Support Organization

IBM WebSphere Portal V5
A Guide for Portlet Application Development

January 2004

Archived

SG24-6076-00

(b)

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

Archived

First Edition (January 2004)

This edition applies to version 5 of IBM WebSphere Portal, WebSphere Studio Site Developer and WebSphere Portal Toolkit.

© Copyright International Business Machines Corporation 2004. All rights reserved.
Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices xi
Trademarks xii

Preface xiii
The team that wrote this redbook. xiii
Become a published author xv
Comments welcome. xvi

Chapter 1. Overview 1
1.1 Portal evolution 2
1.1.1 The generations of portal technology 3
1.2 Overview 4
1.2.1 What is a portal? 5
1.2.2 Enablement for portals 5
1.2.3 The WebSphere Portal framework 7
1.2.4 WebSphere Portal architecture 10
1.2.5 WebSphere Portal tooling 18
1.3 WebSphere Portal 19
1.3.1 Portal concepts 19
1.3.2 Portlets 22
1.3.3 Portlet modes 25
1.3.4 Portlet states 26
1.3.5 Portlets and the model-view-controller (MVC) design pattern. 26
1.3.6 WebSphere Portal runtime: the portlet container 27
1.3.7 Portlet life cycle 27
1.3.8 Portlet events and messaging 29
1.3.9 Page aggregation 32
1.4 Highlights in WebSphere Portal V5 37
1.4.1 Portal install. 37
1.4.2 General infrastructure 38
1.4.3 Event broker 39
1.4.4 Member subsystem. 39
1.4.5 Authentication 39
1.4.6 Authorization 40
1.4.7 URL generation, processing and mappings 41
1.4.8 Search. 42
1.4.9 Content management 43
1.4.10 Content publishing 44

© Copyright IBM Corp. 2004. All rights reserved.

1.4.11 Transcoding 45
1.4.12 Struts Portlet Framework 45
1.4.13 User interface 45
1.4.14 Cooperative portlets (Click-To-Action) 46
1.4.15 Portal Toolkit 47
1.5 Portlet solution patterns 48

Chapter 2. Portlet API 53
2.1 What is a portlet? 54
2.2 Basic portlet terms 54
2.3 MVC architecture 55
2.3.1 Standard MVC architecture 56
2.3.2 Portlet MVC architecture 57
2.3.3 Portlet MVC sample 58
2.4 Servlets versus portlets 59
2.5 What is a portlet application? 61
2.6 Portlet deployment 61
2.6.1 web.xml 64
2.6.2 portlet.xml 67
2.6.3 Parameter summary 76
2.6.4 Descriptors relationship (web.xml and portlet.xml) 76
2.6.5 UID guidelines 77
2.6.6 Building a war file 78
2.7 Portlet life cycle 80
2.8 Portlet API 82
2.8.1 Hierarchy 82
2.9 Core portlet objects 83
2.9.1 Portlet 83
2.9.2 PortletAdapter 83
2.9.3 PortletRequest 84
2.9.4 PortletResponse 85
2.9.5 PortletSession object 86
2.9.6 Client 87
2.9.7 PortletConfig object 88
2.9.8 PortletContext object 88
2.9.9 PortletSettings object 89
2.9.10 PortletApplicationSettings object 90
2.9.11 PortletData object 91
2.9.12 PortletLog object 92
2.9.13 PortletException 93
2.9.14 UnavailableException 93
2.9.15 PortletWindow object 93
2.9.16 User object 94

Archived

2.9.17 PortletURI 94
2.10 Listeners 95
2.10.1 PortletTitleListener 95
2.10.2 PortletPageListener 95
2.10.3 PortletSessionListener 97
2.10.4 WindowListener 97
2.10.5 PortletSettingsAttributeListener 98
2.10.6 PortletApplicationSettingsAttributesListener 98
2.11 Action event handling 98
2.12 Core event objects 99
2.12.1 ActionListener 99
2.12.2 ActionEvent 99
2.12.3 PortletURI 100
2.12.4 ModeModifier 101
2.13 Portlet messaging 102
2.13.1 MessageListener 102
2.13.2 MessageEvent 103
2.13.3 DefaultPortletMessage 103
2.13.4 PortletMessage 104
2.14 PropertyListener interface 105
2.15 EventPhaseListener interface 106
2.16 Attribute storage summary 107
2.17 Portlet services 108
2.17.1 ContentAccessService 109
2.17.2 Custom services 109
2.18 Credential Vault 113
2.19 Core Credential Vault objects 114
2.19.1 Vault 114
2.19.2 Segment 114
2.19.3 Slot 115
2.19.4 Credential 115
2.20 Portlet JSPs 118
2.20.1 Portlet tag library 118
2.21 Resources 124

Chapter 3. Portal Toolkit 125
3.1 Hardware and software requirements 126
3.2 Portal Toolkit installation 128
3.3 Development environment 128
3.4 Portlet application wizard 129
3.5 Developing portlet applications 136
3.5.1 Portlet application contents 137
3.5.2 Generated classes 138

Archived

3.6 Portlet.xml descriptor . . . 140
3.7 Deploying portlets . . . 146
3.8 Adding portlets to applications . . . 149
3.9 Examples . . . 150

Chapter 4. A first portlet application . . . 153
4.1 Sample scenario . . . 154
4.1.1 Creating a portlet project . . . 154
4.1.2 Configuring the Test Environment . . . 163
4.1.3 Running the portlet application . . . 166
4.1.4 Updating the portlet project . . . 171
4.1.5 Adding a JavaBean to your portlet project . . . 174

Chapter 5. Action event handling . . . 181
5.1 Action event . . . 182
5.2 Window events . . . 184
5.3 Simple action String support . . . 186
5.4 Sample scenario . . . 186
5.4.1 Scenario overview . . . 187
5.4.2 Creating the ActionEvent portlet . . . 189
5.4.3 Run the ActionEvent portlet application . . . 209

Chapter 6. Portlet debugging . . . 213
6.1 Overview . . . 214
6.2 Sample scenario . . . 214
6.2.1 Fixing compile errors . . . 214
6.2.2 Debugging a portlet application . . . 216

Chapter 7. Portlet messaging . . . 225
7.1 Portlet messaging . . . 226
7.2 MessageListener . . . 226
7.3 MessageEvent . . . 227
7.4 DefaultPortletMessage . . . 227
7.5 PortletMessage . . . 228
7.6 Sample scenario . . . 231
7.6.1 Description . . . 231
7.6.2 Sending a message . . . 233
7.6.3 Creating the target portlet . . . 236
7.6.4 Running the portlet application . . . 242
7.7 Broadcasting messages . . . 245

Chapter 8. National Language Support (NLS) . . . 249
8.1 Resource bundles . . . 250
8.1.1 Creating resource bundles in WebSphere Studio . . . 252

Archived

8.1.2 Translating resource bundles 254
8.1.3 Accessing resource bundles in portlets 256
8.1.4 Accessing resource bundles in JSPs 257
8.2 Translating whole resources 258
8.3 NLS administration 260
8.3.1 Portlet NLS administration 260
8.3.2 Portal NLS administration 263
8.3.3 Setting NLS titles 263
8.3.4 Adjusting Portal resource bundles 264
8.4 Working with characters 265
8.5 NLS best practices 265
8.6 Sample scenario: NLS bundles 266
8.6.1 NLS bundles 268
8.6.2 Accessing NLS bundles from JSPs 272
8.6.3 Running the NLS scenario 275
8.6.4 Accessing NLS bundles in Java portlets 281
8.7 Sample scenario: translating whole resources 283

Chapter 9. Accessing Web Services 291
9.1 Overview 292
9.2 A simple Web Service project 293
9.2.1 A sample Web Service 298
9.3 Creating a Web Services client portlet 308
9.4 Run the WSClientPortlet application 314

Chapter 10. Using the Credential Vault 319
10.1 Overview 320
10.2 Importing a protected servlet application 325
10.3 Using active credentials 330
10.3.1 Updating the generated portlet 335
10.3.2 Running the portlet 338
10.4 Using passive credentials 341

Chapter 11. Accessing back-end JDBC databases 343
11.1 Creating a database connection 344
11.1.1 Creating a new connection 344
11.1.2 Importing to a folder 346
11.1.3 Creating an SQL statement 347
11.1.4 Generating Java classes 347
11.1.5 Running the SQL statement 351
11.2 Sample scenario 353
11.2.1 Overview 353
11.2.2 Creating HRPortlet 355
11.2.3 Importing the WAR file 359

Archived

11.2.4 Reviewing the portlet code 361
11.2.5 Running the HRPortlet application 366

Chapter 12. Cooperative portlets 371
12.1 Overview 372
12.1.1 The WebSphere Portal property broker 373
12.1.2 Programming model 373
12.1.3 Registering and publishing properties 375
12.2 Sample scenario 376
12.2.1 Development workstation 376
12.2.2 Description 377
12.2.3 Source cooperative portlet 380
12.2.4 Target cooperative portlet 390
12.2.5 Running the cooperative portlets 405
12.3 Hints and tips 409

Chapter 13. Advanced cooperative portlets 413
13.1 Publishing properties programmatically 414
13.2 **Portlet** event handling 415
13.3 Broadcasting source data 417
13.4 **Wiring tool** **418**
13.5 Sample scenario 419
13.5.1 Declarative source cooperative portlet 419
13.5.2 Enabling the portlet for target C2A programmatic 422
13.5.3 Running the cooperative portlets 435
13.5.4 Wire portlets 440
13.5.5 Enabling HRPortlet for programmatic source C2A 440
13.5.6 Running the programmatic source portlet 445

Chapter 14. Struts portlets 447
14.1 Overview 448
14.1.1 The Struts portlet framework 449
14.2 Developing Struts Web applications 450
14.3 Migrating Struts Web applications 456

Chapter 15. Portlet preview 463
15.1 Overview 464
15.1.1 Portlet Preview buttons available in the toolbar 465
15.2 Sample scenario 467
15.2.1 Defining the Portlet Preview preference 467
15.2.2 Previewing the portlet 470

Chapter 16. Remote Server Attach 477
16.1 Overview 478

Archived

16.2 Preparing Portal for Remote Server Attach 479
16.3 Remote Server Attach configuration 484
16.4 Installing a portlet in Remote Portal 486
16.5 Running the portlet 496

Appendix A. Portlet development platform sample installation 501
Prerequisites 502
Installing a loopback adapter 502
WebSphere Studio Site Developer (WSSD) V5.0 506
WebSphere Studio Site Developer - WSSD Fix Pack 1 510
WebSphere Studio Site Developer - WebSphere Application Server Fix Pack 1 516
WebSphere Studio Site Developer - WebSphere Application Server Interim Fixes 518
WebSphere Portal Toolkit V5.0 524
Toolkit installation 524
Configuring Studio Site Developer and the Portal Toolkit 527
Configuration and preparation of the workstation 533
Installing the Cloudscape sample database 533

Appendix B. Automatically redeploying portlets 535
Description 536

Appendix C. Additional material 543
Locating the Web material 543
Using the Web material 544
System requirements for downloading the Web material 544
How to use the Web material 544

Related publications 545
IBM Redbooks 545
Other publications 545
Online resources 545
How to get IBM Redbooks 546
Help from IBM 546

Index 547

Archive

Archived

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 illustrates 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. 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 IBM's application programming interfaces.

© Copyright IBM Corp. 2004. All rights reserved.

Trademarks

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

@server®	CICS®	Lotus®
@server®	Domino®	Notes®
Redbooks (logo)  ™	Dynamic Workplaces™	Redbooks™
ibm.com®	DB2®	SecureWay®
AIX 5L™	DPI®	Tivoli®
AIX®	Informix®	WebSphere®
ClearCase®	IBM®	XDE™
Cloudscape™	IMS™	
CrossWorlds®	Lotus Notes®	

The following terms are trademarks of other companies:

Intel, Intel Inside (logos), MMX, and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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

Archived

Preface

This IBM Redbook helps you design, develop and implement portlet applications using the IBM WebSphere® Studio Site Developer and the Portal Toolkit V5. The information provided in this redbook targets Business-to-Employee (B2E) enterprise applications, but most of the scenarios presented apply to Business-to-Consumer (B2C) applications as well. In this redbook, you will find step-by-step examples and scenarios showing ways to integrate your enterprise applications into an IBM WebSphere Portal environment using the WebSphere Portal APIs provided by the Portal Toolkit to develop portlets as well as extend your portlet capabilities to use other advanced functions such as cooperative portlets, national language support, action events, portlet messaging, Credential Vault, Web Services and portlet debugging capabilities.

Elements of the portlet API are described and sample code is provided. The scenarios included in this redbook can be used to learn about portlet programming and as a basis to develop your own portlet applications. You will also find numerous scenarios describing recommended ways to develop portlets and portlet applications using the APIs provided by the IBM WebSphere Portal Toolkit. The sample scenarios in this redbook have been developed using the WebSphere Studio Site Developer but they can also be developed using the WebSphere Studio Application Developer.

A basic knowledge of Java™ technologies such as servlets, JavaBeans, EJBs, JavaServer Pages (JSPs), as well as XML applications and the terminology used in Web publishing, is assumed.

The team that wrote this redbook

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



Juan R. Rodriguez is a Consultant at the IBM ITSO Center, Raleigh. He received his Master of Science degree in Computer Science from Iowa State University. He writes extensively and teaches IBM classes worldwide on such topics as networking, Web technologies, and information security. Before joining the IBM ITSO, he worked at the IBM laboratory in the Research Triangle Park (North Carolina, USA) as a designer and developer of networking products

© Copyright IBM Corp. 2004. All rights reserved.

Serena Chan is an Advisory IT Specialist in the Portal and Content Management Practice with IBM Global Services in Toronto, Canada. She has in-depth industry experience in investment banking and has extensive Enterprise portal design and architecture experience in various products including IBM WebSphere Portal. Serena holds her Honors Degree in Bachelor of Commerce (H.BCom.) from the University of Toronto and is pursing her Master of Science in Computer Information Technology (M.Sc.) at Regis University.



Belen Gonzalez is an IT Specialist in IBM Global Services in IBM Spain. She holds a degree in Computer Science Engineering from Universidad Autonoma of Madrid. She has participated in e-business projects such as the Sydney Olympic Games. Her areas of expertise include J2EE application development with WebSphere Application Server and WebSphere Studio Application Developer. She has worked in e-commerce projects and now focuses on WebSphere Portal and WebSphere Studio Portal Toolkit.



George Kroner is a Co-op IT Specialist at the IBM ITSO Center in Raleigh, North Carolina. He is currently pursuing a Bachelor of Science degree in Information Sciences and Technology at Pennylvania State University. His interests include Web applications, pervasive computing, intelligent interfaces, and business process refinement.



Monica Parlangelo is an Advisory IT Specialist with the WebSphere Software Platform (Software Group) in IBM Brazil. She has four years of experience with WebSphere products developing solutions with e-Commerce and Portal solutions. She holds a Bachelor's Degree in Systems Analysis from the Universidade Paulista (UNIP) in Brazil and a post-graduate degree from Faculdade de Informática e Administração Paulista (FIAP), Sao Paulo Brazil on enterprise solutions using distributed object technologies with Java.



Sandro Schwedler is an IT Specialist at the WebSphere Innovation Center in IBM Germany. He has been working for IBM since 1998 and was involved in teaching and consulting different WebSphere Studio products such as Application Developer and IBM Rational XDE™. His areas of expertise include middleware, XML, Portlet and Java 2 Enterprise Edition (J2EE) development. He holds a degree in Information Technology from the Berufsakademie Stuttgart, Germany.





Andre Venancio is an Advisory IT Specialist with the WebSphere Software Platform (Software Group) in IBM Brazil. He has four years of experience with WebSphere products developing solutions with Host Integration, Edge Server and Portal solutions. He holds a Bachelor's Degree in Mathematics from the Fundação Santo André in Brazil and a post-graduate degree from Faculdade de Informática e Administração Paulista (FIAP), Sao Paulo Brazil on enterprise solutions using distributed object technologies with Java.

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

Cecilia Bardy
International Technical Support Organization, Raleigh Center

Amber Roy-Chowdhury, Marshall Lamb
IBM Research Triangle Park, North Carolina, USA

Shawn Van Raay
Perficient, Inc, Hamilton, Ontario, Canada

Guillermo Villavicencio
Avatar e-Business Solutions, Lima, Peru

Become a published author

Join us for a two- 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'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll 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

Archived

Comments welcome

Your comments are important to us!

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

- Use the online **Contact us** review redbook form found at:

  `ibm.com/redbooks`

- Send your comments in an Internet note to:

  `redbook@us.ibm.com`

- Mail your comments to:

  IBM® Corporation, International Technical Support Organization
  Dept. HZ8 Building 662
  P.O. Box 12195
  Research Triangle Park, NC 27709-2195

Archived

1

Overview

WebSphere Portal provides a flexible framework based on open standards with the capability to integrate with the best of breed solution. IBM is one of the few vendors to provide an end-to-end portal solution in the solution space.

This chapter provides an overview of the WebSphere Portal technology, IBM's portal tooling, and its use in developing integrated portal applications. A high-level overview of the WebSphere Portal concepts integral to development is presented here.

In this chapter, we explore:

- The evolution of portals
- Fundamental Portal concepts and definitions
- Portal development patterns

Archived

© Copyright IBM Corp. 2004. All rights reserved.

1.1 Portal evolution

As J2EE technology has evolved, much emphasis has been placed on the challenges of building enterprise applications and bringing those applications to the Web. At the core of the challenges currently being faced by Web developers is the integration of disparate user content into a seamless Web application and well-designed user interface. Portal technology provides a framework to build such applications for the Web.

If we take a step back in time to the original PC days when each application took up the entire screen and used all the computer's resources, the advent of Windows® from Microsoft® revolutionized the way we interacted with our desktop. A user no longer had to close one application to interact with another. Each application's content was aggregated to the desktop. This same evolution is taking place on the Web with portal technology.

Taking a shorter step back in time to the advent of the Web, initially interaction with the Web involved entering a single URL to access a single Web site much like the single application model of the early PCs. As the Web quickly evolved, so did the associated browser technology such as applets and browser plug-ins for technologies like Java. Unfortunately, these technologies never standardized and made the job of the Web developer very difficult when trying to provide cross-browser implementations. In parallel with these technologies, the desire grew for dynamic content on the Web and drove the development of Web servers into application servers that could serve dynamic content and technologies such as JSPs.

Support for portals evolved from this application server evolution along with the need to render multiple streams of dynamic content. The early portals fall in the category of *roll your own*. These are proprietary and specific to each implementation. As these portals grew, so did tooling and frameworks to support the building of new portals. The main job of a portal is to aggregate content and functionality. Portal servers provide:

- A server to aggregate content
- A scalable infrastructure
- A framework to build portal components and extensions

Additionally, most portals require personalization and customization. Personalization enables the portal to deliver user-specific information targeting a user based on their unique information. Customization allows the user to organize the look and feel of the portal to suit their individual needs and tastes.

Archived

WebSphere Portal provides a framework for addressing all these issues along with an open flexible infrastructure for creating many types or portals accessible from a wide variety of devices.

1.1.1 The generations of portal technology

Portals have gone through an evolution process of their own.

First generation portals

The first portals, known as first generation portals, were focused on providing static Web content, Web documents and live feeds. They were mostly an aggregation of content. In a corporate environment, they had a similar objective, providing a single interface to corporate information distributed throughout the enterprise. They typically contained information such as company news, employee contact information, company policy documents and other key Web links.

Second generation portals

Second generation portals are first generation portals with added features such as personalized, customized content and a search capability but are often a manual roll-your-own process.

Third generation portals

Third generation portals focus on specific information and applications. Integration has been added at the data level. These portals incorporate the notion of providing services along with the first generation idea of providing content. Another key feature of third generation portals is *collaboration*.

Collaboration portals provide the ability for teams to work in a virtual office. They provide content management services, the mining and organization of related information, along with collaborative services that allow users to chat, e-mail, share calendars and define user communities. Collaborative portals are typically internal corporate portal installations.

Fourth generation portals

Fourth generation portals are intended to address full-function e-business (Figure 1-1 on page 4). This involves integration with legacy applications at the component level. Enterprise portals have evolved from the provision of traditional employee self -service such as the HR policy to providing employees a complete set of comprehensive tools to enhance their productivity.

They take portals beyond the corporate boundaries for use by employees, suppliers and customers. They also provide access from multiple types of

devices to address the diverse user communities in need of services. They offer the richest set of content and application choice via a single user interface to a diverse community including browsers and pervasive devices.They also provide automated personalization via based on business rules. The key to their further evolution is their open framework for common services.

IBM WebSphere Portal is a fourth generation portal providing organizations with a portal framework that connects a wide range of enterprise content and applications. It provides a high degree of integration technologies based on the J2EE platform. Its extensible architecture provides a scalable framework allowing adaptation to the changing needs of business.

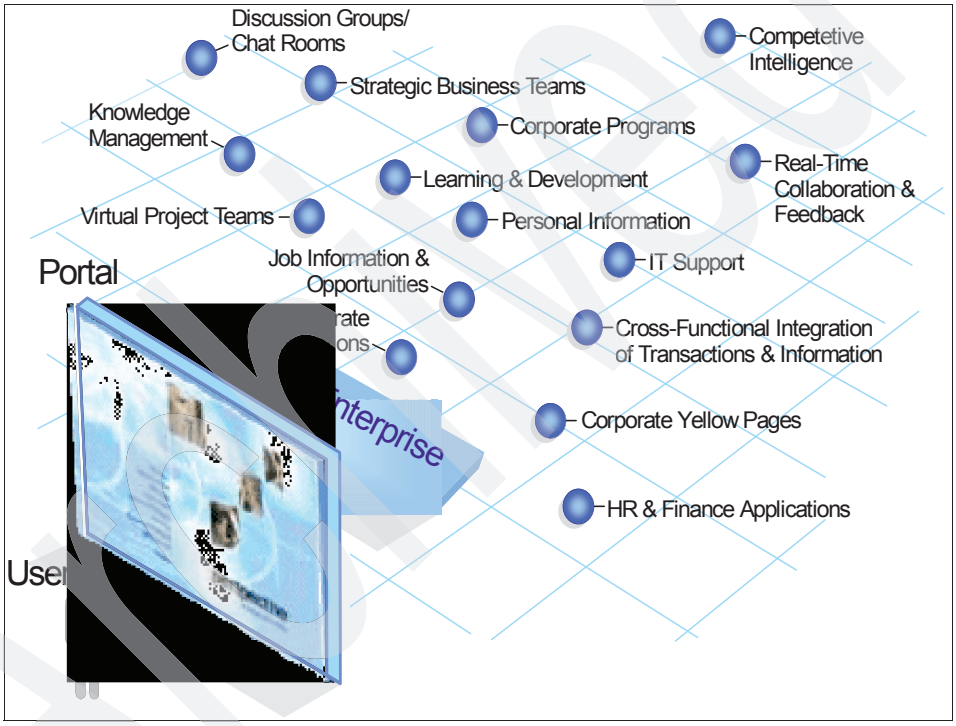


Figure 1-1 e-business needs

1.2 Overview

The primary purpose of implementing an Enterprise portal is to enable a working environment that integrates people, their work, personal activities and supporting processes and technology. Investment in portal technology will remain high amidst economic adjustments. The reason for the sustained growth is that

enterprise portals deliver immediate tangible cost savings, enhance productivity, increase efficiency and generates revenue for the clients.

Most companies have developed their Business to Consumer (B2C), Business to Business (B2B) and Business to Employee (B2E) strategies. A lot of times, the challenge is to tie them together via a comprehensive strategy that is extendable to employees, business partners and customers. Customers are often faced with issues of integrating with legacy systems. Companies are often faced with the decision of whether to build or to buy.

Portal solutions such as IBM WebSphere Portal are proven and shorten the development time. Pre-built adapters and connectors are available so that customers can leverage on the company's existing investment by integrating with the existing legacy systems without re-inventing the wheel.

1.2.1 What is a portal?

Portals are the next-generation desktop, delivering e-business applications over the Web to all kinds of client devices. Portals provide site users with a single point of access to multiple types of information and applications. Regardless of where the information resides or what format it is in, a portal aggregates all of the information in a way that is pleasing and relevant to the user. A complete portal solution should provide users with convenient access to everything they need to get their tasks done.

1.2.2 Enablement for portals

A portal represents a comprehensive approach to delivering Web supported tools and enabling services to employees, customers and business partners. A portal enables services that should be available through Web-enabled devices, on a 24x7 basis.

Authentication/authorization

Authentication provides different mechanisms that can be used to validate the identity of all portal users. Authorization determines whether a user has the necessary privilege to request a service.

Directory services

The Lightweight Directory Access Protocol (LDAP) infrastructure provides a foundation for deploying comprehensive identity management and advanced software architectures such as Web services.

Archived

Content management

Content management provides a way for the company to manage and leverage the enterprise's intellectual assets. Knowledge assets may include business intelligence and competitive intelligence data.

Collaboration

Collaboration enables employees, customers and business partners to work with, interact with, and develop or maintain content with others who share activities or interests.

Search

The portal offers a search service that supports distributed, heterogeneous searches across different data sources. Search and indexing allows users to solve problems quickly, since users often need to make ad-hoc queries to gather new information.

Personalization

Personalization provides the user the ability to establish preferences and profiles. In addition, value-added services for users increase the stickiness of the portal.

e-learning

A portal can provide just-in-time training and development of skills or expertise for work. It allows the individual to select the time and place of learning activities in their own time.

Internationalization

There is an increasing need for providing globalization. As business is getting more global, workplaces are decentralized, often with thousands of individuals working in shifting locations.

Pervasive computing

Portal provides access to applications and systems to mobile, remote users at any time and any place. It provides personalized delivery of integrated content through multiple channels: portal, wireless, kiosk, etc.

e-commerce

Most of the time, the return on investment (ROI) of implementing a portal may accrue through direct savings in self-service as well as reduced transaction costs. Integrating the portal with e-commerce applications can generate revenue and add tangible value that contributes to enterprise competitiveness.

Archived

Host integration

These capabilities provide a single point of entry to applications including legacy systems. This allows processes and data from multiple applications through a single workspace. Most of the time, companies have invested substantially in the legacy systems and the investment can be leveraged.

Site usage

Site analytics provide comprehensive Web site analytics to improve the overall effectiveness of Web initiatives and campaigns and to ensure a high quality, high-availability, error-free Web experience for visitors and customers.

1.2.3 The WebSphere Portal framework

WebSphere Portal's extensible framework allows the end user to interact with enterprise applications, people, content, and processes. They can personalize and organize their own view of the portal, manage their own profiles, and publish and share documents. WebSphere Portal provides additional services (see Figure 1-2 on page 8) such as Single Sign-On, security, directory services, content management, personalization, search, collaboration, search and taxonomy, support for mobile devices, accessibility support, internationalization, e-learning, integration to applications, and site analytics. Clients can further extend the portal solution to provide host integration and e-commerce.

Archived

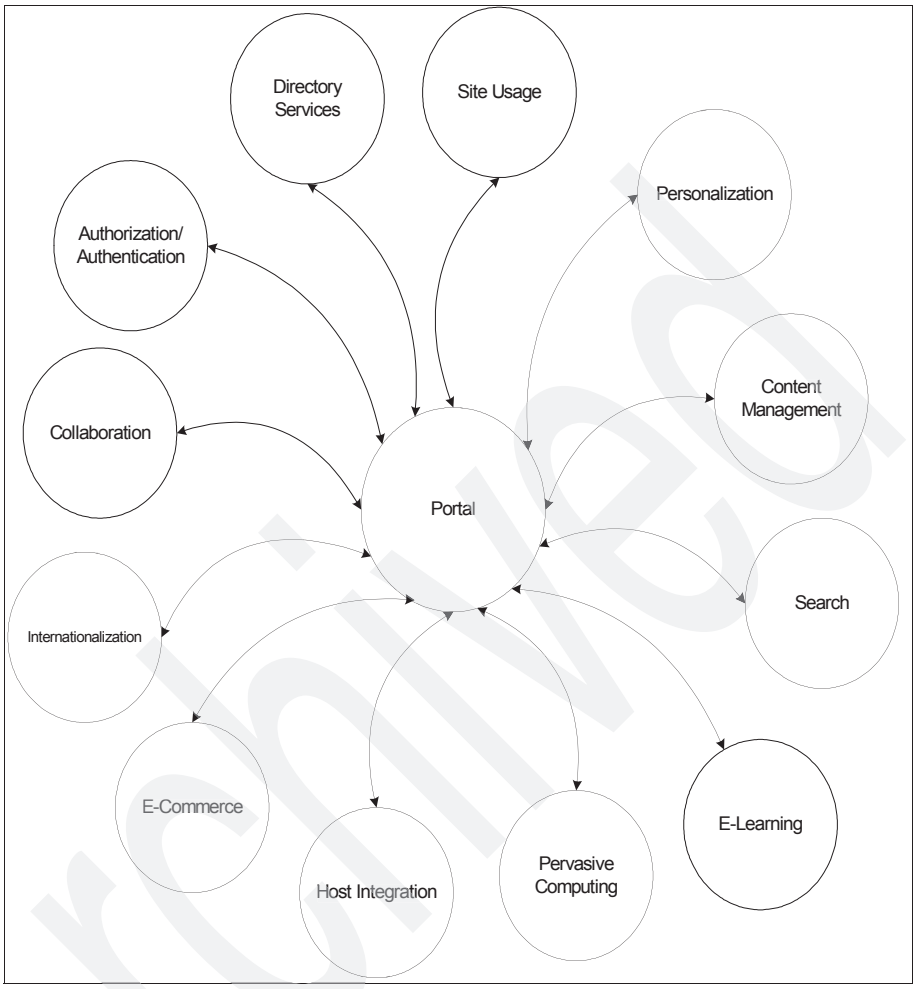


Figure 1-2 Portal context diagram

IBM WebSphere Portal provides a single, secure, interactive point of access to dynamic applications, information, people and processes to help build successful Business to Business (B2B), Business to Employee (B2E) and Business to Consumer (B2C) portals. WebSphere Portal:

- Consists of pre-integrated software which is customizable, extensible and scalable
- Is built on the award-winning WebSphere Application Server 5 platform, using J2EE standards to optimize performance
- Provides integrated Web services so you can quickly deploy portlets

Archived

- Gives users a content publishing and personalization interface that lets them create and target portal content in one step
- Offers numerous portlets for e-mail, calendars, syndicated news, industry applications and many other functions
- Provides award-winning collaborative technology within the portal, in addition to making it available for portlets

WebSphere Portal is a framework that lets you plug in new features or extensions called *portlets*. In the same way that a servlet is an application within a Web server, a portlet is an application within WebSphere Portal. Developing portlets is the most important task in providing a portal that functions as the user's window to information and tasks.

Portlets are an encapsulation of content and functionality. They are reusable components that combine Web-based content, application functionality and access to resources. Portlets are assembled into portal pages which, in turn, make up a portal implementation. Portlets are similar to Windows applications in that they present their contents in a window-like display on a portal page. Like a Windows application, the portlet window has a title bar which contains controls, allowing the users to expand (maximize) and shrink (minimize) the application.

Portlets function within the Portal framework where Windows applications function in the Windows framework. From the portal user's perspective, a portlet is a window on a portal site which provides access to a specific service or resource.

The portal also provides the runtime environment for the portlets that make up the portal implementation. This runtime environment is the portlet *container*.

The portlet container, in the J2EE sense of a container, is responsible for instantiating, invoking and destroying portlets. The portlet container provides the life cycle infrastructure for the portlets. Portlets rely on their container to provide the necessary infrastructure to support a portal environment. The portal infrastructure provides the core sets of services required by the portlets, including:

- Access to user profile information
- A framework for portlets to participate in events
- A framework to communicate with other portlets
- Access to remote content
- Access to credentials
- A framework for storing persistent data.

1.2.4 WebSphere Portal architecture

The WebSphere Portal platform is positioned to enhance the WebSphere family of products, providing tooling for aggregating and personalizing Web-based content and making that content available via multiple devices. WebSphere Portal takes advantage of the strong platform provided by WebSphere Applications Server.

WebSphere Portal finds its roots in Apache Jetspeed. Jetspeed is an Open Source implementation of an Enterprise Information Portal, using Java and XML. Jetspeed was created to deliver an Open Source Portal that individuals or companies could use and contribute to in an Open (Source) manner.

Soon after creation, it became apparent that Jetspeed was going to become an “engine” for Web applications. That, however, was far beyond the scope of the original project. Around that time, there were many discussions on the mailing list that spawned the Turbine project based on technology donated by Jon Stevens/Clear Ink. Turbine is now the Web application framework that Jetspeed shares with many other Web applications.

Typical topology

Building on the Jetspeed implementation, WebSphere Portal provides an architecture for building and running portal applications. The overall WebSphere Portal Architecture can be seen in Figure 1-5 on page 15. WebSphere Portal provides services for Authentication and Authorization though the WebSphere Member Services.

The core of WebSphere Portal architecture is composed of the Presentation Services, the portal infrastructure, and the portal services.

Archive

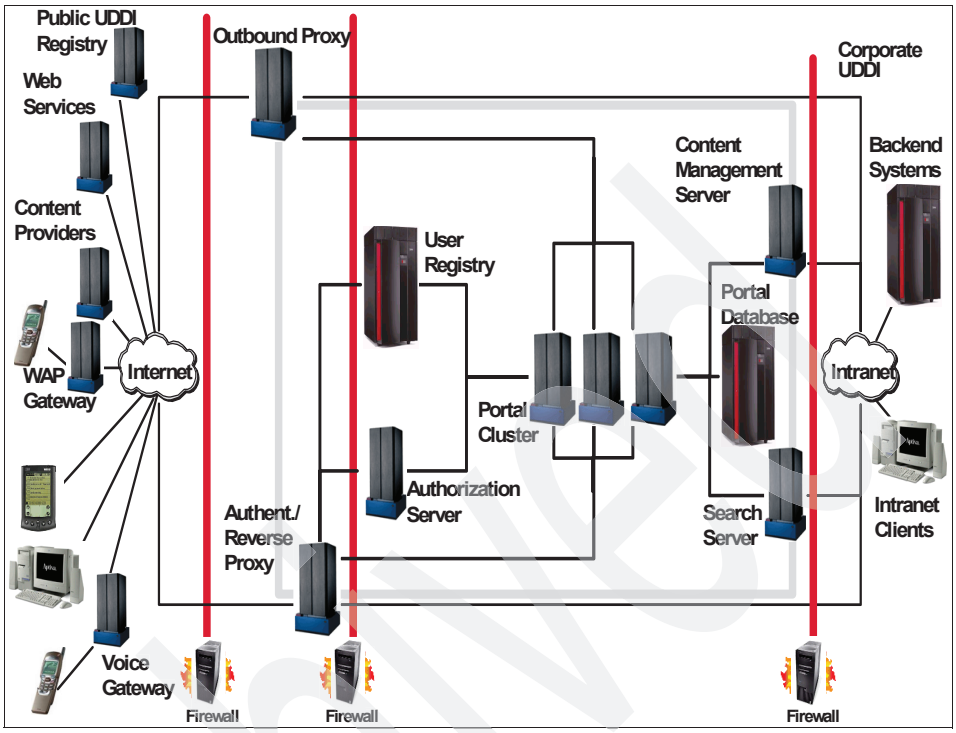


Figure 1-3 Distributed Portal system

Distributed solution

WebSphere Portal can run in a single, two, three, or n - tier environment. This, combined with the delegation capabilities from WebSphere Portal, provides you with a structured management in a distributed environment. WebSphere Portal can be part of an open, architected, and extensible end-to-end geographically distributed solution. The scalable solution incorporates redundancy with high availability design and is proven for a geographically distributed infrastructure. Additional optional components in the Portal Server architecture include a load balancer (WebSphere Edge Server - Network Dispatcher), integration to intrusion detection, and translation (WebSphere Translation Server) within the Demilitarized Zone (DMZ).

Secure demilitarized zone configuration

As shown in Figure 1-4 on page 14 depicted a sample architecture of deploying portal in a multi-tier Demilitarized Zone (DMZ) configuration with high availability. This configuration can be used for an Internet/extranet portal solution.

SSL support

IBM WebSphere Portal supports SSL. SSL support for secure transactions is one of the main reasons to use the IBM HTTP Server as part of your Web development process. The SSL encryption system is used on servers to ensure privacy when information is sent across the Internet. An SSL-enabled server enables clients to verify a server's identity, and ensures that information transmitted between client and server remains private.

Reverse proxy security server

As shown in this configuration, Tivoli® WebSEAL is used to shield the Web server from unauthorized request for external facing users. This approach is desirable when the Web server may contain sensitive data and direct access to it is not desirable. WebSEAL is a Reverse Proxy Security Server (RPSS) that uses Tivoli Access Manager (TAM) to perform coarse-grained access control to filter out unauthorized requests before they reach the domain firewall. WebSEAL uses Tivoli Access Manager (TAM) to perform access control as illustrated in the diagram.

The reverse proxy acts as an authentication gateway node and sits between the browser and the Web servers it protects. It actually acts as a stand-in for these Web servers. The authentication gateway intercepts all requests to the protected resources as well as the responses from the Web servers. To the browser submitting requests, the authentication gateway appears to be the actual Web server, to the Web server responding to requests, the authentication gateway appears to be the client.

Load balancing

In this particular example of integrating with WebSEAL, you can configure WebSphere Application Server to use the LDAP user registry, which can be shared with WebSEAL and TAM. Replicated front end WebSEAL provides the portal site with load balancing during periods of heavy traffics and fail over capability. The load balancing mechanism is handled by a Network Dispatcher such as an IBM WebSphere Edge Server. If the Network Dispatcher fails for some reason, the standby Network Dispatcher will continue to provide access to the portal. In our sample configuration, HTTP Servers and Portal Servers are clustered to provide additional redundancy.

Directory service

The Directory and Security Services provide support for a directory of users accessible through LDAP. These services are used for authentication and can also control and verify the resource access privileges or permissions granted to users. The Directory Server can be replicated to one or more replica LDAP servers to provide redundancy. WebSphere Application Server uses LDAP to perform authentication. The client ID and password are passed from WebSphere Application Server to the LDAP server.

Database service

The database server component is not accessed directly by portal users or administrators. No application-specific tables are created. Database Server is used by WebSphere Application Server, WebSphere Portal, TAM and Directory Server to store the data they need for their operation. Replication can be turned on on the database server which is used by the portal.

Intranet clients

In this configuration, it is optional to use a separate WebSEAL for the internal users for better performance.

Open standards

IBM WebSphere Portal is based on open standards. IBM is leading efforts to standardize the application programming interfaces between portals and other applications. In particular, the Java Community Process (JCP) and the Organization for the Advancement of Structured Information Standards (OASIS) are working cooperatively to standardize the Java and XML technology needed to link portals to disparate applications.

OASIS recently announced the formation of the Web Services for Remote Portals (WSRP) Technical Committee. Chaired by IBM, the WSRP committee has the charter to create an XML and Web services standard that will allow the interoperability of visual, user-facing services with portals or other Web applications.

Syndicated content

IBM WebSphere Portal provides a framework for pre-built, real-time news and syndicated content portlets from third party vendors such as Financial Times, Pinnacor, YellowBrix, Factiva (Dow Jones and Reuters Company), Moreover, CoreMedia, divine, FatWire, Autonomy, ScreamingMedia, X-Fetch, Atomica, Knowmadic and Quiver, just to name a few. The integration is pre-built and seamless. End users and administrators can easily subscribe to the portlets and customize the preference personally to enhance the user experience.

Companies are embracing syndication concepts and standards to automate the publishing of electronic catalogs and other internal information, and to make this information available to workers through enterprise portals.

A popular and useful format for syndicated news and entertainment content is Rich Site Summary (RSS). Content can be published directly from the content management system into Rich Site Summary and Open Content Syndication (OCS) channels, where it can easily be displayed by the Portal Server's built-in RSS portlet. This self-syndication concept defines a procedure for editing, managing, and publishing your own sources of content.

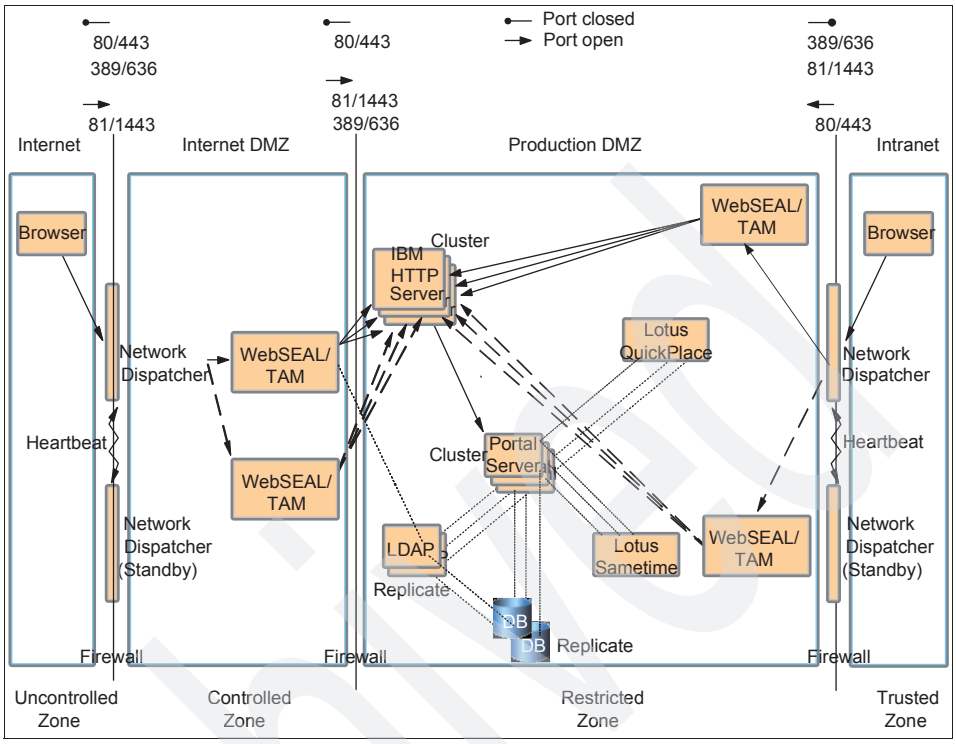


Figure 1-4 High availability portal solution

Logical architecture

WebSphere Portal finds its roots in Apache Jetspeed. Jetspeed is an Open Source implementation of an Enterprise Information Portal, using Java and XML. Jetspeed was created to deliver an Open Source Portal that individuals or companies could use and contribute to in an Open (Source) manner.

Soon after creation, it became apparent that Jetspeed was going to become an “engine” for Web applications. That, however, was far beyond the scope of the original project. Around that time, there were many discussions on the mailing list that spawned the Turbine project based on technology donated by Jon Stevens/Clear Ink. Turbine is now the Web Application framework that Jetspeed shares with many other Web applications.

Building on the Jetspeed implementation, WebSphere Portal provides an architecture for building and running portal applications. WebSphere Portal V5 provides a modular, easily extensible architecture. It is designed as a product that can run stand-alone if required, but allows plugging in alternative implementations for those components that may already be set in customer

environments. The main components of the WebSphere Portal V5 architecture are shown in Figure 1-5. The core of WebSphere Portal architecture is composed of the presentation services, the portal infrastructure, and the portal services.

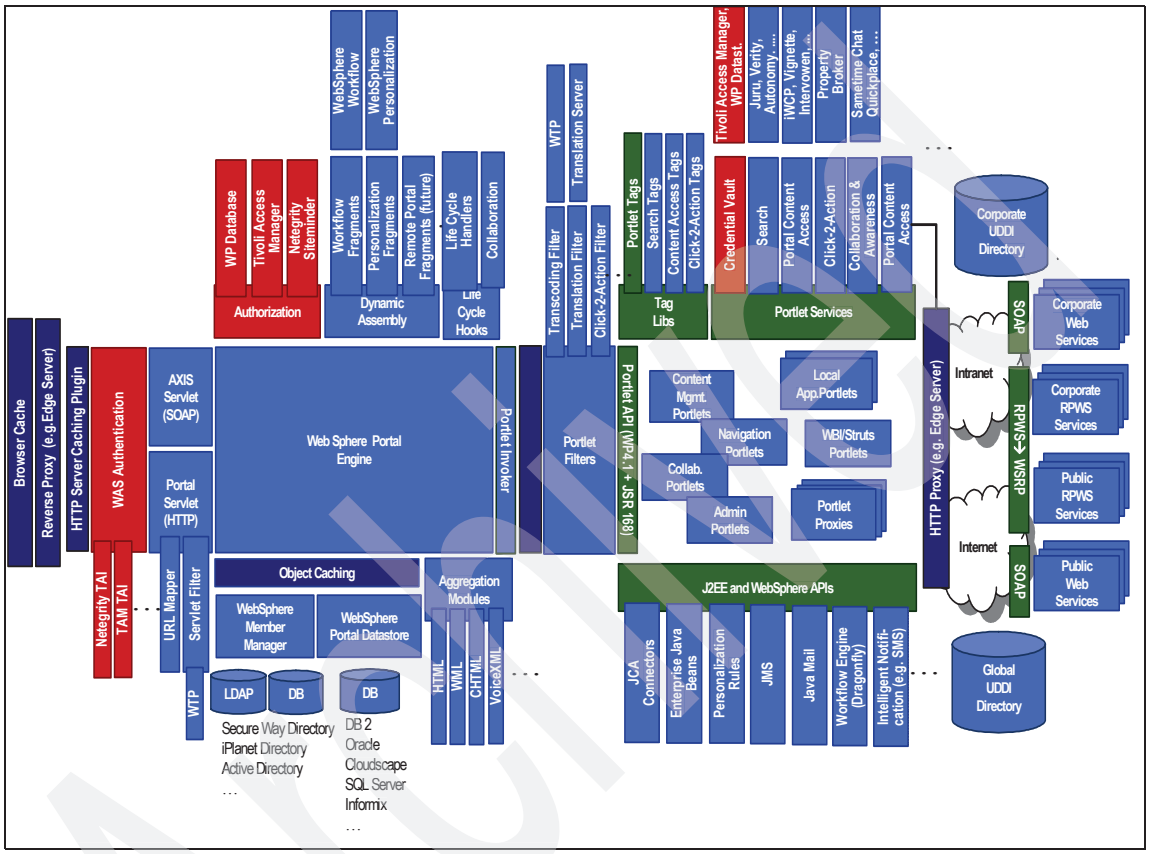


Figure 1-5 WebSphere Portal architecture

Presentation services

WebSphere Portal presentation services provide customized and personalized pages for users though aggregation. Page content is aggregated from a variety of sources via content and applications. The portal presentation framework simplifies the development and maintenance of the portal by defining the page structure independent the portlet definition. Portlets can be changed without impact to the overall portal page structure.

The Portal engine

WebSphere Portal provides a pure Java engine whose main responsibility is to aggregate content from different sources and serve the aggregated content to multiple devices. The Portal engine also provides a framework that allows the presentation layer of the portal to be decoupled from the portlet implementation details. This allows the portlets to be maintained as discrete components. Figure 1-6 shows the WebSphere Portal Engine Components.

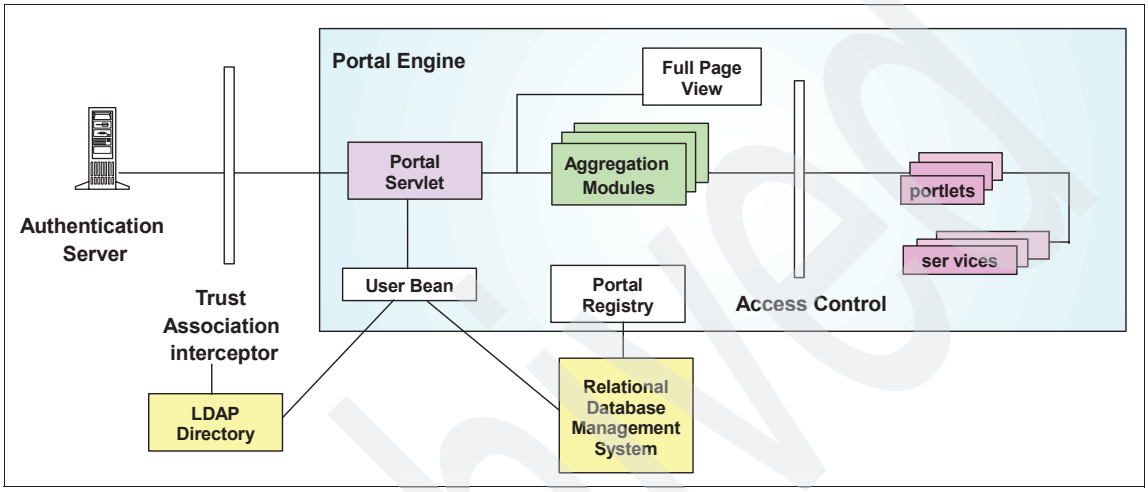


Figure 1-6 WebSphere Portal engine

The Authentication Server is a third-party authentication proxy server that sits in front of the Portal engine. Access to portlets is controlled by checking access rights during page aggregation, page customization, and other access points.

The Portal Servlet is the main component of the Portal engine. The portal servlet handles the requests made to the portal. The portal requests are handled in two phases. The first phase allows portals to send event messages between themselves. In the second phase, the appropriate Aggregation Module for the requesting device renders the overall portal page by collecting information from all the portlets on the page and adding standard decorations such as title bars, edit buttons, etc.

Portlet container

Portal Services are components WebSphere Portal uses to extend the portal functionality. Key functionality is provided with WebSphere Portal for personalization, search, content management, site analysis, enterprise application integration collaboration and Web services. Portlets can access these services via their container.

Archived

Portal infrastructure

The WebSphere Portal infrastructure is the framework that provides the internal features of the portal. Functionality such as user and group management via self registration, as well as portal administration, are provided by the Portal infrastructure.

User and group management

The WebSphere Portal infrastructure provides facilities to allow user self management along with enterprise integration with user directories such as LDAP or database structures.

Security services

Since WebSphere Portal runs within the WebSphere Application Server platform, it makes use of the standard Java Security APIs to provide authentication. The WebSphere Portal is configured so that incoming requests pass through an authentication component such as WebSphere Application Server, WebSEAL or other proxy servers. A user's authorization for a particular resource such as page or a portlet is handled by the portal engine.

User Beans are provided to allow programmatic access to the User information for use within portlets.

Page transformation

WebSphere Transcoding Technology is integrated with WebSphere Portal to transform the portal markup produced by WebSphere Portal to markup for additional devices such as mobile phones and PDAs.

Portal services

Portal services are built-in features the WebSphere Portal provides to extend and enhance the full portal solution. These services are provided via the Portlet container as seen in Figure 1-5 on page 15. Among the services are the following:

- **Personalization**

  The IBM WebSphere Personalization functionality enables advanced personalization capabilities. Base customization, such as choosing which portlets are desired on a page, is accomplished by the user via administration functionality. Advanced personalization via rules engines, user preferences and profiles is accomplished by the provided personalization services.

- **Content management**

  WebSphere Portal provides services to facilitate connections to content management sources. Built-in support is provided for several common content types such a as Rich Site Summary (RSS), News Markup Language

Archived

(NewsML) and Open Content Syndication (OCS) along with most XML and Web browser markup.

- **Search**

  WebSphere Portal offers a simple search service. The Portal Search capability enables search across distributed HTML and text data sources. The search can crawl a Web site and is configured so as to force it to follow several layers in a site or to extend beyond several links in a site. Furthermore, IBM Extended Search and Enterprise Information Portal can be fully incorporated into the portal environment. These search engines are industrial-strength tools that provide federated searches across numerous data sources.

- **Site analysis**

  You can take advantage of the underlying WebSphere Application Server technology and Site Analyzer to provide information about Web site visitor trends, usage and content. This detailed information can then be used to improve the overall effectiveness of the site.

- **Collaboration**

  Collaboration services are provided by WebSphere Portal through a set of pre-defined portlets. These portlets allow for team-room function, chat, e-mail, calendering and many other collaborative technologies.

- **Web Services**

  WebSphere Portal provides services for exposing and integrating portlets as remote portlets hosted on another portal platform via Web Services technology. The entire process of packaging and responding to a SOAP request is hidden from the developer and the administrator.

1.2.5 WebSphere Portal tooling

WebSphere Portal and WebSphere Portal Toolkit, along with their prerequisite products, provide the basic tooling for developing and deploying portals and their associated portlets.

WebSphere Portal

WebSphere Portal contains built-in support for portlet deployment, configuration, administration and communication between portlets.

WebSphere Portal provides the framework for building and deploying portals and the portal components, portlets. Portlet content is aggregated by the WebSphere Portal to provide the desired portal implementation.

WebSphere Portal makes use of the WebSphere Application Server technology to provide a portal platform.

WebSphere Portal Toolkit

The WebSphere Portal Toolkit is provided with WebSphere Portal and provides an environment for developing portal using WebSphere Portal. The WebSphere Portal Toolkit is a plug-in for WebSphere Studio Application Developer (WSAD) or WebSphere Studio Site Developer (WSSD) which adds the portal development environment.

The WebSphere Portal Toolkit provides the ability to quickly create complete, MVC-compliant portlet applications. It also provides intuitive editors for working with the deployment descriptors required by your portlet applications. Furthermore, it allows you to dynamically debug your portlet applications.

The WebSphere Portal Toolkit is explored in detail in Chapter 3, “Portal Toolkit” on page 125.

1.3 WebSphere Portal

WebSphere Portal takes the advantage of the WebSphere Application Server, making use of its J2EE services. WebSphere Portal itself installs as an Enterprise application in WebSphere Application Server.

1.3.1 Portal concepts

The following are some definitions and descriptions of Portal concepts.

Portlet

A portlet is an application that displays page content.

Portlet application

Portlet applications are collections of related portlets and resources that are packaged together. All portlets packaged together share the same context which contains all resources such as images, properties files and classes. Important also is the fact that portlets within a portlet application can exchange messages.

Page

A portal *page* displays content. A page can contain one or more portlets. For example, a World Market page might contain two portlets that displays stock tickers for popular stock exchanges and a third portlet that displays the current

Archived

exchange rates for world currencies. To view a page in the portal, you select its page.

Note: WebSphere Portal V4.x uses the concept of *Place* for grouping pages. In WebSphere Portal V5, the concept of *Place* does not exist. Places are treated as top-level pages in WebSphere Portal V5.

Layout

The page *layout* defines the number of content areas within the page and the portlets displayed within each content area. In many cases, the portal administrator defines the page layout. The administrator can permit specified users or user groups to change the page layout to reflect individual preferences. If you have authority to change a page, use the **configure** icon (wrench icon) to alter the page layout.

Roles

Each portal page is subdivided into one or more content areas. Each content area can contain one or more portlets. The portal administrator or a user who has authority to manage a page can control whether others who have authority to edit the page can move, edit or delete the content areas and the portlets on the page. Portal V5 permission is role based. A role is a set of permissions. Roles can be assigned (or mapped) to individual principals granting those principals the corresponding permissions. If you have authority to make changes to a portal page, use the *Resource Permissions* page in Access under Administration to set the permissions for the page. By default, there are seven roles and they are as follows:

- **Administrators** are allowed to have unrestricted access on all portal resources
- **Security Administrators** are allowed to grant access on a resource
- **Delegators** are allowed to grant access to other principals
- **Managers** are allowed to create, edit, and delete shared resources
- **Editors** are allowed to create and edit shared resources
- **Privileged Users** are allowed to create private resources
- **Users** are allowed to view portal resources

Comparison of V4.x permission vs. V5.x roles

Permissions that a principal (a user or group) had in WebSphere Portal V4.x are mapped to the appropriate roles in WebSphere Portal V5.0. The following table illustrates this role mapping.

Archive

Table 1-1 Role mapping

V4.x Permissions	V5.0 Roles
View	User
Edit	Privileged User
Manage	Manager
Delegate	Security Administrator
View + Edit	Privileged User
View + Manage	Manager
View + Delegate	Security Administrator + User
Edit + Manage	Manager
Edit + Delegate	Security Administrator + Privileged User (Migration option: Security Administrator + Editor)
Manage + Delegate	Administrator
View + Edit + Manage	Manager
View + Edit + Delegate	Security Administrator + Privileged User (Migration option: Security Administrator + Editor)
View + Manage + Delegate	Administrator
View + Edit + Manage + Delegate	Administrator
Create	No longer necessary. In WebSphere Portal V5.0, principals with the Administrator, Manager, Editor, or Privileged User roles on a resource are automatically allowed to create new resources underneath that resource in the resource hierarchy.

Themes

Themes represent the overall look and feel of the portal, including colors, images and fonts. There are several default themes provided with the standard installation of WebSphere Portal. Each page in the portal may have a different theme associated with it, thereby creating the appearance of virtual portals. Use the *Themes and Skins* under *Portal User Interface* to manage themes.

Archive

Skins

The term *skin* refers to the visual appearance of the area surrounding an individual portlet. Each portlet can have its own skin. The skins that are available for use with a portlet are defined by the portal theme that is associated with the place. The portal administrator or the designer determines the theme for places and the available skins for the theme. The administrator can permit specified users to change the skins to reflect individual preferences. If you have authority to make changes to a portal page, use the *Themes and Skins* under *Portal User Interface* to manage themes.

1.3.2 Portlets

The base building blocks of a Portal are the portlets. Portlets are complete applications following the Model-View-Controller design pattern. Portlets are developed, deployed, managed and displayed independent of all other portlets.

Portlets may have multiple states and View modes along with event and messaging capabilities. Based on the J2EE container model, portlets run inside the Portlet Container of WebSphere Portal analogous to the way servlets run inside the Servlet Container of WebSphere Application Server. Portlets are a special subclass of HTTPServlet that includes properties and functionality that allows them to run inside the Portlet Container. Though portlets actually run as servlets under the WebSphere Application Server, they cannot send redirects or errors to the browser directly, forward requests or write arbitrary markup to the output stream. All communication back to the end user from a portlet is done via the aggregation modules.

To understand the portlet model used by WebSphere Portal, let us take a step back and examine the Flyweight pattern. This pattern is used by WebSphere Portal as the design pattern for the portlet model.

The Flyweight pattern

The Flyweight pattern was originally presented by the GofF or Gang of Four (Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides) in E.Gamma, et al., *Elements of Reusable Object-Oriented Software*, Addison Wesley, 1995.

Flyweight is a structural pattern used to support a large number of small objects efficiently. Several instances of an object may share some properties. Flyweight factors these common properties into a single object, thus saving considerable space and time otherwise consumed by the creation and maintenance of duplicate instances. Key to the Flyweight Design Pattern is the fact that the objects share some information. It is then possible to greatly reduce the overhead problem and make the presence of so many objects possible.

The flyweight object is a shared object that can be used in multiple contexts at the same time; the object functions independently in each context.

The state shared by the objects falls into two categories, intrinsic and extrinsic.

Intrinsic state State stored in the object and independent of object's context. Thus the information is sharable across the objects. The more stateless and intrinsic information shared between objects in the flyweight, the better. This allows for greater savings in memory, since less context information needs to be passed around.

Extrinsic state State that depends on a single request varies with the objects context and therefore cannot be shared. This information must be stateless and determined by context, having no stored values, but values that can be calculated on the spot. Client Objects are responsible for passing the extrinsic state to the object when the object needs it.

This separation into extrinsic and intrinsic information allows great numbers of similar objects to exist, differing only in the context in which they exist.

The different components involved in the Flyweight Pattern are the Flyweight, the ConcreteFlyweight, the UnsharedConcreteFlyweight, the FlyweightFactory and the Client.

- The Flyweight: the shared object with intrinsic state. The flyweight declares an interface through which flyweights can receive and act on intrinsic data.
- ConcreteFlyweight: implements the flyweight interface and adds storage for the intrinsic state.
- UnsharedConcreteFlyweight: the flyweight interface enables sharing but does not enforce it. Not all flyweights are shared. It is common for UnsharedConcreteFlyweight objects to have ConcreteFlyweight objects as children at some level in the hierarchy.
- FlyweightFactory: serves to dispense particular flyweights that are requested. When a Flyweight with certain properties is requested, it checks to see if one already exists, and if so, returns that flyweight. If the requested flyweight does not exist, it creates the requisite flyweight, stores and returns it.
- Client: when creating an object, a client must assign a flyweight to it, so it asks the FlyweightFactory for a particular flyweight, receives that flyweight, and creates a reference to it in the object it is creating.

The parameterization of portlets is based on the flyweight pattern, the Portlet Container being the Flyweight Factory.

Archive

Portlets

Portlets are invoked by the portlet container. Every portlet has to inherit from the abstract org.apache.jetspeed.portlet.Portlet class, either by deriving directly from it, or by using one of the abstract portlet implementations.

A portlet is a small Java program that runs within a portlet container. Portlets receive and respond to requests from the portlet container. There is only ever one portlet object instance per portlet configuration in the Web deployment descriptor. There may be many PortletSettings objects parameterizing the same portlet object according to the Flyweight pattern, provided on a per-request basis.

When the portal administrator deploys a new portlet or copies an existing one, PortletSettings are created. A Portlet parameterized by its PortletSettings is referred to as a *concrete portlet*. The settings of concrete portlets may change at runtime since administrators modify the portlet settings by using the configuration mode of the portlet. The PortletSettings initially contain the elements defined in the deployment descriptor and are changeable by the portlet administrator.

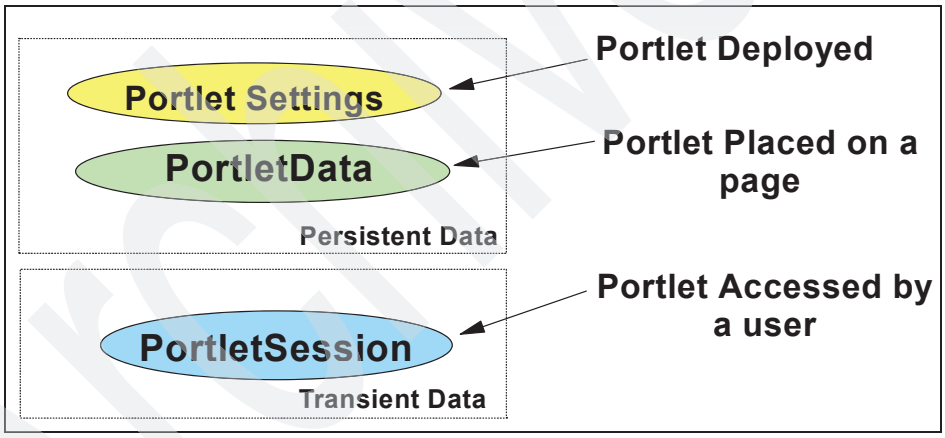


Figure 1-7 Portlet parameterization objects

Additionally, users can have personal views of concrete portlets. Therefore, the transient PortletSession and persistent concrete PortletData carries vital information for the portlet to create a personalized user experience.

When a portlet is added to a page, PortletData is created to parameterize the portlet. The PortletData can only be accessed by the portlet itself, for example when changing a list of desired stocks to watch in a stock portlet. A concrete portlet in conjunction with portlet data creates a *Concrete Portlet Instance*. PortletData scope depends on the scope of the page. If the administrator places the portlet on a page, the portlet data contains data stored for the group of users

associated with the page. If a user puts the portlet on the page, the portlet data contains data for that user. For more information on the portlet data object, refer to Chapter 2, “Portlet API” on page 53.

Finally, when the user initially accesses a portlet, a PortletSession is created. The portlet session stores transient data associated with an individual use of the portlet. The concrete portlet instance parameterized by the PortletSession is referred to as the *User Portlet Instance*.

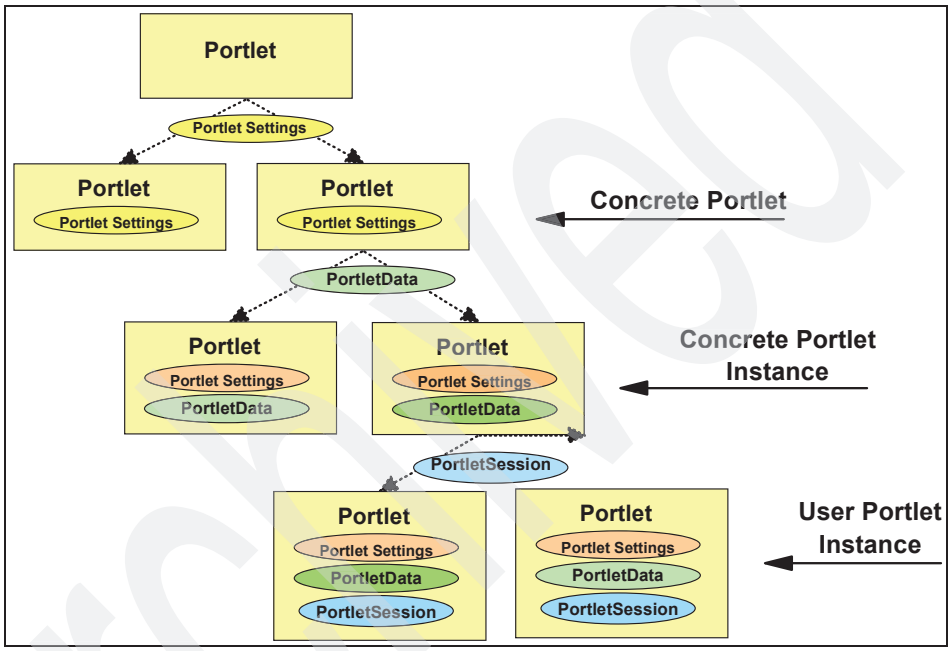


Figure 1-8 The portlet parameterization

1.3.3 Portlet modes

Portlet modes are a facet of the Portal display model. Modes allow the portlet to display a different “face” depending on its usage. There are four modes:

View	Initial face of the portlet when created. The portlet normally functions with this face.
Help	If the portlet supports the help mode, a help page will be displayed for the user.
Edit	This mode allows the user to configure the portlet for their personal use, for example, specifying a city for a localized weather forecast.

Archived

Configure	If provided, this mode displays a face that allows the portal administrator to configure the portlet for a group of users or a single user.

1.3.4 Portlet states

Portlet states determine how the portlet is displayed in the portal. The state of the portlet is stored in the PortletWindow.State object and can be queried for optimizing processing based on state. The three states of a portlet are:

Normal	The portlet is displayed in its initial state as defined when it was installed.
Maximized	The portlet view is maximized and takes over the entire body of the portal replacing all the other portal views.
Minimized	Only the portlet title bar is visible inside the portlet page.

1.3.5 Portlets and the model-view-controller (MVC) design pattern

Because portlets must be capable of supporting multiple views for multiple devices, the key design pattern used for portlets is the *model-view-controller* (MVC) design pattern. This design pattern contains three entities:

- The *model*, the data source to be retrieved for the portlet

  Model data for a portlet is typically retrieved from an external data source and loaded into Java display beans, or arrives formatted in an XML document.

- The *view* or views, the output mechanism used to display the data of the portlet

  Display views are typically implemented as either JSPs, more typically used when the data model is implemented in Java beans, or XSLT style sheets when the incoming data is formatted in an XML document.

- The *controller*, which joins the selected view to the data and conducts the operation of the portlet.

  The controller selects the view for display based on the target device or browser, and then passes the data model to the view. The view extracts the specific display data, formats the data for the browser and renders its output to the browser as part of the portal aggregation of portlet outputs.

Archive

For portlet development, the MVC pattern has the following characteristics:

- The portlet is only responsible for calling the right controller, depending on the markup supported by the client.
- Connectors are responsible for accessing content sources. Typically, there is one connector per content source type, for example, one connector for POP3 access and one for file-based cache.
- Models represent the content as retrieved through the connector. A model is independent of the presentation.
- Controllers can be used to provide markup-specific content (HTML, cHTML, or WML).

In the MVC structure, there is a distinct separation of data from presentation along with a controller component for managing the interaction between the data (model) and the presentation or view. The controller knows the environment in which the application is invoked, gathers information from the data object to be displayed, and then applies the appropriate view to render the data using the markup language appropriate for the current device.

A comprehensive discussion of the MVC pattern and how it is applied to portlets is provided in Chapter 2, “Portlet API” on page 53. See 2.3, “MVC architecture” on page 55 for details.

1.3.6 WebSphere Portal runtime: the portlet container

WebSphere Portal is a J2EE application based on the servlet technology. In fact, portlets inherit from HTTP Servlet in the Java hierarchy, providing the servlet functionality. The WebSphere Portal portlet container is not, however, a standalone container as is the servlet container. The portlet container is a thin layer implemented on top of the servlet container designed to reuse the functionality provided by the servlet container.

The Portlet API provides the standard interfaces for accessing the services provided by the portlet container. The Portlet API is explored in detail in Chapter 2, “Portlet API” on page 53. As previously mentioned, the Portlet Container is implemented on top of the servlet container and thus the Portal API is similar to the servlet API.

1.3.7 Portlet life cycle

Much like the Servlet Container, the Portlet Container manages the portlet life cycle along with providing services to the portlets running in the container.

The portlet container loads and instantiates the portlet class. This can happen during startup of the portal server or later, but no later then when the first request to the portlet has to be serviced. Also, if a portlet is taken out of service temporarily, for example while administrating it, the portlet container may finish the life cycle before taking the portlet out of service. When the administration is done, the portlet will be newly initialized.

During the portlet life cycle, the portlet container invokes the following methods on the Portlet class (subclass of a the Portlet Adapter class) on behalf of user requests as seen in Figure 1-9.

- init()
- initConcrete()
- login()
- service()
  - doView()
  - doEdit()
  - doHelp()
  - doConfigure()
- logout()
- destroyConcrete()
- destroy()

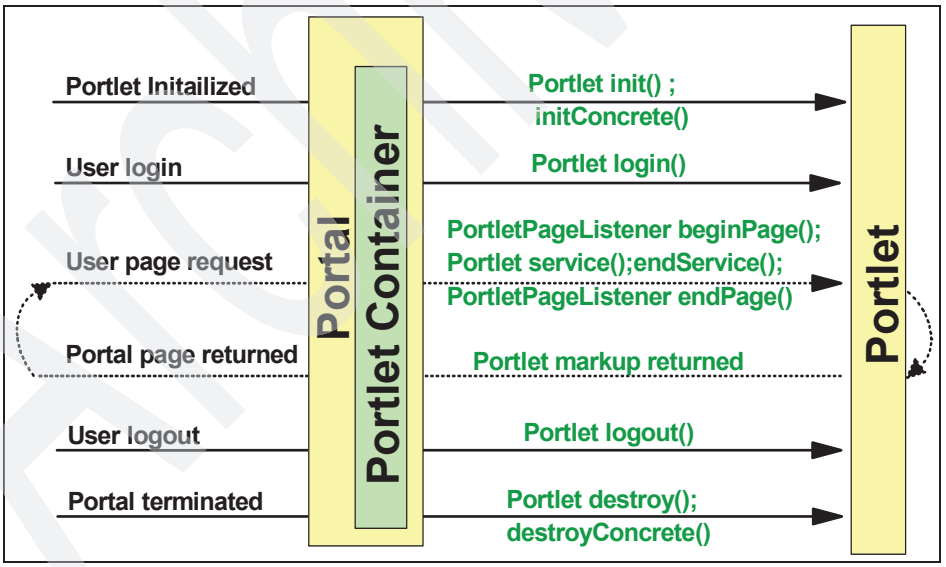


Figure 1-9 Portlet life cycle

The portlet container calls the following methods of the abstract portlet during the portlet's life cycle:

Archived

- **init()**

  The portlet is constructed after portal initialization and initialized with the init() method. The portal always instantiates only a single instance of the portlet, and this instance is shared among all users, exactly the same way a servlet is shared among all users of an application server.

- **initConcrete()**

  After constructing the portlet and before the portlet is accessed for the first time, the portlet is initialized with the PortletSettings. This is known as the concrete portlet.

- **service()**

  The portal calls the service() method when the portlet is required to render its content. During the life cycle of the portlet, the service() method is typically called many times. For each portlet on the page, the service() method is not called in a guaranteed order and may even be called in a different order for each request.

- **destroyConcrete()**

  The concrete portlet is taken out of service with the destroyConcrete() method. This can happen when an administrator deletes a concrete portlet during runtime on the portal server.

- **destroy()**

  When the portal is terminating, portlets are taken out of service, then destroyed with the destroy() method. Finally, the portlet is garbage collected and finalized.

1.3.8 Portlet events and messaging

Many portals today display static content in independent windows. The ability for portlets to interact within a portal is key to giving a portal a “live” feeling. In “live” portals, quite often the user is presented with one portlet on a page that presents a choice of data, a list of stocks for example, and choosing from the list causes another portlet to be updated with the details of the choice. This type of list-detail processing via multiple portlets is done with portlet events and messaging. This same type of process could be accomplished using a single portlet but consider the example of a stock list, stock details and news associated with the stock. Giving the user this function via three portlets allows the user to customize the portal experience by choosing which information about the chosen stock is displayed by simply adding the associated portlet to the page.

In portlet messaging, one portlet typically detects a condition and formats a message as a result of that condition, then sends the message to the receiver. The receiving portlet receives the message from the event handler and

processes the message as you would expect. Portlets can both send and receive messages.

Portlets communicate using portlet actions and portlet messages. For example, an account portlet creates a portlet action and encodes it into the URL that is rendered for displaying transactions. When the link is clicked, the action listener is called, which then sends a portlet message to send the necessary data to the transaction detail portlet.

There are some basic rules to portlet messaging:

- Portlets in different applications can only communicate through default portlet message objects. Default portlet message objects can only carry strings.
- In order for portlets to communicate through custom messages, they must be part of the same portlet application. WebSphere Portal uses a unique class loader for each portlet application to provide security between applications. The message is typically a custom Java object unique to the application. Since messaging portlets must share this message object, they must share the same class loader and therefore they must be part of the same portlet application.
- For performance reasons, portlets that communicate through messaging must reside on the same page. Since only one page is displayed at a time, there is little need to send messages to portlets not currently displayed.

Portlet events contain information about an event to which a portlet might need to respond. For example, when a user clicks a link or button, this generates an action event. To receive notification of the event, the portlet must have the appropriate event listener implemented within the portlet class.

Action events: generated when an HTTP request is received by the portlet container that is associated with an action, such as when the user clicks a link.

Message events: generated when another portlet within the portlet application sends a message.

Window events: generated when the user changes the state of the portlet window.

The portlet container delivers all events to the respective event listeners (and therefore the portlets) before generating any content to be returned to the portal page. Should a listener, while processing the event, find that another event needs to be generated, that event will be queued by the portlet container and delivered at a time point determined by the portlet container. It is only guaranteed that it will be delivered and that this will happen before the content generation phase. There is no guarantee for event ordering.

Archived

Once the content generation phase has started, no further events will be delivered. For example, messages cannot be sent from within the service, doView or other content generation methods. Attempts to send a message during the content generation phase will result in an org.apache.jetspeed.portlet.AccessDeniedException.

The event listener is implemented directly in the portlet class. The listener can access the PortletRequest.

It is important to understand the underlying event handling and message processing to ensure delivery of all send messages. The portal event handling and message processing sees four steps executed in the following order.

1. Processing all action events

   The user makes a request of the portal, the portal receives the request and decodes the action URI sent by the client and propagates an action event to the appropriate portlet. The receiving portlet's action listener is called to process an action event. An appropriate time to send messages to other portlets is during the processing of the action event.

2. Processing all message events

   If a message is sent to a portlet, the portlet's message listener is called to process the message. Since portlets can send multiple messages and send messages as a result of receiving a message, this process continues until there are no more messaging events pending. Cyclical messaging is prevented by the WebSphere Portal architecture.

3. Processing all window events

   Sizing operations such as maximize, minimize and restore, along with the portlet's ability to request a specific size, causes multiple window events to be sent to all portlets affected by the sizing activity. This processing of window events continues until there are no more window events pending.

4. Portlet rendering process

   Upon completing the event processing in the order specified above, the portal aggregator begins calling each container on the page being displayed, causing its contents to be rendered. The rendering process is explored in detail in 1.3.9, "Page aggregation" on page 32. When aggregation is complete, the page is returned to the user.

Archive

Important: It is important to note that events are not processed in the last step of the process, page rendering. If a message is sent by a portlet during rendering, the message will not be delivered or processed. This is a result of the fact that the event queue is help in the portlet request and at the time of rendering, the portlet request is no longer available. Therefore, if portlet interaction is desired, portlet messages must be sent during the first three steps of the event and aggregation process.

1.3.9 Page aggregation

Portals allow users to choose sets of portlets they would like to work with and provides a framework for displaying those portlets in a consistent fashion.

A defined set of applications, which should be presented in a common environment are referred to as a *page*.

Page aggregation is the process that collects information about the user's choices, the device being used and the selected portlets, then takes that information and combines it to create a display that is appropriate for the device.

The aggregation process involves three basic steps:

- Collecting user information
- Selecting the active applications
- Aggregating the output

Once the active page is determined, the layout of this page is used to aggregate the content of the defined applications, arrange the output and integrate everything into a complete page. Basic Portal Page Layout can be seen in Figure 1-10 on page 34.

Rendering of page components is done using JSPs, images, style sheets, and other resources. These resources are located in the file system in a path-naming convention that the portal server uses to locate the correct resources for the client. WebSphere Portal provides dynamic aggregation of pages from page descriptors held in the portal database.

Archived

Collecting user information

During the collection of user information, the following information is collected:

User The user is authenticated at login and the user identification is available throughout the session.

Client The user's device is determined by information contained in the request header. Once determined, this information is also stored in the session.

Markup The markup is associated with the device category. There are currently three markups defined, HTML, cHTML and WML. New markup scan be added via the Markup Manager Portlet.

Markup version The version for the supported markup. For example, `ie5` for the Internet Explorer family of browsers, `ns` for the Netscape family of browsers.

Language The portal determines the language to be displayed via the following algorithm.

If the user is logged in, the portal user interface is displayed in the preferred language of the user.

If no preferred language is set, the portal UI is displayed in the language set by the client browser if available.

If no browser language is available, the portal UI is displayed in the default language set for the portal.

Portlets not supporting any of the above scenarios display their UI in the portlet's default language.

Page The access control list determines which pages and labels a user has access to.

Theme The name of the active theme is taken from the currently active page.

Screen Depending on the interactions of the user with the portal, different screens are presented. The screen holds the output of the portlets on a page.

Selecting the active applications

During this phase of aggregation, the portal determines the active applications or portlets to be displayed. When the portal receives a request, it determines the active place and the active page for the current user. Aggregation then continues with the rendering of the page.

Archived

Aggregating the output

Once the active page is determined, the portal uses the layout of the page to aggregate the content of the defined applications, to place the output and build the complete page. A page contains components such as row or column containers that contain other components or portlets. Figure 1-10 shows the layout of a portal page.

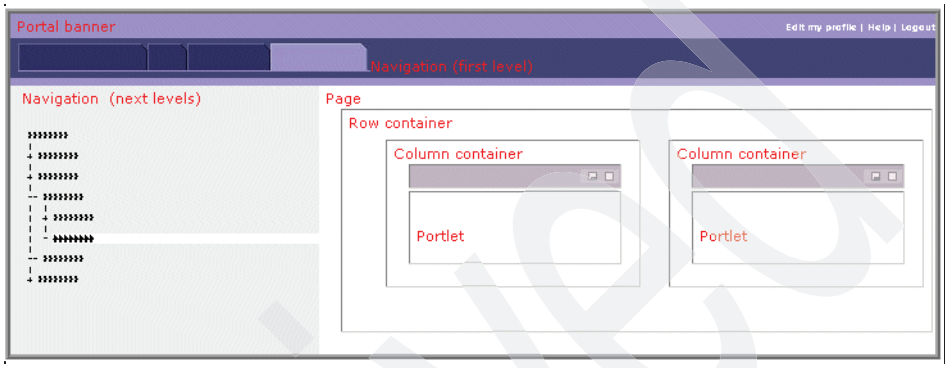


Figure 1-10 Portal page layout

A portal page is made up of the following elements.

Portal window The content inside the displayed window. It is made up of the banner and the portal page.

Banner The top area of the window that holds the company information, the greeting, a page selection box, tabs to select the current page in the page group being displayed and some additional controls for interacting with the portal such as logging in, logging out and help.

Screen Hold the output of the portlets on the currently selected page. The layout is determined by its row and column containers.

Node Node is a level of hierarchy in the portal. Nodes include pages, labels, or URLs, and are used to navigate the portal structure. The portal has a tree structure that is used to organize the portal into branch nodes, which belong to other nodes that are higher in the tree. The single highest node in the portal is called the content root. Nodes are represented and accessed from the portal navigation menu.

Page Page is a type of node that provides portal content, similar to a page on any Web site. However, portal pages display

Archive

	content in the form of portlets, which are arranged on the page by row and column containers. Each page displays content that has been customized for the portal user.
Label	Label is a type of node that has a name and is used to hold other nodes.
URL	URL is a type of node that can open locations within the portal or external Web sites.
Container	A container is an area on a page that contains content. A container can be structured as a row, column, or cell in a table. That is, when you are arranging content on the page, the content can be placed in a container that spans the width of the page (row) or the height of a page (column).
Row	A container inside a page that allows portlets to be arranged in a horizontal format.
Column	A container inside a page that allows portlets to be arranged in a vertical format.
Control	The frame around the portlet is constructed by the frame. It builds the bar above the portlet output including buttons to control the state and view of the portlet.
Portlet	A portlet is a type of application that can be accessed through a small box or window in a portal page. Portlets provide access to specific services or information, for example, a calendar or news feed.

Themes and skins

Window and component layouts can be controlled by themes and skins. Themes refer to the window templates. Themes represent the look and feel of the portal, including background colors, images and fonts, and is also used to render to portal banner. Skins refer to the component templates. It defines the border, margins, and title bar of the portlets on a page. Skins use the theme name to select the graphics that match the theme colors.

Templates

Aggregation uses the concept of templates to perform window, screen and component layout. When a corresponding part needs to be rendered, a template loader will load the requested template. If the requested template cannot be found, the default template will be used. A template consists of the template class that controls the rendering, the localization and the launch of the template JSP. The template JSP performs the actual rendering. There are three types of templates:

Archive

- Window templates

  The Window template is responsible for the layout of the parts of the banner area and the placement of the screen. You can change, for example, the navigation tab location via the window template.

- Screen templates

  The Screen template is responsible for the layout and the content of the screen, the portion of the portal page containing the output of the portlets.

- Component templates

  Component templates are responsible for rendering the component itself and for starting the rendering of its children components. The children of container components (row and column) may be other containers or controls. The child of a control will always be a single portlet.

Page aggregation processing

The rendering process is a domino process starting with the root container. The root container triggers the rendering of all the child components in the page hierarchy as seen in Figure 1-11 on page 37.

Rendering the screen triggers the aggregation of a page and its portlets. The *pageRender* tag in the screen starts the rendering process. If no portlet is maximized, then the *pageRender* tag calls the RootContainer.

The Root Container holds all the containers and controls for this page. The pageRender tag tells the Root Container to invoke the rendering of all its children. Since the Root Container is used only as a starting point for the aggregation, it does not render itself and therefore is not visible on the screen.

Each child of the Root Container invokes its template which is responsible for rendering all the content of its child. If the child contains further child components the *componentLoop* and *componentRender* tags execute the rendering of all these components one at a time.

Each component invokes its own template which in turn invokes more components until the leaf of each branch is reached. Thus, control moves between component classes and their respective JSPs. Each component writes its content to the servlet output stream.

When a control is reached, it invokes the rendering of the portlet, which adds its output to the output stream via its rendering. When the entire tree has been traversed, the output stream is closed and the output is sent back to the requesting client.

Archived

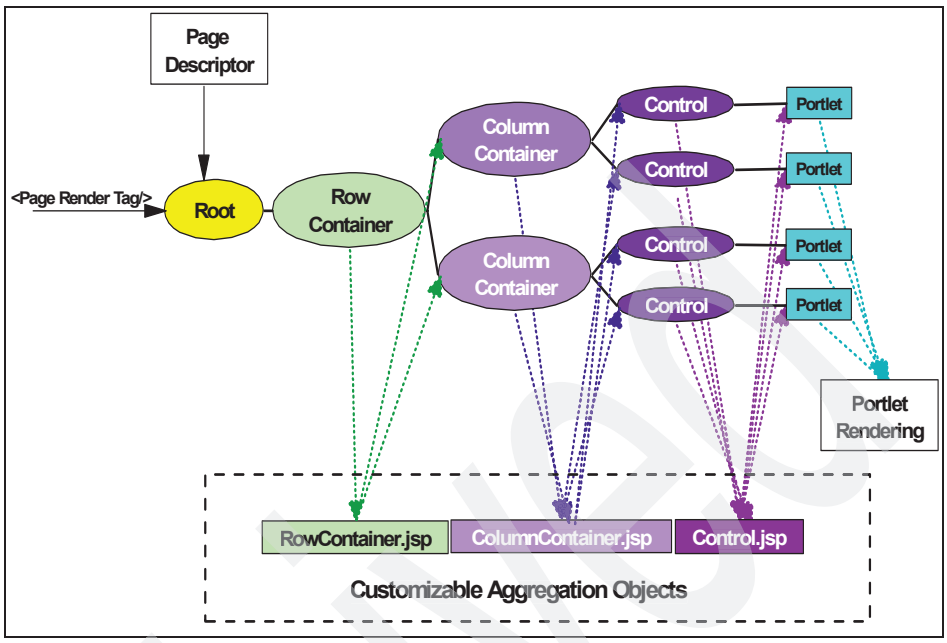


Figure 1-11 Page aggregation

1.4 Highlights in WebSphere Portal V5

In this section, we present general information about Portal V5.

1.4.1 Portal install

While WebSphere 4.x has an install procedure that tried to address all needs and adapt virtually all portal settings to the specific customer environment which resulted in a complex install procedure that required systems administrators to know and specify many things at install time, WebSphere Portal 5.0 has a redesigned install that follows a more modular approach, consisting of the following steps:

1. Installation - This step lays down the required files and only asks for some basic configuration settings. By default, the portal installation installs WebSphere Application Server 5.0.1 EE and the Cloudscape™ database, plus the portal software on top of those. Alternatively, the install can be done on a preexisting installation of WebSphere Application Server 5.0.1 EE optionally using a preexisting database (for example Cloudscape, DB2®,

Oracle, SQL Server, Informix®). Installation of the portal is quick and simple, using common defaults.

2. Configuration - This step allows tailoring a portal installation fit the specific customer environment by running configuration scripts to change the database being used by the portal, switch from using the WMM database as a user registry to using one of the supported LDAP directories, enabling use of a proxy for access of remote content through the portal, etc.

1.4.2 General infrastructure

Support for WebSphere Application Server V5 Enterprise

WebSphere Portal 5.0 runs on WebSphere Application Server 5.01 Enterprise to take advantage of better performance and scalability.

Property Broker incorporating C2A and Cooperative Portlet Data sharing

The Property Broker is intended to support brokering of properties between independently developed portlets. It is intended to support the collaborative requirements of both the Click-to-Action and Dynamic Workplaces™ features. The main runtime pieces are as follows:

- A property broker service (an instance of PortletService), which provides public and private APIs for using or implementing the property brokering function.
- A property match broker, which maintains a repository of property matching rules and carries out matches as required.
- Portlets which use the public APIs on the property broker service to enable the sharing of properties and inform it of changes in shared properties. Portlets may use the existing action mechanism for receiving notification of changes to shared properties, or may implement a new property provider interface, through which such notifications are delivered.
- A property broker portlet filter (an instance of PortletFilter), which filters all calls to portlets.
- The portlet container, which invokes callbacks on the portlets to indicate the start and end of the event processing phase, and which provides an API to the portlet service to invoke selected methods on portlets.

Enabling for Communities

WebSphere Portal 5.0 through its WebSphere Member Manager Component provides support for Communities as special groups with additional meta-information.

Archived

1.4.3 Event broker

WebSphere Portal 5.0 has a portal event broker to which portal components can fire typed events which the broker dispatched to the listeners previously registered for those events. The portal event broker is used to deliver portal internal events across portal components as well as for producing events for event listeners for BEI and site analyzer integration.

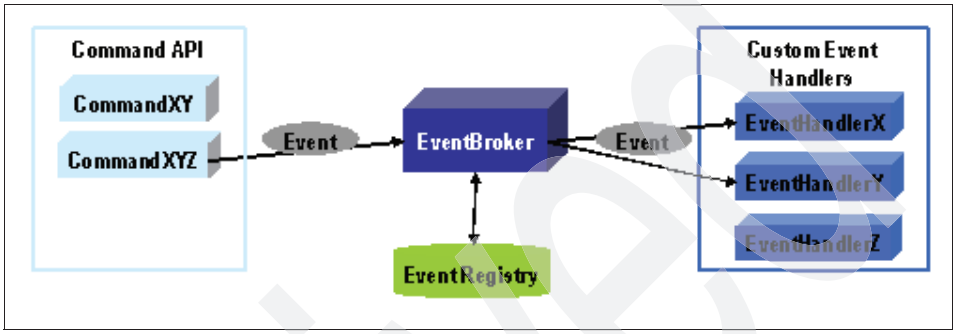


Figure 1-12 Event broker

1.4.4 Member subsystem

WebSphere Portal Server 5.0 uses WebSphere Member Manager instead of WMS.

WebSphere Member Manager can access user information in different types of repositories using WMM Repository Adapters which implement the WMM Member Repository Interface. WMM provides repository adapters for LDAP user profile repositories and the WMM Database user profile repository (supporting the same set of databases as WebSphere Portal). It is also possible to connect custom repositories by implementing a custom profile repository adapter, for example, in service projects.

1.4.5 Authentication

J2EE Security

The authentication function in WebSphere Portal Server 5.x uses the J2EE Security calls to authenticate users instead of the SSO Authenticator calls that had been used in WebSphere Portal 4.x.

Deprecating old SSO functionality

In WebSphere Portal Server 5.x, the old JAAS-based SSO functionality allowing portlets to take user ID and password from the JAAS Subject for the special case

Archived

that no authentication proxy is used is no longer supported. Instead, portlets have to use the Credential Vault that also works in the general case.

1.4.6 Authorization

Enhancing access control for roles and inheritance

WebSphere Portal uses a role-based approach to manage user authorization for accessing portal resources such as portlets and pages. Access control administration can be performed using corresponding portlets within the running portal or via the XMLAccess scripting interface.

Portal access control (PAC) is the single access control decision point within the portal. It controls access to all sensitive portal resources, like for example pages and portlets. PAC is used by various components including the customizer, the different aggregation modules, and the SOAP RPI router that allows for remote invocation of portlets. All these components will allow actions on particular portal resources only if these actions are allowed by PAC.

PAC directly supports access control configuration of hierarchical resource topologies through the concept of permission inheritance. This concept reduces the administration overhead for an administrator when controlling access to a large number of portal resources. Inherited permissions are automatically assembled into roles that can be assigned to individual users and user groups, granting them access to whole sets of logically related portal resources. The "user-to-role-assignments" can be managed within the portal or within external authorization systems (for example Tivoli Access Manager).

To allow for pluggable implementations, the authorization component defines a Service Provider Interface (SPI). WebSphere Portal Server 5.x has a built-in authorization component implementation that plugs into the SPI so that it can be replaced by other implementations easily.

The summarized access control facilities provided by PAC include:

- Instance-level access control for the complete set of portal resources
- Granting/revoking of permissions based on roles
- Support for predefined action sets for convenient creation of roles based on the intrinsic portal resource topology
- Flexible blocking of permission inheritance on a per resource/per action set basis
- Notion of Private Resources to reduce the number of defined roles within the portal for high volume personalized resources

- Delegated administration concept supporting an unlimited number of delegation levels
- Leveraging a sophisticated caching infrastructure for high performance access control decisions
- SPI plug-point for external access control components like, for example, Tivoli Access Manager
- First alignment towards upcoming JSR 115 based authorization facilities that will be provided by WebSphere in the future

1.4.7 URL generation, processing and mappings

WebSphere Portal 5.0 has mechanisms for generating URLs to be embedded in portal or portlet markup and for analyzing the URLs in incoming requests to determine what actions to process and what to display.

Canonical Portal URLs

WebSphere Portal 5.0 supports a canonical URL format that consists of the server name plus one or more GUIDs or Unique Names of URL addressable resources within the portal such as places, pages, and portlet instances.

User-friendly Portal URLs

In addition to canonical URLs, WebSphere Portal 5 can support arbitrary user-friendly URLs defined by administrators explicitly for selected portal resources. To define user-friendly URLs, the portal administrator defines URL contexts organized as trees which have context names as their nodes. The user-friendly URLs result from traversals from the root to a leaf of such a tree.

URL mappings

To translate user-friendly URLs (which in general have an arbitrary structure and do not contain GUIDS or unique names that can be understood by the portal's URL processing) into canonical portal URLs (which contain the correct GUIDs or unique names for portal resources), URL mappings are required.

WebSphere Portal allows administrators to define URL mappings for the parts of URL spaces for which they have the appropriate access rights in two ways: They select a node in the URL spaces and may map it to a URL addressable portal resources they start by selecting a URL addressable portal resource and then selecting the node(s) in the user-friendly URL space which should map to the resource.

Administrators may only define URL mappings for those friendly URL contexts for which they have been granted the appropriate access rights.

Archived

1.4.8 Search

WebSphere Portal 5.0 introduces major improvements in its search functionality, adding categorization, summarization and support of more than 225 document formats through document filters.

Document Categories and Summaries

WebSphere Portal Server 5.0 provides function for automatic categorization of documents as well as automatic generation of document summaries.

Eureka! categorizer

The Eureka! categorizer is a high-accuracy system for categorizing text documents, including those from highly heterogeneous sources such as the Web. Currently, it is used by ibm.com® to categorize IBM's Web pages with 80% accuracy (relative to humans categorizing to the same taxonomy) and 80% coverage. The system consists of two major components, a taxonomy creation system and a categorizer. It is the use of the categorization system which produces the high accuracy of Eureka!

The Eureka! categorizer and associated data can be viewed as a black box that accepts text in either HTML, XML, or flat text, and outputs a list of one or more categories into which the text has been categorized, and a score for each. Optionally, it may also detect the presence of one or more phrases or terms that are then mapped to a specific category. The categorizer is available in multiple languages; however, each language requires a separate invocation of the categorizer.

The categorizer requires that the calling application fetch the text to be categorized and handle the resulting output of categories from the categorizer. In a multilingual application, it may also be necessary for the calling application to determine the language of the text in order to dispatch it to the appropriate categorizer. The categorizer will operate as a server within the UIM Architecture framework.

The Eureka! data consists of a language-specific dictionary of words (stemmed words in English); a language-specific hierarchy of categories; and a set of definitions for each category. The category definitions are produced by the Eureka! "back-end" system, which will remain at HAW. However, the Eureka! system currently consists only of categories in the following areas: computers (excluding computer science), financial markets, and telecommunications. Additional effort is included in the scope of work to expand this set of category definitions to other areas

Summarizer
The Summarizer is configurable to run in three client-selectable modes:

- For certain types of news articles, it will return the initial 255 characters of the text document.
- For documents which have certain narrative quality, it will return N (client-settable) most salient sentences. Additionally, the computation of salience can be made sensitive to a query or to a user profile, presented as a parameter to Summarizer, thus enabling query-biased and profile-biased summaries.
- For applications where it makes sense to define/assume a "domain" and where a domain dictionary/glossary is provided, it will 'highlight' occurrences of domain-specific terms in documents. This mode may also produce "keyword summaries" which list important domain terms in the documents.

Document filter technology
WebSphere Portal 5.x integrates the Outside-In document filters from Stellent. This technology allows the portal to search over 225 document types.

Web crawler
The system is a collection of crawlers, folders, index and filters and build in categorizers/summarizers, which allow the user to build a fixed taxonomy documents (either Web pages or manually uploaded) via a rule-based categorizer or the Eureka! categorizer.

1.4.9 Content management

Portal Document Manager
Portal Document Manager (PDM) is a portlet application which provides a simple, real-time document viewing and contribution solution for Portal users. It is built according to the WebSphere Portal 5.0 portlet style and architecture guidelines and uses the new WPCP Portal Content Management (PCM) API to provide the necessary folder, document and user management functions needed for the PDM solution. PDM will be shipped in all versions of WebSphere Portal V5.0 (including Express). One of PDM's major usability objectives is to provide a simple interface, one that can be used without training, often referred to as a "walk up and use" interface. PDM's target audience includes business professionals, and content contributors who demand a nontechnical interface.

Archived

This release of PDM provides the following functions:

- Document Management: Navigate a hierarchy of documents organized into user-defined folders; Authorized users can add, view, modify and delete folders and documents.
- Access Control: Portal users/user groups used for access control; assign access control rights for folders and documents using Portal access control interface.
- Search: PDM documents and folders are searchable using Portal search engine.
- Workflow Process: Using built-in workflow, assign approvers for workflow process during PDM configuration. Approvers must approve new and changed documents before they are made public. Work items show up in the Tasks folder.
- Subscription: Allows subscription to documents and folders. Subscription folder shows subscribed documents.
- Integration with On-Demand Client (ODC) components: ODC editors (RichTextEditor, Presentation Editor and Spreadsheet Editor) can be used to edit PDM documents. ODC Mailbox portlet can save attachments as PDM documents or attach PDM documents to e-mail. ODC document conversion services are used when needed to change PDM document formats.
- Versioning: The user can create new versions of documents. The user can view and retrieve document versions. PDM provides built-in versioning support but can be configured to support CVS, IBM CM, and ClearCase®.

1.4.10 Content publishing

WebSphere Portal content publishing (WPCP) provides a Web content management solution that gives nontechnical users greater control over content published to portals and other Web sites. Users benefit from the combined power of having one place to manage content for their Portal environment or other Web sites and an easy-to-use Web interface. This interface puts content management back into the hands of nontechnical business users and provides them with tools such as personalization rules, templates, workflow, and versioning, that make the content creation process simple, yet controlled. WPCP decreases Web maintenance and administration costs, increases sales and profits by deploying timely and personalized content, and improves efficiency by getting all content produced in an enterprise to the Web.

1.4.11 Transcoding

Transcoding technology was incorporated into WebSphere Portal 4.1. As transcoding technology serves different market through various IBM offerings, including WebSphere Portal, number of markup language transformation were not enabled in WebSphere Portal.

Starting with WebSphere Portal V5, plug-ins for WML and cHTML markup transformation are enabled. WebSphere Transcoding Publisher will be bundled as part of the portal server core install package. This will alleviate the need to have a separate installation for WebSphere Transcoding Publisher and Portal.

The aim is to simplify the installation process and reducing the chance of an error during installation of portal and later during migration. In an effort to do this, now transcoding is installed inside the portal directory; this includes moving transcoding classes to the *shared app* directory. Configuration steps are also simplified by pre-configuring portal property files with transcoding information.

1.4.12 Struts Portlet Framework

Struts is a very popular framework for Web applications using a mode-view-controller design pattern. The Struts framework can be used to effectively design Web applications and support development teams of different sizes and organization.

For WebSphere Portal 4.2, the Struts Portlet Framework was updated to include the Struts 1.1 Beta 3 release and support for Tiles and File upload was added.

The Struts Portlet Framework in the WebSphere Portal 4.2 implementation is closely tied to Portal Core API, so changes there will effect the Struts Portlet framework and require changes to function in the WebSphere Portal version 5 product. There are also WebSphere Application Server V5 dependencies that need to be addressed. The new functions that end users will see again are supported for newer releases of Struts.

1.4.13 User interface

WebSphere Portal V5 implemented a new containment model.The functions of the containment model can be grouped into two parts: information supply and administration.

The next two sections deal with these aspects, a further section explains the structure of the containment model.

Archive

Information supply

The containment model provides the information needed to perform tasks such as content aggregation or building navigation to browse the aggregated content. The information supplied can be dependent on a specific user; it would be a user-specific view on the containment model.

Administration

The information in the containment model must be changeable, of course. This can only be done via the Command API. The commands can manipulate information on the content, navigation, derivation and any other information stored in the containment model. Elements can be managed via PAC to enable the permission concept of the portal, this means each element of the containment model is a resource which can be protected in regards to what action can be performed on it for a specific user.

The administration of the containment model allows you to:

- Add and delete root nodes
- Add, move, reorder and delete child nodes of a node
- Modify a node, including:
  - Changing associated information
  - Implicitly or explicitly deriving a content node (a page)

1.4.14 Cooperative portlets (Click-To-Action)

One of the most significant advantages of the Portlet architecture is the portlets' ability to communicate with one another to create dynamic, interactive applications. Portlets can use messages to share information, notify each other of a user's actions or simply help better manage screen real estate.

Messages can be sent to all portlets on a page, to a specific named portlet or to all portlets in a single portlet application.

User-Driven Process Integration extensions to C2A

Enhancements to C2A which would contribute to the realization of the User-Driven Process Integration (UDPI) idea would be remembering the user choice for each step (so that only that choice is presented or automatically executed during subsequent interactions), supporting cross-page data transfer, so that the next step in the task is automatically surfaced to the user, supporting the notion of "sticky notes" which the user can attach to chosen sources (as reminders in a partially completed process of what he intends to do next), etc.

Archive

Also, a user with special privileges should be able to save his choices (which in effect will define a particular process connecting a set of diverse applications) for import and use by other users or all users in a group or organization.

Property wiring tool

The wiring tool may be invoked as part of editing a page. It provides the capability to view sharable properties on each portlet instance and create wirings between them. It also provides the capability to view the existing set of wiring rules for the current page.

In order to obtain information about the sharable properties, the tool invokes listProperties on the property broker. In order to obtain information about the existing rules, it invokes getMatchRules on the PropertyBrokerServiceInternal interface. This allows the user to pairwise choose compatible properties on two portlet instances and wire them up, or to specify that type-based matching be used for a specified property on a portlet.

After the user has created the matching rules using this tool, the tool will invoke setMatchRules on the PropertyBrokerServiceInternal interface. The property broker service will store the rules persistently and cause the property match broker to update its in-memory data structures to add the new rules.

The Portlet Wiring Tool is not provided with the WebSphere Portal V5 product package but it can be downloaded from the Portlet Catalog at `http://www.ibm.com/websphere/portal/portlet/catalog` and search for IBM Portlet Wiring Tool V5.0. The Navigation code is 1WP10004E. This portlet should be placed on a page called Wires under the Page Customizer.

1.4.15 Portal Toolkit

The Portal Toolkit for WebSphere Portal is an add-on component that installs into WebSphere Studio Site Developer and adds portlet development and debug functionality. The toolkit includes two primary components and a set of example portlets which demonstrate portlet programming techniques.

The Portlet Wizard components allows a developer to begin development of a new portlet application. The developer specifies the portlet application name, portlet name, Java class name for the portlet, and the markups to be supported by the portlet. The wizard then generates a skeleton portlet application as a project in WebSphere Studio Site Developer. This project includes a Java source file that represents the portlet controller, a Java class that implements a Java bean to transfer display data from the controller class to the display JSPs, and sample display JSPs for all supported portlet modes and display markups. The project also includes properly completed web.xml and portlet.xml documents.

The Portlet Application debug components allow the developer to source debug a portlet. The developer defines a server instance for local debug, with WebSphere Portal running inside WebSphere Studio Site Developer, remote debug, with WebSphere Portal running on a remote instance of WebSphere Application Server, and remote attach, which allows the developer to debug a portal within a full portal production runtime stack.

The toolkit also includes interactive, dialog driven editors for the portlet.xml and web.xml documents. As the developer changes Java files or JSPs, these resources are automatically recompiled and validated.

A portlet application project may be packaged as a standard portal WAR file and exported to a portal server at any time.

In Portal Toolkit V5.0, the following enhancements are included:

- WebSphere Portal 5.0 Currency / Portlet API support
- Updated portlet wizard
- Additional portlet examples

1.5 Portlet solution patterns

Enterprise Resource Planning (ERP) and Customer Relationship Management (CRM) systems are excellent candidates for portlets because efficient, personalized access to these functions provide measurable returns on your portal investment. WebSphere Portal includes portlets that help you access a variety of ERP and CRM systems.

Enterprise Applications running on a back-end or host system are another group of candidates for portlets, especially when the portal addresses the business-to-employee pattern and you want to provide a common working environment to your users, whatever application and system they may need for their work.

There are many ways to perform application integration in a Web environment. Not all of them are based on portlets and amongst the portlet-based solutions, several different architectural approaches can be applied. Depending on technical circumstances, the given time frame and the goals of the integration, typically, different approaches may be combined in one portal solution.

We try to list some of the patterns you might think of. One way we can differentiate is shrink-wrapped versus roll-your-own.

Archived

Customizable portlets from a vendor

In this pattern, a portlet is provided which can be installed in Portlet Server and, after a configuration effort, the system or application in question can be accessed through the portal. Often, such a portlet is delivered by the vendor of the system that should be accessed. Both the Host On-Demand portlet and the Host Publisher portlet we use in the following examples are of this type.

Custom developed portlets

This pattern comes into play when either no vendor offers a portlet for the requested application, or the requested level of functionality, usability, accessibility or security is not met by the existing portlets. Another reason might be that you want to combine information or functionality of multiple applications seamlessly into one portlet.

Most probably, this integration will include using the Java Connector Architecture (JCA). JCA is a standard architecture for integrating J2EE applications with Enterprise Information Systems that are not relational databases. Each of these systems provides native APIs for identifying a function to call, specifying its input data, and processing its output data. The goal of the JCA is to achieve an independent API for coding these functions.

JCA also defines a standard Service Provider Interface (SPI) for integrating the transaction, security and connection management facilities of an application server with those of a transactional resource manager. Thus, JCA is a standards-based approach to managing connections, transactions, and secure access to enterprise application systems. IBM's JCA connectors provide access to systems such as SAP, People Soft, CICS®, and IMS™. Leveraging its CrossWorlds® acquisition, IBM will also develop and integrate JCA connectors to many other systems.

Another way to look at portlets for application integration is from a topology point of view.

Client to remote application

In this pattern, for example used by IBM Host On-Demand, the portlet is just a bootstrap to allow the client to get in touch with the requested system or application, and Portal Server is the framework for the user interface. This implies that normally, an applet is involved which makes a direct network connection to a remote system.

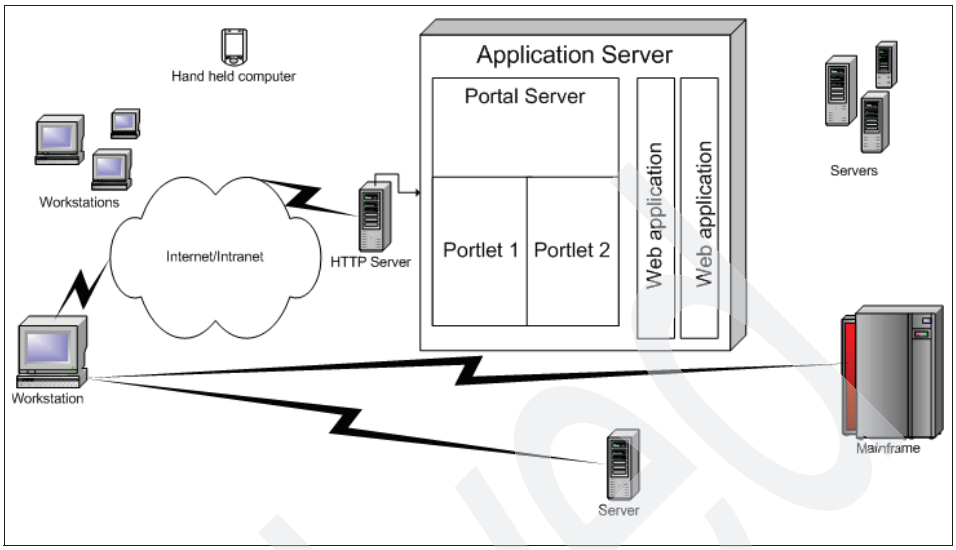


Figure 1-13 Portal Solutions - client to remote application

Portlet to remote application

This is the topology most likely used if you write your own application integration portlet. Access to the requested application or information is gained through standardized interfaces such as JCA connectors, JDBC and JMS, or by using a proprietary API provided by the application that is to be integrated (for example SAP Business Connector).

Archived

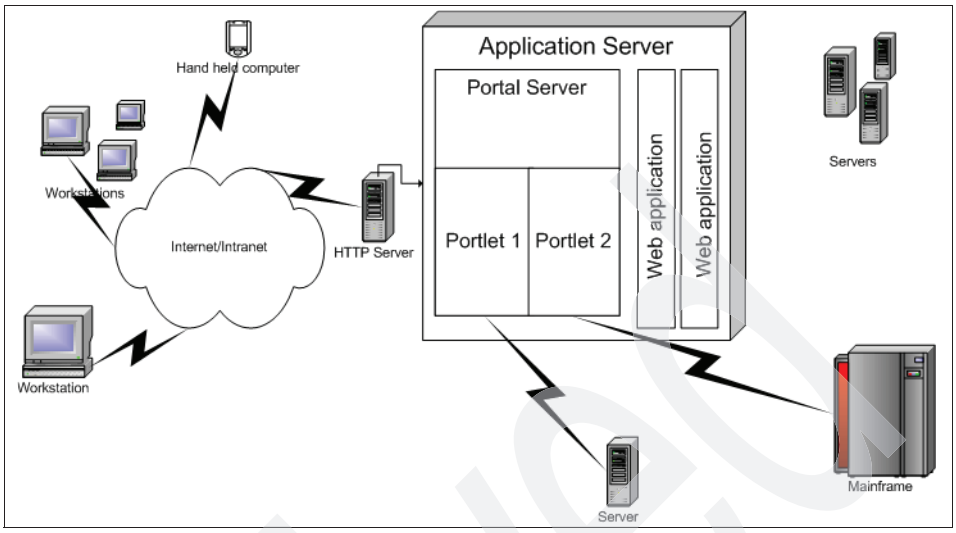


Figure 1-14 Portal Solutions - portlet to remote application

Portlet to Web application

In this pattern, most of the work is done in a Web application. Also, if you write a Web application using the JCA or EJB and create a portlet interface to it, you follow this pattern. The enterprise application integration does not stop here at integrating with ERP and CRM systems.

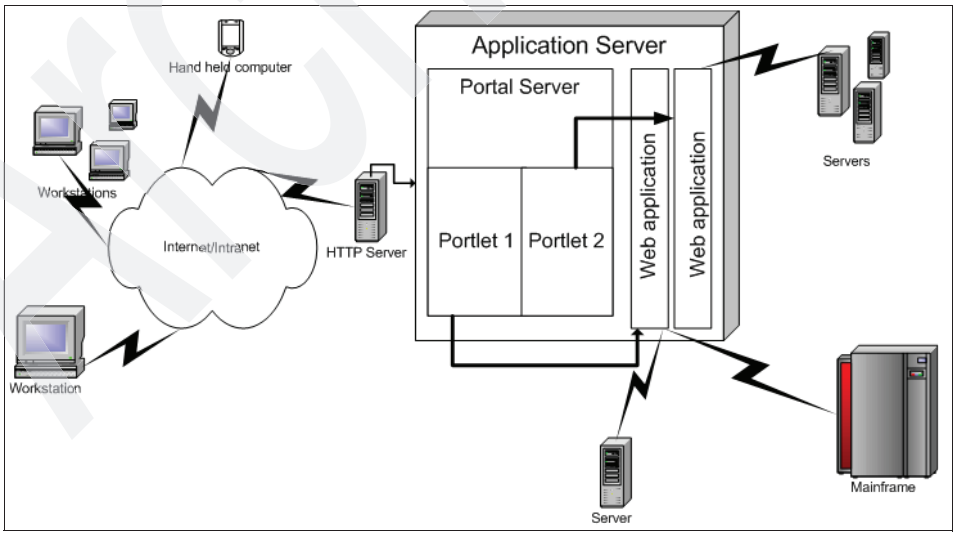


Figure 1-15 Portal Solutions - Portlet to Web application

Archive

Archived

2

Portlet API

This chapter provides details on the Portlet life cycle, Portlet API and deployment concerns.The goal of this chapter is to provide you with the ability not only to design and build dynamic portlet applications, but also to recognize opportunities to portalize existing applications and services.

At the end of this chapter, you should be able to work with the Portlet API to design and build new portlet applications. You will have the requisite skills to deploy new applications as well as existing portalized applications. The WebSphere Studio Application Developer environment is covered in Chapter 3, “Portal Toolkit” on page 125 and as such will not be discussed here. In this chapter, all development and deployment information will be development environment independent.

Archived

© Copyright IBM Corp. 2004. All rights reserved.

2.1 What is a portlet?

A portlet is a server side application that runs in the context of the WebSphere Portal Server. It inherits from the javax.servlet.http.HttpServlet class and as such is treated as a servlet by the application server. The portlet is executed inside a Web container managed by the application server. In the Portlet API, this container is referred to as the Portlet container.

Note: It is not possible to directly execute the portlet functionality by addressing the portlet via http.

Though a portlet may provide dual functionality as both a servlet and a portlet, it is certainly best practice to keep these controller functions separate. A portlet is visible on a portal page as a single small window, of which each portal page may have many. The portlet is the content inside the window, not the window itself. The window is defined by the selected skin.

2.2 Basic portlet terms

In order to fully understand some of the introductory topics, it is necessary to define a few of the most basic terms used when discussing portlets.

Portlet window

This is the window that surrounds the portlet, including the title bar and any border images.

State

This is the current state of the portlet window. Valid states are Normal, Minimized and Maximized.

Mode

This defines the current condition of the portlet. The modes that are available for any particular user depend on the permissions for that user, the device used to access the portlet and the configuration and implementation of the portlet.

Note: All portlets support the default mode, View.

Archived

The following portlet modes are supported:

- **View**. When a user is simply viewing the portlet, likely with other portlets on the page, it is in View mode.
- **Edit**. When the user selects the **Edit** button to change some configuration information, the portlet is in Edit mode. Users only have access to the Edit mode if they have been granted edit access by the administrator.
- **Configure**. The Configure mode is conceptually similar to Edit mode in that it is used to adjust the configuration of the portlet. However, only users with manage permissions on a portlet have access to the Configure mode. In practice, the average user may have edit permissions on a portlet to change certain personal settings such as user IDs and passwords. Typically, only administrators would have manage permissions on a portlet in order to adjust non-user specific settings such as server names, etc. The actual implementation of the Edit and Configure modes, however, is entirely up to the portlet developer.
- **Help**. The Help mode is used to present help information.

2.3 MVC architecture

To help you understand the role of a portlet and prepare you for effective and well-designed portlet development, a review of the model-view-control (MVC) architecture is necessary. This section will briefly review the MVC Model 2 architecture.

In the simplest of forms, the MVC model 2 architecture is illustrated as in Figure 2-1 on page 56.

Archived

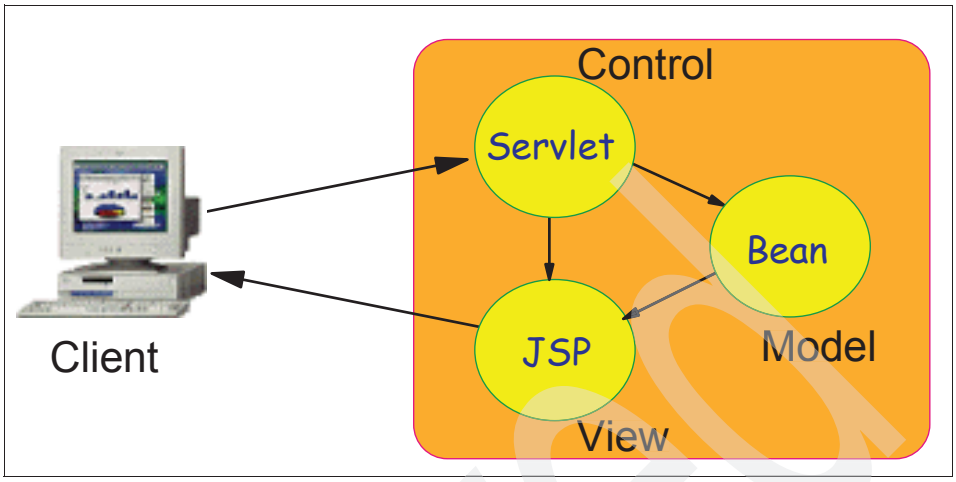


Figure 2-1 Simple MVC architecture

2.3.1 Standard MVC architecture

The Model View Control architecture is concerned with separation of responsibilities. The objective, no matter how it is applied or to what type of application, is to segregate a system into components. Each component should be small, identifiable, self-contained and reusable. These components are identified by the role they play in the system. Each role in that system may have several classes working in conjunction to achieve the goal of that role. This section will cover the three roles of MVC: Model, View and Control.

Though the MVC architecture was originally applied to Swing applications, it has gained popularity and widespread acceptance throughout the servlet community. This section is technology-independent, but will use the servlet technology to demonstrate the application of MVC.

Model

This component is responsible for encapsulating all the business logic required by the system. It must be independent of the other components in the system. To achieve this objective, it must be able to retrieve the data required to complete the business rules data by itself or accept very generic receive parameters. Furthermore, it must be able to return the results in a generic form that any potential View component could use. In a typical servlet environment, the Model is represented by one or more Java beans, EJBs or other Java classes.

View

This component is responsible for creating a presentation resource for the results of the Model component. Like all MVC components, the View must be independent of the other components in the system. Its success or failure must not depend on the success or failure of the Model component. In practice, several different View components may be developed in order to create a dynamic, complete and possibly multi-purpose application. In a typical servlet environment, the View is created using Java Server Pages.

Control

At the heart of the MVC architecture is the Controller component. Each client request of the system is routed through the Controller class. Its responsibility is threefold. First, it should evaluate the validity of the request, including the user's state and any parameter information passed as part of the request. The Controller then decides which Model component has the requisite functionality to satisfy the business requirements of the request. Once the Model component has completed its work, the Controller is responsible for deciding on the appropriate View component to present the results back to the client. If either one of the Model or View components fails, the Controller is responsible for either attempting to satisfy the request in another fashion or deciding on an appropriate View component capable of presenting an error message. In a typical servlet environment, the servlet itself plays the role of the Controller.

2.3.2 Portlet MVC architecture

The MVC architecture can be applied as a design pattern to any system needing to achieve separation of responsibilities. In fact, you will see as you continue through this redbook that the Portal Server itself is architected this way. Furthermore, several of the benefits of the portlet architecture are available to you only if you employ a good MVC design.

Model

The Model in a portlet application is not necessarily different from the Model in any other Java server side application.The Model represents business logic and should not be concerned with the Controller or the View. The Controller could be a servlet, portlet, or a simple Java class. The View could be a JSP or even simple HTML. In theory, then, provided that existing applications employ solid MVC practices, porting the functionality WebSphere Portal should not require any changes to the logic. However, in practice, there are always applications that lack this foresight. The rich API covered later in this chapter will arm you with the tools to tackle this situation. Implementing a rigid commitment to the MVC architecture now will conserve an enormous amount of effort in later migrations or maintenance duties.

View

Like the servlet MVC implementation, the View is traditionally implemented using JSPs or simple HTML. However, because the HTML the View returns will be aggregated, it must not contain page-level tags and must be very mindful of the environment in which it is executing. Furthermore, the Portlet API provides tag libraries which aid in creating dynamic view resources for the portlet environment.

Control

The Controller is responsible for determining the requested mode, executing an appropriate Model and selecting the correct View. The portlet class itself acts as the Controller. Instead of determining the request method as in servlets, portlets need to determine the mode the user has requested. In a normal presentation, where a page is built with several portlets on it, the mode is View. The user, with appropriate permissions, may click the **Edit** button in order to perform some edit functionality. In this case, the mode is Edit. WebSphere Portal also supports Help and Configure modes.

2.3.3 Portlet MVC sample

In the simplest of portlet applications, the MVC architecture would be applied as in Figure 2-2. Note that Figure 2-2 does not reflect the architecture of the Portal Server, simply the portlets executed by the server in any given single request.

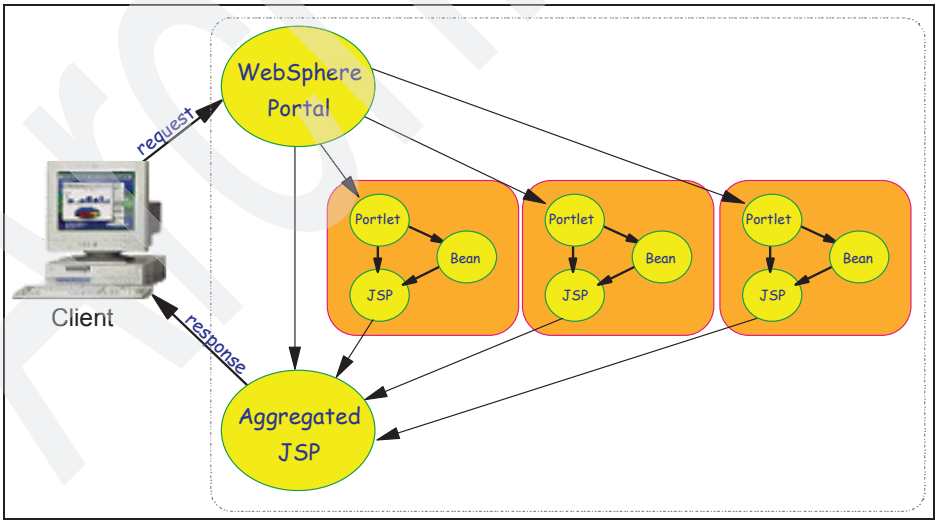


Figure 2-2 Simple Portlet MVC architecture

Archive

2.4 Servlets versus portlets

Those coming from a servlet background will find many similarities when first working with portlets. This section will address some of the more important conceptual differences between servlets and portlets. When designing your portlet applications, the most important factor to initially consider is that unlike servlets, portlets are only a small piece of a large presentation.

Servlets have the luxury of knowing they will be the only presentation resource returned to the client at any given time. Portlets, on the other hand, must understand that the presentation resource they return will be aggregated into a larger resource returned to the client. As a result, they are forced to consider constraints such as screen real estate, portlet interactivity, and events as well as overall performance.

Real estate

Portlets can access a variety of information through the API to help it understand its current condition in the portal. The PortletState informs the portlet if the user has requested the portlet to be minimized, maximized or restored (normal). A portlet should attempt to tailor the content it returns in accordance with the requested state.

For example, if the user has maximized the portlet window, the content returned should adequately fill the portal page. However, if the user has requested that the portlet be minimized, there is no need to return any content. It is important to remember that although the portlet is simply deciding not to return content, it is still executed and any business logic encapsulated in the model will still be performed. It is not possible to dynamically change the state of the portlet except during event handling.

Page aggregation

Although a servlet may be a single piece of a much larger Web application, at any given point in time only a single servlet is fulfilling a user's request. This provides a great deal of predictability in that as the master controller, it can guarantee what is executed and returned to the client. This is not true of portlets. Each portal page is potentially the aggregation of several portlets.

Furthermore, when a servlet executes and returns content to the user, it can be sure that the content it returns will not be affected by any other servlet in the system. This is not true of portlets. A portlet has the ability to write markup to the top of the page even though its normal content is placed inside a cell in a table. This provides a mechanism to include JavaScript functionality the portlet may need. Be aware, however, that as one portlet has that ability, so do all. As such, you must properly encode variable names and functions.

Archive

This functionality must be used with care as there is no inherent mechanism for one portlet to control the presence or absence of another portlet on a page, and as such it cannot reasonably predict what other page-level code may be present.

Inter-portlet communication

Servlets have the ability to share data through a variety of scopes but since they are executed serially by the client, they cannot interact with each other during a single request. Because portlets are pieces of a larger portal, they have the ability to communicate with other portlets and to be affected by other portlets in a single request. This inter-portlet communication provides a way to create a dynamic portlet application crossing multiple portlets on the same page.

For example, one portlet can inform other portlets in the same portlet application or the same page that a user has performed some action. The listening portlets can then alter their presentation, perform alternative logic or otherwise change their behavior.

Event handling

In the servlet architecture, events are represented via HTTP methods. For example, when a user submits a form, the doPost method is called. The portlet event model, however, closely mirrors the traditional Java event model in that portlets implement appropriate interfaces and are notified by the Portal Server when these events are fired. For example, when a user clicks button, an action event is generated and sent to the registered listener. The Portlet API also provides WindowEvents and MessageEvents.

Security

Servlets execute in a neutral environment and are inherently responsible for validating the user's authenticity and/or authority to make a specific request. This is traditionally a function of the controller role. A portlet, on the other hand, operates only in the context of the portal server and cannot be called directly.

The Portal Server is responsible for authentication and authorizing all user access. Therefore, portlets can be reasonably assured that authentication and authorization has been performed prior to their execution. They may however perform some authorization in order to tailor content to a specific user or role. Where in servlets, authentication is a daily concern of developers, it is an option for portlet developers.

Archive

2.5 What is a portlet application?

A portlet application is a group of logically associated portlets. At a minimum, a portlet application defines a single portlet, such as in weather portlet application. In practice, the application may contain several portlets such as the exchange2000 portlet application which contains five portlets as illustrated in Figure 2-3.

Portlets defined in the same application may share configuration parameters set in the deployment descriptor or by the administrator at runtime. They also have the ability to communicate with each other with custom messages.

Portlet Applications are defined in the deployment descriptors at development time and cannot be created dynamically by the administrator. From an administrative perspective, portlet applications allow repetitive administrative tasks to be completed on a group of portlets instead of on individual portlets.

From a development perspective, portlet applications allow developers to provide for deployment all the portlets needed to achieve a business requirement and to ensure, at a minimum, that all these components are installed into the Portal Server.

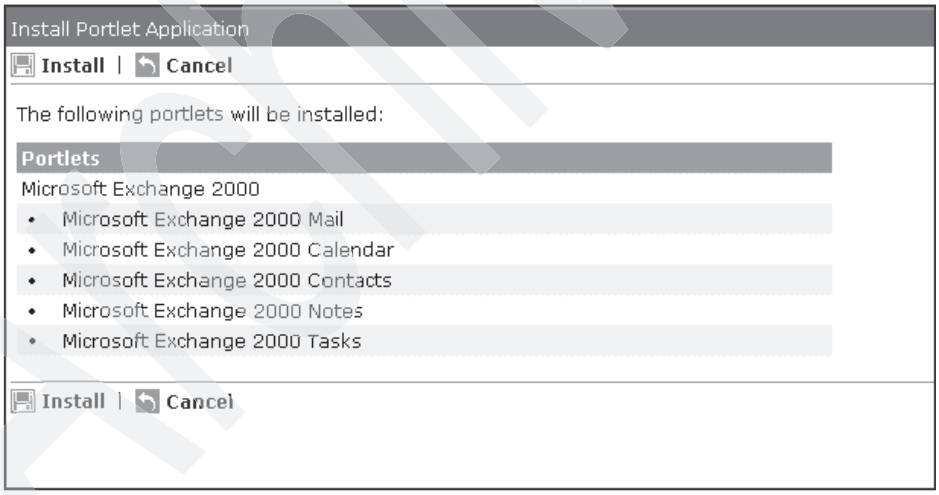


Figure 2-3 A Portlet application with multiple portlets

2.6 Portlet deployment

When a portlet is installed into the portal server, the deployment information is contained in two deployment descriptors. The Web.xml deployment descriptor is

used by the WebSphere Application Server to register the portlets being deployed. The application server is not aware of the portlet hierarchy and is simply installing a Web application containing servlets. The portlet.xml deployment descriptor is used by the portal server to retrieve parameters and to set the initial configuration.

When the portlet application is deployed, it is deployed as a Web archive (war), *not* an enterprise application (ear). When the war file is deployed, the portal server actually creates an associated ear folder in the WebSphere\AppServer\installedApps directory. The new ear folder will contain the original name of the war file with _WPS_PA_191.ear appended to the end of the name of the war file. The last number indicates the order in which the portlet application was installed.

The war file that was deployed is placed into the ear file with a META-INF folder. This folder contains the application.xml deployment descriptor, the ibm-application-ext.xmi and the Manifest.MF files. All these files are generated by the portal server when the application is installed.

Abstract and concrete portlet applications

The portlet.xml deployment descriptor is used by the portal server to identify the abstract and concrete portlet applications you wish to deploy. An abstract portlet application contains one or more abstract portlets. A concrete portlet application contains one or more concrete portlets. The abstract application is used as a foundation for concrete applications. The combination of an abstract application and configuration parameters creates a concrete application. Each portlet.xml may only define a single abstract portlet application. However, any number of concrete applications may be defined based on the single abstract application.

The concrete applications are the applications presented in the portal server administration. They are the actual applications available to the users. This allows the same application to be deployed repeatedly with different parameterization, effectively creating multiple applications.

Arihco

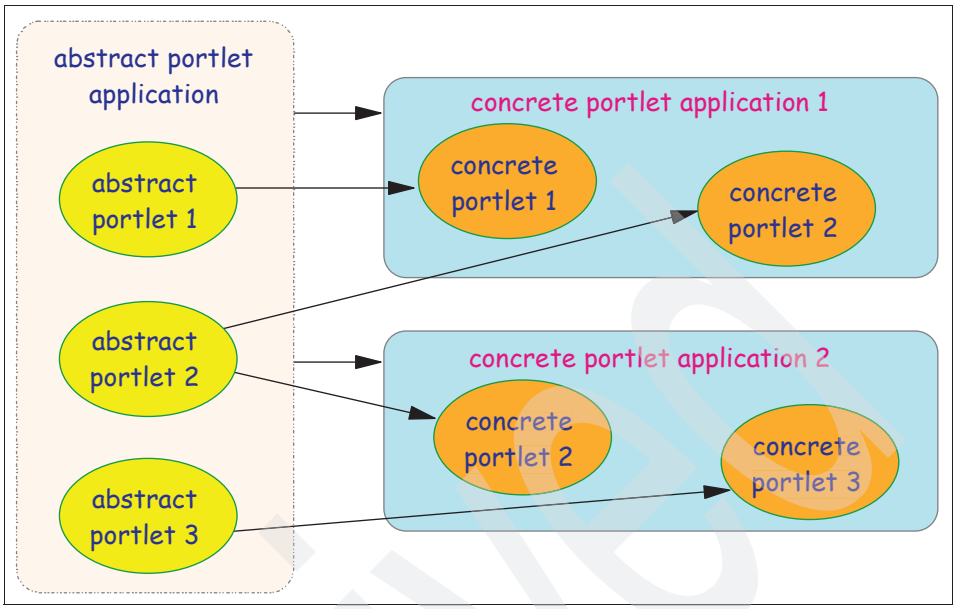


Figure 2-4 Abstract and Concrete relationship in a single portlet.xml

Certain configuration parameters are set at the abstract level while others are set at the concrete level. As expected, parameters set at the abstract level affect each concrete application. The parameters and configuration information unique to each concrete application are represented in a PortletApplicationSettings object. Parameter and configuration information unique to each portlet are represented in the PortletSettings object. When the portlet is actually placed on a page, it is parameterized by a PortletData object.

When a user logs on and accesses a portlet, it is associated with a session object. This sequence is represented in Figure 2-5 on page 64.

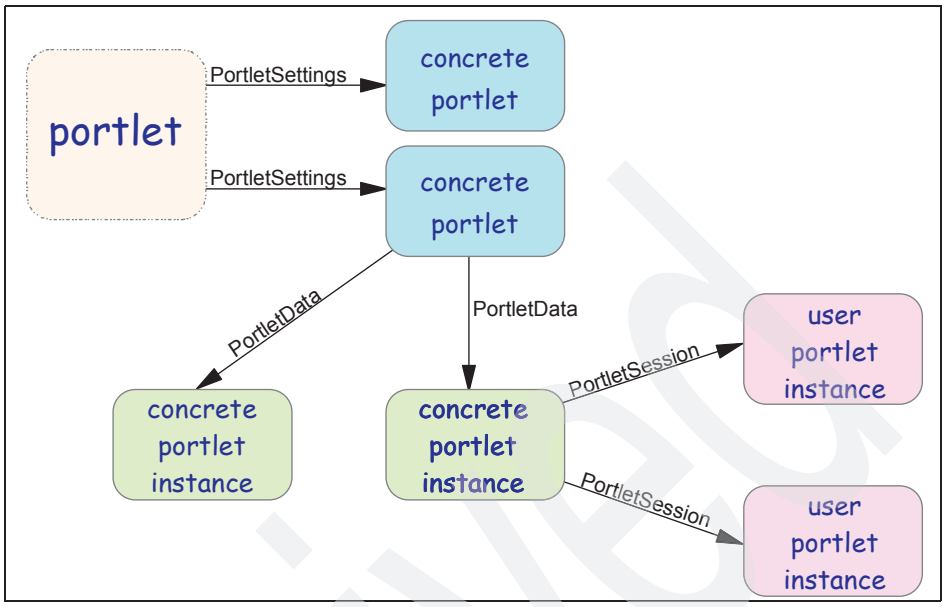


Figure 2-5 Portlet parameterization

2.6.1 web.xml

The web.xml defines the Web application being deployed. This section will detail the required elements of the web.xml when deploying a portlet application. For details on all of the elements of the web.xml deployment descriptor, refer to http://java.sun.com/j2ee/tutorial. Example 2-1 provides an example web.xml document. To keep this section simple, the deployment descriptors will define only a single portlet in the application.

Example 2-1 Simplest web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.2//EN" "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app id="WebApp">
   <display-name>HelloWorld</display-name>
   <context-param>
      <param-name>webmaster</param-name>
      <param-value>the webmaster</param-value>
   </context-param>
   <servlet id="Servlet_1">
      <servlet-name>HelloWorldPortlet</servlet-name>
      <servlet-class>
            com.ibm.itso.ral.portlets.HelloWorldPortlet
```

```
      </servlet-class>
      <init-param>
         <param-name> init_param1 </param-name>
         <param-value> An initialization parameter </param-value>
      </init-param>
   </servlet>
   <servlet-mapping>
      <servlet-name>HelloWorldPortlet</servlet-name>
      <url-pattern>/HelloWorldPortlet/*</url-pattern>
   </servlet-mapping>
</web-app>
```

DOCTYPE *required*

This will be the same for each and every web.xml deployed. This tag defines the DTD that is to be used when this document is parsed. Only one is allowed.

web-app *required*

This is the top-level tag wrapping the entity of the web.xml. Only one is allowed.

– **id** *required*

  This attribute identifies the web-app in the application server but is not seen in the portal environment.

display-name *required*

Though this name is never seen in the portal environment, it will be seen in the WebSphere Administrator's console Event Message screen when the Web module is loaded. Only one is allowed.

context-param *optional*

This element provides an opportunity to set context parameters that will be shared by all portlets deployed via this Web application. Every portlet subsequently based on this web.xml will share access to the context parameters via the PortletContext object. There is no limit on the number of context parameters that may be set. These parameters cannot be changed at runtime by the administrator. For more information on parameters see "Parameter summary" on page 76.

param-name *required*

This indicates the name of the parameter. This name is used in code to retrieve the parameter value. For a summary on the various parameters set in the deployment descriptors, see "Parameter summary" on page 76.

Archived

param-value *required*

The String value held by the parameter.

servlet *required*

This tag wraps the definition of the servlet class. There must be one or more of these tags.

– **id** *required*

  This provides an identifier for the defined servlet. This id will be used in the portlet.xml to refer to the defined class. The id must be unique within the Web application only.

servlet-name *required*

This name is not used by the portlet environment.

servlet-class *required*

The full class name must be provided. This class must be accessible either in the deployed war, via the application server library or through the classpath.

init-param *optional*

This element provides an opportunity to set parameters that will be shared by all portlets based on this servlet. These parameters are available in code through the PortletConfig object. There is no limit on the number of init parameters that may be set. These parameters cannot be changed at runtime by the administrator. For a summary on the various parameters set in the deployment descriptors, see “Parameter summary” on page 76.

param-name *required*

Indicates the name of the parameter. This name is used in code to retrieve the parameter value.

param-value *required*

The value held by the parameter.

servlet-mapping *required*

Though this element is not used by the portal environment it is a required element of the web-app element and therefore must be included.

servlet-name *required*

This is required and must be the same as the servlet name defined in the servlet element.

Archived

url-pattern *required*

This tag is required and may not be empty. The URL pattern is not used in the portal environment.

There are numerous other elements in traditional web.xml but they are not required in the portal environment.

2.6.2 portlet.xml

The portlet.xml elements are defined by the portlet_1.1.dtd which can be found in the WebSphere\PortalServer\app\wps.ear\wps.war\dtd directory. Figure 2-2 displays a simple portlet.xml.

Example 2-2 portlet.xml deployment descriptor

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE portlet-app-def PUBLIC "-//IBM//DTD Portlet Application 1.1//EN"
"portlet_1.1.dtd">
<portlet-app-def>
    <portlet-app uid="DCE:d798f9c6c:1" major-version="2" minor-version="1">
        <portlet-app-name>HelloWorld application</portlet-app-name>
        <portlet id="HWPortlet_1" href="WEB-INF/web.xml#Servlet_1"
                        major-version="3" minor-version="2">
            <portlet-name>HelloWorld portlet</portlet-name>
            <cache>
                <expires>30</expires>
                <shared>yes</shared>
            </cache>
            <allows>
                <maximized/>
                <minimized/>
            </allows>
            <supports>
                <markup name="html">
                    <view output="fragment"/>
                    <configure/>
                    <edit/>
                    <help output="document"/>
                </markup>
                <markup name="wml">
                    <view/>
                </markup>
            </supports>
        </portlet>
    </portlet-app>
    <concrete-portlet-app uid="DCE:d798f9c6c:1.1">
        <portlet-app-name>HelloWorld application</portlet-app-name>
        <context-param>
```

Archived

```
            <param-name>context_param1</param-name>
            <param-value>A context parameter</param-value>
        </context-param>
        <concrete-portlet href="#HWPortlet_1">
            <portlet-name>HelloWorld portlet</portlet-name>
            <default-locale>en</default-locale>
            <language locale="en">
                <title>HelloWorld</title>
                <title-short>Hello</title-short>
                <description>A simple hello world portlet.</description>
                <keywords>Hello World simple</keywords>
            </language>
            <language locale="it">
                <title>ciao mondo</title>
                <title-short>ciao</title-short>
                <description>Un portlet semplice del mondo di ciao.
                </description>
                <keywords>Ciao mondo semplice.</keywords>
            </language>
            <language locale="es">
                <title>hola mundo</title>
            </language>
            <config-param>
                <param-name>config_param1</param-name>
                <param-value>A configuration parameter</param-value>
            </config-param>
        </concrete-portlet>
    </concrete-portlet-app>
</portlet-app-def>
```

DOCTYPE *required*

This will be the same for each and every portlet.xml deployed. This tag defines the DTD that is to be used when this document is parsed. Only one is allowed.

portlet-app-def *required*

This is the top-level tag which encapsulates all abstract and concrete portlet application definitions. It is required and not more than one is allowed.

portlet-app *required*

This tag is used to define the abstract portlet application. This abstract application will be used as a basis for the concrete portlet applications defined later in the descriptor. Only one portlet-app tag is allowed per portlet.xml and only one portlet.xml is allowed per war file. Therefore, each war file may only deploy a single abstract portlet application.

Archived

– **uid** *required*

  This ID uniquely identifies this abstract application in the portal server. This ID must unique throughout the entire portal environment. For guidelines on ensuring the ID is unique, refer to “UID guidelines” on page 77. This ID will be used if the portlet application is updated. Once the ID is determined, it should not be changed between iterations. Doing so will cause updates to fail. The ID may contain any combination of letters and characters to maximum length of 255.

– **major-version** *optional*

  An optional tag only used by administrators to track releases; it is not used in the portal. It must be a number and only one is allowed. If this attribute is not supplied, the major-version will always be 0. If this attribute is supplied, the minor-version must also be included or the default value of 0 will be assumed.

– **minor-version** *optional*

  An optional tag only used by the administrators to track releases and not used in the portal. Must be a number and only one is allowed. If this attribute is not supplied, the minor version will be 0. If this attribute is supplied, the major-version must also be included or the default value of 0 will be assumed.

portlet-app-name *required*

Only one is allowed. Since only concrete portlet applications are visible in the portal, this name is visible in the portal environment when the full information for the portal application is requested on the Manage Portlet Applications portlet. This name need not be unique.

portlet *required*

One or more of these tags is required. This tag defines the abstract portlets contained in the abstract portlet application. Each portlet tag maps to a single servlet defined in the web.xml. There is a one-to-one relationship between the servlets defined in the web.xml and the abstract portlets defined in the portlet.xml. You may not map two abstract portlets to the same servlet. Therefore, if there are two servlets defined in the web.xml, there will be two abstract portlets defined in the portlet.xml

– **id** *required*

  This ID must be unique within the abstract portlet application only. This ID will be used by the concrete portlets to create a link to the abstract definition. It may be any combination of letters and numbers to a maximum of 64 characters.

Archived

– **href** *required*

This tag creates the link between the abstract portlet and the servlet defined in the web.xml. The link is formatted as in Example 2-3 where `Servlet_1` is the ID defined in the `<servlet id>` tag of the web.xml.

Example 2-3 href syntax

```
href="WEB-INF/web.xml#Servlet_1"
```

– **major-version** *optional*

An optional tag only used by the administrators to track releases. Not used in the portal. Must be a number and only one is allowed. If this attribute is supplied, the minor-version must also be included or the default value of 0 will be assumed.

– **minor-version** *optional*

An optional tag only used by the administrators to track releases. Not used in the portal. Must be a number and only one is allowed. If this attribute is supplied, the major-version must also be included or the default value of 0 will be assumed.

portlet-name *required*

Defines the name of the abstract portlet. This name will only be seen in the show info screen of the Manage Portlet Application portlet, not during normal portlet administration or execution. Must be unique within the abstract portlet application only.

cache *optional*

This tag indicates the type and level of caching this portlet will perform. If this tag is included, it must contain the expires and shared elements. If the cache element is not included, the default values for expires and shared are 0 and no respectively.

expires *required*

Indicates in seconds the when the cached content should be considered expired.

– 0 indicates the content immediately expires and should always be refreshed

– -1 indicates the content does not expire.The content will not be refreshed once the portlet is initialized.

– Any other value measures the cache in seconds.

Archived

shared *required*

Indicates if the content is to shared among all users or if a cache must be maintained for each user. Use the NO option carefully as a large cache will result. Valid values are yes or no.

allows *required*

This tag indicates the portlet states that are supported by this portlet. The normal state is assumed and may not be unsupported. The other valid values are:

- **maximized** *optional*

  When selected, the portlet will take ownership of the portal screen and other portlets will not be able to return content for inclusion in the portal page. Each portlet on the page will state have the opportunity to execute any listeners it implements, such as PortletPageListener. However, the service method, and by extension doView method of the other portlets will not be executed.

- **minimized** *optional*

  When selected, the portlet will be displayed as a title bar only. Any listeners implemented by the portlet will be executed but the portlet's service, and by extension doView method will not.

- **detached, moving, resizing, closed** *optional*

  Though these are valid values according to the DTD, they have no corresponding support in the portal server. As such, including or omitting them will have no effect at this point.

supports *required*

This element indicate which markup languages this portlet can render its content. It is required and at least one markup may be supported.

markup *required*

This tag provides a definition for a single markup that this portlet will support. Each markup that is to be supported is defined in a markup element.

- **name** *required*

  This attribute identifies the name of the markup defined in this element. Valid strings are html, wml, chtml. If custom markups have been defined, they too would be valid.

view *required*

Indicates that at a minimum, this portlet supports View mode. This is required for all markup types.

Archive

– **output** *optional*

This attribute indicates the type of output the portal server should expect from this portlet. Valid values are document and fragment

- *Document:* Not used in V4.1.2
- *Fragment:* All HTML portlets should use this value.

configure *optional*

Indicates this portlet supports the Configure mode. As with the View mode, it may specify as output fragment or document. This tag has no effect on non-html markup types. The developer is required to implement configure support by including a doConfigure method in the portlet. This tag simply instructs the portal server to include the appropriate link on the portlet title bar.

edit *optional*

Indicates this portlet supports the Edit mode. As with the View mode, it may specify as output fragment or document. This tag has no effect on non-html markup types. The developer is required to implement edit support by including a doEdit method in the portlet. This tag simply instructs the portal server to include the appropriate link on the portlet title bar.

help *optional*

Indicates this portlet supports the Help mode. As with the View mode, it may specify as output fragment or document. This tag has no effect on non-html markup types. The developer is required to implement help support by including a doHelp method in the portlet. This tag simply instructs the portal server to include the appropriate link on the portlet title bar.

concrete-portlet-app *required*

This element defines the concrete portlet application to be deployed into the portal server. One or more of these elements are required. This concrete application is based upon the abstract portlet application defined earlier in the portlet.xml. A concrete portlet application is not required to contain all of the portlets defined in the abstract application. However, it may not define more portlets than the abstract application. Each concrete portlet contained in the concrete application maps to one and only one abstract portlet. An abstract portlet may not be mapped twice in the same concrete application.

– **uid** *required*

This uid must be unique throughout the entire portal environment. Refer to 2.6.5, “UID guidelines” on page 77 for more information on ensuring UIDs are unique. This UID will be used by the portal server when the portlet is updated or deleted. If the ID changes between iterations, the original concrete application will not be update. Instead, a new concrete

application will be installed, resulting in multiple concrete applications. Generally, once the ID has been determined, it should not be changed. The ID may contain any combination of letters and characters to a maximum length of 255.

portlet-app-name *required*

This is the application name that will be used in the portal server. When the war file is deployed, each of the concrete applications will be listed. This is the name that will appear in that list. This name need not be unique in the portlet.xml or the portal server. However, deploying more than one concrete portlet application with the same name may cause some administrative confusion. If two or more applications are deployed with the exact same name, only one will be initially active. The other application must be manually activated. In practice, when there is a one-to-one relationship between the abstract and concrete portlet applications, the application names are often the same. This name may contain any combination of letters and numbers to a maximum length of 255 and only one is allowed per concrete application.

context-param *optional*

This element provides an opportunity to set parameters that will be shared by all concrete portlets defined in the concrete portlet application. These parameters are available in code through the PortletApplicationSettings object. There is no limit on the number of context parameters that may be set. Be aware that these parameters may be changed at runtime by the administrator via the Manage Portlet Applications portlet. For a summary on the various parameters set in the deployment descriptors, see 2.6.3, “Parameter summary” on page 76.

param-name *required*

Indicates the name of the parameter. This name will be seen by the administrator if they decide to work with these parameters at runtime. This is also the name used in code to retrieve the parameter value. The name has a maximum length of 255.

param-value *required*

The value intended to be held by the parameter. This value can be changed at runtime by the administrator. The maximum length of the parameter value is 1048576.

concrete-portlet *required*

This element wraps the definition of the concrete portlet being deployed in this concrete application. Any number of concrete portlets may be deployed,

up to the number of abstract portlets defined in the abstract portlet application.

– **href** *required*

This attribute creates a link between the concrete portlet and the abstract portlet. The syntax dictates that the portlet id defined in the abstract application be prefixed with a # symbol as illustrated in Example 2-4.

Example 2-4 Concrete portlet href

```
<concrete-portlet href="#HWPortlet_1">
```

portlet-name *required*

This tag indicates the administrative name of the portlet. This name is not the title of the portlet and will not be seen by the average end user. This name need not be unique in the portlet.xml or the portal server. However, take care to properly name your portlets to prevent confusion. If two or more portlets are deployed in the same portlet.xml with the exact same name, only one will be initially active. The name may be any combination of characters to a maximum length of 255.

default-locale *required*

This element indicates which language is the default language for this concrete portlet alone. This setting will not override the user's preferred locale or locale settings provided by the client browser. If the client's locale cannot be determined, this value is used. Also, if the portlet does not support the locale requested by the user, this default locale is used instead. The value must be a recognized country code including, if appropriate, any variants. This value cannot be changed at runtime by the administrator.

language *required*

At least one language block must be included. Though not required, it is a best practice to ensure that at a minimum, the default locale is implemented in a language block. In practice, a language block should be provided for each language this portlet intends to support. ideally this includes all ten group 1 languages. Only the languages defined in the portlet.xml will be available. Though the strings can be changed, there is no mechanism to add new languages at runtime.

– **locale** *required*

This attribute indicates the locale being defined by this language block. The value must a recognized country code, including any applicable variants.

title *required*

This tag specifies the language specific title of this portlet. This title will be seen in the title bar of the portlet at runtime.This value may be changed at runtime by the administrator. The maximum length of the title is 255 characters.

title-short *optional*

This tag specifies the language specific short title of the portlet. The maximum length of the short title is 128.

description *optional*

This description is used in several places in the portal. For example, in the Edit Layout and Content portlet, the description will display in a hover box over the portlet. The maximum length for the description is 1024 characters.

keywords *optional*

This tag specifies the language specific keywords of the portlet. The maximum length of the keywords is 1024 characters.

config-param *optional*

This element allows parameters to be passed to the concrete portlet. Unlike context and servlet-config parameters, these parameters are not shared between portlets. Any number of portlet-config parameters may be supplied. The values can be changed at runtime by the administrator via the Manage Portlets portlet. These parameters are accessed in code via the PortletSettings object. For a summary on the various parameters set in the deployment descriptors, see “Parameter summary” on page 76.

param-name *required*

Indicates the name of the parameter. This name will be seen by the administrator if they decide to work with these parameters. This is also the name used in code to retrieve the parameter value. The name has a maximum length of 255.

param-value *required*

The value intended to be held by the parameter. This value can be changed at runtime by the administrator. The maximum length of the parameter value is 1048576.

Archived

2.6.3 Parameter summary

There are four types of parameters that can be specified in the deployment descriptors.

Parameter Name	Location	Visibility	Programmatic Access	Runtime Accessibility
Context-Param	web.xml - web app definition	Every portlet deployed in the .war	PortletContext.getInitParameter()	Read-only
Init-Param	web.xml - servlet definition	Each portlet based on the particular servlet	PortletConfig.getInitParameter() Portlet.getInitParameter()	Read-only
Context-Param	portlet.xml concrete app definition	All portlets defined in a single concrete app	PortletApplicationSettings.getAttribute()	Read/Write
Config-Param	portlet-xml concrete portlet definition	Individual Concrete portlets	PortletSettings.getAttribute()	Read/Write

2.6.4 Descriptors relationship (web.xml and portlet.xml)

The relationship between servlets, abstract portlets and concrete portlets is best described in Figure 2-6 on page 77. Note that some required elements have been omitted for clarity.

Archive

```
<web-app id="WebApp">
   <display-name>SimplePortlet</display-name>
   <servlet id="Servlet_1">
      <servlet-name>Portlet</servlet-name>
      <servlet-class>com.yourco.portlets.Portlet</servlet-class>
   </servlet>
   <servlet id="Servlet_2">
      <servlet-name>AnotherPortlet</servlet-name>
      <servlet-class>com.yourco.portlets.AnotherPortlet</servlet-class>
   </servlet>
   <servlet-mapping> ... </servlet-mapping>
   <servlet-mapping> ... </servlet-mapping>
</web-app>

<portlet-app-def>
    <portlet-app uid="DCE:4604:1">
        <portlet-app-name>SimplePortlet application</portlet-app-name>
        <portlet id="Simple_Portlet_1" href="WEB-INF/web.xml#Servlet_1">
            <portlet-name>SimplePortlet portlet</portlet-name>
        </portlet>
        <portlet id="Another_Portlet_1" href="WEB-INF/web.xml#Servlet_2">
            <portlet-name>New Portlet</portlet-name>
    </portlet-app>
    <concrete-portlet-app uid="DCE:4604:1.1">
        <portlet-app-name>Simple Portlet application</portlet-app-name>
        <concrete-portlet href="#Simple_Portlet_1">
            <portlet-name>SimplePortlet portlet</portlet-name>
        </concrete-portlet>
    </concrete-portlet-app>
    <concrete-portlet-app uid="DCE:4604:1.2">
        <portlet-app-name>Second Simple Portlet Application</portlet-app-name>
        <concrete-portlet href="#Another_Portlet_1">
            <portlet-name>Another Simple portlet</portlet-name>
        </concrete-portlet>
    </concrete-portlet-app>
</portlet-app-def>
```

Figure 2-6 web.xml and portlet.xml relationship

2.6.5 UID guidelines

When determining the UID for either concrete or abstract portlet applications there are several steps to follow to ensure the ID is guaranteed to be unique throughout the entire portal environment. It is recommended that your organization develop style guidelines to ensure uniqueness in your environment.

- Include the portlet's namespace in the UID, using the same format that is used for Java packages
- Add some portlet application specific description
- Add some arbitrary characters to guarantee uniqueness within the namespace, for example: com.ibm.wps.samplet.mail.4969
- Add postfixes for the corresponding concrete portlet applications, for example: com.ibm.wps.samplet.mail.4969

2.6.6 Building a war file

All the elements of the portlet need to be deployed in a Web archive (.war) file. This file can be created with any zip creation tool, the jar command line utility or the export utility of the WebSphere Studio Application Developer tool. The WebSphere Studio Application Developer environment will be covered in detail in Chapter 3, “Portal Toolkit” on page 125. Each war file should contain elements listed in Figure 2-7.

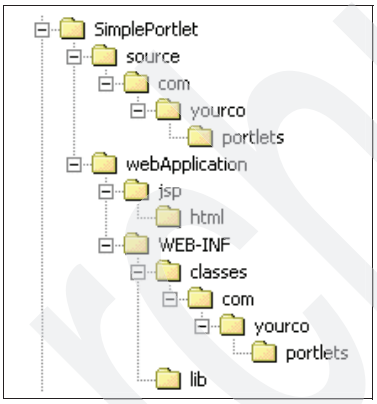


Figure 2-7 Basic WAR contents

The source folder is optional and you may choose what source to include for distribution.

The webApplication folder is optional and may be used to contain the WEB-INF folder. Alternatively, the WEB-INF folder can be placed directly under the root without modification.

Generally, it will also contain a JSP folder to hold all JSPs used throughout the entire portlet application. The JSP folder will organize the contained JSPs in folders representing the markup and languages they are intended to support. For example, if a portlet supported HTML, WML and cHTML and provided limited

Archived

National Language support for HTML requests, the JSP folder would be organized as in Figure 2-8.

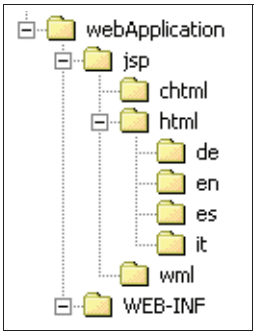


Figure 2-8 WAR structure for a portlet with NLS and multi-device support

The WEB-INF folder must contain at a minimum the two required deployment descriptors. The web.xml and portlet.xml must be placed directly under the WEB-INF folder.

The classes that make up the portlet application must be stored in one of two locations. Those classes that have been packaged into jar files should be stored in the lib directory. Classes that are not packaged in a jar file are stored in the classes folder with the complete package structure. Both approaches are illustrated in Figure 2-9.

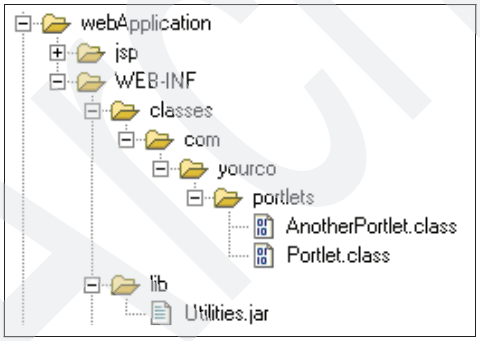


Figure 2-9 Storing classes in the WEB-INF folder

Once the contents have been organized correctly, you can use the jar command line utility to create the war file. There is no compression requirement for the war file so you may choose to compress the file or not without affecting deployment. For a complete discussion regarding the jar utility, refer to:

`http://java.sun.com/docs/books/tutorial/jar/basics/index.html`

Archived

2.7 Portlet life cycle

This section will explain the portlet life cyle and when certain objects become available. For a complete discussion of the portlet object, refer to 2.9.1, “Portlet” on page 83. The basic life cyle of each portlet is displayed in Figure 2-10. Though the login and logout methods are part of SessionListener interface, they are covered here since they are usually included in normal portlet implementations. Other listeners are covered in 2.10, “Listeners” on page 95.

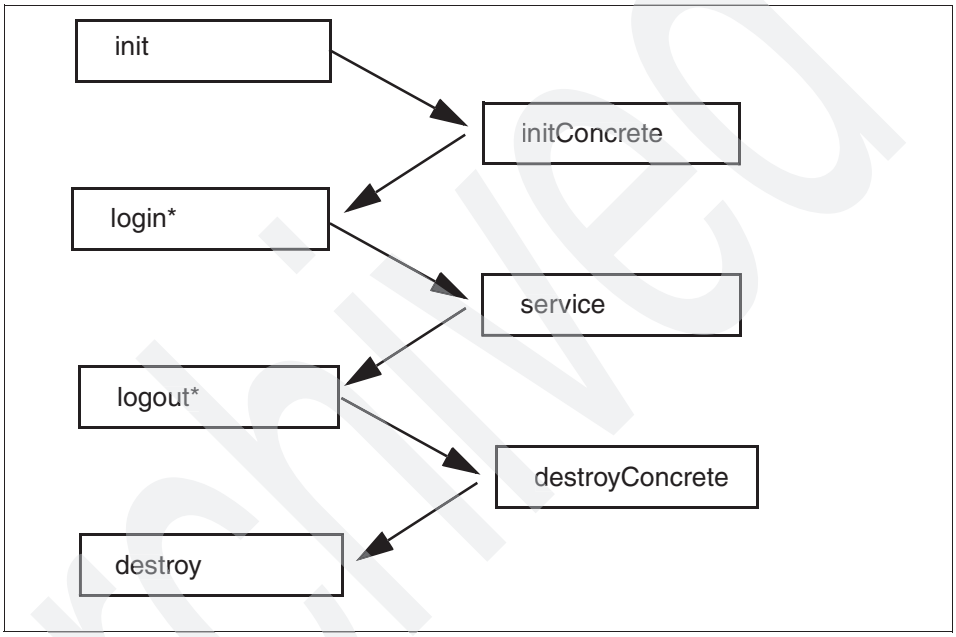


Figure 2-10 Basic portlet lifecyle

init(PortletConfig config)

This method is called by the portlet container on the abstract portlet when the portlet is first loaded. As with servlets, portlets are loaded when they are first requested. Any subsequent calls to the portlet will not execute this method. Generally, initialization that is applicable to *every* concrete portlet based on this abstract portlet is placed in this method. If you choose to override this method, at a minimum it should make a call to its parent via `super.init(portletConfig)`. At this point in the portlet life cycle, no portlet-specific storage objects are available. This includes PortletSession, PortletData, PortletApplicationSettings and PortletSettings.

Archived

initConcrete(PortletSettings settings)

This method is called by the portlet container on the concrete portlet. The initialization code performed in this method is not shared by other concrete portlets even though they may be based upon the same abstract portlet. It is in this method that the PortletSettings object is first available. The PortletSettings encapsulates the concrete portlet configuration parameter information. From the PortletSettings object, the PortletApplicationSettings object is available. The PortletApplicationSettings object encapsulates concrete portlet application context parameters. In this method, no user-specific objects are yet available.

login(PortletRequest request)

If the concrete portlet has been placed on a page that requires authorization, the login method is called by the portlet container to associate a user with the portlet. It is at this point that the PortletData object is first available. The PortletSession is created by the container for the registered user at this point and is available in this method via the request object. If the request for the portlet is made by an anonymous user, this method is not called. If this method is not called, a default session object can still be created with no user association, though it may be of little practical use. This method is actually defined in the PortletSessionListener interface which is implemented by the abstract class Portlet. Since your custom portlets will extend from Portlet, it is included in this discussion even though other oft-used listeners are not.

service(PortletRequest request, PortletResponse response)

This method is called on each and every request of the portlet. After the portlet has been added to a page and initially accessed by a user, this is the only method that will be called by the portlet container on subsequent requests. Generally, this method will delegate the request to the appropriate do method to render content. At this point, all portlets and, if applicable, user-specific objects are available.

logout(PortletSession session)

Only when a user specifically selects the **Log Off** button on the portal is this method called. This method provides you with the opportunity to manage any user-specific information once the user has logged out and to clean up user-related resources. If the user removes the portlet from their page, the logout method is not called until the user actually logs out of the portal, even though they no longer are accessing the portlet. When the portlet is taken out of service by the Portal server or the administrator, this method will not be called. The PortletSettings object is still available in this method, although the PortletRequest is not. This method is actually defined in the PortletSessionListener interface which is implemented by the abstract class Portlet. Since your custom portlets

will extend from Portlet, it is included in this discussion even though other oft-used listeners are not.

destroyConcrete(PortletSettings settings)

This method is called when the concrete portlet is taken out of service either because of the portal server stopping or the application being uninstalled from the portal server. The portlet container will call each running concrete portlet in the application individually when the application is deleted. In this method, the PortletSettings object is passed in as a parameter and cannot be retrieved from the normal getPortletSettings method.

destroy(PortletConfig config)

The portlet container executes this method on the abstract portlet when the portlet is taken out of service. Since it is executed on the abstract portlet and not the concrete portlets, it is executed only once. This method provides an opportunity to execute clean-up code on each and every concrete portlet in the application derived from this abstract portlet.

2.8 Portlet API

Portlets are descendents of HttpServlets and as such inherit much of the basic functionality from that class. However, as illustrated in 2.4, “Servlets versus portlets” on page 59, there are some key differences. This section will introduce many of the key objects in the portlet API. This section is not intended to replace the javadoc and therefore will discuss the primary function of certain objects and some of their key methods. The complete javadoc for the portlet API can be found in the \WebSphere\PortalServer\app\wps.ear\wps.war\doc\Javadoc\WPS directory. For the most up-to-date API information, refer to:

www7b.software.ibm.com/wsdd/zones/portal/

2.8.1 Hierarchy

The abstract class Portlet descends from the HttpServlet interface as illustrated in Figure 2-11 on page 83. Note that the package structure indicates the portlet belongs to the org.apache.jetspeed.portlet package. It is important to understand that the WebSphere Portal API and the Jetspeed API are not the same, or even compatible at this time.

Archived

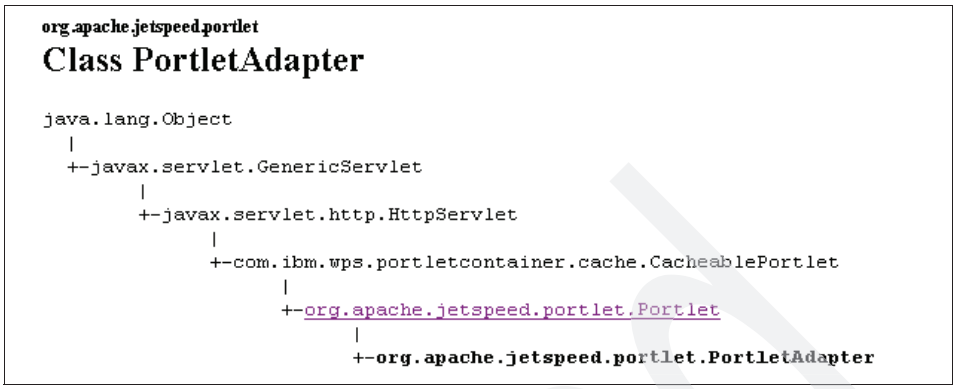


Figure 2-11 Portlet hierarchy

2.9 Core portlet objects

This section will cover many of the objects you will use in day-to-day portlet development.

2.9.1 Portlet

The abstract class Portlet defines the abstract methods that comprise the base functionality of each portlet. All life cycle methods such as init, service and destroy are defined in this class.

For convenience, these abstract methods have been implemented in the PortletAdapter class. The PortletAdapter implements the service method with the basic functionality to determine the type of request and delegate the request to the appropriate do method. As such, it also defines the doView, doConfigure, doHelp and doEdit methods. Most portlet development will extend from the PortletAdapter class.

2.9.2 PortletAdapter

This class is provided as a default implementation of the Portlet class. It is recommended that your portlet classed extend from this abstract class rather than from the Portlet class. The adapter only provides implementations of the portlet-specific methods. It does not provide an implementation for the doXXX methods of the servlet parent (for example, doPost, doGet, etc.). In addition to the methods of the Portlet class, this class defines several additional methods.

Archived

The methods getVariable, setVariable and removeVariable provide access to the variables you can set on the concrete portlet. It is important to remember that these variables are at the concrete level and therefore will not be shared with other concrete portlets even though they may be based upon the same abstract portlet. These variables are available only in code and are not presented in portal administration, nor are they configurable in the portlet.xml deployment descriptor. Example 2-5 illustrates the usage of these methods.

Example 2-5 Setting and Accessing the concrete portlet variable

```
setVariable("var", "Some Value");
String var = (String) getVariable("var");
```

2.9.3 PortletRequest

The PortletRequest interface inherits from the HttpServletRequest and ServletRequest interfaces. It represents the user's request and like ServletRequest, encapsulates information about the user and the client. An implementation of PortletRequest is passed to the service method and subsequently to the delegated do methods (doView, doEdit and so on). In addition to client and user information, the PortletRequest object can be used as a short term bucket for storing information, such as JavaBeans. JSPs then have access to the information stored in the PortletRequest to create dynamic presentations. Some of the more frequently used methods of this object are listed below. Example 2-6 on page 85 illustrates some common usage of the PortletRequest object.

- **getAttribute/setAttribute/removeAttribute**

  These methods allow you to store data in a short term bucket. The PortletRequest is portlet-specific and therefore data stored in this object is not available to other portlets. The storage is only valid during the single request. All objects placed in this scope should be serializable.

- **getParameter**

  This method provides access to the parameters passed as part of the HttpServletRequest. There is no need to distinguish whether the parameter is passed via an HTTP get or post method. This method is often used in event-handling.

- **getCookies**

  This method provides access to the cookies stored by the current domain on the client's machine. An array of cookie objects is returned and the portlet is responsible for iterating through the collection.

- **getHeader**

  This method provides access to the headers supplied by the client. Some of the more common headers you may want to access include accept, accept-encoding and cache-control.

- **getLocale**

  This method returns the preferred locale for the user. The Portal Server determines the locale by first retrieving the user's preferred language set during registration. If the preferred language is not set, the locale is retrieved from the accept-language header supplied by the client.

- **getPreviousMode**

  This method is intended to return the previous mode visited by the user. In WebSphere Portal V4.1.2, this method always returns null. Look for updates to this functionality in future releases.

Example 2-6 Working with the PortletRequest

```
request.setAttribute("uri", uri);
String fNmame  = request.getParameter("f_name");
java.util.Locale locale = request.getLocale();
```

2.9.4 PortletResponse

The PortletResponse interface extends from the HttpServletResponse and ServletResponse interfaces. This object encapsulates the response sent to the Portal Server for aggregation. Unlike the ServletResponse, the response is sent to the Portal Server, not the client machine directly. Therefore, attempting to influence the overall request, such as setting a status code, will have no effect. Some of the most commonly used methods of this object are listed below:

- **getWriter**

  This method returns a java.io.PrintWriter object that can be used to return markup to the Portal Server. The content returned by the PrintWriter is aggregated into the entire portal page. While it is possible to use a PrintWriter as well as include a JSP, it is generally considered bad practice to do so.

- **encodeNamespace**

  This method takes a String and attaches the name of the portlet application as a prefix. For example, the value "`variable_one`" when encoded would be returned as "`PC_175_variable_one`". Any variables that will become part of the aggregated portal page should be encoded. JavaScript functions and variables are good examples of values that should be encoded to prevent name collisions.

Archived

- **addCookie**

  This method allows you to add a cookie to the ultimate HTTP response that is sent by the Portal Server to the client. In order to ensure the name of cookie is unique throughout the portal, it is recommended that you use the encodeNameSpace method.

- **addHeader/setHeader/containsHeader**

  This method provides access to the headers sent back to the client via the portal server.

- **encodeURL**

  This method will append the passed string to the complete URL of the Portal Server. For example, the string "`example.gif`" becomes "`http://www.yourco.com/wps/WPS_PA_351/example.gif`" when passed to the encodeURL method.

- **createURI/createReturnURI**

  These methods will create URI object that contains a URL pointing the portlet in particular mode. For more information see 2.9.17, "PortletURI" on page 94.

Example 2-7 Working with the PortletResponse

```
java.io.PrintWriter out = response.getWriter();
out.println("Hello World");
PortletURI uri = response.createURI();
String functionName = response.encodeNamespace("myFunction");
```

2.9.5 PortletSession object

The PortletSession object extends from HttpSession and serves much the same purpose. The PortletSession is intended to represent an ongoing conversation between the client and the portlet. To this end, the PortletSession can be used to store information needed between requests. The PortletSession is intended to store data between requests, not between portlets. As such, data stored in the session by one portlet is not accessible by another. The PortletSession is retrieved from the request object as illustrated in Example 2-8. Since a PortletSession object is created when a user logs in, there is no need to create one. However, the getPortletSession(boolean) can be used to create a session for an anonymous user.

Example 2-8 Retrieving a PortletSession

```
   PortletSession session = request.getPortletSession();
```

The most important methods of the PortletSession are getAttribute/setAttribute/removeAttribute: these methods allow you to store, retrieve and delete objects in the PortletSession. Objects stored in the PortletSession must be serializable.

2.9.6 Client

The Client interface represents the device making the request, not the user. The Client object can be retrieved from the PortletRequest object as illustrated in Example 2-9. Figure 2-12 illustrates the result of most of the methods of the client object when requested via Internet Explorer and a Nokia WAP emulator.

Example 2-9 Working with the client object

```
Client client  = request.getClient();
out.print("<P>Manufacturer: " + client.getManufacturer() + "<br/>");
out.print("MarkupName:" + client.getMarkupName() + "<br/>");
out.print("MimeType   " + client.getMimeType() + "<br/>");
out.print("Model:     " + client.getModel() + "<br/>");
out.print("UserAgent: " + client.getUserAgent() + "<br/>");
out.print("Version:   " + client.getVersion() + "</P>");
```

Generally, the client object is used to determine the markup language to which the device is mapped. Based on that information, device-specific markup can be generated.

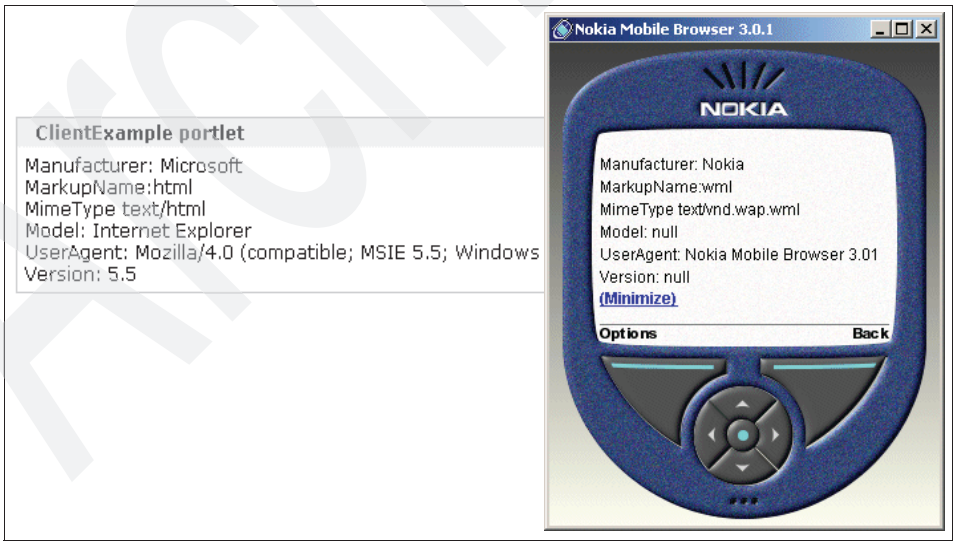


Figure 2-12 Client Information displayed on various clients

2.9.7 PortletConfig object

The PortletConfig object represents the abstract portlet. Therefore, any information contained in the PortletConfig is shared by all concrete portlets deployed based on the same abstract portlet. This object can be used to access the initialization parameters set in the web.xml deployment descriptor's servlet definition. Unlike other parameters, these are read-only and cannot be altered dynamically. This object can also be used to determine which modes and states are supported. Furthermore, this object provides access to the PortletContext object. The PortletConfig is retrieved via the getPortletConfig method of the PortletAdapter class or the getConfig method of the AbstractPortlet class. There are some useful methods available in this object. They are listed below and illustrated in Example 2-10.

- **supports**

  This method can accept a PortletWindow.State object or a Portlet.Mode object and return a boolean indicating whether or not the state or mode is supported by the portlet.

- **getContext**

  This method will return a PortletContext object. For more information on the PortletContexr, refer to 2.9.8, "PortletContext object" on page 88.

Example 2-10 Working with PortletConfig

```
boolean maxSup = getPortletConfig().supports(PortletWindow.State.MAXIMIZED);
boolean minSup = getPortletConfig().supports(PortletWindow.State.MINIMIZED);
boolean viewSup = getPortletConfig().supports(Portlet.Mode.VIEW,
                                              request.getClient());
boolean editSup = getPortletConfig().supports(Portlet.Mode.EDIT,
                                              request.getClient());
boolean configureSup = getPortletConfig().supports(Portlet.Mode.CONFIGURE,
                                              request.getClient());
boolean helpSup = getPortletConfig().supports(Portlet.Mode.HELP,
                                              request.getClient());
PortletContext context = getPortletConfig().getContext();
```

2.9.8 PortletContext object

The PortletContext provides a mechanism for the portlet to access the services of the portlet container in which it is running. For example, the Context provides access to the PortletLog, servlet context parameters as well as any services hosted by the portal such as Credentials Vault, PersistentConnection and possibly other custom services. The parameters accessed by the PortletContext are the context parameters set in the web.xml. These parameters are common to

all portlets deployed in the same web.xml, regardless of their organization into various portlet applications. The PortletContext object is retrieved from the PortletConfig object as illustrated in Example 2-11.

Example 2-11 Accessing Context Parameters via the PortletContext

```
PortletContext context = getPortletConfig().getContext();
String webmaster = context.getInitParameter("webmaster");
```

The PortletContext can also be used to store attributes that will be shared by all portlets deployed via the same web.xml regardless of concrete portlet application. These attributes are not distributed in a clustered environment.

- **include**

  This is the most commonly used method of the PortletContext object. In a well-designed MVC architecture, the portlet executes one or more business objects to satisfy the logic of the request. Once the logic has completed, the include method generally calls a JSP to produce the output. Unlike Servlets, there is no ability to forward to a JSP. Example 2-12 illustrates this approach.

- **getContainerInfo**

  This method indicates the Portal Server version the portlet is executing. It only indicates the major version, not the minor one. In WebSphere Portal Server V4.1.2, this method returns the String '`IBM WebSphere Portal Server/4.1`'.

- **getText**

  This method provides access to Resource Bundles to use in providing National Language Support (NLS). For more information on NLS, see Chapter 8., "National Language Support (NLS)" on page 249.

Example 2-12 Including a JSP

```
public void doView(PortletRequest request, PortletResponse response)
                        throws PortletException, IOException {
    //Business logic completed
    getPortletConfig().getContext().include("/jsp/View.jsp",
                                            request, response);
}
```

2.9.9 PortletSettings object

This object is best thought of as wrapping the information defined in the `<concrete-portlet>` section of the portlet.xml deployment descriptor. The

PortletSettings object encapsulates the configuration information of the concrete portlet instance. The parameter information is retrieved from the portlet.xml but can be modified at runtime while the portlet is in Configure mode. Therefore, the PortletSettings object can be used as a storage for attributes to be shared by all the concrete portlet instances. When attributes are adjusted or added, be sure to call the store method to persist the new values. The administrator can add new parameters and alter existing parameter values via the Manage Portlets portlet in Administration place. The PortletSettings object also provides access to configuration information such as the title of the concrete portlet and the default locale. This object can be retrieved from the PortletRequest object or is passed as a parameter to the initConcrete and destroyConcrete methods of the portlet. The main methods are:

- **getAttribute/setAttribute/removeAttribute**: these methods provide access to attributes.
- **getTitle**: this returns a string indicating the title of the portlet for the current client and the specified locale. Note that this method returns the active title, not necessarily the title specified in the deployment descriptor. If the administrator has changed the title at runtime for example, that value is returned.
- **getDefaultLocale**: this method returns a Locale object specifying the default locale as determined by the portlet.xml.
- **getPortletApplicationSettings**: this method will return the PortletApplicationSettings object discussed in 2.9.10, “PortletApplicationSettings object” on page 90.

Example 2-13 Working with PortletSettings

```
String title = request.getPortletSettings().getTitle(
                        request.getLocale(),
                        request.getClient()));
java.util.Locale locale = request.getPortletSettings().getDefaultLocale());
PortletApplicationSettings portletAppSettings =
             request.getPortletSettings().getApplicationSettings();
String attribute = request.getSettings().getAttribute(“attName”);

//Only available in doConfigure:
request.getSettings().setAttribute("attribute", "Some Value");
request.getSettings().store();
```

2.9.10 PortletApplicationSettings object

This object is best thought of as wrapping the information defined in the <concrete-portlet-app> section of the portlet.xml deployment descriptor. It is used

to encapsulate the information pertaining to all concrete portlets\ deployed as part of the same concrete portlet application. The context parameters defined in the concrete portlet application section of the portlet.xml are available through this object's getAttribute method. These parameters can be adjusted and new ones added only while a portlet is in configure mode.

- **getAttribute/setAttribute/removeAttribute**: these methods provide access to attributes of the concrete portlet application.

Example 2-14 Working with PortletApplicationSettings

```
PortletApplicationSettings portletAppSettings =
            request.getPortletSettings().getApplicationSettings();
String attribute = portletAppSettings.getAttribute("attribute"):

//Only available in doConfigure:
portletAppSettings.setAttribute("attribute", "Some Value");
portletAppSettings.store();
```

2.9.11 PortletData object

The PortletData object represents a ConcretePortlet instance on a users page. It provides a quick, secure and effective method of attribute persistence with no JDBC code required. The PortletData is not dependent on the life cycle of the portlet. The PortletData is user-specific. However, when a user first accesses a portlet utilizing the PortletData object, the PortletData is not unique. In fact, until the user sets some value in the PortletData, they continue to use a shared Data. This PortletData is shared with the administrative user who first place the portlet on the page. All values stored in the PortletData must be serializable. Since a null object is not serializable, be sure to test the validity of your object prior to setting them into the PortletData object.

For example, the HelloWorld portlet uses PortletData to persist the greeting String and the moniker the user wishes to be addressed by. The Administrator installs this portlet, grants edit permissions to the All Authenticated Users group and places it on the Welcome page. The Administrator chooses to edit the portlet and enters "`hello there`" as the greeting String and "`admin`" as the moniker. When user JohnSmith logs into the portal page and opens the welcome page, he sees the name admin and the greeting "`hello there`". The administrator decides to change the greeting to "`Greetings`". Since JohnSmith has not edited the PortletData, he continues to share the PortletData and sees the changes the admin has made. JohnSmith chooses to edit the PortletData to use his name instead of admin. Once he edits the PortletData, he has his own PortletData object. Changes he makes will be seen by no one else. Furthermore, he will no longer see any changes to the PortletData made by the administrator.

Archived

Example 2-15 Working with PortletData

```
PortletData data = request.getData();
String greeting = (String) data.getAttribute("greeting");
String moniker = (String ) data.getAttribute("moniker");

//Only available in doEdit or possibly actionPerformed:
PortletData data = request.getData();
data.setAttribute("greeting", greeting);
data.setAttribute("moniker", moniker);
```

2.9.12 PortletLog object

This allows you to quickly write error messages or other information to the log files. All messages are written to the same file location regardless of the level currently enabled. The log file is named wps_<*time-stamp*>.log where the <*time-stamp*> is formatted as YYYY.MM.DD-HH.MM.SS. For example: wps_2002.10.14-12.32.41.log. The time stamp reflects the time the log file was created, typically when the server was first started. The log file is stored in <WPS-ROOT>\log. To change the location of the directory, uncomment the baseGroup.FileHandler.fileName attribute in jLog.properties and enter the new location. If the directory does not exist, it will be created for you.

There are four levels of severity when writing to the log: info, debug, warn and error. By default, error and warn are enabled. Debug and info levels are enabled for your portlets by enabling the PortletTraceLogger in the EnableTracing portlet in the Portal Administration. Since there is an associated expense with logging, the API provides a mechanism to determine if a logging level is currently enabled prior to writing the message. Example 2-16 illustrates this approach. Finally, if you pass an exception to a particular write method such as error or debug, the portlet container will print out the stack trace to the log file.

Example 2-16 Simple Logging

```
PortletLog log = getPortletConfig().getContext().getLog();
if (log.isDebugEnabled())log.debug("debug enabled:" + someMsg);
if (log.isWarnEnabled()) log.warn("warn enabled:" + someMsg);
if (log.isInfoEnabled()) log.info("info enabled:" + someMsg);
if (log.isErrorEnabled())log.error("error enabled:" + someMsg);
```

If the portlet you are writing extends PortletAdapter, a convenience method has been provided for you as illustrated in Example 2-17 on page 93.

Example 2-17 Accessing the PortletLog in PortletAdapter

```
PortletLog log = getPortletLog();
```

2.9.13 PortletException

The Portlet Exception inherits from the ServletException and is used as the basis for most exceptions thrown in the Portal environment, including UnavailableException

2.9.14 UnavailableException

This exception is thrown if the portlet fails to initialize. Generally, your portlets will include an init method which calls the super.init. Since this call may produce an UnavailableException, the functionality is provided to evaluate what to do if the initialization fails.

- **getUnavailableSeconds**: this method returns an int (integer) indicating how long this portlet is unavailable for.
- **isPermament**: this method returns a boolean indicating this portlet is no permanently unavailable.

The length of time the portlet is unavailable is determined when the exception is first created.

- **UnavailableException(String msg)**: this constructor indicates the portlet is permanently unavailable.
- **UnavailableException(String msg, int time)**: this constructor will reflect the length of time for which this portlet is unavailable.

2.9.15 PortletWindow object

This object represents the window surrounding the portlet only. Generally, this class is useful when determining the real state a portlet has to work with. Example 2-18 on page 94 illustrates this approach. Minimized, Normal and Maximized are defined as constants in the PortletWindow.State class.

Archived

Example 2-18 Determining portlet window state

```
PortletWindow.State state = request.getWindow().getWindowState();
if (state.equals(PortletWindow.State.NORMAL)){
   getPortletConfig().getContext().include("/jsp/View.jsp", req, resp);
} else if (state.equals(PortletWindow.State.MAXIMIZED)){
   getPortletConfig().getContext().include("/jsp/MaxView.jsp", req, resp);
} else {
   //Window is minimized, no need to generate content.
}
```

2.9.16 User object

The User object represents the authenticated user and is retrieved from the PortletRequest object. The API provides predicable getters and setters for the most common attributes of the user such as GivenName, FamilyName and UserID. This class provides access to both Basic and Extended attributes of the user. Basic attributes are those stored in the LDAP directory as part of the schema used throughout the portal. Extended attributes are those attributes stored in the Portal Server database. Example 2-19 illustrates accessing both basic and extended attributes.

Example 2-19 Working with User attributes

```
User user = request.getUser();
String familyName = user.getFamilyName();
String favoriteColor = user.getAttribute(“favColor”);
String phoneNumber = user.getAttribute(“phoneNumber”);
```

The getID returns as a String the complete DN of the user. For example, wpsadmin in a typical SecureWay® environment would return `uid=wpsadmin, cn=users,dc=<domain>,dc>=<com> ‘‘`

There are two User interfaces defined in the Portlet API. The org.apache.jetspeed.portlet.User class represents the logged in user and is the User object you will use day-to-day. The com.ibm.wps.puma.beans.User interface is an EJB and is not used to access individual user information

2.9.17 PortletURI

The PortletURI is used in organizing navigation through the portal as a user moves from mode to mode in a portlet. When a user is on a normal page (for example when the portlets are presented in View mode), the page is an aggregation of all the portlets. In order for any one portlet to be able to navigate

back to that aggregated state, the PortletURI can store the URL. The PortletURI is then placed in a bucket such as the request or session object. For more information on the PortletURI object, see 2.12.3, “PortletURI” on page 100.

2.10 Listeners

The event model of the Portal API is very similar to the traditional Java event model. However, there are two main points of distinction. First, there is no need to register listeners. When a portlet in installed, the Portal Server determines the listeners it implements and registers them on behalf of the portlet. Secondly, since the registration is taken care of by the Portal Server, it is not possible to specify which portlets a particular portlet wishes to register for. Therefore, portlets implementing listeners need to carefully plan for unsollicited and unexpected events.

There are several listeners defined in the Portal API. The ActionListener is covered in the Event handling section and the MessageListener is covered in the Messaging section.

2.10.1 PortletTitleListener

This listener allows you to dynamically set the title of the portlet. This listener requires the single method as shown in Example 2-20. This interface is particularly useful when tailoring the title to certain modes or devices. To return a title, simply use a PrintWriter object or include a JSP using the PortletContext object. While the second approach allows you to create a more dynamic title including images and so forth, you must remain mindful of the limited space in the title bar.

Example 2-20 PortletTitleListener example

```
public void doTitle(PortletRequest request, PortletResponse response)
                    throws PortletException, IOException {
   PrintWriter out = response.getWriter();
   String title = getPortletSettings().getTitle(
                    request.getLocale(), request.getClient());
   out.print(title + "(" + request.getMode() + ")");
}
```

2.10.2 PortletPageListener

This interface provides the opportunity to add content to the top and bottom of the aggregated page. Example 2-21 on page 96 illustrates a simple

implementation of the PortletPageListener interface. It is important to note that content returned from the beginPage method is not placed at the top of the page but rather at the top of the aggregated content as displayed in Figure 2-13.

Example 2-21 PortletPageListener implementation

```
public class AgendaPortlet extends PortletAdapter implements
PortletPageListener {
........
   public void beginPage(PortletRequest request, PortletResponse response)
               throws PortletException, IOException {
      PrintWriter out = response.getWriter();
      out.println("This page contains my agenda.");
   }

   public void endPage(PortletRequest request, PortletResponse response)
               throws PortletException, IOException {
      PrintWriter out = response.getWriter();
      out.println("End of my agenda.");
   }
}
```

The resulting page including the top and bottom messages is illustrated in Figure 2-13.

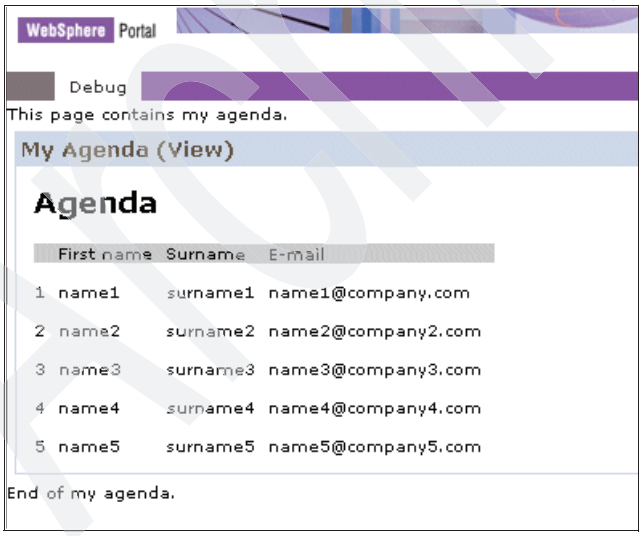


Figure 2-13 beginPage and endPage placements

The beginPage is a convenient method when you need to include Javascript functions needed by your portlet. However. be very conscious of any content you

decide to display in the beginPage method as it may adversely affect the overall aggregation of the page. Furthermore, because the page is aggregated, be sure that any functions or global variables you declare have properly encoded the namespace of the portlet to ensure there are no naming collisions. Use the response.encodeNamespace to do this.

Restriction: The Home.jsp can choose to cancel calls to the PortletPageListener via the <wps:pageRender includeBeginPage="no" includeEndPage="no"> tag. In this case, your beginPage and endPage methods will not be called.

2.10.3 PortletSessionListener

This interface requests the Portal Server to notify the portlet if an authenticated user has accessed the portlet. This interface is already implemented by the PortletAdapter class which is traditionally the parent of most custom portlets. This interface defines the two methods shown in Example 2-22. Figure 2-10 on page 80 illustrates where in the life cycle of the portlet these methods are called. The functionality of the login and logout methods is detailed in 2.7, “Portlet life cycle” on page 80.

Example 2-22 PortletSessionListener methods

```
public void login(PortletRequest request) throws PortletException{ ... }
public void logout(PortletSession session) throws PortletException{ ... }
```

2.10.4 WindowListener

Note: The WindowEvent interface is deprecated; you should use the PortletWindow.getWindowState() method instead. It is included here for information and as a reference for portlets developed using previous releases.

This interface will notify the portlet that the user has changed the window state. Presently, there are only three supported window states, despite the javadoc. NORMAL, MAXIMIZED and MINIMIZED states are supported. The portlet is notified of these three states through windowMaximized, windowMinimized, and windowRestored, respectively. Though only three states are currently supported, the WindowListener defines methods for windowClosing, windowOpening, windowDetached and windowClosed. This methods are never called. However, in order to implement this interface, all methods must be implemented even though several will contain empty bodies.

Note: You will need to make sure you implement the interface org.apache.jetspeed.portlet.event.WindowListener and not the AWT counterpart since some development environments will offer both.

Example 2-23 Implementing the WindowListener

```
public void windowMaximized(WindowEvent arg0) throws PortletException {
   // Some action can be performed
}
public void windowMinimized(WindowEvent arg0) throws PortletException {
   // Some action can be performed
}
public void windowRestored(WindowEvent arg0) throws PortletException {
   // Some action can be performed
}
public void windowClosing(WindowEvent arg0) throws PortletException { }
public void windowClosed(WindowEvent arg0) throws PortletException { }
public void windowDetached(WindowEvent arg0) throws PortletException { }
```

2.10.5 PortletSettingsAttributeListener

The PortletSettings object encapsulates the concrete portlet defined in the portlet.xml. Part of that definition includes configuration parameters that may be declared at deployment time. These parameters can be altered and new ones can be added at runtime. The PortletSettingsAttributeListener notifies your portlet if the configuration parameters are changed at runtime.

2.10.6 PortletApplicationSettingsAttributesListener

Similar to the PortletSettingsAttributeListener, this listener provides notification when the context parameters of the concrete application have changed, been added or removed.

2.11 Action event handling

The event model in WebSphere Portal is very similar to the traditional Java event model. When a portlet wishes to be notified that a user has performed an action, it simply implements the ActionListener correctly and the portal server will take care of calling the appropriate method when the event is generated. Unlike in the traditional Java event model, only the portlet generating the event may listen for that event. That is, there will always only be a single listener for any particular ActionEvent. In order to notify other portlets of an event, the listening portlet may

choose to send messages. For more information on sending messages, see 2.13, “Portlet messaging” on page 102.

When the Portal server services a request, it acts in two distinct phases. The first phase is the event processing phase. All events, including WindowEvents, ActionEvents and MessageEvents are generated, delivered and processed in this phase. Once this phase is complete, the content generation phase begins. Once content generation has begun, no events can be generated. Attempting to generate events during the content generation phase, for example doView, doEdit, etc., will cause an exception.

2.12 Core event objects

This section will cover the objects you will need to work with when managing event handling in action events.

2.12.1 ActionListener

The org.apache.jetspeed.portlet.event.ActionListener interface defines a single method to be implemented as illustrated in Example 2-24.

Example 2-24 ActionListener Interface

```
org.apache.jetspeed.portlet.event.ActionListener
public void actionPerformed(org.apache.jetspeed.portlet.event.ActionEvent
         event) throws PortletException;
```

2.12.2 ActionEvent

An implementation of the org.apache.jetspeed.portlet.event.ActionEvent interface is passed to the actionPerformed method by the PortalServer when a PortletURI with an action is executed. The ActionEvent object provides access to the PortletRequest and the action.

Note: The DefaultPortletAction class and the PortletAction interfaces are deprecated in this release and you should use the Simple Action string instead, as illustrated in Example 2-25.

Example 2-25 Working with the ActionEvent

```
public void actionPerformed(ActionEvent event) throws PortletException {
   PortletRequest request = event.getRequest();
   String action = event.getActionString();
```

Archived

2.12.3 PortletURI

The portletURI represents a URL that can be used to navigate between modes. The PortletURI can be used to navigate to a previous mode, such as from Edit to View, or to navigate back to the same mode, such as a multi-part form in View or Edit. There is no ability to create a PortletURI object pointing to a mode not yet visited by the user.

PortletRequest.createURI returns a portletURI object pointing to the portlet in its current mode. For example, if the portletURI is created in the doView mode, the URL points to the portlet in View. The createReturnURI method returns a PortletURI object pointing to the last mode the portlet was in. This mode is commonly used in the doEdit method when the URI needs to point back to the View mode. The edit.jsp would use the PortletURI to bring the user back to the View mode when they have completed the edit or configure process.

In order for a portlet to be notified of an event, such as the user clicking a button, the portletURI must contain an associated PortletAction. Typical PortletURI construction and usage is shown in Example 2-26.

In this release of the Portlet API, the process of adding actions to PortletURI objects has been simplified. The addAction(PortletAction) method has been deprecated and replaced with addAction(String). Since the vast majority of work with PortletActions involves no more than setting a name, this new implementation is much more convenient.

Developers are advised to use simple action string instead. For details, see Chapter 5, “Action event handling” on page 181.

Note: Deprecated classes and interfaces are still supported in the current release but are not recommended for use because they might not be supported in future releases.

Since the DefaultPortletAction class and the PortletAction interfaces are deprecated in this release, we show the use of the Simple Action string instead, as illustrated in Example 2-26.

Example 2-26 Working with PortletURI

```
PortletURI uri = response.createReturnURI();
uri.addAction("save");
request.setAttribute("uri", uri.toString());
```

It is possible to add parameters to the PortletURI object. Parameters added to the PortletURI via code or through a form are accessed the same way via the

portlet request object. This provides a mechanism to pass default values or to pass parameters not displayed in the form. Example 2-27 displays the code for adding a parameter. Be aware that parameters set via the PortletURI are not passed in the traditional HTML syntax.

Note: The DefaultPortletAction class and the PortletAction interfaces are deprecated in this release and you should use the Simple Action string instead, as illustrated in Example 2-27.

Example 2-27 Add URI

```
public void doView(PortletRequest request, PortletResponse response) throws
                        PortletException, IOException {
   PortletURI viewURI = response.createReturnURI();
   viewURI.addAction("save");
   viewURI.addParameter("Param1", "Param1Value");
   request.setAttribute("viewURI", viewURI.toString());
   getPortletConfig().getContext().include("/jsp/View.jsp", request,
                        response);
   }
```

2.12.4 ModeModifier

When a PortletURI is created, it points to a portlet in particular mode. When that PortletURI is executed and it contains a PortletAction, it will notify the appropriate listener. If, in the actionPerformed method, you need to redirect the user to a mode other the one specified, the request.setModeModifier method can be used to redirect the user to another mode. The ModeModifier can only be set during event processing. Calling this method in doView or doEdit, etc., will have no effect. There are three possible modes the user can be redirected to:

- **REQUESTED** This ModeModifier will navigate the user to whatever mode was originally set by the PortletURI. Essentially, this is the default. If the ModeModifier is changed, it cannot be changed back to REQUESTED.
- **CURRENT** This ModeModifier will keep the user in the current mode. For example, if the user tries to save some information and the actionPerformed determines it is incorrect, setting ModeModifier to CURRENT will return them to the Edit screen.
- **PREVIOUS** This ModeModifier will return the user to the mode the user was in prior to the CURRENT regardless of previous ModeModification. Therefore, setting ModeModifier to CURRENT in one event process will not make that mode PREVIOUS in the next event process.

Archived

2.13 Portlet messaging

One of the most significant advantages of the Portlet architecture is the portlets' ability to communicate with each other to create dynamic, interactive applications. Portlets can use messages to share information, notify each other of a user's actions or simply help better manage screen real estate.

Messages can be sent to all portlets on page, a specific named portlet or to all portlets in a single portlet application. To send a message to all portlets on a page, you must send an instance of the DefaultPortletMessage.

In order to make full use of the potential, you need to adequately architect the entire portlet application anticipating inter-portlet communication. Attempting to implement effective and meaningful message after significant portlet development will cause some difficulty and may require the entire application to be overhauled. This is true for several reasons. For example, access to certain storage objects, such as PortletData, is limited to certain modes. Therefore, if the initial design of an application makes significant use of the PortletData object, implementing messaging later to share configuration information would require a considerable effort. Furthermore, in order to reduce or eliminate code, action event and message event functionality can be combined into a common method. However, to achieve this, it is necessary to consider the information passed via the action or message objects.

To help you understand where messaging may fit into your applications, it is important to become familiar with some of the common uses of portlet messaging. This section will present two examples demonstrating common usage of portlet messaging. The first example illustrates how one portlet can use messaging to control the navigation of another portlet. The second example will demonstrate how a portlet can notify other portlets when a user has altered their configuration information via the Edit mode.

First, however, you must become familiar with the core objects used in the messaging architecture.

2.13.1 MessageListener

The MessageListener interface must be implemented by the portlets you want the portal server to send messages to. The interface defines the single method listed in Example 2-28 on page 103. Since the portlet may be notified by more than one other portlet and therefore may receive different types of messages, it should validate the type of message received prior to working with the object. This is illustrated in Example 2-28 on page 103.

Archive

Example 2-28 Implementing the MessageListener interface

```
public void messageReceived(MessageEvent event) throws PortletException {
   if (event.getMessage() instanceof DefaultPortletMessage) {
      DefaultPortletMessage msg = (DefaultPortletMessage) event.getMessage();
      String message = msg.getMessage();
      //Do something based on the message
   }
}
```

Be aware that when a portlet receives a message, it is not in Edit or Configure mode and therefore faces certain restrictions. For instance, portlets do not have write access to the PortletData object when they are not in Edit mode. Also, they cannot adjust the attributes stored in the PortletSettings object unless they are in Configure mode. Attempts to store attributes in these objects when not in the appropriate mode result in an AccessDeniedException.

Therefore, when attempting to share configuration or settings information between portlets, you need to choose your scope carefully or decide to persist to an outside resource.

2.13.2 MessageEvent

This object is sent to registered MessageListeners by the portlet container when a portlet executes the send method of the PortletContext object. There are two important methods available in this object

- **getMessage**: returns the message object sent with this event. Since this method returns a PortletMessage, the result must be casted to the appropriate type as illustrated in Example 2-28.
- **getRequest**: this method returns the current PortletRequest. The request can be used to access the PortletSession object or to store data to be used in the doView method.

2.13.3 DefaultPortletMessage

This object implements the PortletMessage interface and provides the basic functionality needed for sending string messages between portlets. If you broadcast a DefaultPortletMessage to null, it will be sent to all portlets on the page implementing the MessageListener interface. Example 2-29 on page 104 illustrates sending a simple broadcast message to all portlets on the same page regardless of application affiliation.

Archived

Example 2-29 Broadcasting a message to all portlets on a page

```
PortletMessage msg = new DefaultPortletMessage(“Some Message”);
getPortletConfig().getContext().send(null, msg);
```

2.13.4 PortletMessage

This interface defines the message object that will be sent between portlets. Since it is a flag interface, it does not define any methods to be implemented. Therefore, you are free to create message objects that can store a wide variety of information. Example 2-30 illustrates a simple custom message used to carry an employee object.

Example 2-30 Creating a custom message

```
import org.apache.jetspeed.portlet.*;
import java.net.*;

public class EmployeeMessage implements PortletMessage {
   private Employee emp;
   public Employee getEmployee() { return emp; }
   public void setEmployee(Employee employee) { this.emp = employee;}
}
```

If you simply need to send a string message between portlets, the DefaultPortletMessage provides this basic functionality. It is not possible to send a broadcast message using custom messages. Sending a custom message to null will only send the message to portlets implementing the MessageListener interface on the same page and deployed as part of the same portlet application. This is illustrated in Example 2-31.

Example 2-31 Sending a custom message

```
public void actionPerformed(ActionEvent event) throws PortletException {
      Employee employee = new Employee();
      //Create an employee object with parameters from a form
      EmployeeMessage msg= new EmployeeMessage();
      msg.setEmployee(employee);
      getPortletConfig().getContext().send(null, msg);
}
```

Archived

2.14 PropertyListener interface

This interface is implemented for cooperative portlets using the programmatic approach. This interface may optionally be implemented by portlets. It is an alternate mechanism by which interested portlets may be notified of changed properties.

Other options are to be notified through portlet actions (the actionPerformed method of the ActionListener interface), or Struts actions. The PropertyListener interface may be implemented by portlets that only wish to update their current state based on property changes, rather than execute an action immediately. For more information and a sample scenario of a portlet using this interface, see Chapter 13, “Advanced cooperative portlets” on page 413.

Note: The PropertyListener interface requires that you implement the setProperties method to be notified of property changes.

setProperties

Invoked by the Property Broker to deliver new property values which were changed in the current event cycle of the current request. The Property Broker may be notified of such changes when a portlet invokes the changedProperties in the PropertyBrokerService interface (explicit notification), or when a portlet action which has declared output parameters is invoked (implicit notification).

This method is only invoked during the event phase. Since multiple explicit or implicit property change notifications may be made during an event cycle, one or more setProperties calls may be invoked on a single portlet instance during a single event cycle. The runtime may batch property values from multiple changedProperties calls in a single setProperties call. All properties are guaranteed to be delivered before the first endEventPhase call is delivered, which marks the start of the render phase.

Source cooperative portlets report property changes may be made by using the changedProperties method.

changedProperties method

This method publishes changes in properties and may be used during the portlet's event phase only. This includes the beginEventPhase method of the EventPhaseListener interface, the actionPerformed method of the ActionListener interface, and the setProperties method of the PropertyListener interface.

All properties must have been registered earlier, implicitly or explicitly. A simpler alternative to explicitly invoking this method is often applicable. For example, declare output parameters for registered actions (either programmatically or via an WSDL declaration). In this case, the action may bind the values of the output

parameters on invocation, and at runtime the values will be transferred as if the changedProperties method had been explicitly invoked.

2.15 EventPhaseListener interface

This interface is mainly for programmatic cooperative portlets. It allows developers to get control of the portlet before the portlet receives a notification of a changed property. For more information and a sample scenario of a portlet using this interface, see Chapter 13, “Advanced cooperative portlets” on page 413.

This interface provides the following methods:

- **beginEventPhase()**: at any point during the event phase, a portlet may explicitly publish the value of an output property to the property broker by invoking the changedProperties() method. This is an alternative to the declaration of output parameters for actions and binding the output parameter values to the request or session when the action is invoked. This may happen in the callback method associated with the start of the event phase (beginEventPhase()), in the invocation of the setProperties() method, or in the portlet action method invocation.

  Such publish calls are dealt with by the property broker in the same manner as output parameters of actions: wires associated with output properties are examined and the property values propagated using the information in the target end of the wire. To register properties, you will use the beginEventPhase method of EventPhaseListener, because only during the event phase is it possible to register and unregister properties.

- **endEventPhase()**: the property broker guarantees the completion of all property value notifications to target portlets by the end of the event phase, whether through portlet actions or through the special setProperties() method. The end of the event phase is indicated by the invocation of the endEventPhase() method.

  During the render phase of each request cycle, source portlets can write visual controls representing source data to their output stream. The end user interacts with the visual control in the response to trigger one or more actions on other portlets on the page. During the event phase of the subsequent request, the action is invoked on the corresponding target portlet or portlets.

Redbooks

2.16 Attribute storage summary

There are many objects in the portal environment for storing attributes. In order to help you choose the right object for the right situation, refer to the following chart.

Object	Scope	Attribute Type	Programmatic Access	Best Practice
PortletRequest	Limited to request between the portal server and the portlet	object	getAttribute() setAttribute() removeAttribute()	Use a short term bucket for communication between portlet and JSP (ex: Portlet URI)
PortletSession	Limited to subsequent requests by the same user on the same concrete portlet instance	object	getAttribute() setAttribute() removeAttribute()	Use as an open line of communication between requests. (for example Shopping cart)
PortletSettings	Shared by all instances of the concrete portlet. Editable only in configure mode.	String	getAttribute() setAttribute() removeAttribute()	Use only for configuration information that is applicable to all instances (for example user ID)
PortletApplication Settings	Shared by all concrete portlet instances deployed in the same concrete application.Editable only in configure mode.	String	getAttribute() setAttribute() removeAttribute()	Use only for configuration information that is applicable to all concrete portlet instances in the same application (for example server name)
PortletData	Persistently available to a single concrete portlet instance.	object (serializa ble)	getAttribute() setAttribute() removeAttribute()	Use for information that needs life beyond a session (for example portlet preferences)
PortletURI	One request through to the actionPerformed method	String	addParameter()	Use to provide default parameter values in case the user does not enter a value in a form

Archive

Object	Scope	Attribute Type	Programmatic Access	Best Practice
PortletMessage	Only available to registered message listeners in the event processing phase	Object	Since each custom portlet message can be implemented uniquely, access is not pre-defined	Use to adequately capture all the information necessary to complete the message. There is no predictably regarding order of execution for listeners so do depend on this.
PortletConfig	Same config object is available to every concrete portlet instance derived from the same abstract portlet	String	getInitParameter()	This vale can only be set during development or deployment. Since its scope is very broad, use carefully.
DefaultPortlet Action	Available as long as the PortletURI it is attached to is available.	object	setParameter() getParameters()	It is not recommended to store objects such as PortletResponse etc. Use sparingly.
PortletAdapter	Available to all instances of the concrete portlet. Value is not unique between users.	object	getVariable() setVariable()	Use this object to store attributes that are not unique to any one user, and can be lost if the server shuts down

2.17 Portlet services

A PortalService is a discoverable extension to the Portal functionality. A portlet can query the container for a specific service and use that service without ever knowing the implementation or concerning itself with its life cycle management. Their life cycle is managed by the portal and as such does not have container restrictions placed on portlets. Example 2-26 illustrates accessing a service in a portlet.

Example 2-32 Accessing a service

```
ContentAccessService service = (ContentAccessService)
      getPortletConfig().getContext().getService(ContentAccessService.class);
```

The default installation of WebSphere Portal Server ships with the ContentAccessService. Other services could be implemented by various vendors or by yourself as seen in 2.17.2, “Custom services” on page 109. WebSphere Portal Server also supplies the CredentialsVaultService, which is discussed in detail in 2.18, “Credential Vault” on page 113.

2.17.1 ContentAccessService

The ContentAccessService provides a convenient mechanism for accessing content outside the Portal Server. Whereas the PortletContext include method is limited to content relative to the Portlet Application, the ContentAccessService has no such limitations. Example 2-33 illustrates simple usage of the ContentAccessService. There are two important methods defined in this service:

- **include(String url, PorltetRequest request, PortletResponse response)**

  This method will write the results of the URL to the response unfiltered. There is no opportunity to remove undesirable or malformed HTML. There is no URL rewriting whatsoever so relative links, such as images, will not be displayed properly. Use this method only when the URL can be trusted to return reliable content.

- getURL(String url, PortletRequest request, PortletResponse response)

  This method returns a java.net.URL object. This object can then be used to open a URLConnection, access an inputStream or access the host, port and other important information. This method provides the opportunity to filter the content prior to including it in the response.

Example 2-33 Using the ContentAccessService

```
public void doView(PortletRequest request, PortletResponse response) throws
                   PortletException, IOException {
   ContentAccessService cas = (ContentAccessService)
                              getPortletConfig().
                              getContext().
                              getService(ContentAccessService.class);
   cas.include(“http://www.ibm.com”, request, response);
}
```

2.17.2 Custom services

The Portlet API allows you to create your own services that you can install into the portal server. The main benefits of services are twofold. First, they execute outside of the Portlet Containers. Secondly, the are not tied to any given portlet and therefore their life cycle is not dependent on individual portlets. This means

that once the service has been initialized, it is available to all portlets with no further initialization cost. Likewise, the destruction cost is not absorbed by any single portlet.

To create your own service, there are four steps. Some of these steps are optional. This section will use a custom MailService as an example. This example allows a portlet to locate the MailService, send an e-mail and verify that it was in fact sent. The actual implementation of the JavaMail API is not included for clarity.

1. Define the service

   First, you must define an interface that defines the functionality this service will provide. The custom service interface must extend PortletService. The PortletService interface is a flag interface and therefore does not define any methods.

Example 2-34 Defining the Service Interface.

```
package com.yourco.services.mailservice;

import org.apache.jetspeed.portlet.service.*;

public interface MailService extends PortletService {

   public boolean sendEMail(String address, String subject, String message);

}
```

2. Implement the service

   The Service interface then needs to be implemented. The implementation class must implement the custom service interface you defined as well as the PortletServiceProvide interface. The PortletServiceProvide defines the init and destroy methods that must be implemented. The init method may be called by the factory when the implementation class is first created. In practice, while your custom factories may choose not to utilize this method, the default factories do. The init method is an appropriate location to load initialization parameters, establish connection pools, etc. Initialization parameters are discussed in step 4. The destroy method is an appropriate location to release any resources or perform any other common clean-up code.

Archived

Example 2-35 Implementing the custom service

```
package com.yourco.services.mailservice.impl;

import org.apache.jetspeed.portlet.service.*;
import org.apache.jetspeed.portlet.service.spi.*;
import com.yourco.services.mailservice.MailService;

public class MailServiceImpl implements PortletServiceProvider, MailService {

   private String server_name;

   public void init(PortletServiceConfig config)
      throws PortletServiceUnavailableException {
      //Set Mail Server name based on inititialization parameters
      server_name = config.getInitParameter("SERVER_NAME");
   }

   public void destroy() {
      //No resources to destroy
   }

   public boolean sendEMail(String address, String subject, String message) {
      //Send mail using JavaMail API
      return true;
   }
}
```

3. Create the service factory

   This step is optional when creating custom services. The factory is used by the PortletContext object to retrieve an instance of the service. Two default factories are provided with Portlet API. *PortletServiceDefaultFactory* will always return a new instance of the service. *PortletServiceCacheFactory* will always return the same instance of the service. Both of these factories call the init method of the service they are instantiating. Generally, either of the two default factories will provide the functionality you need when creating custom services. However, to ensure this example is complete, Example 2-36 illustrates a custom factory for the MailService service.

Example 2-36 Creating a custom factory

```
package com.yourco.services.mailservice.factory;

import java.util.*;
import javax.servlet.ServletConfig;
import org.apache.jetspeed.portlet.service.*;
import org.apache.jetspeed.portlet.service. spi.*;
import org.apache.jetspeed.portletcontainer.service.*;
```

Archived

```
import com.yourco.services.mailservice.impl.MailServiceImpl;

public class MailServiceFactory  implements PortletServiceFactory{

   private PortletServiceProvider psp = null;

   public PortletService createPortletService(Class service, Properties props,
             ServletConfig config) throws PortletServiceUnavailableException {
      if (psp != null) {
         return psp;
      } else {
         psp = new MailServiceImpl();
         psp.init(new PortletServiceConfigImpl(service, props, config));
         return psp;
      }
   }
}
```

4. Register the service

   Once the service interface has been defined, the implementation class created and the factory decided upon, the classes should be packaged into a jar file. This jar should be placed in the <WAS-ROOT>lib\app directory. If you have decided to use one of the default factories, they are already in this directory in the wps.jar file.

   Once the files have been deployed, the service must be registered. Open the PortletService.properties file in the <WP-ROOT>\app\wps.ear\wps.war\WEB-INF\conf directory. It is recommended that you make a backup of this file prior to modifying it. The service and its factory must be registered as illustrated in Example 2-37. The first mapping indicates that when a service is requested, the specified implementation class should be returned. The second mapping indicates which factory should be used to create this service when requested. This mapping should specify your custom factory, org.apache.jetspeed.portletcontainer.service.PortletServiceCacheFactory or org.apache.jetspeed.portletcontainer.service.PortletServiceDefaultFactory.

Example 2-37 Registering the service in PortletServices.properties

```
com.yourco.services.mailservice.MailService =
      com.yourco.services.mailservice.impl.MailServiceImpl
com.yourco.services.mailservice.impl.MailServiceImpl.factory =
      com.yourco.services.mailservice.factory.MailServiceFactory
```

Archived

Initialization parameters are also supplied in the PortletService.properties file as illustrated in Example 2-38. Accessing these parameters is illustrated in Example 2-35 on page 111.

Example 2-38 Setting Unit parameters in PortletService.properties

```
com.yourco.services.mailservice.impl.MailServiceImpl.SERVER_NAME =
   "SERVER_NAME"
```

5. Test the service

   In order for the service to become available in the Portal, the Portal Server must be restarted. Using the WebSphere Administrator's Console, restart the WebSphere Portal Application Server. Example 2-39 shows a simple portlet making use of the MailService service.

Example 2-39 Using the MailService service

```
public void actionPerformed(ActionEvent event) throws PortletException {
   PortletRequest request = event.getRequest();
   String address = request.getParameter("address");
   String subject  = request.getParameter("subject");
   String msg = request.getParameter("msg");
   MailService mailService = (MailService)
      getPortletConfig().getContext().getService(MailService.class);
   String result = "" + mailService.sendEMail(address, subject, msg);
   request.getPortletSession().setAttribute("EmailResult", result);
}
```

2.18 Credential Vault

WebSphere Portal can be configured to exist in a single sign-on environment using a number of different approaches. If the various systems participating in the SSO realm all authenticate to Domino®, WebSEAL can provide the SSO functionality. Third-party authentication mechanisms such as Tivoli Access Manager can also be used to create a unified environment for the user.

However, on the portlet level, there may be systems outside the current SSO realm or applications that simply require an explicit login. To facilitate the storage, retrieval and usage of the credentials necessary to access these back-end systems, WebSphere Portal provides the Credentials Vault Service. This service is based on the Portlet Service architecture discussed in 2.17, "Portlet services" on page 108. The CredentialsVaultService allows you to easily and securely

Archived

persist user IDs and passwords without concerning yourself with database access code.

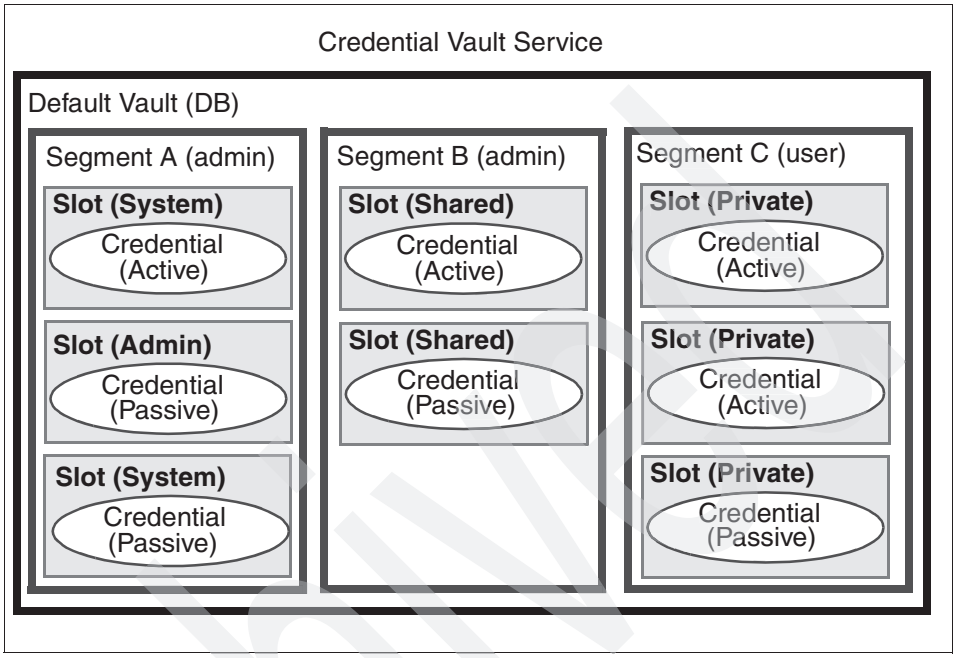


Figure 2-14 Credential Vault objects

2.19 Core Credential Vault objects

There are several key objects used when working with or administering the Credential Vault.

2.19.1 Vault

This is a persistent store where credentials are actually stored. WebSphere Portal provides the default database vault. The Tivoli Access Manager lock box could also be registered and used as a vault. You can create and register your own custom vault implementations that may store credentials in some database, in memory or even a simple file system.

2.19.2 Segment

A vault can be separated into segments to distinguish the access control portlets have when working with the credentials stored in the vault. Portlets can retrieve

credentials from any type of segment. A vault can only be segmented by the administrator.

- **Administrator Managed** A segment flagged as Administrator Managed prevents portlets from creating new slots in the segment.
- **User Managed** This type of segment allows a portlet to dynamically create new slots and to place credentials in that slot. Only the default vault provided by WebSphere portal provides user-managed segments.

2.19.3 Slot

A slot is “drawer” in a segment that actually contains the credential. A slot can only contain a single credential. When retrieving credentials, a portlet searches the vault for a slot based on the slot ID. This ID is usually persisted in the PortletData object. The definition and implementation of slots is dependent on the vault containing the slot. The default vault implementation provided by WebSphere Portal provides four types of slots.

- **System slot** The credentials stored in this type of slot are available to all users and portlets. This type may be used when a user ID/password is company-specific and not unique for each employee.
- **Administrative slot** The credentials stored in this type of slot are applicable to individual users but are associated with administrator-defined resources such as Lotus® Notes®.
- **Shared slot** The credentials stored in this type of slot are available to all the portlets of a specific user. This type may be used when several portlets will access the same back-end system on behalf of the same user.
- **Portlet Private slot** The credentials stored in this type of slot are available to the single portlet instance that stored it. The credential is not accessible from any other portlet. This type may be used when the credentials are required only by a single portlet and are not applicable to any other user.

2.19.4 Credential

This object actually contains the user ID/password pair. There are two base types of credentials.

- **Passive credential** This type of credential simply persists the user ID/password pair. When a portlet needs to access some back-end system with credentials stored in a passive credential, it is required to retrieve the user ID string and password character array from the credential and manually construct the connection to the back end. Example 2-40 on page 116 illustrates using a passive credential.

Example 2-40 Accessing a Passive Credential

```
UserPasswordPassiveCredential cred =
       (UserPasswordPassiveCredential) vault.getCredential(
                    slotID,
                    "UserPasswordPassive",
                    null,
                    request);
if (cred != null ){
   String pass = cred.getPassword().toString();
   String userid = cred.getUserId();
}
// Use ID and password to connect to some back end
```

- **Active credential** This type of credential encapsulates the user ID/password pair as well as the all the logic required to access the back-end system. Portlets do not have access to the user ID or password persisted in the credential. However, the credential provides connection methods and utilizes the persisted user ID and password to establish the necessary connection. Example 2-41 illustrates how an active credential never returns the user ID or password but instead provides the requisite connection functionality.

Example 2-41 Accessing and using an Active Credential

```
JavaMailCredential credential =
       (JavaMailCredential) vault.getCredential(
                 slotID,
                 "JavaMailCredential",
                 config,
                 request);
javax.mail.Session mailSession =
       javax.mail.Session.getDefaultInstance(props, null);
if (credential != null ){
   mailSession = credential.getAuthenticatedSession(mailSession, host);
   mailSession.getTransport().send(someMsg);
}
```

  Since an active credential inherently provides more security, it is the preferred type of credential.

WebSphere Portal ships with several predefined types of credentials.

- Active credentials
  - HTTPBasicAuthCredential
  - HTTPFormBasedAuthCredential
  - JavaMailCredential
  - LtpaTokenCredential

  - WebSealTokenCredential
  - SiteMinderTokenCredential
- Passive credentials
  - SimplePassiveCredential
  - UserPasswordPassiveCredential
  - JassSubjectPassiveCredential

Example 2-42 illustrates sample code that can be used to store credentials using the Credential Vault Service provided by WebSphere Portal.

Example 2-42 Storing credentials

```
PortletContext context = getPortletConfig().getContext();
CredentialVaultService vault = (CredentialVaultService)
            context.getService(CredentialVaultService.class);
ObjectID defaultSegmentId = vault.getDefaultUserVaultSegmentId();
Map descripMap = new HashMap();
descripMap.put("en", "A simple test slot");
CredentialSlotConfig slot = vault.createSlot(
         "",
         defaultSegmentId,
         descripMap,
         null,
         CredentialVaultService.SECRET_TYPE_USERID_STRING_PASSWORD_STRING,
         false,
         true,
         request);
request.setAttribute("Test_SlotID", slot.getSlotId());
int passLength = password.length();
char[] passChars = new char[passLength];
password.getChars(0, passLength, passChars, 0);
vault.setCredentialSecretUserPassword(
         slot.getSlotId(),
         userid,
         passChars,
         request);
```

CredentialVaultService methods

- **getCredentialTypes** Returns an Iterator of all Credential Types that are registered in the Credential Type Registry.

Archived

2.20 Portlet JSPs

When designing your portlet applications, you will generally use the MVC Model 2 architecture discussed in 2.3.2, “Portlet MVC architecture” on page 57. For the development of dynamic portlet JSPs, a rich tag library is provided with WebSphere Portal Server. There are several custom tag libraries supplied with WebSphere Portal Server depending on the installation type and what additional components are installed.

- **portlet.tld** This tag library contains the tags used in day-to-day JSP development when working with JSPs.
- **c2a.tld** This tag library contains the tags to be used in cooperative portlets using the declarative approach. For details, see Chapter 12, “Cooperative portlets” on page 371.
- **engine.tld** This tag library is intended to be used in the construction of themes and skins.
- extend.tld This tag library is only supplied if the installation type is extend or experience. These tags are not available with the enable installation.
- content.tld This tag is used in JSPs working with the PortletContent Organizer.
- menu.tld This tag library provides access to Collaborative functionality in the themes.
- person.tld This tag library provides access to Collaborative functionality inside your portlets.

2.20.1 Portlet tag library

Like all tag libraries in the WebSphere Portal Server, the portlet.tld is located in the <WP-ROOT>app\wps.ear\wps.war\WEB-INF\tld directory. Example 2-43 illustrates referencing the tag library at the beginning of a JSP.

Example 2-43 Referencing a tag library

```
<%@ taglib uri="/WEB-INF/tld/portlet.tld" prefix="portletAPI" %>
```

This section will cover the tags available in the portlet.tld tag library and some of their most common uses.

- `init <portletAPI:init />`

  This tag must be called if you wish to access the PortletRequest, PortletResponse or PortletConfig objects in the JSP. This tag simply initializes three variables for you: `portletRequest`, `portletResponse`, and

portletConfig. Attempting to access these variables without calling the init tag will cause the page compilation of the JSP to fail. However, you still have full access to the javax.servlet.http.HttpServlet objects via the normal variable names.

- createReturnURI <portletAPI:ecreateReturnURI />

  This tag returns a String pointing to the portlet in the previous mode. The resulting URI could be used to create a Back button or to specify and an action on a FORM. If you wish to add a PortletAction to the URI object in order to notify any applicable listeners, you can include the URIAction tag in the body of the createReturnURI tags. Example 2-44 illustrates this approach.

Example 2-44 Adding a PortletAction to the PortletURI

```
<portletAPI:createReturnURI >
   <portletAPI:URIAction name="submit" />
</portletAPI:createReturnURI>
```

You can also add a parameter to the PortletURI object using a similar approach to the one used with the PortletAction.

Example 2-45 Adding a Parameter to the PortletURI and the resulting URI

```
<portletAPI:createReturnURI >
   <portletAPI:URIParameter name="fname" value="john" />
</portletAPI:createReturnURI>
```

Result:

```
/wps/myportal/.cmd/ActionDispatcher/_pagr/104/_pa.104/113/.md/-/.piid/188/.ciid
/223/.reqid/-1/PC_188_fname/john#223
```

- createURI <portletAPI:createURI />

  This tag returns a String pointing to the portlet in the current mode. As with the createReturnURI tag, PortletActions and parameters can be added to the resulting URI. Though the documentation indicates that the state can be controlled by passing a string attribute, this functionality is not implemented.

- URIAction <portletAPI:URIAction name=”sting”/>

  This tag is only used when creating a PortletURI object. Example 2-44 illustrates the use of this tag. This tag requires that a name attribute be specified.

Archived

- URIParameter <portletAPI:URIParamter name="string" value="string"/>

  This tag is only used when creating a PortletURI object. Example 2-45 on page 119 illustrates the use of this tag. This tag requires that name and value attributes be specified.

- dataAttribute <portletAPI:dataAttribute name="string" />

  This tag will retrieve from the PortletData object the attribute specified by the name attribute. If the attribute does not exist in the PortletData, nothing is returned. When the dataAttribute tag is used in the body of the a dataLoop tag, it does need to specify the name of the attribute.

Example 2-46 Retrieving a single PortletData attribute

```
Welcome <portletAPI:dataAttribute name = "pref.nick_name" /> to your page.
```

- dataLoop <portletAPI:dataLoop pattern="string">

  </portletAPI:dataLoop>

  This tag provides a loop through all the attributes stored in the PortletData object. Though by default, it will iterate through all attributes, it is possible to specify a pattern to limit the attributes it locates. Omitting the pattern attribute will return all attributes. Example 2-47 illustrates the usage of this tag. Notice the loop simply iterates through the collection of attributes; it does not retrieve the value. To retrieve a PortletData value, use the dataAttribute tag.

Example 2-47 Looping through the attributes in the PortletData object

```
<portletAPI:dataLoop pattern="pref.*">
   <portletAPI:dataAttribute/><br>
</portletAPI:dataLoop>
```

  Though using an asterisk in the pattern is helpful for readability and reliability, the pattern attribute in fact does not need to use an asterisk at all. The tag will attempt to find the value specified by the pattern attribute anywhere in the name of the attribute. For example, if an attribute is stored in the PortletData with the name "pref.greeting", the code in Example 2-48 would successfully locate the attribute. However, it is important to note that the pattern is case-sensitive. Therefore, the pattern "name" would not locate the attribute "Name".

Example 2-48 Using the Pattern attribute

```
<portletAPI:dataLoop pattern="eet">
```

- settingsAttribute <portletAPI:settingAttribute name="string" />

This tag provides access to the parameters set in the <config-param> blocks in the portlet.xml's concrete portlet section. When the dataAttribute tag is used in the body of the settingsLoop tag, it does need to specify the name of the attribute.

Example 2-49 Accessing the PortletSettings attributes

```
For support contact <portletAPI:settingsAttribute name = "author" />
```

- settingsLoop <portletAPI:settingsLoop pattern="string">

  </portletAPI:settingsLoop>

If several configuration parameters have been set in the portlet.xml, they can all be retrieved with this tag. The pattern tag is optional.

Example 2-50 Looping through the PortletSettings attributes

```
<portletAPI:settingsLoop pattern="info.">
    <portletAPI:settingsAttribute/><BR>
</portletAPI:settingsLoop>
```

If you do not include the pattern attribute or enter an empty string, it will return all attributes in the PortletSettings object. As with the dataLoop tag, the settingsLoop tag will attempt to locate the specified pattern anywhere in the attribute's name. For example, if a <config-param> were set in the portlet.xml with a name of "info.author", the code in Example 2-51 would successfully retrieve the attribute. However, it is important to note that the pattern is case-sensitive. Therefore the pattern "author" would not locate the attribute "Author".

Example 2-51 Using pattern to locate an attribute

```
<portletAPI:settingsLoop pattern="thor">
```

- encodeNameSpace <portletAPI:encodeNamespace value="string" />

When including JavaScript functions or other variables that will be returned to the aggregated portal page, it is important to ensure the values are unique in order to avoid name collisions. This tag prefixes the namespace of the portlet to the string it is passed. This tag should be used when creating the variable and when accessing it. Example 2-52 on page 121 illustrates the usage and result of this tag.

Example 2-52 Encoding the name space

```
<portletAPI:encodeNamespace value="function1" />
```

Result:

```
PC_189_function1
```

- `encodeURI`

  This tag will prefix the full URL of the portal to the passed path value. For example, if the image yourco.jpg is in the images folder directly under the root of the deployed portlet application, the code shown in Example 2-53 would successfully locate the image and create a fully qualified URL.

Example 2-53 Creating afully qualified URL

```
<img src= <portletAPI:encodeURI path="/images/yourco.jpg" /> >
```

Result

```
http://ka0kkhc.sg246897.com/wps/WPS_PA_206/images/yourco.jpg
```

- `if <portletAPI:if attribute= "string">`

  `</portletAPI:if>`

  This tag allows you test some of the more common conditions a portlet may face. When the attribute evaluates to true, the body of the if tag is executed. There are several attributes you can evaluate.

  - mode
  - state
  - locale
  - mime type
  - markup
  - capabilities

  Though the infocenter indicates that a previous mode can be evaluated as well, the previousMode attribute is not functional. You may choose to execute several if statements individually as shown in Example 2-54 on page 122.

Example 2-54 Executing If tags individually

```
<portletAPI:if state = "Normal"> state is normal </portletAPI:if>
<portletAPI:if state = "Maximized"> state is maximized </portletAPI:if>
<portletAPI:if locale = "en"> Locale is english </portletAPI:if>
<portletAPI:if markup = "html"> Markup is html </portletAPI:if><
<portletAPI:if mimetype = "text/html"> mime type is text html
</portletAPI:if><BR>
<portletAPI:if mode="view"> Mode is View </portletAPI:if ><BR>
```

Archived

You can evaluate more than one condition on a single attribute. In this case, if any of the conditions are true, that attribute will evaluate to true.
Example 2-55 illustrates this.

Example 2-55 Evaluating multiple conditions on a single attribute

```
<portletAPI:if state="Normal, Maximized" >
```

You may also evaluate multiple attributes in the same tag as illustrated in Example 2-56. All conditions must evaluate to true for the if tag to return true.

Example 2-56 Evaluating multiple attributes

```
<portletAPI:if state="Normal" mode="view" locale="en">
   Displaying the normal English view
</portletAPI:if>
```

- `log <portletAPI:log text="string" level="string"/>`

  This tag will write the value passed to the log file located in the <WP-ROOT>\log directory. The text attribute contains the string you wish to write to the log file. The level attribute indicates which level this message should be written under. This tag does not evaluate whether the requested level is enabled before it attempts to write the message. For more information on writing to the log, see 2.9.12, “PortletLog object” on page 92.
  Example 2-57 illustrates the usage of this tag. The valid values for the level attribute are error, warn, debug and info. If you omit the level tag, the default level is error.

Example 2-57 Using the log tag

```
<portletAPI:log text="There was an error" level="warn"/>
```

- `text <portletAPI:text key="sting" bundle="string">`

  Note: The `text` tag has been deprecated in this release. You should now use the `fmt` tag from the JSP Standard Tag Library (JSTL). For details, see Chapter 8, “National Language Support (NLS)” on page 249.

  This tag was used to provide access to key-value pairs in resource bundles.

- `bidi <porteltAPI:bidi locale=”string” dir=”ltr | rtl” />`

This tag is used to support text for bidirectional languages. Bidirectional languages are read from right to left or from bottom to top. The attributes are not required. For example, if the request indicates that the client is Hebrew or Arabic, it will execute the tag contents if dir is set to `rtl`.

2.21 Resources

- For the most up-to-date information on WebSphere Portal, refer to the Portal zone at:

  `http://www7b.boulder.ibm.com/wsdd/zones/portal/`

- For help via a news group, visit `new.software.ibm.com` and locate the ibm.software.websphere.portal-server news group.
- For other Redbooks discussing installation and administration, refer to:

  `http://www.redbooks.ibm.com`

Archived

3

Portal Toolkit

This chapter provides an overview of the WebSphere Portal Toolkit V5. It includes general information about requirements and the new function provided in this release to create and deploy portlet application projects.

This chapter discusses the following topics:

- Portal Toolkit installation
- Portlet application wizard
- Developing portlet applications
- Deploying portlets
- Adding portlets to applications
- Examples

Archived

© Copyright IBM Corp. 2004. All rights reserved.

3.1 Hardware and software requirements

Portal Toolkit V5.0 can be installed on the following operating systems:

- Windows 2000 Server + Service Pack 3
- Windows XP Professional + Service Pack 1

Portal Toolkit 5.0 supports the following in the WebSphere Studio family:

- WebSphere Studio Enterprise Developer V5.01
- WebSphere Studio Application Developer Integration Edition V5.01
- WebSphere Studio Application Developer V5.01
- WebSphere Studio Site Developer V5.01

The following databases are supported by the WebSphere Portal V5.0:

- WebSphere Portal built-in database (Cloudscape)
- IBM DB2 Enterprise Edition V7.2 with Fix Pack 7 or Fix Pack 8
- IBM DB2 Enterprise Server Edition V8.1
- Oracle V8.1.7 or V9.2

Note: The hardware and software requirements depend on the configuration of the development environment.

Local debug configuration

The hardware and software requirements for a full portlet development environment in a single machine are shown in Table 3-1.

Table 3-1 Development workstation with local debug:

Components	Requirement
Hard disk space	1 GB plus storage for portlet development projects
Memory	768 MB minimum, 1 GB recommended
Operating system	Windows 2000 with Service Pack 3 or Windows XP Professional with Service Pack 1
Software	Portal Toolkit V5.0 WebSphere Studio V5.01 WebSphere Portal V5.0

Note: Local debugging on WebSphere Portal V4.2 is also supported.

Archived

To debug a personalized portlet application, the WebSphere Studio type must be either WebSphere Studio Enterprise Developer or WebSphere Studio Application Developer.

Remote debug configuration

The hardware and software requirements for remote test and debugging are shown in Table 3-2 and Table 3-3 on page 139.

Table 3-2 Development machine using remote server attach

Components	Requirement
Hard disk space	2 GB plus storage for portlet development projects
Memory	768 MB minimum, 1 GB recommended
Operating system	Windows 2000 with Service Pack 3 or Windows XP Professional with Service Pack 1
Software	Portal Toolkit V5.0 WebSphere Studio V5.01

Table 3-3 Remote server

Components	Requirement
Hard disk space	1.5 GB plus storage for portlets
Memory	768 MB minimum, 1 GB recommended
Operating system	Windows 2000 with Service Pack 3
Software	WebSphere Portal V5.0, including all prerequisite software

Note: You can also set up the remote server attach for WebSphere Portal V4.2 running on a remote machine.

Archive

3.2 Portal Toolkit installation

IBM Portal Toolkit V5.0 provides the capabilities to create, test, debug and deploy individual portlets and Web content. Portal Toolkit is implemented as a plug-in to IBM WebSphere Studio Workbench. The Portal Toolkit provides the following:

- Portlet project development wizards, editing and debugging capabilities.
- Portal projects, in which you can publish your portlet application onto your target WebSphere Portal server machine. Your portlet will appear on the debug page of your Portal Server. Both remote and local test environment are supported in this release.
- Portlet application samples for enterprise applications.

Note: Portal Toolkit can be downloaded from the following Web site:

`http://www.ibm.com/websphere/portal/toolkit`

The Portal Toolkit installation step-by-step procedure is described in Appendix A, “Portlet development platform sample installation” on page 501.

3.3 Development environment

Portal Toolkit V5.0 allows you to set up a portlet development environment when installed as a plug-in in WebSphere Studio.

Development environment configurations

For WebSphere Portal V5, you can set up your development environment to support testing and debugging portlets on the local development machine or on a remote server.

- Local debug configuration: for debugging portlets on the local development machine, you must use the *WebSphere Portal Test Environment* configuration. In this environment, you can run and debug your portlets without having to manually deploy them to the server. During the installation of Portal Toolkit, the portal server is installed to the WebSphere Test Environment of WebSphere Studio.
- Remote debug configuration: for debugging portlets on a remote server, you must use the *WebSphere Portal Remote Server Attach* configuration.

Archived

3.4 Portlet application wizard

In this section, you will see how to build a portlet application project with the wizard and then test it in the Test Environment. Portlets are WAR (Web Archive) files so they need to be tested in a server.

Follow these steps to create a portlet application project:

1. Start WebSphere Studio Site Developer by clicking **Start -> Programs -> IBM WebSphere Studio -> Site Developer 5.0**.
2. In the pop-up window, enter the Site Developer workspace directory. The workspace directory is the place where the projects and servers are stored. You can create different workspace directories for different projects.

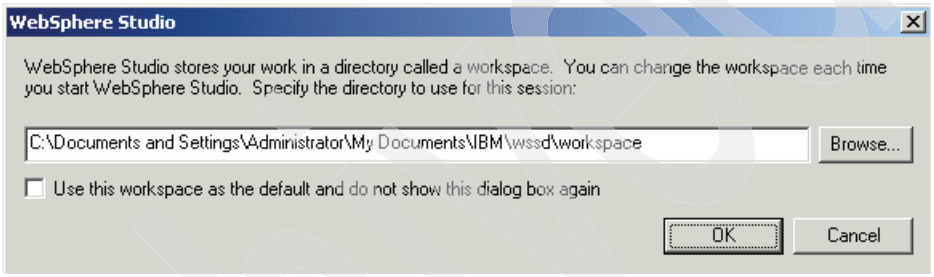


Figure 3-1 Workspace directory

3. Once WebSphere Studio has started, switch to the Portlet Perspective by clicking **Window -> Open Perspective -> Other** and selecting the **Portlet** perspective.
4. To start the Portlet Application wizard, from the menu bar select **File -> New -> Portlet Application Project**. The wizard can also be started by selecting **File -> New -> Other** and then **Portlet Development** on the left side and **Portlet Application Project** on the right side.

   When the wizard starts, you will be presented with the window shown in Figure 3-2 on page 130.

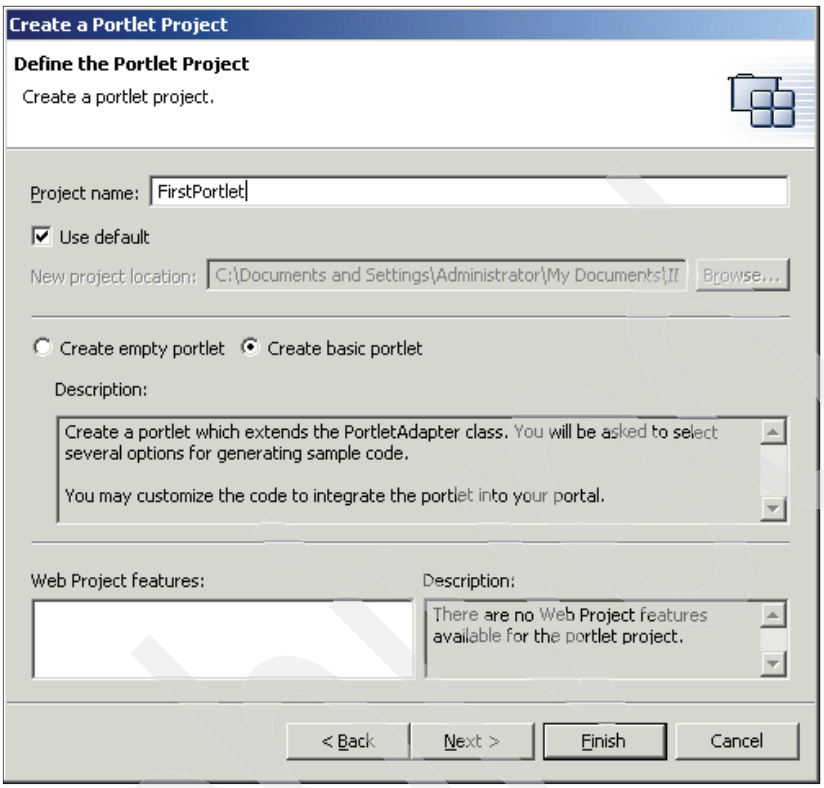


Figure 3-2 Portlet project wizard initial window

The fields in the first window are:

- **Project Name**: this value will determine the name of the project created by this wizard. The value entered here will be used throughout the remainder of the wizard as the default value for other parameters.
- **Use default**: this checkbox indicates that you would like the entire contents of the application stored in the workspace. If you would like the contents of the application stored somewhere else on the file system, deselect this box.
- **New project location**: if you deselect the **Use default** check box, this field is enabled and allows you to specify where the application will be saved.
- **Create an empty portlet**: this option will create a portlet entry in the portlet deployment description, but no portlet Java classes. If you choose this option, you will need to add portlet code.

– **Create a basic portlet**: this is the default option and will create a basic portlet in the portlet application. The portlet will extend from PortletAdapter and contain meaningful implementations of all four do methods. The implementations will adhere to the MVC approach. A bean will also be created for you to encapsulate your business logic. The resulting folder structure will appear as in Figure 3-10 on page 137.

– **Web Project features**

5. When you select **Finish** in this window, a portlet application will be created with the default values. By selecting **Next**, you can modify the default values for the J2EE Settings Page, as illustrated in Figure 3-3.

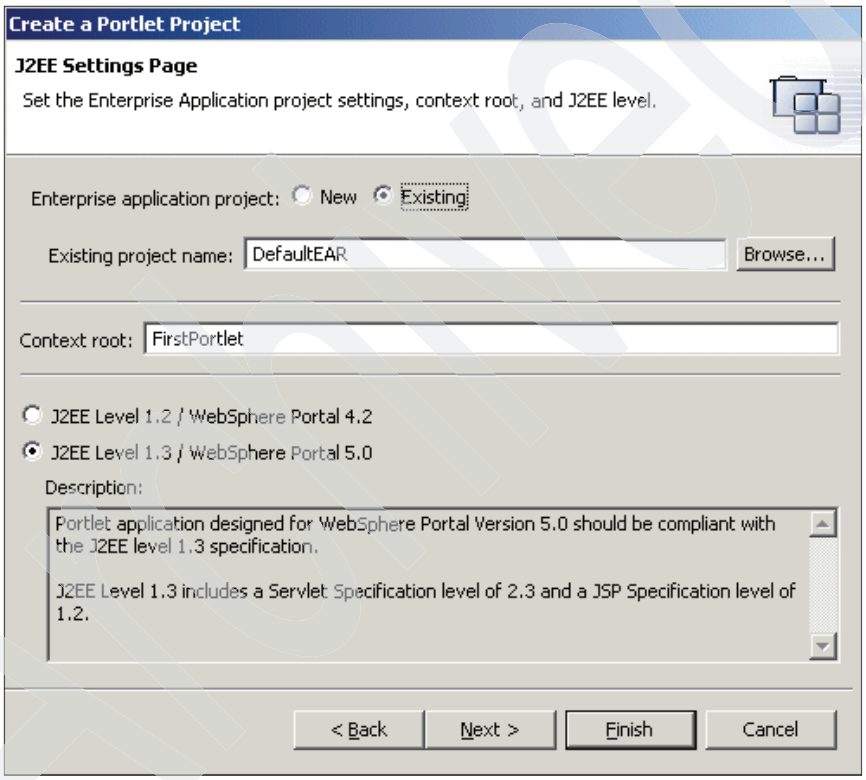


Figure 3-3 J2EE Settings Page

The fields in this window are:

– **Enterprise Application project**: although the Portal Server does not recognize EAR files, the portlet application in WebSphere Studio must be contained in an Enterprise Application. When using the Portal Server debugging environment, all portlet applications contained in an EAR file are deployed together. You may choose an existing EAR file or enter a

new one to be created. If you enter a new one then you can choose the location where your application will be stored.

- **Context Root**: this value will be used in the application.xml and Web settings files. It will not be the context root of the portlet application when deployed. Since the EAR is not used to deploy the portlet application into a full server, this value is only used when the EAR is published to debug Portal Server connected to WebSphere Studio Application Developer.
- **J2EE level**: the portlet application designed for WebSphere Portal V5.0 should be compliant with J2EE level 1.3 specification.

6. If you click **Next**, you can select dependent JARs existing within the Enterprise Application project selected in the second window.

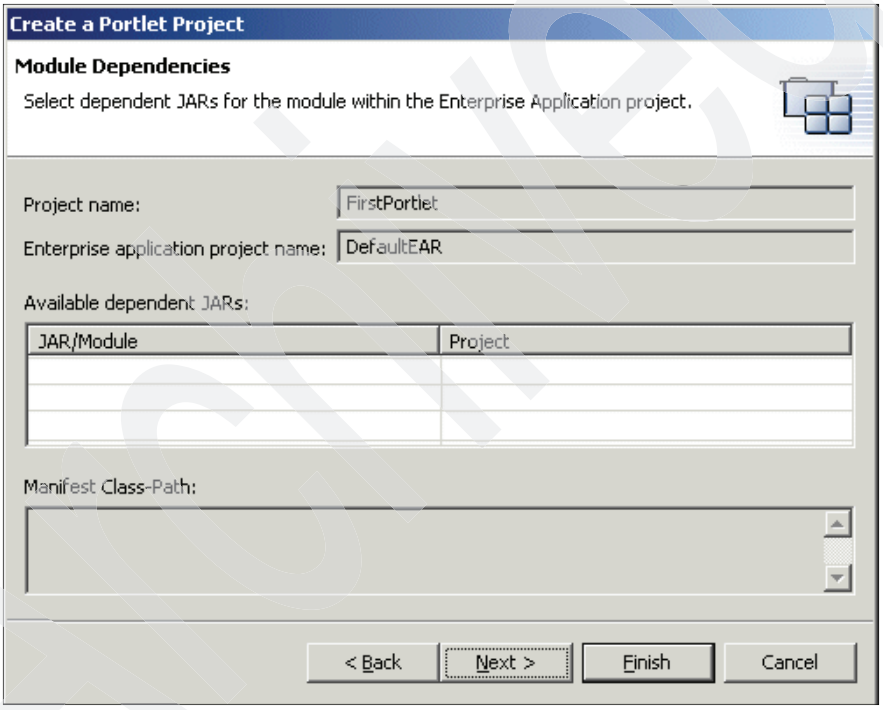


Figure 3-4 Module dependencies window

7. By selecting **Next** again, you can see the general settings of the portlet.

Archived

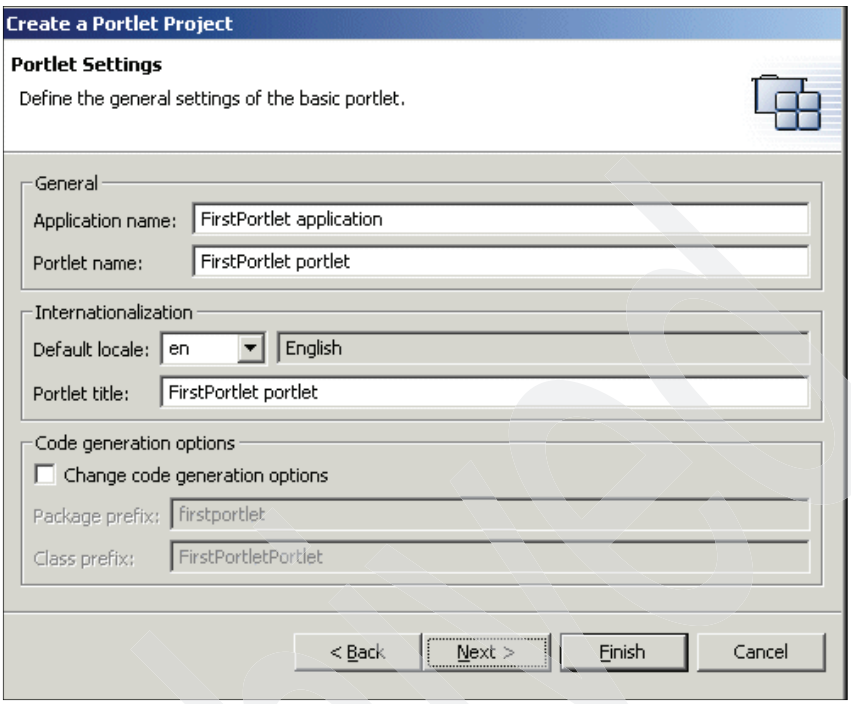


Figure 3-5 Portlet settings

The fields in this window are:

- **Application name**: this name is used in the portlet.xml to specify the abstract application name. This value will never be seen by the administrator of the portlet or the end user. Generally, there is no need to change this value.
- **Portlet name**: this value is used to identify the abstract portlet. This name will never been seen by the administrator or the end user. There is typically no need to alter this value.
- **Default locale**: this value adds the default locale and the language block to the portlet.xml.
- **Portlet title**: this will complete the language block with the title. The description, short title and keyword elements are included in the language block but left empty.
- **Code generation options**: this check box allows you to change the default prefix names given to the Java package which will be created in the Java Source folder and the class name of the default source created by the wizard.

8. The next window of the wizard allows you to define event handling options, for example action event and message event handling. Checking the **Add action listener** check box or **add message listener** check box will cause the portlet code generated by the wizard to implement the ActionListener or MessageListener interface, respectively. Form samples for those events are also provided by the wizard.

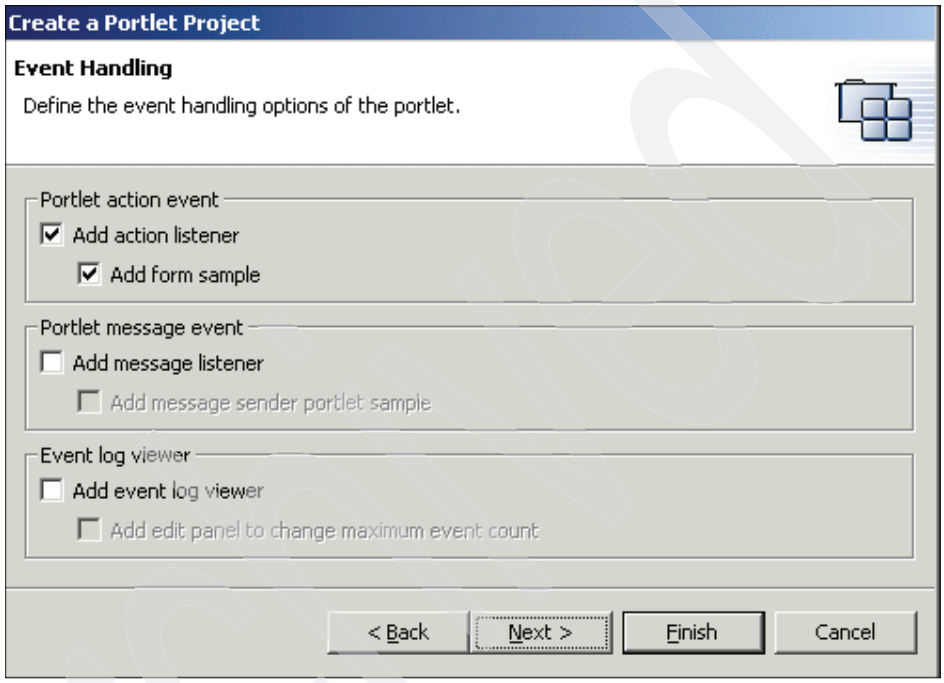


Figure 3-6 Fifth screen of the Portlet application project wizard

9. In the next window of the wizard, you can add Credential Vault handling to the portlet. For more information about the different options of Credential Vault, see Chapter 10, “Using the Credential Vault” on page 319.

Around

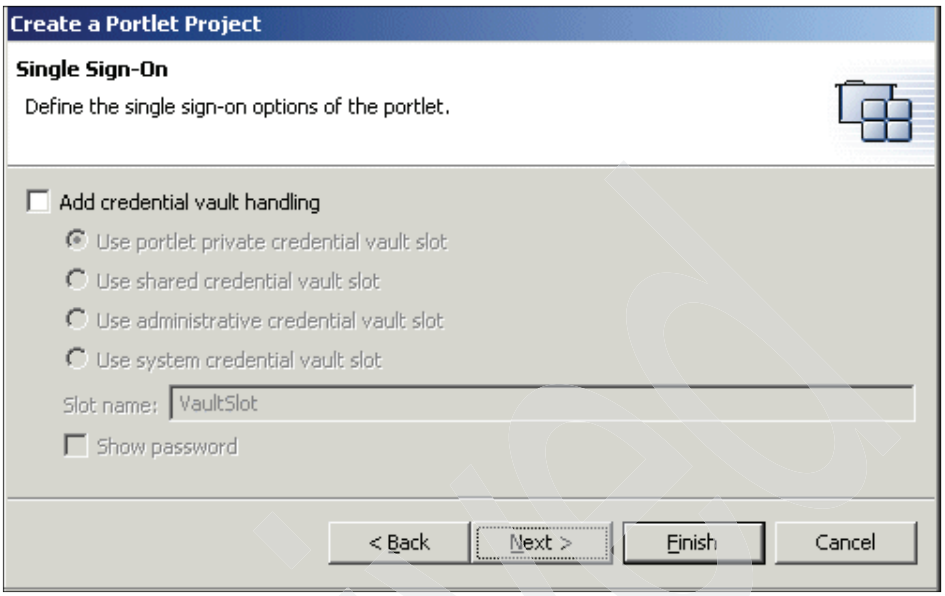


Figure 3-7 Credential Vault window

10. This next window allows you to add more miscellaneous options of the portlet. By default, the html markup and portlet View mode are created.

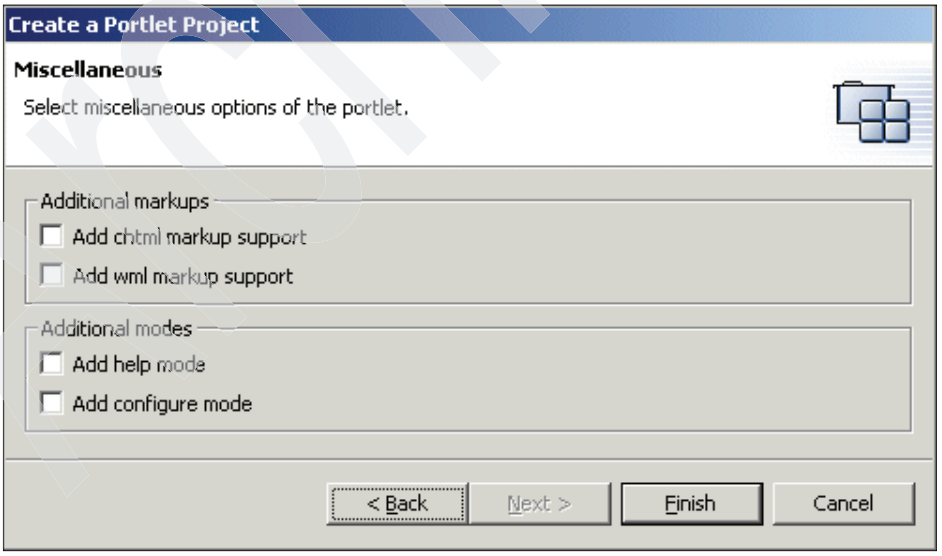


Figure 3-8 Miscellaneous options

The fields in this window are:

– **Additional markups**: these check boxes indicate which markup languages you intend to support. By selecting a value, a new folder will be created under the JSP folder containing JSPs specifically for the markup. A basic portlet specifying all three markups will produce the folder structure shown in Figure 3-9.

– **Additional modes**: these check boxes allow you to enable fragment for Help and Configure mode for the html markup. Two new JSPs files will be created as you can see in Figure 3-9.

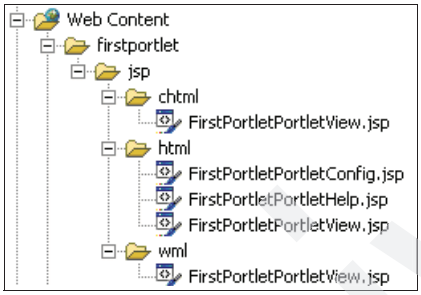


Figure 3-9 Supporting additional markups

11. Select **Finish** and the wizard will create the necessary folder structure, classes, JSPs and deployment descriptors.

3.5 Developing portlet applications

The wizard will create a sample skeleton you can use as a foundation for your portlet development. Figure 3-10 on page 137 shows the result of creating a basic portlet application with the wizard.

Archived

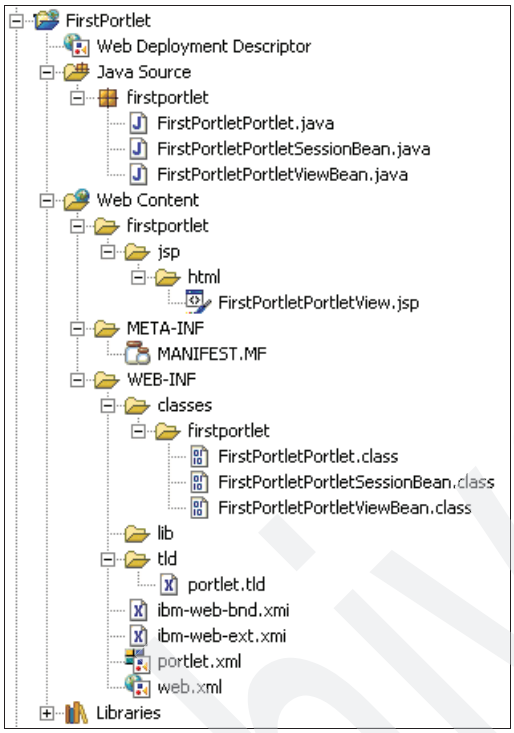


Figure 3-10 Folder structure and contents of a basic portlet application

3.5.1 Portlet application contents

The generated project contains the following folders and files by default:

- **Java Source**: this folder contains the Java files that make up the portlet application. By default, the wizard assumes you will create a basic portlet following an MVC approach and creates a simple Java bean for you. Whatever package name was specified in the fourth screen of the wizard will be created. If a simple class name was specified without a package, the wizard will place the portlet in a package named portlet.
- **Web Content**: this folder contains everything needed to deploy the application to the portal. Essentially, this folder will become the .war file. This folder contains three sub folders:
- **Package / jsp**: this folder will contain all the JSPs used by the application to create the content of the portlet. For each markup you choose to support, a directory will be created containing JSPs for each mode a portlet may support.
- **META-INF**: this folder contains the MANIFEST.MF file.

Archived

- **WEB-INF**: this folder contains the compiled code and deployment descriptors used by the Portal to install the application.
  - **classes**: if your compiled portlet class files are not packaged into a jar file, they are included in this directory. The complete package structure is created in this folder.
  - **lib**: this directory contains any jar files that your application makes use of and which are not normally available in the portal environment via the classpath. Also, if you have packaged your compiled portlets into a jar, the jar file is placed in this directory.
  - **tld**: this is included to allow JSPs to compile and recognize the custom tags available in the portal environment. This folder and file are not required at deployment time since the tld is installed with Portal. To make maintenance easier and more reliable, you may choose to delete this file upon deployment.
  - **ibm-web-bnd.xmi**: this file is not used by the portal environment but is included with all Web applications created in WebSphere Studio.
  - **ibm-web-ext.xmi**: this file is not used by the portal environment but is included with all Web applications created in WebSphere Studio.
  - **portlet.xml**: this is the deployment descriptor required by the Portal server to install the portlet application. It must be located under the WEB-INF folder or installation will fail.
  - **web.xml**: this deployment descriptor is required by the application server to install the Web application. It must be located under the WEB-INF folder or installation will fail.
- **images**: this folder is not created for you by the wizard. However, if the JSPs you create use images, it is a good practice to place them in an images folder under the Web Content directory and access them as demonstrated in Example 3-1.

Example 3-1 Accessing images in JSPs

```
<IMG src='<%=response.encodeURL("/images/database.gif")%>' />
```

3.5.2 Generated classes

Selecting **Basic portlet** in the wizard will generate some classes for you, depending on the number of modes your portlet supports.

- **Portlet**: this is a simple portlet extending from PortletAdapter and implementing the four modes a portlet may support. Each method is very

Archive

similar to the code in Example 3-2. Of course, each do method (doView, doEdit, doHelp, doConfigure) will need to call (include) the appropriate JSP.

Example 3-2 Sample doView method

```
public void doView(PortletRequest request, PortletResponse response) throws
PortletException, IOException {
   // Make a View mode bean
   FirstPortletViewBean viewBean = new FirstPortletViewBean();
   request.setAttribute(View_BEAN, viewBean);

   // Save name in the View mode bean
   viewBean.setPortletName("FirstPortlet");

// Invoke the JSP to render
getPortletConfig().getContext().include(View_JSP + getJspExtension(request),
request, response);
   }
```

- **PortletSessionBean**: this simple JavaBean stores portlet instance data in a portlet session. Example 3-3 demonstrates the code in the bean.

Example 3-3 Sample PortletSessionBean

```
public class FirstPortletPortletSessionBean {
   private String formText = "";

   public void setFormText(String formText) {
        this.formText = formText;
    }

   public String getFormText() {
        return this.formText;
    }
}
```

- PortletViewBean: this simple JavaBean passes data from the portlet to a JSP for markup rendering in View mode. If you have defined that your portlet will support more modes then it will create PortletEditBean, PortletConfigBean and PortletHelpBean to render their respective modes. Figure 3-4 on page 140 demonstrates the code in the bean.

Archived

Example 3-4 Sample PortletViewBean

```
public class FirstPortletPortletViewBean {


    private String formActionURI = null;

   public void setFormActionURI(String formActionURI) {
        this.formActionURI = formActionURI;
    }

   public String getFormActionURI() {
        return this.formActionURI;
    }
}
```

- **PortletView.jsp**: several JSPs are created for you by the wizard. They are simple in functionality. They retrieve their respective bean from the request object and print out the name property. Example 3-5 displays the code. Initially, all JSPs have the same functionality, but different text.

Example 3-5 JSP code

```
<%@ page contentType="text/html" import="java.util.*, firstportlet.*"%>
<%@ taglib uri="/WEB-INF/tld/portlet.tld" prefix="portletAPI" %>
<portletAPI:init/>

<DIV style="margin: 6px">

<H3 style="margin-bottom: 3px">Welcome!</H3>
This is a sample <B>view mode</B> page. You have to edit this page to customize
it for your own use.<BR>
The source file for this page is "/Web
Content/firstportlet/jsp/html/FirstPortletPortletView.jsp".

</DIV>
```

3.6 Portlet.xml descriptor

To facilitate creating, organizing and updating the portlet.xml deployment descriptor, the Portal Toolkit provides an intuitive interface. This interface is accessed by double-clicking the **portlet.xml** file. For details on the elements in the portlet.xml deployment descriptor, see 2.6.2, “portlet.xml” on page 67. Since the interface is rather intuitive, this section will only cover some of the more important fields and points of concern.

The interface allows you to select the contained components and work with a window dedicated to that element. For example, Figure 3-11 demonstrates working with the abstract portlet application.

The wizard will create unique IDs for the abstract and concrete applications. These values are large and entirely unmemorable. Feel free to change the value to something more meaningful but still unique. A best practice is to create the application name from your organization name. See 2.6.5, “UID guidelines” on page 77.

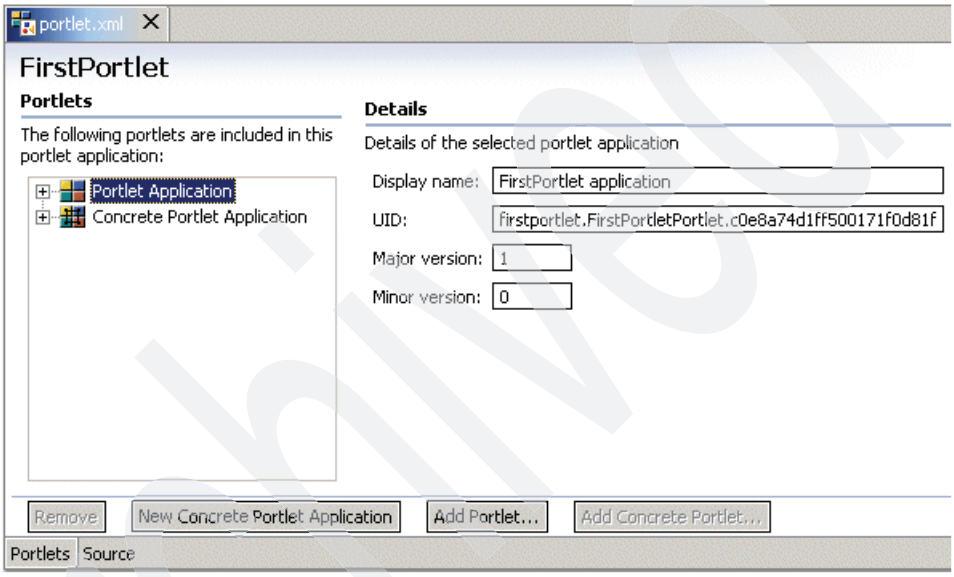


Figure 3-11 Working with the abstract portlet application

If you want to add more portlets for deployment, select the **Add portlet** button. The resulting dialog, shown in Figure 3-12 on page 142, will allow you to add portlets already defined in the associated web.xml. You cannot add portlets already defined in the abstract application.

Archive

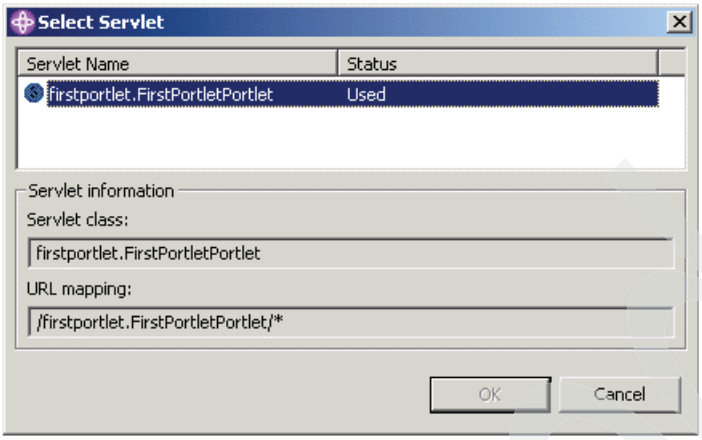


Figure 3-12 Add portlet

Contained in the abstract application are the abstract portlets to be deployed in this application. Figure 3-13 on page 143 shows the interface for working with the abstract portlets.

Archived

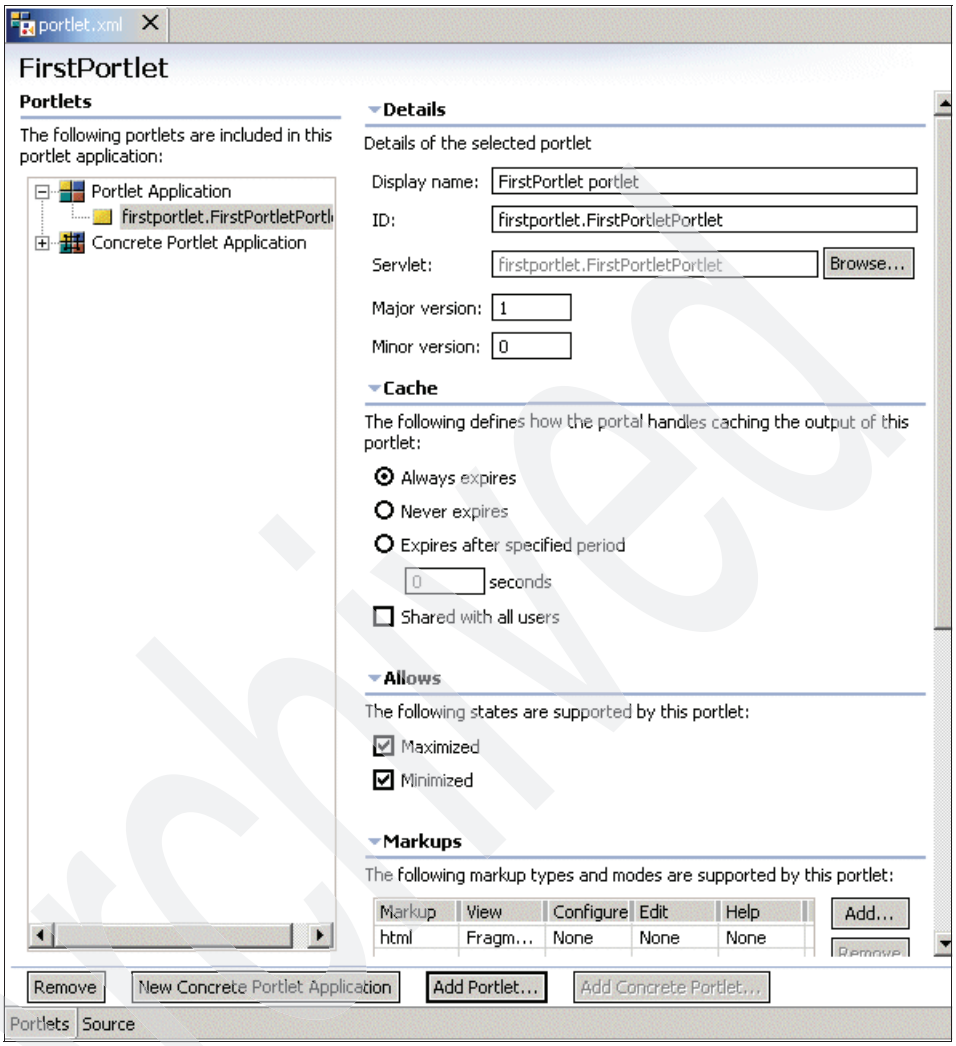


Figure 3-13 Working with abstract portlets

The support section allows you to add or remove predefined markup languages. If you need to add custom markup languages, they must be manually entered into the portlet.xml via the Source tab.

The configuration parameters (not shown in the figure will actually enter servlet-init parameter definitions into the corresponding servlet in the web.xml.

Each portlet.xml may only define a single abstract portlet application. However, since it may contain any number of concrete applications, you can choose to add new concrete portlet applications based on the abstract application. If you click

Archived

the **New Concrete Portlet Application** button, a new, empty application will be created. Although the ID and so forth will be created for you, you must add portlets.

The Concrete Portlet Application interface provides the opportunity to set the context parameters of the application. The Concrete Portlet Applications window is shown in Figure 3-14. You are also free to change the UID if you like, but keep in mind that it must be unique throughout the entire portal environment.

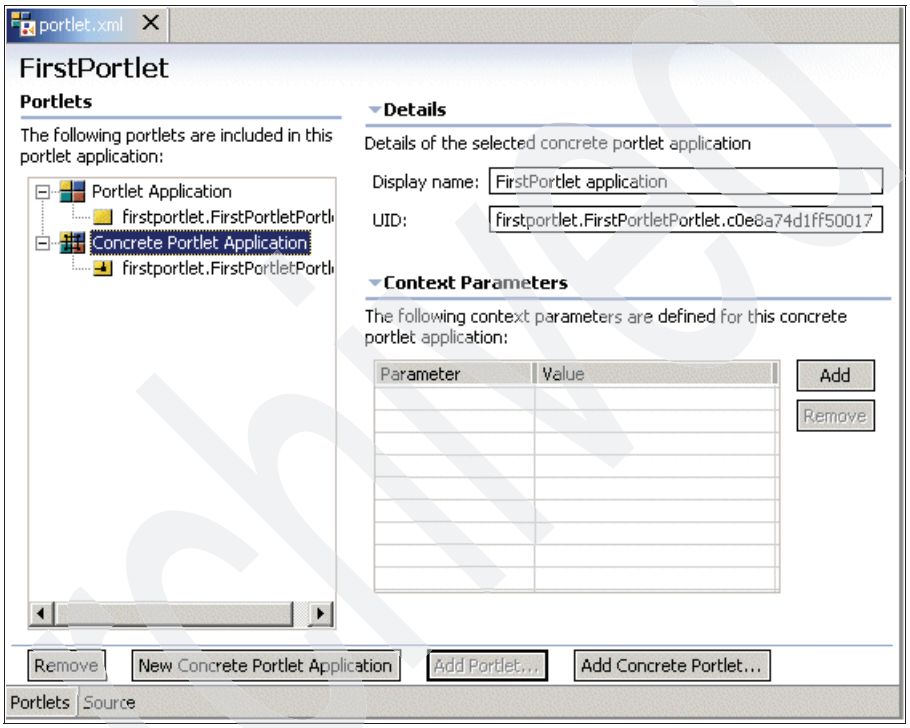


Figure 3-14 Working with Concrete Portlet Applications

The final window allows you to work with the actual concrete portlets that will eventually be seen by end users. This window allows you to set the title that will be seen in the title bar of the portlet. You can choose to set the contents for language blocks by selecting **Add**, choosing the locale and completing the title field. You must define at least one locale. You may also choose to complete the keywords, description and short title fields. Figure 3-15 on page 145 displays the interface for this final window.

The configuration parameters can be set for the portlet using settings in the text area.

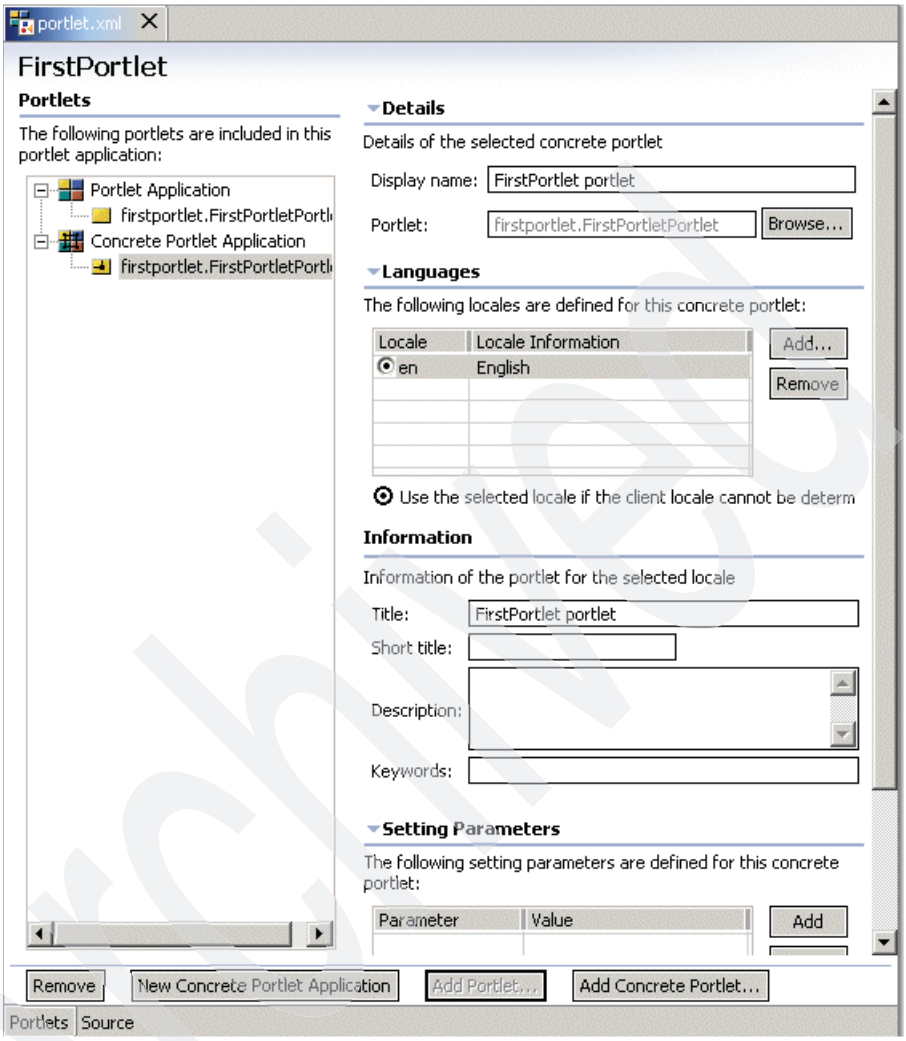


Figure 3-15 Working with concrete portlets

At any time, you can choose to manually edit the source file by selecting the **Source** tab at the bottom of the interface. When you have finished working with the portlet.xml, press **CTRL-S** to save the file.

Tip: When a file has unsaved changes, an asterisk will appear beside the name in the Title tab. Be sure to check for unsaved changes in other perspectives as well.

Archive

3.7 Deploying portlets

Once you have created, edited and configured your portlet application, you must deploy the portlet application. To deploy your portlet application in WebSphere Studio Site Developer, you have to configure a server and server instance and choose the option **Run on Server**.

Configuring a server and server instance

1. Select **File -> New -> Other**. Then select **Server** in the left frame and **Server and Server Configuration** in the right. Click **Next**.

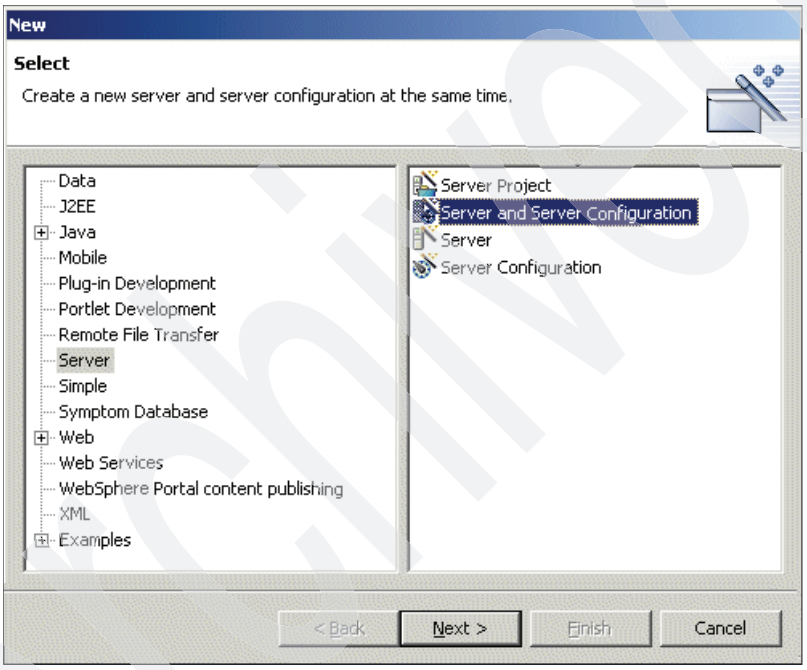


Figure 3-16 Create a New Server and Server Configuration.

2. In the Create a New Server and Server Configuration window, enter the following values and click **Next**:
   a. Server name: `WebSphere Portal`
   b. Folder: `Servers`
   c. For the Server type, select **WebSphere Portal version 5.0 -> Test Environment**.

Archived

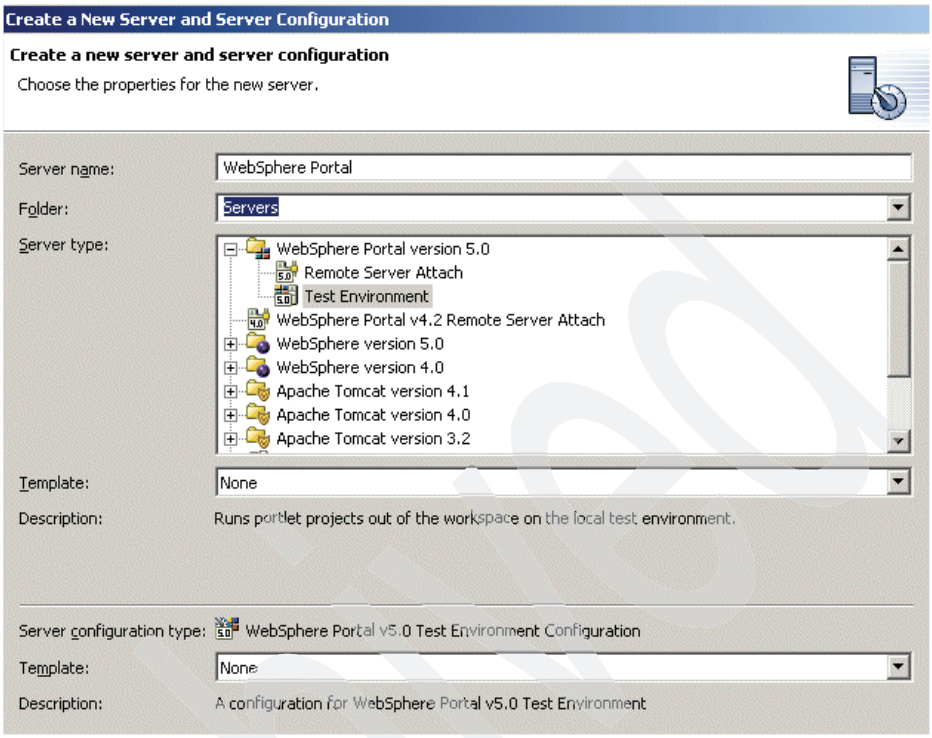


Figure 3-17 Values for a New Server and Server Configuration

3. In the next window, enter the HTTP port number. Click **Finish**.

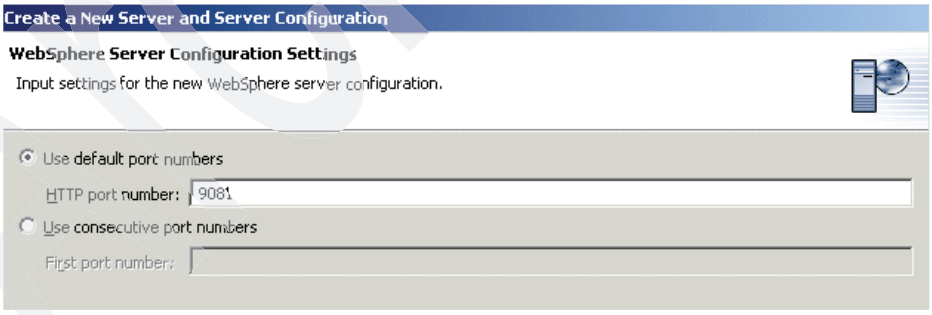


Figure 3-18 Default port number

A directory called Servers is created in the Server Configuration panel and a directory called WebSphere Portal is created under Servers.

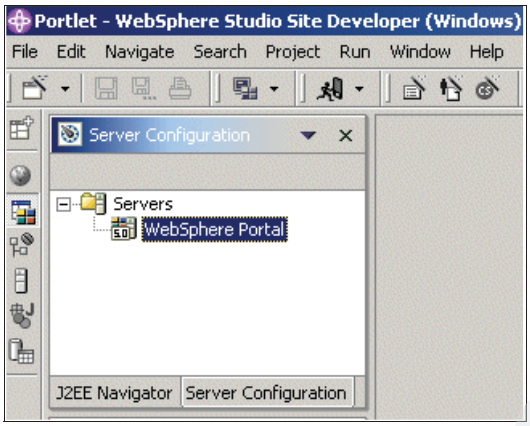


Figure 3-19 Server Configuration panel

Running the application

To publish the portlet in the target server, the EAR containing the WAR must be associated with the Server. In the Server Configuration panel, right-click the server name created and select **Add -> DefaultEAR** as shown in Figure 3-20.

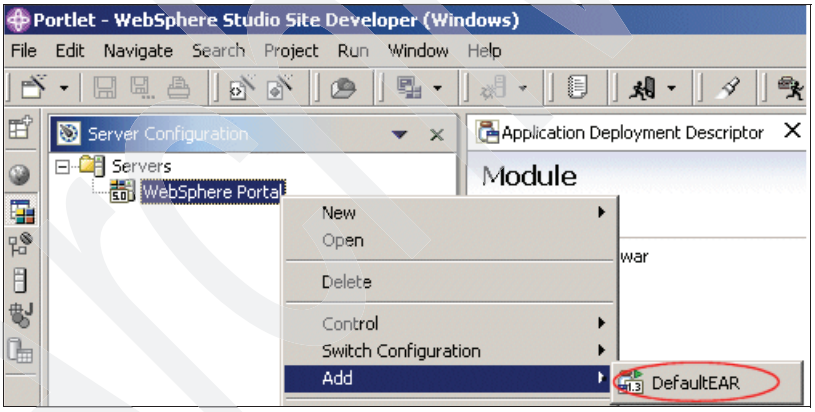


Figure 3-20 Associating EAR with a server

Once the project DefaultEAR has been added to the server configuration, you can switch back to the portlet perspective and select the portlet application you want to test. Right-click and from the context menu select **Run On Server**. The project will be published and the server will be started.

Once the server has been published and started, an internal Web browser is used to call the Portal. Notice that you do not need to install the portlet or place it on a page. This is done for you, as shown in Figure 3-21 on page 149

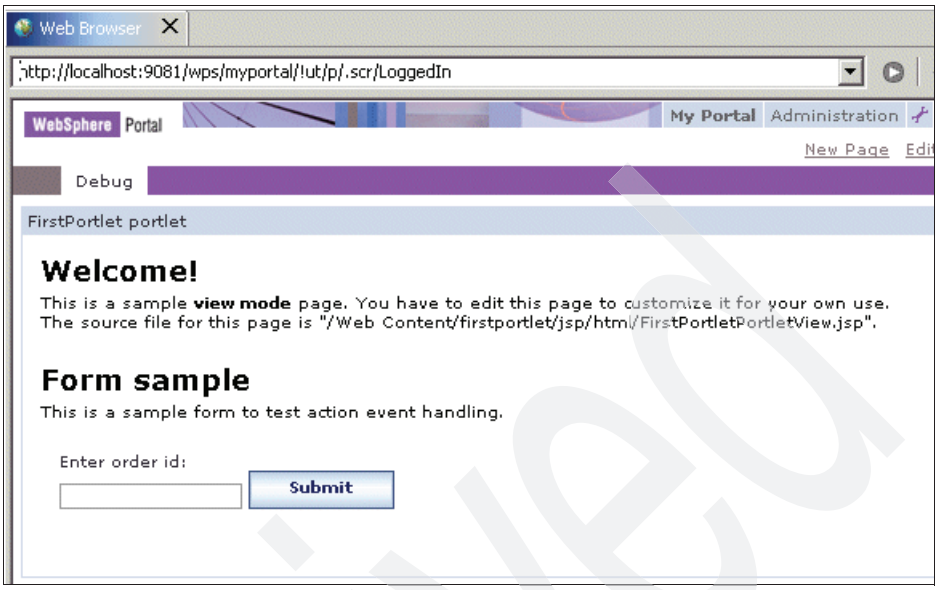


Figure 3-21 Deployed portlet

3.8 Adding portlets to applications

Normally, your application will need to contain more than one portlet. To add more portlets, select the portlet application where you want to add the new portlet, right-click and select **New -> Other**. You will see the window shown in Figure 3-22 on page 150; select **Portlet Development** in the left panel and **Portlet** in the right.

Archived

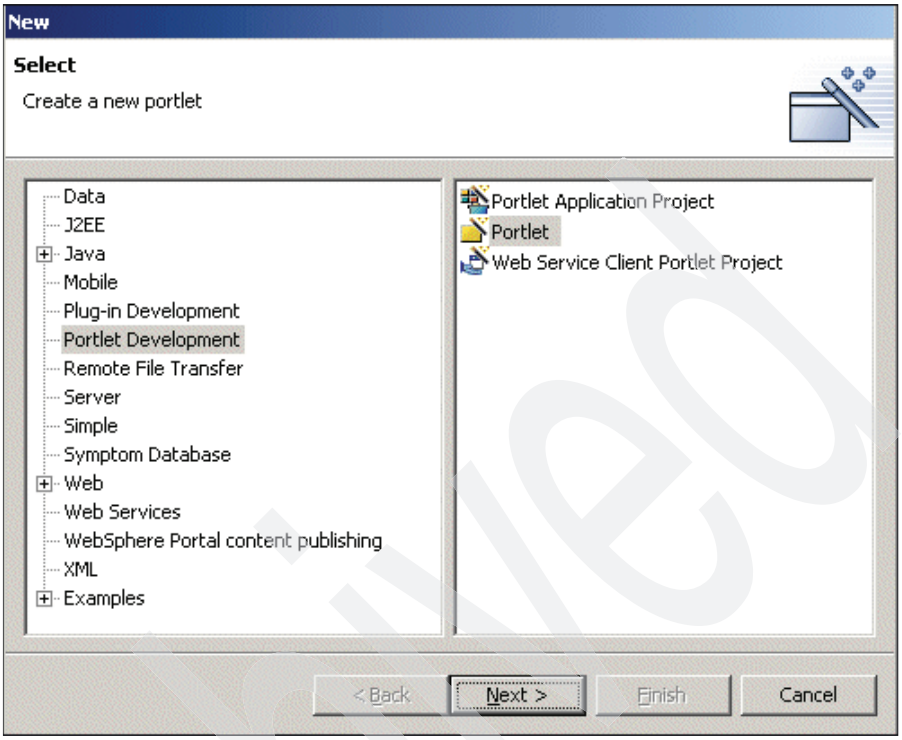


Figure 3-22 Creating a new portlet

Select **Next** and you will begin to create a new portlet; the process is the same as when you created a new portlet application project. Now you can see the new portlet name and the options which have been added to the portlet.xml and web.xml descriptor files.

3.9 Examples

WebSphere Studio Site Developer provides some examples of portlet applications. In this section, we will explain how to add an example of Click-to-Action to your workspace and deploy it.

Open WebSphere Studio Site Developer and select **File -> New -> Other**; in the left panel, select **Examples -> Portlet Applications** and in the right panel select **Click-to-Action Shipping demo** (Cooperative Portlets).

Archived

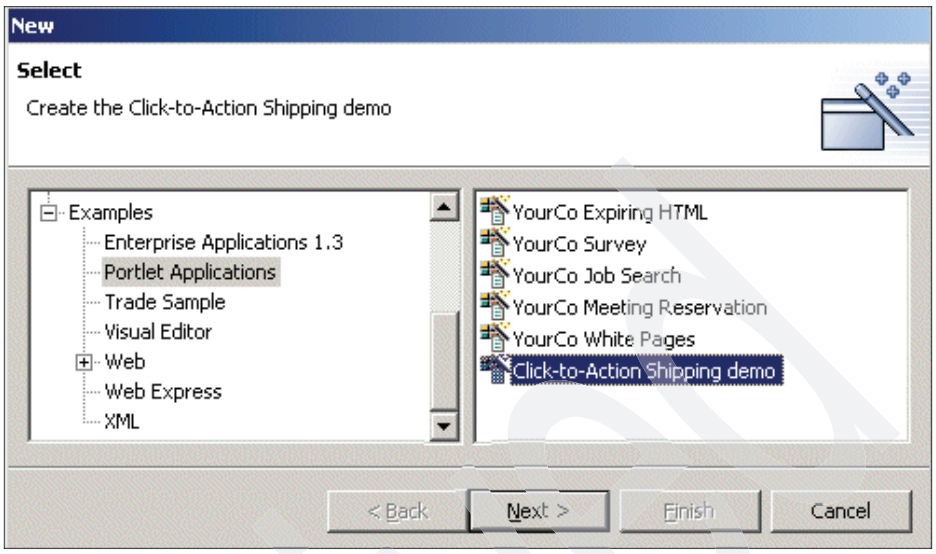


Figure 3-23 Creating a Click-to-Action demo

Select **Next** and you will see the options to define the portlet project.

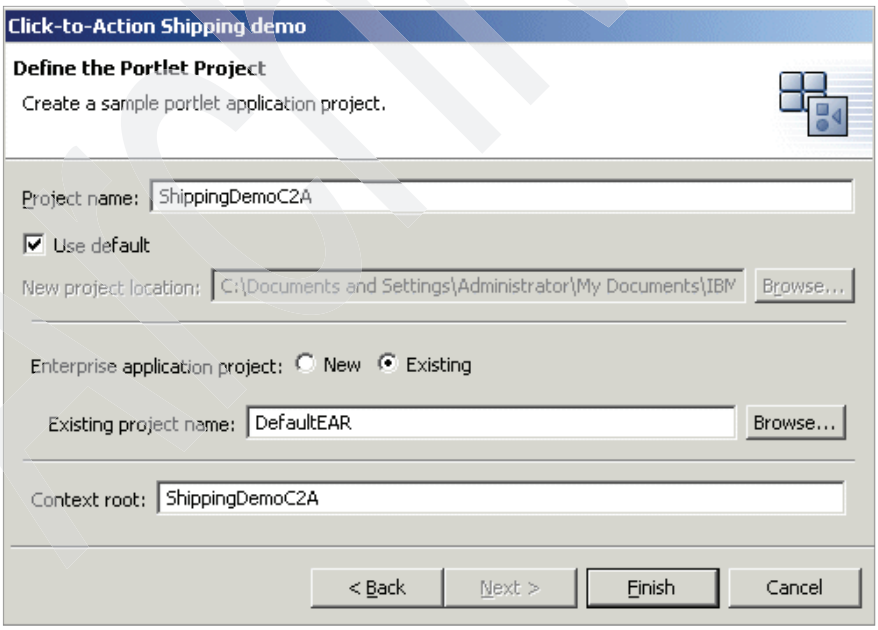


Figure 3-24 Defining the Portlet Project

When you click **Finish**, the project will be created and added to your workspace. If the repair server configuration window is prompted, click **OK**.

The new project has added to the J2EE navigator window. Notice the directory structure, which is the same as we described in Figure 3-10 on page 137. Now there is a new folder called *nls* which contains properties files to support different languages. In the example, only *english* and *default properties* files have been created.

If there are more projects in the application.xml file of the DefaultEAR, select the **Module** tab and remove them, except for the Click-to-Action war file. Save the file. Right-click the project and select **Run on Server** to test the application.

Note: If you experience an error publishing to the server, go to Server Configuration and open the server by double-clicking. Click **Yes** in the window where you are prompted to update the entries of the server configuration and save the file.

Archive

4

A first portlet application

This chapter provides a sample scenario for creating and testing the simplest example of a portlet project, the Hello World example. After running this portlet, you will modify it using JSP expressions and a JavaBean and, lastly, verify your changes. These activities will allow you to understand the techniques used to develop portlet projects.

Archived

© Copyright IBM Corp. 2004. All rights reserved.

4.1 Sample scenario

In this sample scenario, you will complete the following tasks:

1. Create and run a portlet project using the basic portlet type to become familiar with how portlets work
2. Modify the portlet to use JSP expressions and verify your changes
3. Add a JavaBean to your project and verify the changes

The development workstation has already been created for you and its components can be seen in Figure 4-1.

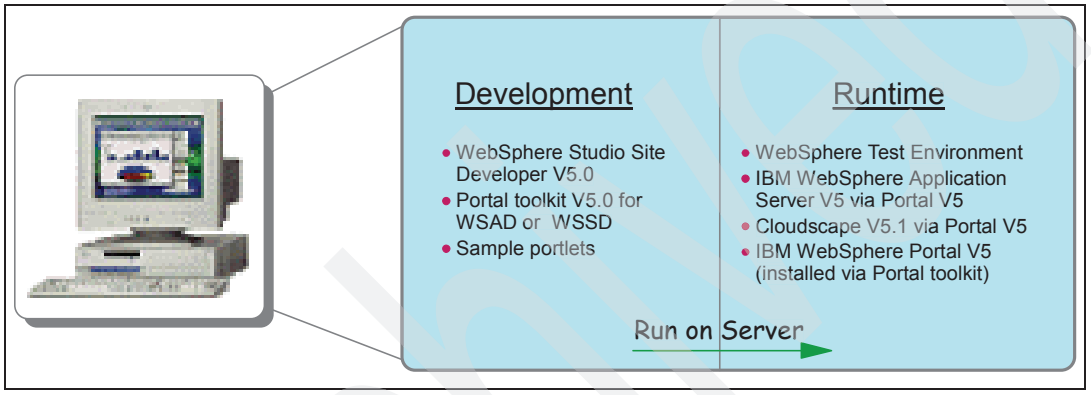


Figure 4-1 Development workstation

4.1.1 Creating a portlet project

You will start by creating a portlet application using WebSphere Studio Site Developer. This sample portlet is based on the Model-View-Controller (MVC) design pattern. The MVC methodology allows efficient relationships between the user interface and the underlying data model. The main components of this design pattern are:

- Model - represents the logical structure of data in a software application
- View - represents all elements in the user interface
- Controller - represents the elements connecting the Model and View elements

Figure 4-2 on page 155 illustrates the MVC portlet application. The portlet application also includes a Configure.jsp, but it is not shown here.

Note: In this initial project, the Edit mode will not be implemented and therefore Edit.jsp is not required.

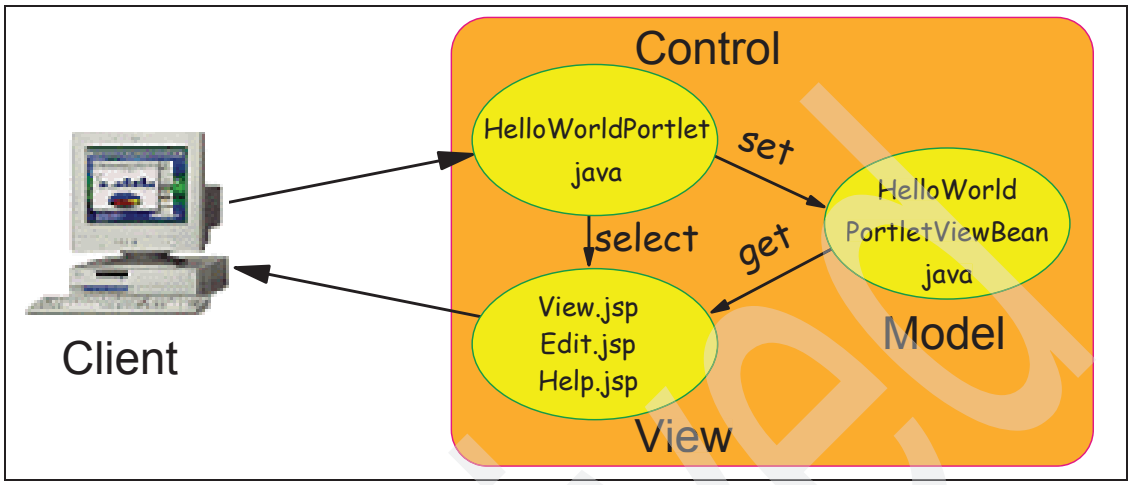


Figure 4-2 HelloWorld portlet application

You will start by creating a portlet application using WebSphere Studio Site Developer. This sample portlet uses the basic portlet type.

In this section, you will create a portlet application and then test the application in the WebSphere Studio Site Developer Test Environment. The sample portlet HelloWorld displays a `Hello World` message and a message indicating the portlet current mode. Portlet modes allow a portlet to display different content to the user, depending on the task required by the portlet. The WebSphere Portal API provides the modes View, Help, Edit and Configure.

Creating the portlet project

Execute the following steps to create the portlet application:

1. Open the IBM WebSphere Studio Site Developer by clicking **Start -> Programs -> IBM WebSphere Studio -> Site Developer 5.0**.

   Note: If prompted, click **OK** to use the default workspace directory.

2. Select **File -> New -> Project**.

Archived

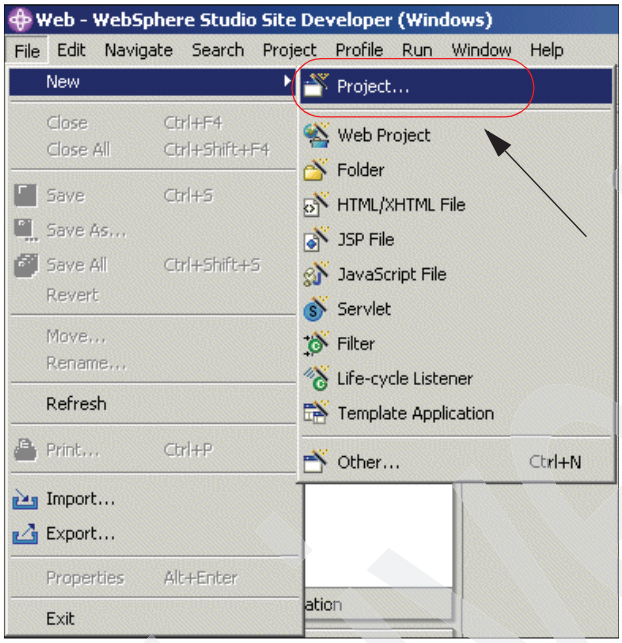


Figure 4-3 Creating a new project

3. Select **Portlet Development** from the left panel and **Portlet Application Project** from the right panel. Then click **Next**.

Archived

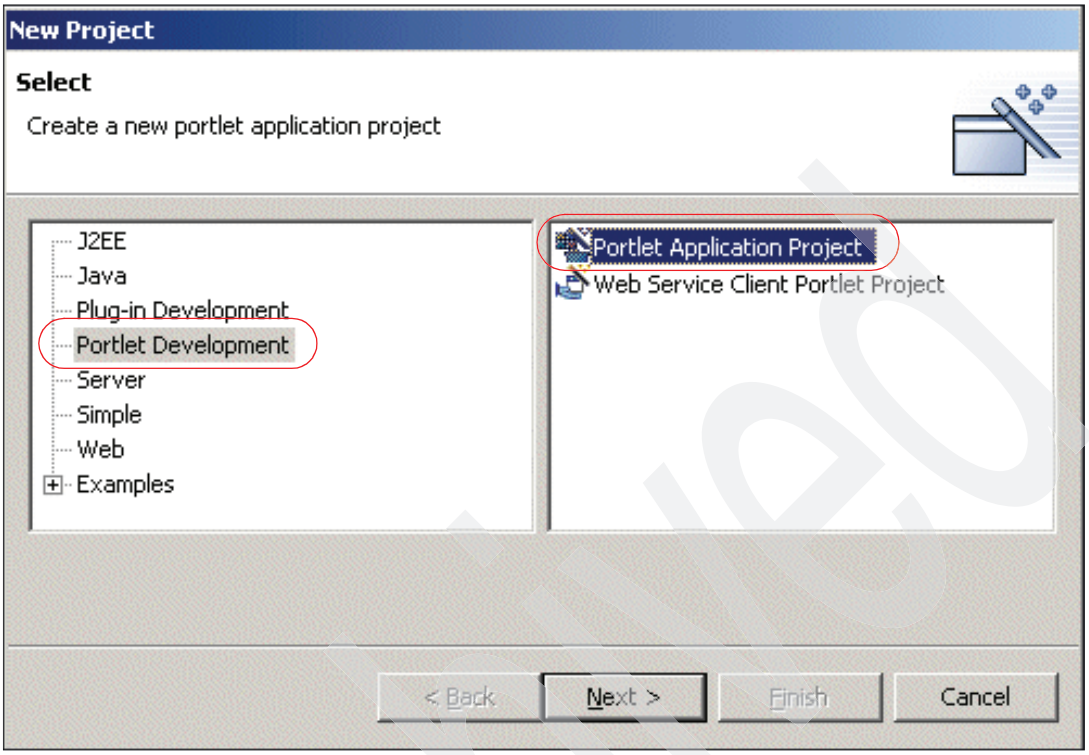


Figure 4-4 Selecting Portlet Application Project

4. In the Define the Portlet Project window, enter `HelloWorld` as the Project Name. You will be creating a *basic* type portlet which extends PortletAdapter in the Portlet API. Click **Next** to continue.

Archive

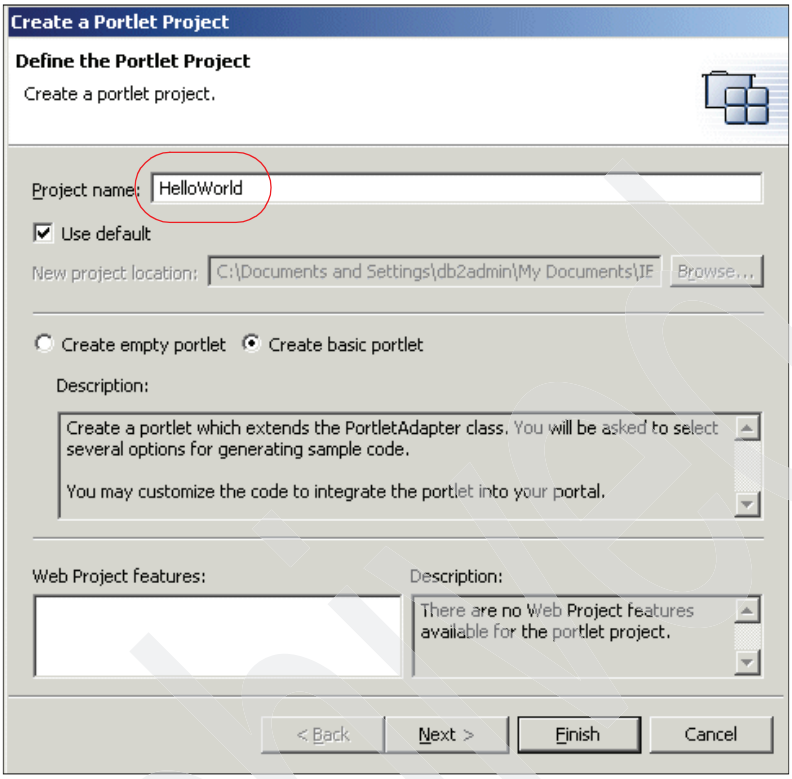


Figure 4-5 Defining the portlet project

5. On the J2EE Settings Page, accept the default values. This will create a portlet project compatible with WebSphere Portal V5. Click **Next**.

Archive

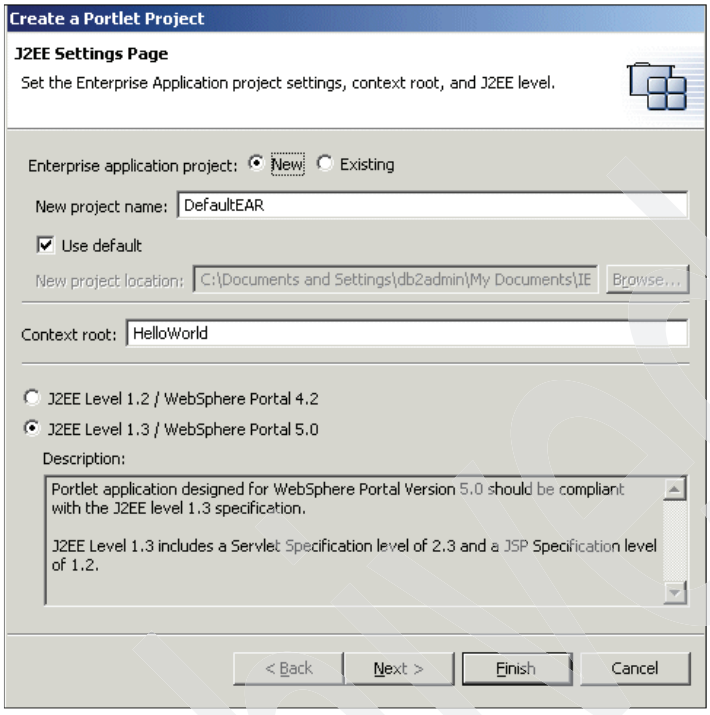


Figure 4-6 Creating a portlet project for WebSphere Portal V5

6. Accept the default values on the Portlet Settings Page. Click **Next**.

Archived

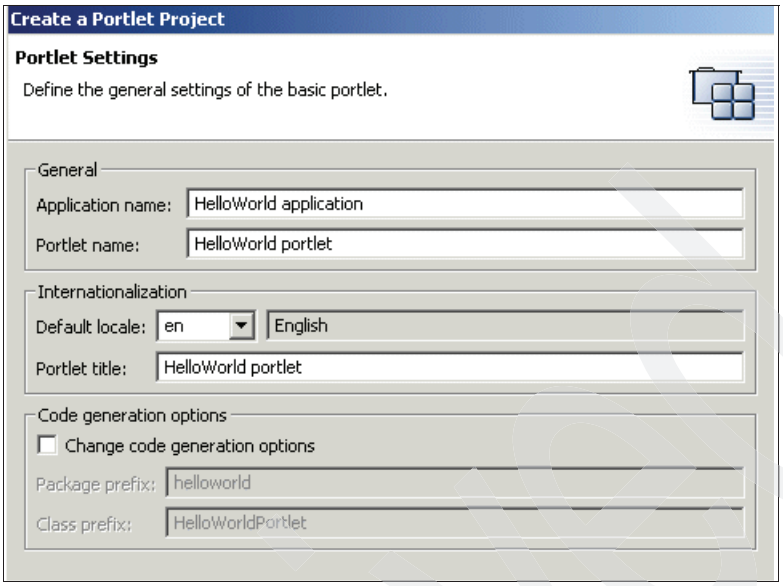


Figure 4-7 Portlet settings window

7. Deselect the **Add form sample** and **Add actionlistener** check boxes. The Hello World example does not need these. You will learn more about the action listener in Chapter 5, “Action event handling” on page 181. Click **Next**.

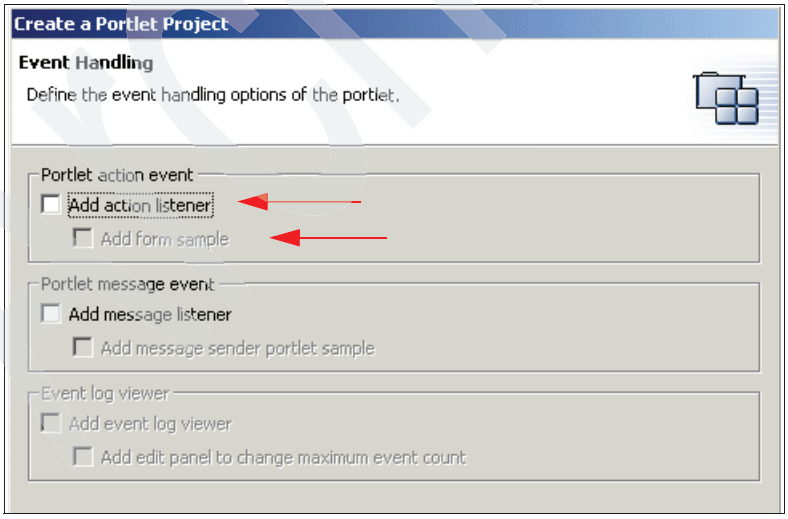


Figure 4-8 Event handling

8. Accept the default values for the Single Sign-On (SSO) page. The Hello World application will not use this feature. You will learn more about the Credential Vault in Chapter 10, “Using the Credential Vault” on page 319. Click **Next**.

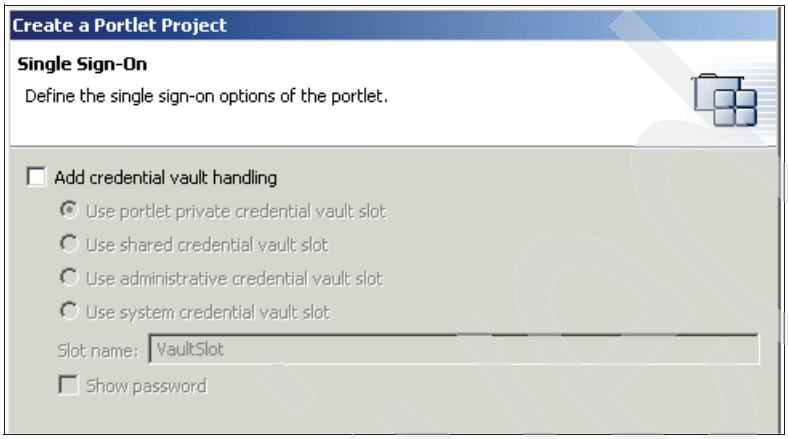


Figure 4-9 Single Sign-On window

9. Select the check boxes to **Add help mode** and **Add configure mode** to your portlet project (see Figure 4-10). We will show you how to navigate to these modes in this exercise. Click **Finish** to generate your project.

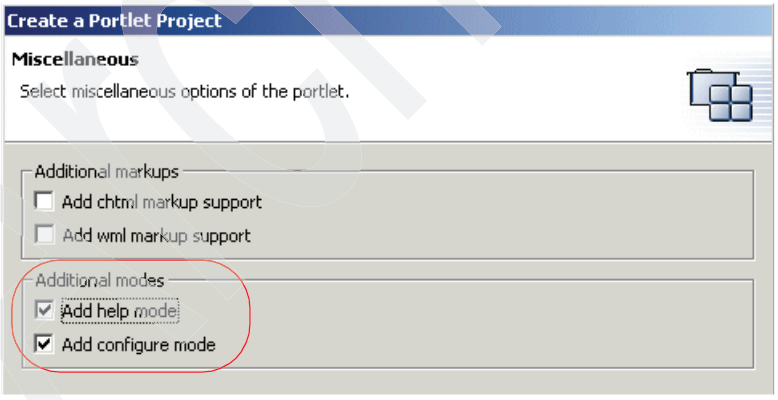


Figure 4-10 Additional markup support and additional portlet modes

Note: Configure mode is for administrators only.

10. You will now see the HelloWorld project listed in the J2EE Navigator panel. This panel is located in the upper left hand of the WebSphere Studio Site Developer workbench.

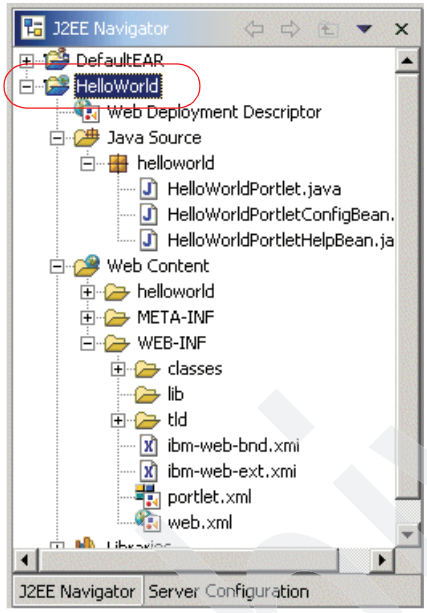


Figure 4-11 The Navigator panel

11. Expand the /Java Source/helloworld/ folder to view the Java files used in this project. Do the same to the /Web Content/helloworld/jsp/html/ folder to view the JSPs used to render the content of your project.

Archived

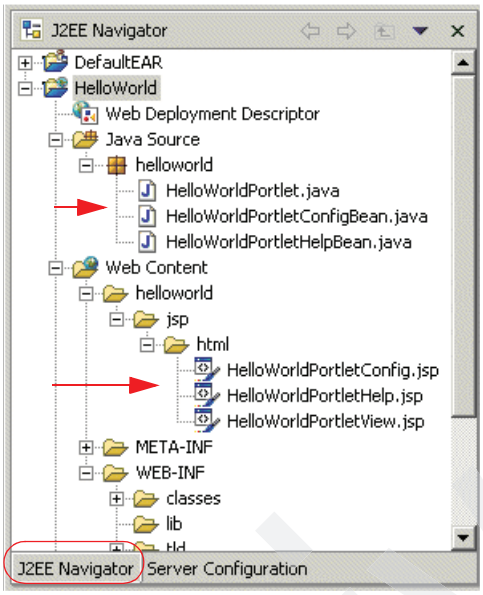


Figure 4-12 Java and JSP files

4.1.2 Configuring the Test Environment

Initially, you will need to configure the WebSphere Portal V5.0 Test Environment. If it is not already configured, follow these steps:

1. Click the **Server Configuration** tab.

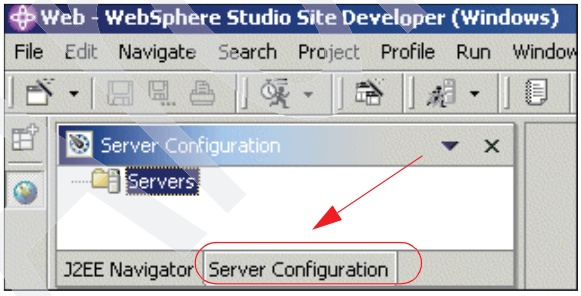


Figure 4-13 Server Configuration tab

2. Right-click **Servers**, then click **New**, then **Server and Server Configuration**.

Archived

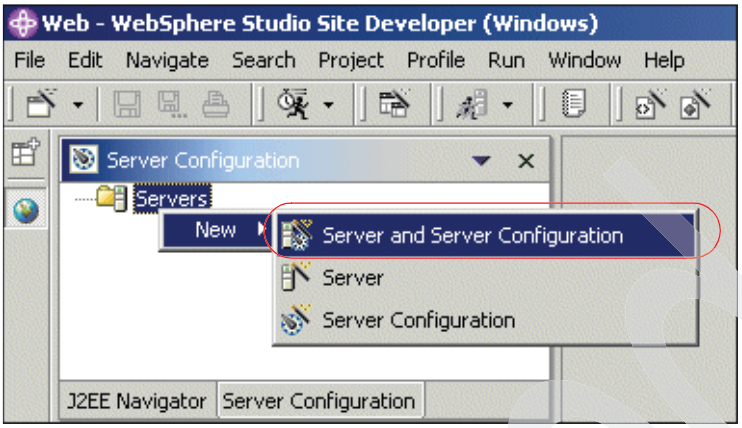


Figure 4-14 Creating a new server and configuration

3. A new window will be displayed to allow you to configure the new server. Enter `Test Environment` as the server name. Select **Test Environment** under WebSphere Portal V5.0 as the server type. Click **Next** to continue.

   Note: You must enter a server name to continue. Also, be sure to select **Test Environment**.

Archived

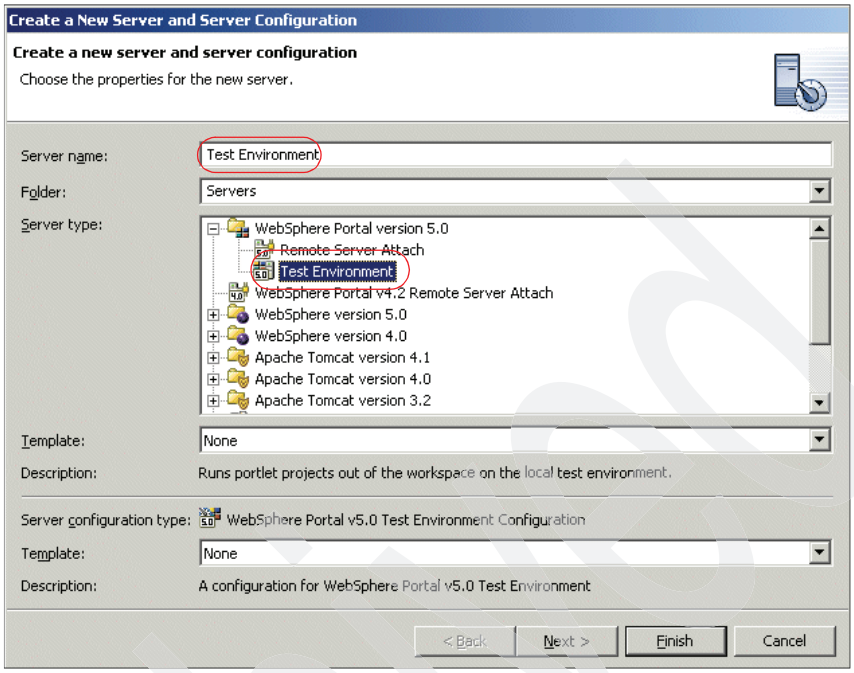


Figure 4-15 Server configuration settings

4. This will bring up a window to select the *HTTP port number* to be used by Site Developer. Use port 9080. Click **Finish** to add the Test Environment.

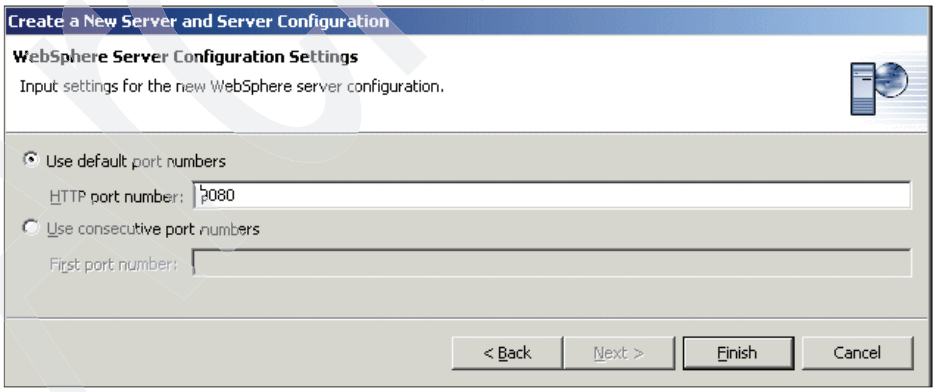


Figure 4-16 HTTP port selection

5. The server has been successfully added and it can now be seen in the Server Configuration tab.

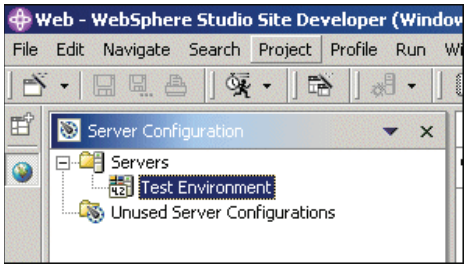


Figure 4-17 Test Environment has been added successfully

4.1.3 Running the portlet application

To run a project in the WebSphere Studio Site Developer Test Environment, you will need to add the portlet project to the Test Environment.

1. Add your portlet project to the Test Environment:
   a. Click the **Server Configuration tab** (on the navigator panel).
   b. Expand the Servers tree.
   c. Right-click **WebSphere Portal V5.0 Test Environment** (or **Test Environment**).
   d. Click **Add -> DefaultEAR** to add your project to the Test Environment.

Archived

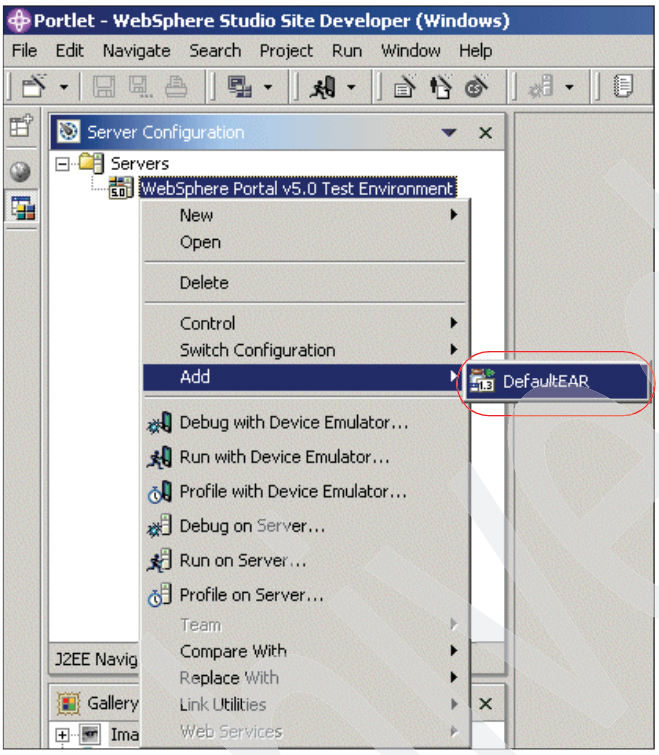


Figure 4-18 Adding a project to the Test Environment

2. Expand the Servers tree. You will see the HelloWorld project under the DefaultEAR project in the Test Environment.

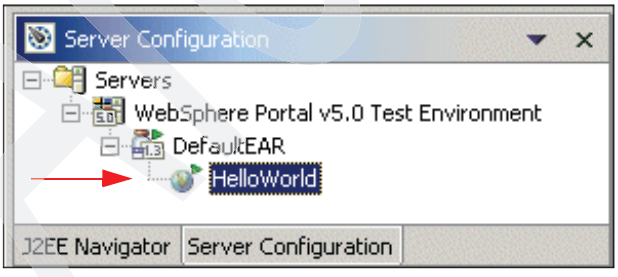


Figure 4-19 Project shown in the Servers tree

3. Now click the **J2EE Navigator** tab to see your project again. Right-click **HelloWorld**. Then click **Run on Server**. This will load your project into the Test Environment so that you can view it in the WebSphere Studio Site

Developer Web browser. It may take a minute or two for this process to complete.

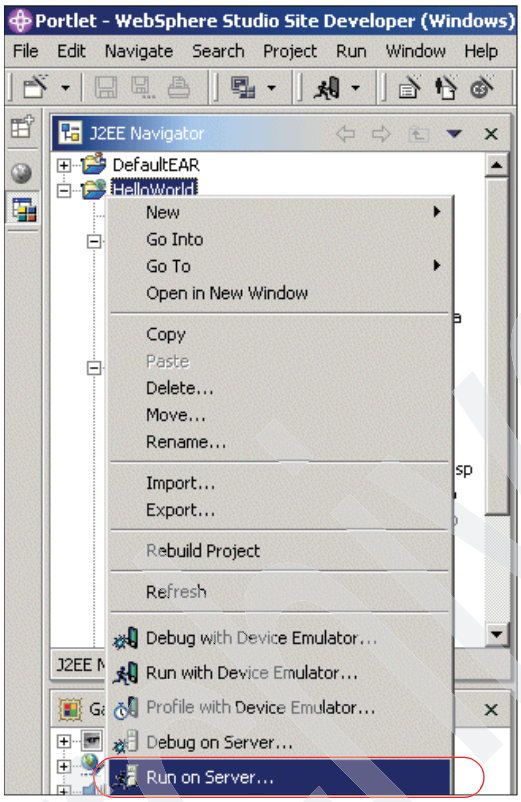


Figure 4-20 Running the project in the Test Environment

4. You will now see your newly created portlet project running in the Web browser.

Archived

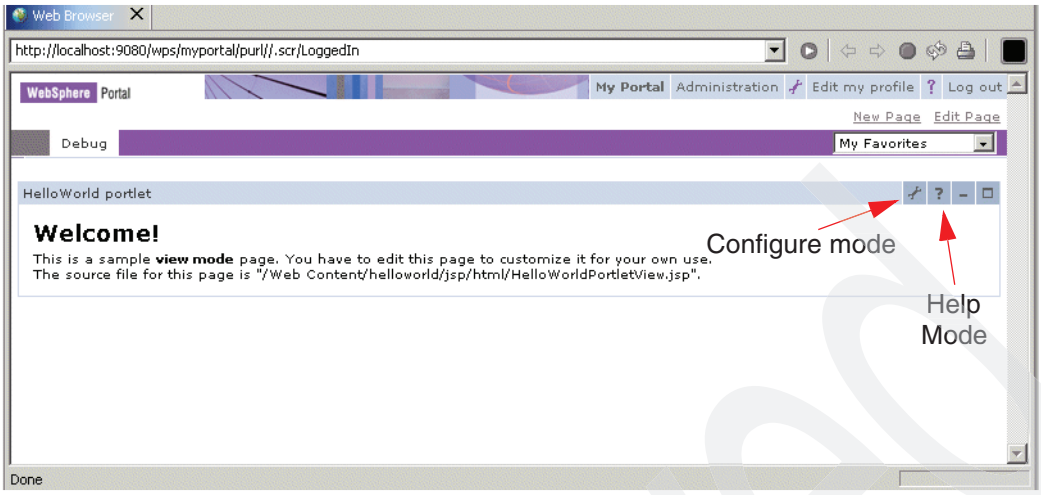


Figure 4-21 Viewing the portlet project

5. Click the wrench icon in the title bar of the HelloWorld portlet. This will take you to the Configure mode of the portlet (administration).

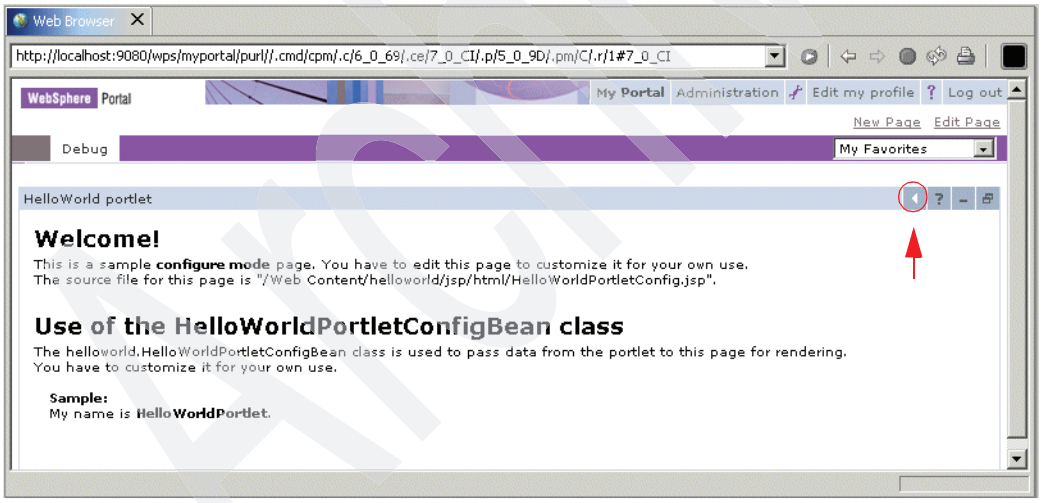


Figure 4-22 Configure mode

6. Similarly, clicking the **?** icon will take you to the Help mode. The Help mode window will be quickly displayed as a pop-up window and you will not be able to see it. However, you can open an external browser instance like Internet Explorer and then point it to the same Portal URL.

a. In IE, enter `http://localhost:9080/wps/portal`.

b. Log on to the portal with user ID wpsadmin and password wpsadmin.

c. Click the **?** icon to see the help page.

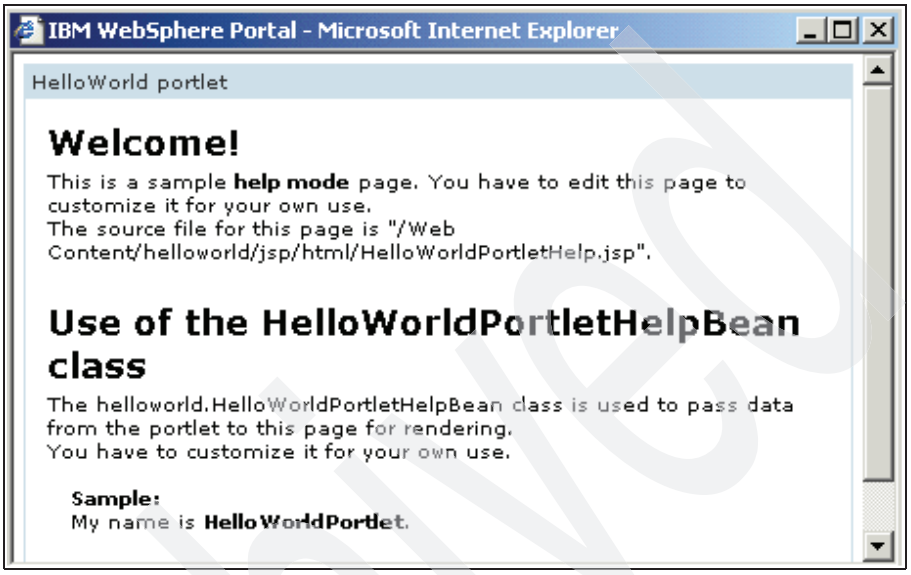


Figure 4-23 Portlet help mode page

7. Click the small white triangle highlighted to return to the View mode. See Figure 4-22 on page 169.
8. Click the minimize button (-). This will change the state of your portlet to minimized. When a portlet is minimized, the content of the portlet is not rendered, and only the title bar is shown.
9. Now click the maximize icon (rectangle). This will set the state of your portlet to maximized and the entire screen will be filled with one portlet.

Archived

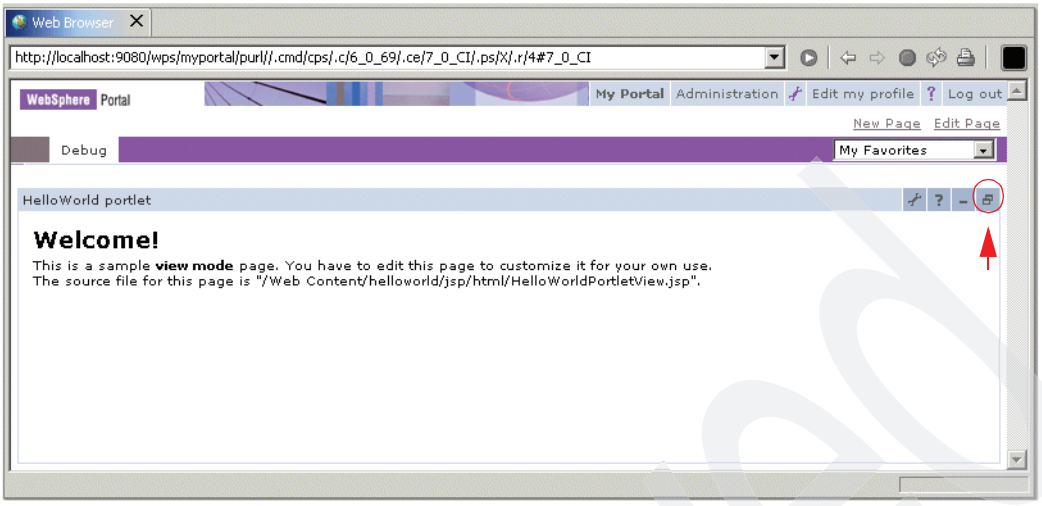


Figure 4-24 Maximized state

10. Now click the restore icon in the title bar as highlighted in Figure 4-24. This will return your portlet to its normal viewing state.

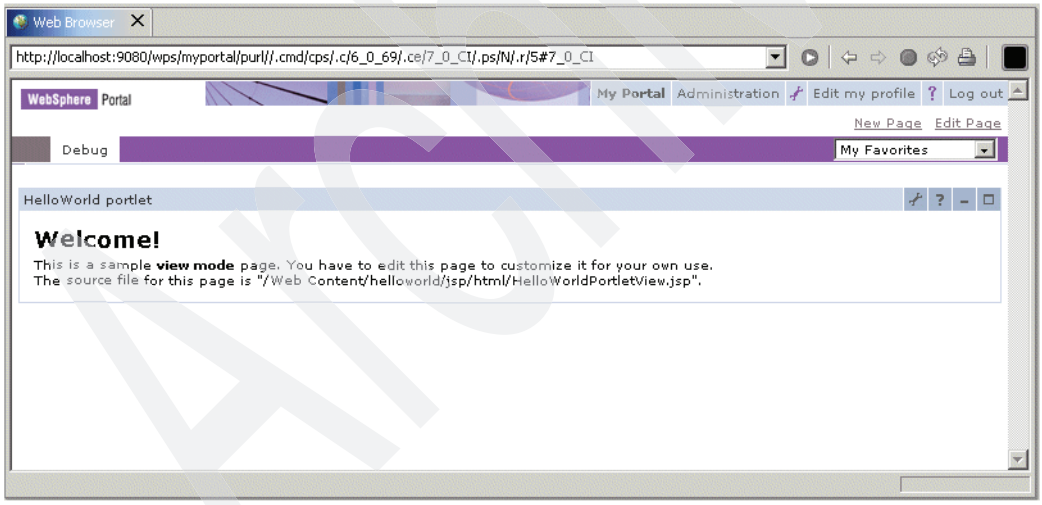


Figure 4-25 Normal viewing state

4.1.4 Updating the portlet project

In this section, you will use JSP expressions to add content to the View mode of your portlet.

Archived

1. Double-click **HelloWorldPortletView.jsp** in the /Web Content/helloworld/jsp/html/ directory of your portlet project to open it for editing.

   Note: Make sure you edit the JSP in the /html/ directory.

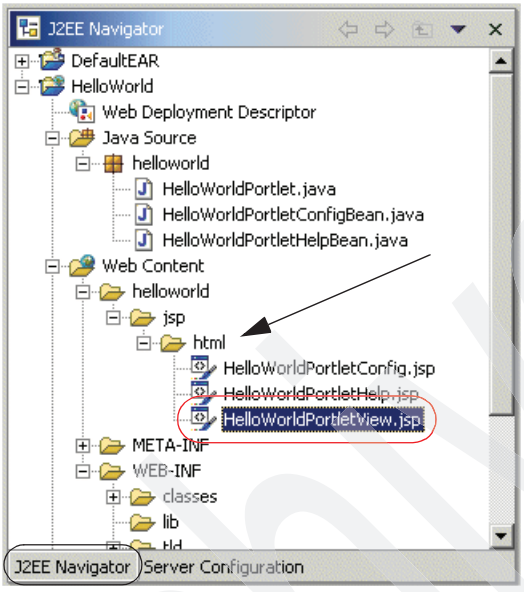


Figure 4-26 Editing the view JSP

2. You will see the code for this page in the integrated JSP editor.

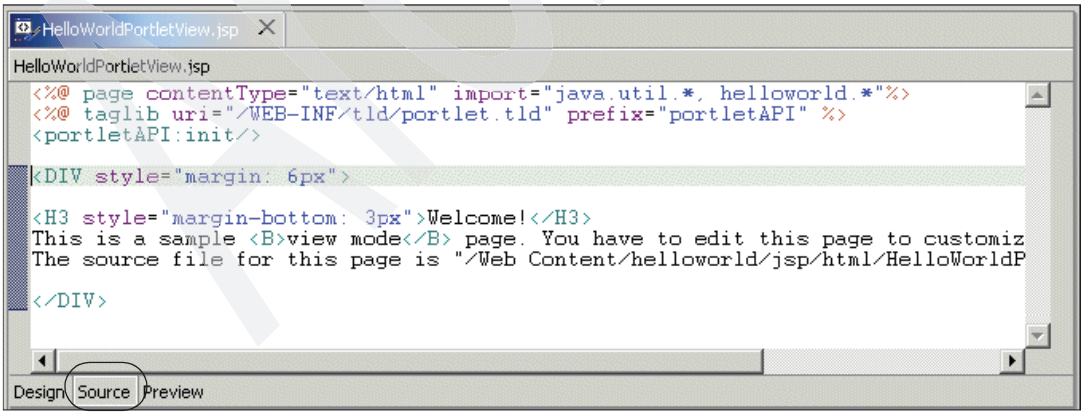


Figure 4-27 HelloWorldPortletView.jsp code

3. Modify the code according to the following example. Add the text shown in bold.

Example 4-1 HelloWorldPortletView.jsp

```
...
<DIV style="margin: 6px">

<H3 style="margin-bottom: 3px">Welcome!</H3>
This is a sample <B>view mode</B> page. You have to edit this page to customize
it for your own use.<BR>
The source file for this page is "/Web
Content/helloworld/jsp/html/HelloWorldPortletView.jsp".

<br>
Current time: <%=new java.util.Date() %>
<br>
Hostname: <%= request.getRemoteHost()%>

</DIV>
```

4. Save all your changes. For example click **File -> Save All.**
5. Right-click the HelloWorld project in the navigator panel and click **Run on Server** to view your changes.

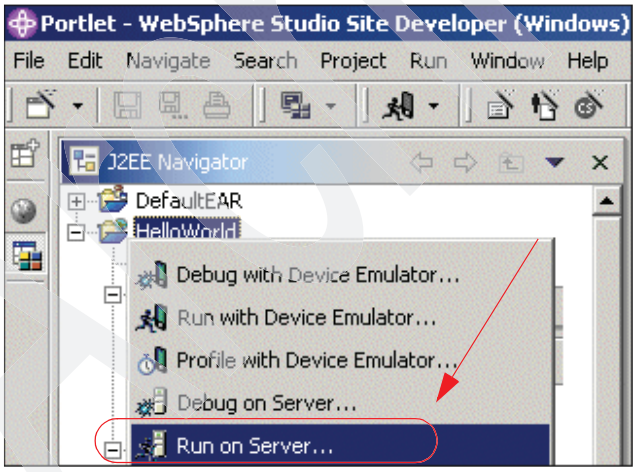


Figure 4-28 Running the updated project on the server

6. You will see your changes in the integrated Web browser.

Archived

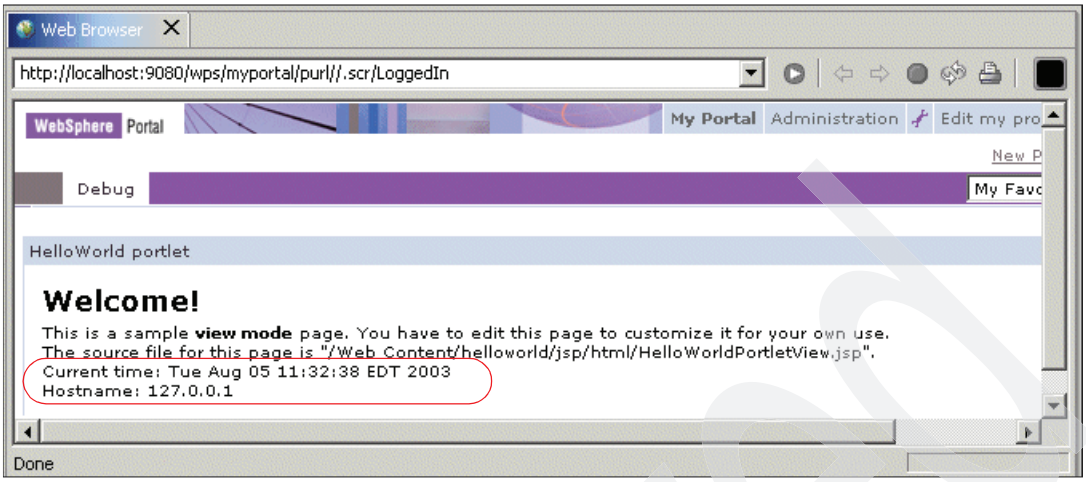


Figure 4-29 Viewing your updated portlet results

4.1.5 Adding a JavaBean to your portlet project

Another way to store information to be accessed and displayed by the View mode JSP is to use a JavaBean. In this exercise, you will add a JavaBean to your project and use it to display information when it is run. JavaBeans are a special type of Java class that contain the business logic of the application. They are used to temporarily store and process data and access back-end resources such as databases.

1. The Java files used to invoke the JSPs to render content are located in the /Java Source/helloworld/ folder of your portlet project.

Archived

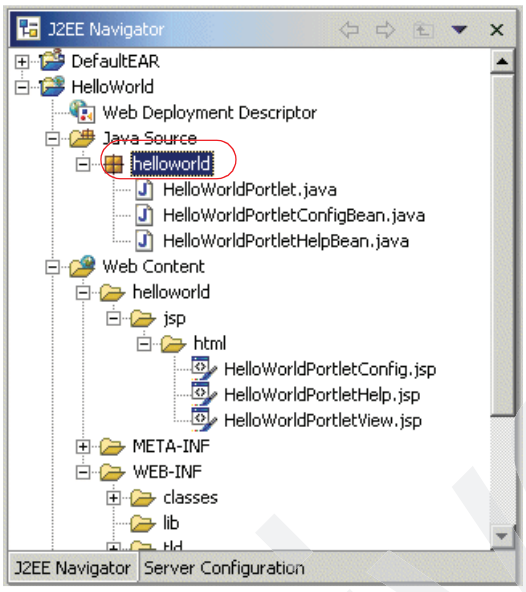


Figure 4-30 /Java Source/helloworld directory

2. To add a JavaBean, you will add a class to this project. There is an easy-to-use wizard in WebSphere Studio Site Developer for doing this. To start the wizard, right-click the **/Java Source/helloworld/** folder and click **New -> Class**.

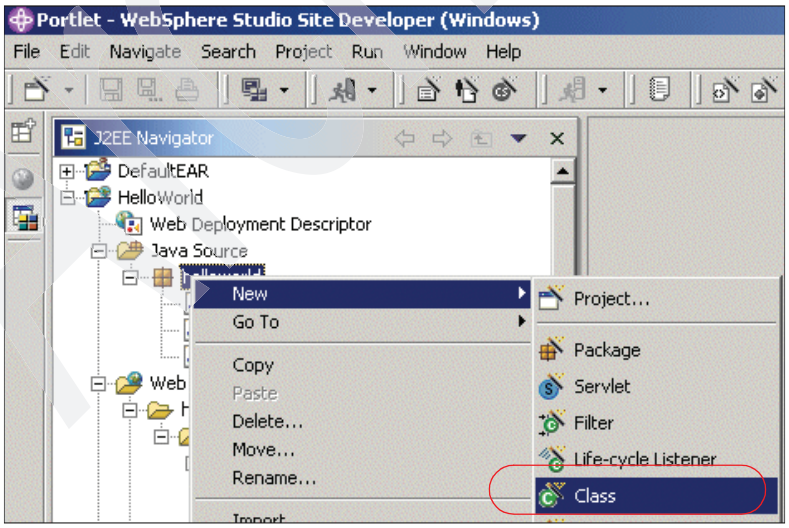


Figure 4-31 Adding a new class to your project

Archived

3. Give this class the name `HelloWorldPortletViewBean`. Click **Finish** to add this file to your project.

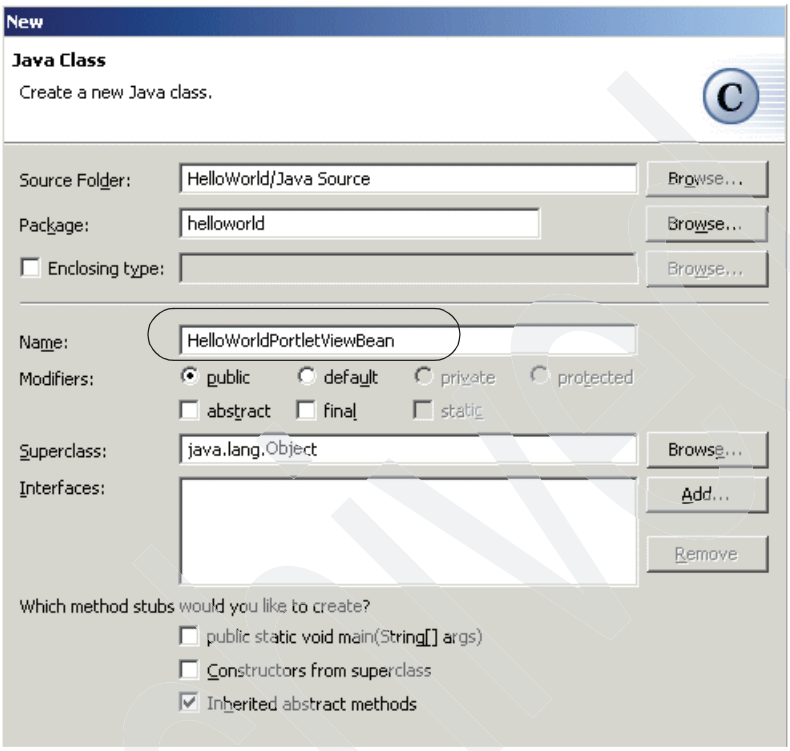


Figure 4-32 Adding a class wizard

4. The HelloWorldPortletViewBean.java file will now appear under the /Java Source/helloworld/ folder.

Archive

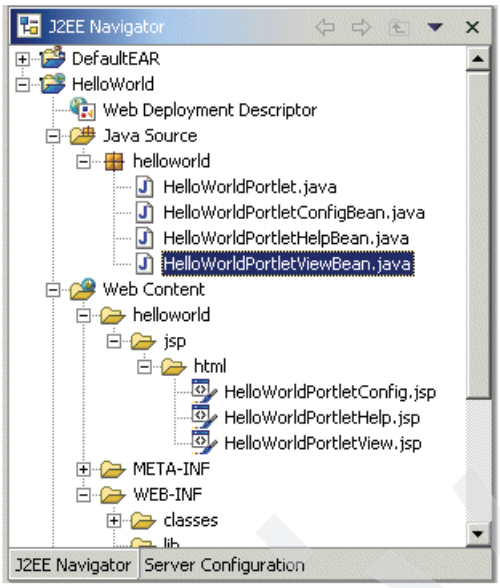


Figure 4-33 The newly added file is shown

5. Double-click the **HelloWorldPortletViewBean.java** file for editing. Modify its code according to the following example. The addition of a *set* and *get* method allows you to store and retrieve a value from this bean.

Example 4-2 HelloWorldPortletViewBean.java

```
public class HelloWorldPortletViewBean {

   private String myName = "";

   public void setMyName (String s){
      myName = s;
   }

   public String getMyName (){
      return myName;
   }

}
```

6. Next, you will need to modify the HelloWorldPortlet.java file.You will add code that will use the set information in the bean you just created. This code is inserted in the doView() method of the HelloWorldPortlet.java file.

Archived

Example 4-3 HelloWorldPortlet.java

```
public class HelloWorldPortlet extends PortletAdapter {

   ....
   ....
   public void doView(PortletRequest request, PortletResponse response) throws
   PortletException, IOException {

      //Make a bean
      HelloWorldPortletViewBean viewBean = new HelloWorldPortletViewBean();

      //Set your name
      viewBean.setMyName("John Doe");

      //Save bean in request so the view jsp can read it
      request.setAttribute("HelloWorldPortletViewBean", viewBean);

        // Invoke the JSP to render

getPortletConfig().getContext().include(VIEW_JSP+getJspExtension(request),
request,
      response);
   }
```

7. Now that the bean is created and the portlet can successfully store a value in this bean, it is necessary to modify the code to the HelloWorldPortletView.jsp file so that the value can be extracted from the bean and shown on the screen. Double-click the **HelloWorldPortletView.jsp** file and make the following changes. The *useBean* tag tells the JSP file that it will be accessing values stored in a JavaBean.

Example 4-4 HelloWorldPortletView.jsp

```
<jsp:useBean id="HelloWorldPortletViewBean"
class="helloworld.HelloWorldPortletViewBean" scope="request" />

<%@ page contentType="text/html" import="java.util.*, helloworld.*"%>
<%@ taglib uri="/WEB-INF/tld/portlet.tld" prefix="portletAPI" %>

<portletAPI:init/>

<DIV style="margin: 6px">

<H3 style="margin-bottom: 3px">Welcome!</H3>
This is a sample <B>view mode</B> page. You have to edit this page to customize
it for your own use.<BR>
The source file for this page is "/Web
Content/helloworld/jsp/html/HelloWorldPortletView.jsp".
```

```
<br>
Current time: <%=new java.util.Date() %>
<br>
Hostname: <%= request.getRemoteHost()%>

<br>
Updated by <%=HelloWorldPortletViewBean.getMyName()%>

</DIV>
```

8. Save all your changes. For example, click **File -> Save All**.
9. Again, right-click the **HelloWorld** project in the Navigator panel and click **Run on Server**.

   You will now see the changes you made to your project when the portal loads in the Web Broswer.

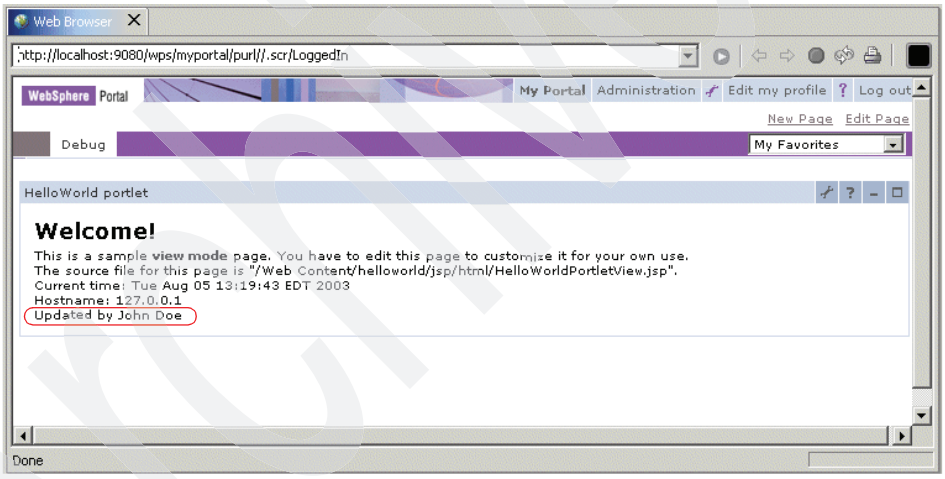


Figure 4-34 Viewing the changes made to your portlet

Archived

Archived

5

Action event handling

This chapter covers the ActionListener interface and the objects you will need to work with when managing event handling. To receive these events, an event listener must be implemented in the portlet class. In addition, the required method, actionPerformed, must be added to process the action event.

Archived

© Copyright IBM Corp. 2004. All rights reserved.

5.1 Action event

When a portlet wishes to be notified that a user has performed an action, it has to implement the ActionListener interface and a portlet action. Only the portlet generating the event may listen for that event. There will always be only a single listener for any particular action event. In order to notify other portlets of an event, the listening portlet may choose to send messages. For portlet messaging details, see Chapter 7, “Portlet messaging” on page 225.

A portlet has two phases of processing and rendering sequences. The first phase is the action processing phase. All events are generated, delivered and processed in this phase. Once this phase is complete, the service phase begins, in which portlets' outputs are rendered. Once this phase has begun, no events can be generated without causing an exception. The service method is also called when a portal page is refreshed.

The objects you will need to work with when managing event handling in action events are described below.

ActionListener

The org.apache.jetspeed.portlet.event.ActionListener interface defines a single method to be implemented as illustrated in Example 5-1.

Example 5-1 Implementing ActionListener interface

```
org.apache.jetspeed.portlet.event.ActionListener
public void actionPerformed(org.apache.jetspeed.portlet.event.ActionEvent
event) throws PortletException;
```

ActionEvent

An ActionEvent is sent by the portlet container when an HTTP request is received that is associated with a portlet action.

Note: The getAction() method returns the action that this action event carries but it is deprecated in favor of a getActionString() method.

The getActionString() method returns the action string that this event carries. Simple portlet actions use a single string as portlet action which can be executed multiple times and does not require a session.

Archived

Example 5-2 Working with the ActionEvent.

```
public void actionPerformed(ActionEvent event) throws PortletException {
   String actionString = event.getActionString();
   PortletRequest request = event.getRequest();
}
```

PortletURI

The PortletURI represents a URL that can be used to create navigation between modes. The PortletURI can be used to navigate to a previous mode, such as from Edit to View, or to navigate back to the same mode, such as for a multi-part form in View or Edit. There is no ability to create a PortletURI object pointing to a mode not yet visited by the user.

PortletResponse.createURI returns a portletURI object pointing to the portlet in its current mode. For example, if the portletURI is created in the doView mode, the URL points to the portlet in View. The createReturnURI method returns a PortletURI object pointing to the last mode the portlet was in. This mode is commonly used in the doEdit method when the URI needs to point back to the View mode. The edit.jsp would use the PortletURI to bring the user back to the View mode when the edit or configure process has been completed.

In order for a portlet to be notified of an event, such as the user clicking a button, the portletURI must contain an associated PortletAction. Typical PortletURI construction and usage is shown in Example 5-3.

Example 5-3 Working with PortletURI

```
PortletURI uri = response.createReturnURI();
uri.addAction(“save”);
request.setAttribute("uri", uri.toString());
```

It is possible to add parameters to the PortletURI object. Parameters added to the PortletURI via code or through a form are accessed in the same way via the portlet request object. This provides a mechanism to pass default values or to pass parameters not displayed in the form. Example 5-4 displays the code for adding a parameter. Be aware that parameters set via the PortletURI are not passed in the traditional HTML syntax. Example 5-4 shows how parameters are added to the URI.

Example 5-4 Adding a parameter to the PortletURi

```
public void doView(PortletRequest request, PortletResponse response) throws
                    PortletException, IOException {
   PortletURI viewURI = response.createReturnURI();
   viewURI.addAction(“save”);
   viewURI.addParameter("Param1", "Param1Value");
   request.setAttribute("viewURI", viewURI.toString());
   getPortletConfig().getContext().include("/jsp/View.jsp", request,
                    response);
   }
```

Portlet.ModeModifier

When a PortletURI is created, it points to a portlet in a particular mode. When that PortletURI is executed and if it contains a PortletAction, it will notify the appropriate listener. If, in the actionPerformed method, you need to redirect the user to a mode other than the one specified, the request.setModeModifier method can be used to redirect the user to another mode. The ModeModifier can only be set during event processing. Calling this method in doView or doEdit will have no effect. There are three possible modes to which the user can be redirected.

- **REQUESTED:** this ModeModifier will navigate the user to whatever mode was originally set by the PortletURI. Essentially, this is the default. If the ModeModified is changed, it cannot be changed back to `REQUESTED`.
- **CURRENT:** this ModeModifier will keep the user in the current mode. For example, if the user tries to save some information and the actionPerformed determines it is incorrect, setting ModeModifer to `CURRENT` will return the user to the edit screen.
- **PREVIOUS:** this ModeModifier will return the user to the mode the user was in prior to `CURRENT` regardless of previous ModeModification. Therefore, setting ModeModifer to `CURRENT` in one event process will not make that mode `PREVIOUS` in the next event process.

5.2 Window events

Window events are sent by the portlet container when a user modifies the window’s state by clicking control buttons such as the maximize or minimize buttons. To receive window events, you have to implement the WindowListener interface at the portlet class.

Note: The WindowEvent interface is deprecated; use PortletWindow.getWindowState() instead.

In the next example, the doView() method checks whether the window is maximized to set a parameter that the JSP page uses to display more information about each entry of the agenda.

Example 5-5 Get the window state.

```
public void doView(PortletRequest request, PortletResponse response)
            throws PortletException, IOException {
   PortletWindow.State state = request.getWindow().getWindowState();
   if ( state == PortletWindow.State.MAXIMIZED )
      request.setAttribute("detail", "1");
   else
      request.setAttribute("detail", "0");
   .......
```

The JSP checks *detail* attribute and displays information depending on its value. Figure 5-1 and Figure 5-2 show the results depending on the window state.

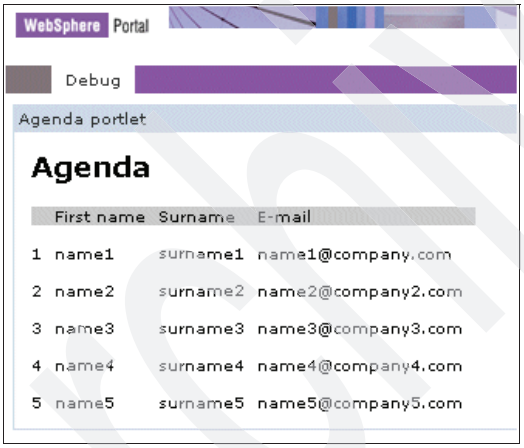


Figure 5-1 Information shown when the window is not maximized

Archived

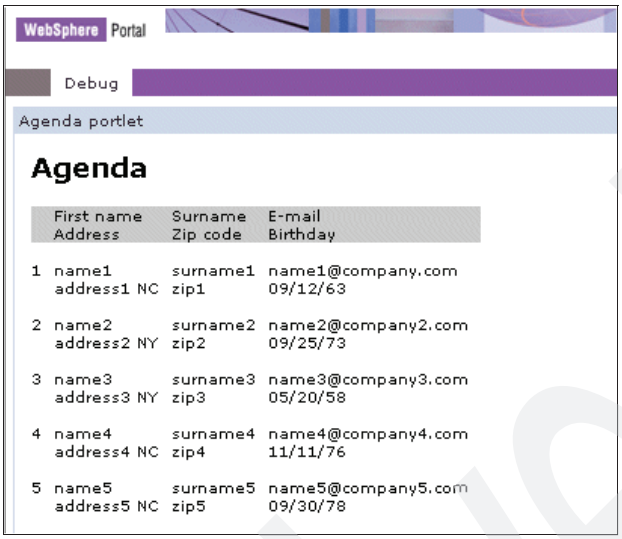


Figure 5-2 Information shows when the window is maximized

5.3 Simple action String support

Simple portlet actions are not available in the Portlet API prior to V4.2. To check if a simple action String is supported, you can use getMajorVersion() and getMinorVersion() methods of the PortletContext, which return, respectively, the major and minor version of the Portlet API that the portlet container supports. The sample code is shown in Example 5-6.

Example 5-6 Check PortletAPI version

```
if ( (getPortletConfig().getContext().getMajorVersion() <= 1) &&
       (getPortletConfig().getContext().getMinorVersion() <= 1) ) {
   // simple actions not supported
} else {
   // simmple actions supported
}
```

5.4 Sample scenario

The sample scenario illustrates the process of creating a sample portlet project that handles action events. You will create, deploy and run this portlet application. These exercises will allow you to understand the techniques used to develop portlets that process action events.

Archived

The development workstation is illustrated in Figure 5-3.

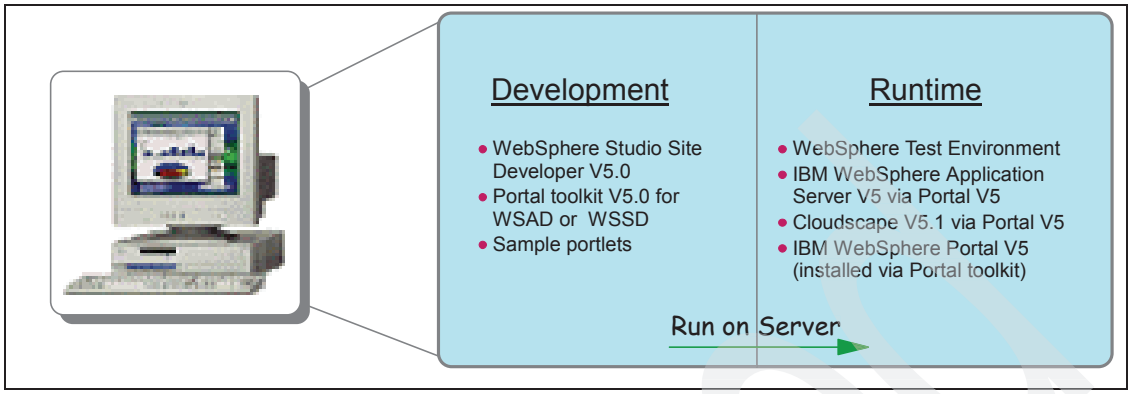


Figure 5-3 Development workstation

5.4.1 Scenario overview

Portlet events contain information about an event to which a portlet might need to respond; to receive these events, an event listener must be implemented in the portlet class. Three types of events are found in the Portlet API: *ActionEvents, MessageEvents* and *WindowEvents*.

There are also two additional events, PortletSettingsAttributeEvent and PortletApplicationSettingsAttributeEvent, which are used for notifications about changes to the attributes of the portlet settings of a concrete portlet or concrete portlet application.

In this scenario, you will create a sample portlet based on a Basic portlet type using the Portlet Wizard; you will then add code to support ActionEvent handling. To send an ActionEvent, you must associate a PortletAction with the http request. The PortletAction is normally linked with URLs or buttons in HTML form, providing a way to the portlet programmer to implement different processing actions for different user selections. The sample scenario is illustrated in Figure 5-4 .

Archive

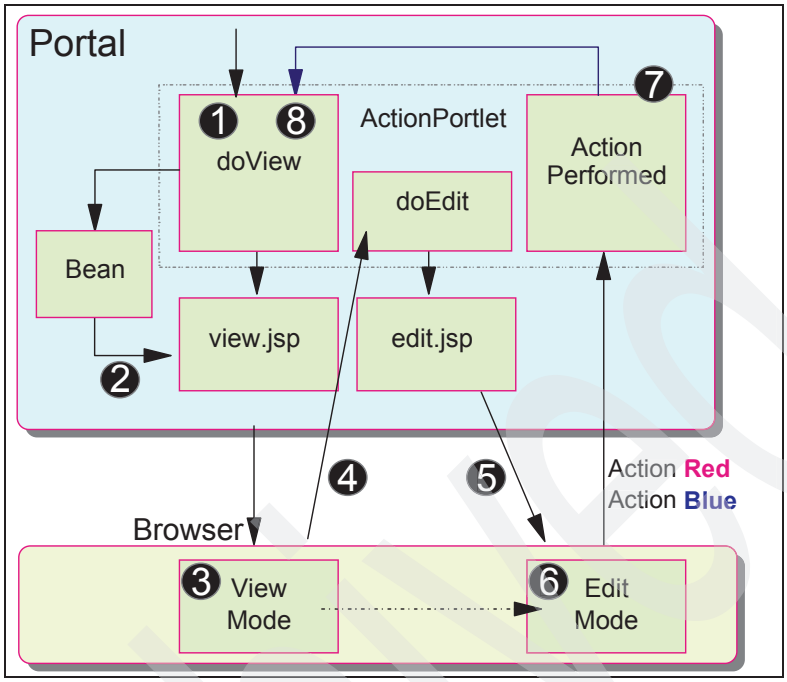


Figure 5-4 Event handling scenario

The sequence flow for this scenario is as follows:

1. Initially, the doView method is executed.
2. A JSP is called to render an initial screen. A message is obtained from the request object.
3. The Portlet View mode screen is shown in the browser window.
4. The user clicks **Edit** to go into Edit mode.
5. The Edit mode screen is displayed (the doEdit method is executed).
6. The user selects the desired action button (red or blue).
7. The actionPerformed method is executed to process the action. A resulting message is stored in the request object.
8. The doView method is executed to complete the cycle and a message is obtained from the request object.

Arihog

5.4.2 Creating the ActionEvent portlet

In this section, you create a Basic type portlet application with the name ActionEvent. The portlet application will be published and executed in the test environment.

1. If WebSphere Studio Site Developer is not running, click **Start -> Programs -> IBM WebSphere Studio -> Site Developer 5.0**.
2. Select **File -> New -> Project**.

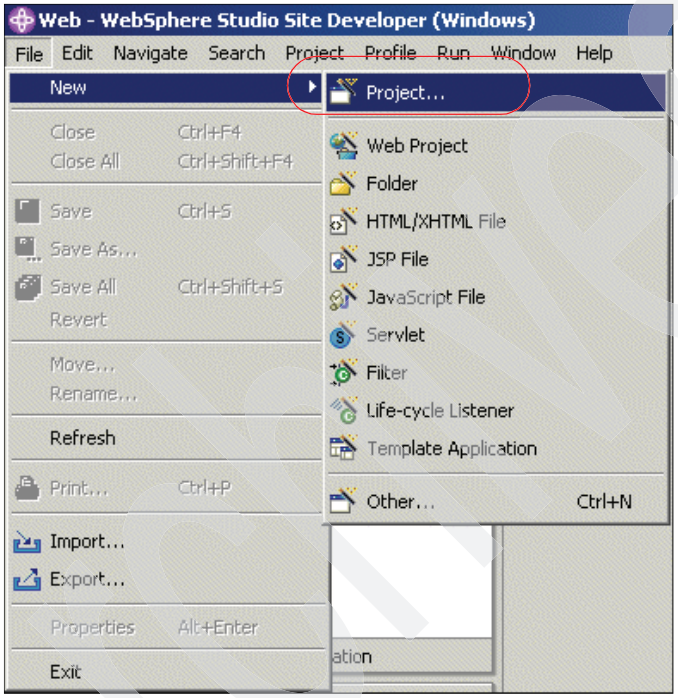


Figure 5-5 Starting creation of Portlet project

3. Select **Portlet development** -> **Portlet application project**. Click **Next**.

Aired

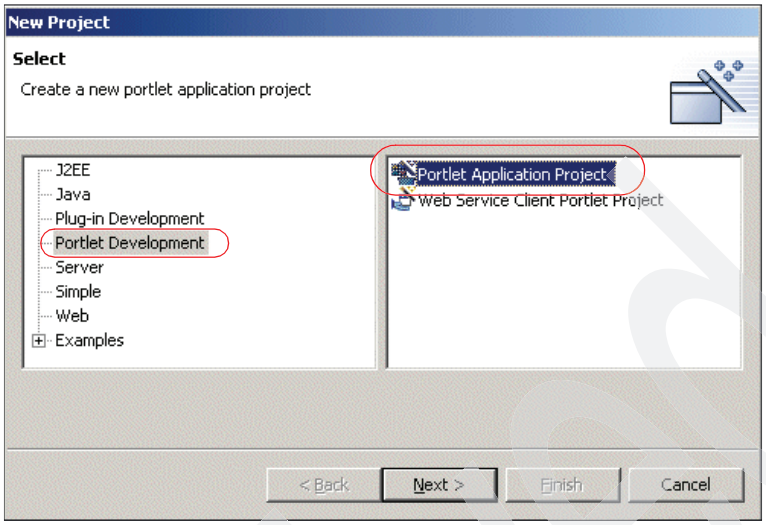


Figure 5-6 Selection of Portlet application project

4. The Portlet wizard will load. Enter `ActionEvent` as the project name. Then click **Next**.

Archived

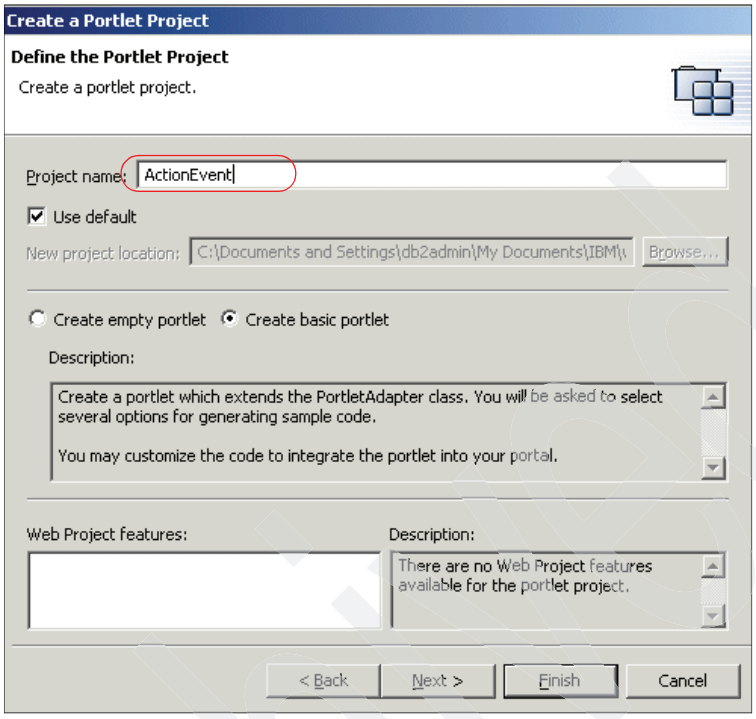


Figure 5-7 Defining the portlet project

5. You will be adding this project to the existing DefaultEAR project. Accept the default and click **Next**.

Archive

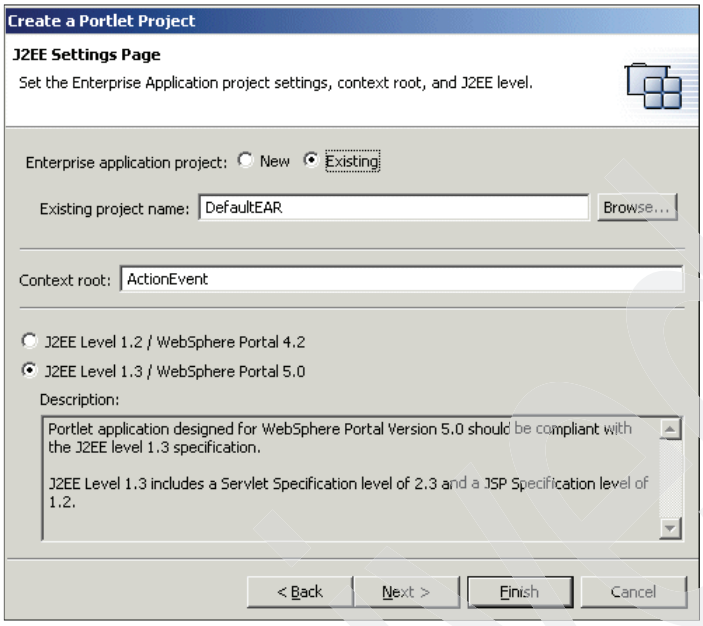


Figure 5-8 J2EE settings page

6. There are no module dependencies for this project. Click **Next**.

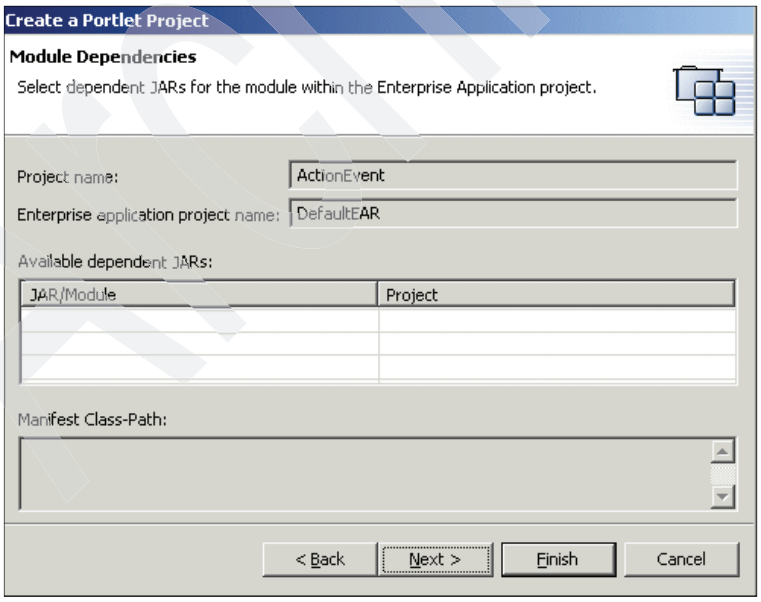


Figure 5-9 Module dependencies

7. Review and accept the default portlet settings. Click **Next**.

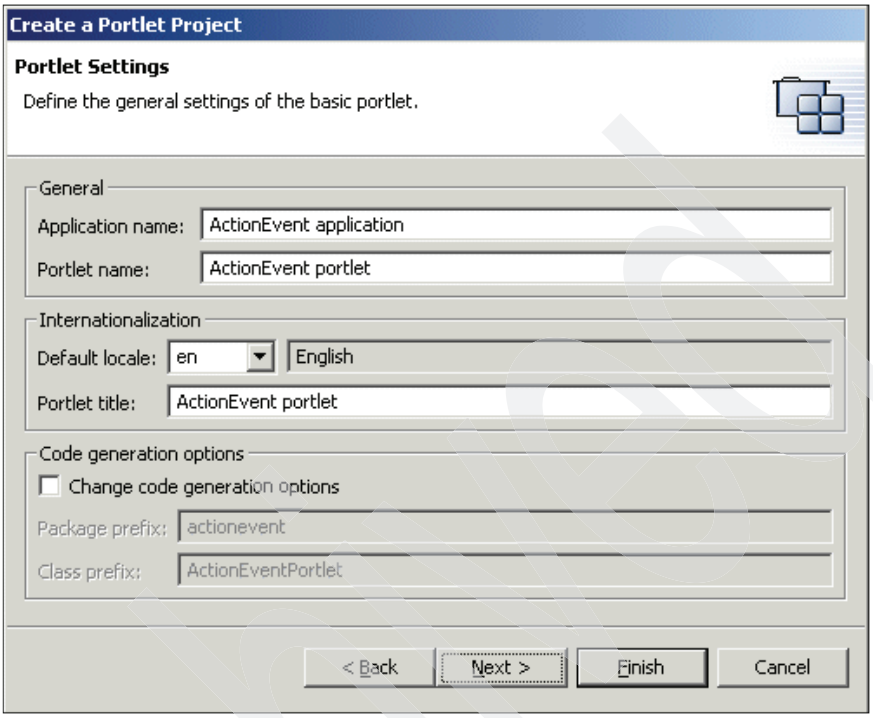


Figure 5-10 Portlet settings

8. Deselect the option for adding a form sample. In this project, you will create your own form. However, make sure that **Add action listener** remains checked. Click **Next**.

Archive

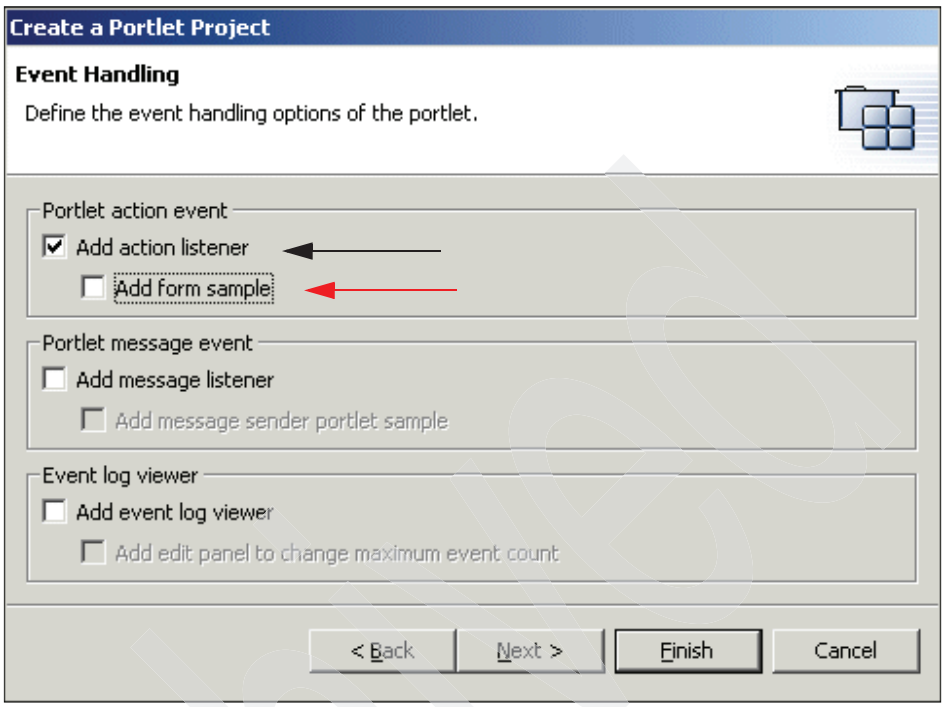


Figure 5-11 Event handling

9. This portlet project does not use the Credential Vault. Leave this option unchecked and click **Next**.

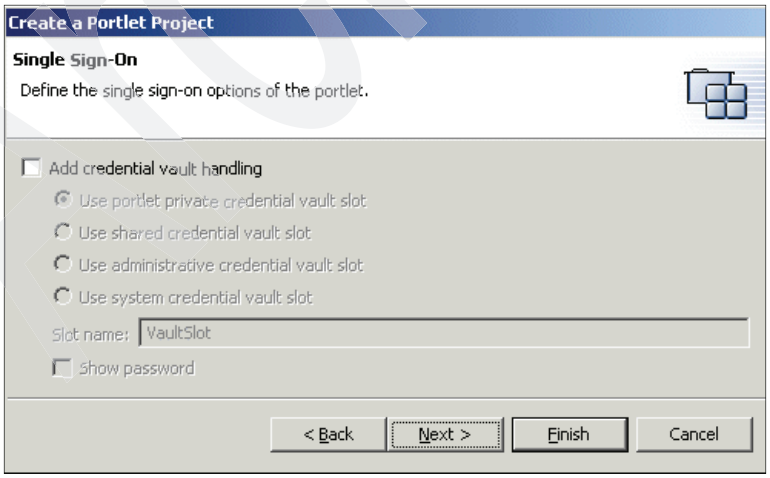


Figure 5-12 Single Sign-On

10. Leave the markup and mode options unchecked. This project will not require them. Click **Finish** to generate your project.

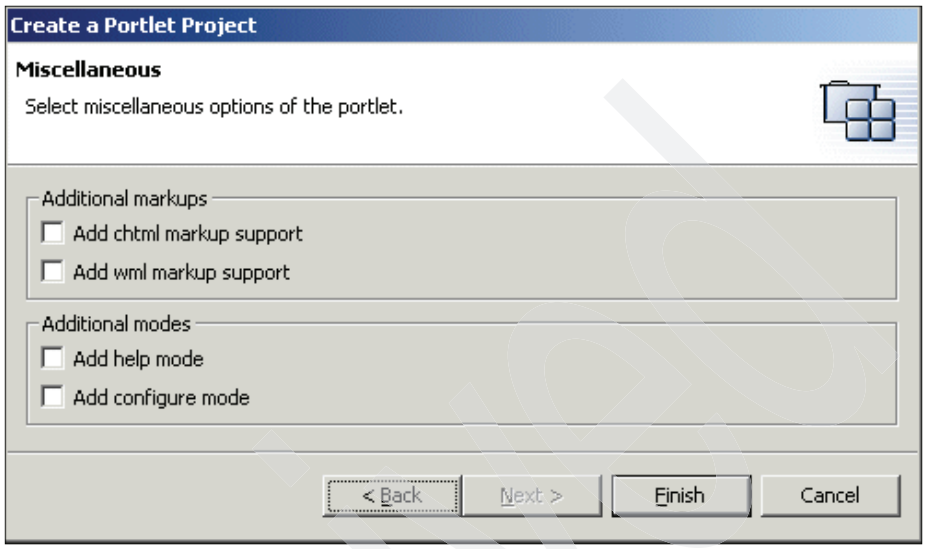


Figure 5-13 Modes and markups

11. Since you are adding an additional portlet project to the DefaultEAR project, you will receive a dialog informing you of this. Click **OK**.

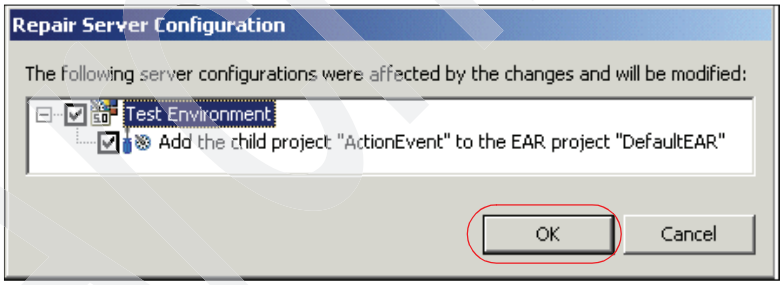


Figure 5-14 Server configuration dialog

12. In the Portlet.xml file that is now open on your screen, expand the Portlet Application tree and select **actionevent.ActionEventPortlet**.
13. Scroll down to the markups category and click the drop-down menu to enable fragment editing for the HTML markup.
14. Click **File -> Save Portlet.xml** to save the changes you made to this file.

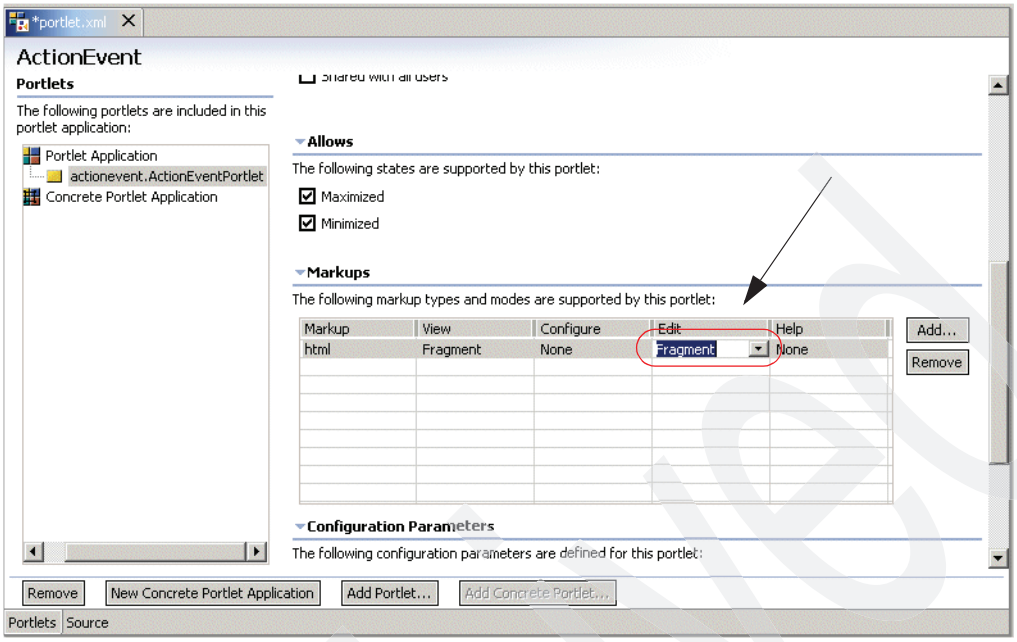


Figure 5-15 Editing portlet.xml

15. Now that the new project has been successfully added, you will want to remove the HelloWorld project so it is not deployed when you test this project later on in this exercise. To do this:

    a. Open the application.xml in the /DefaultEAR/META INF/ folder by double-clicking it.

Archived

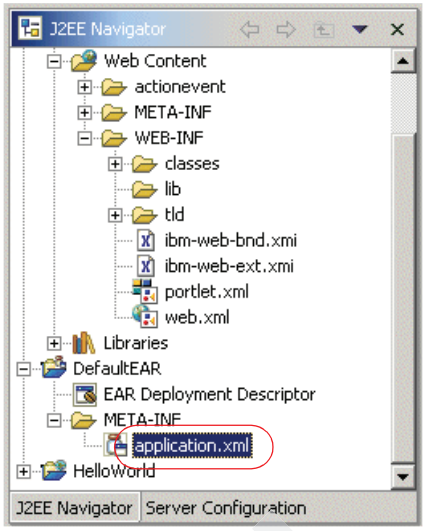


Figure 5-16 Application.xml

b. Select **Module**.

c. Remove all WAR modules except for this project called ActionEvent.

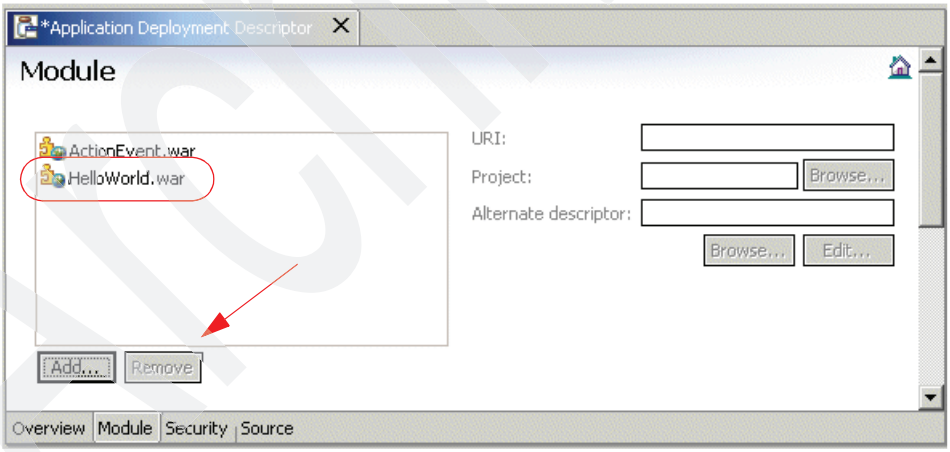


Figure 5-17 Removing WAR modules

d. Click **File -> Save All**.

e. If you receive the Repair Server Configuration dialog box, click **OK**.

Archived

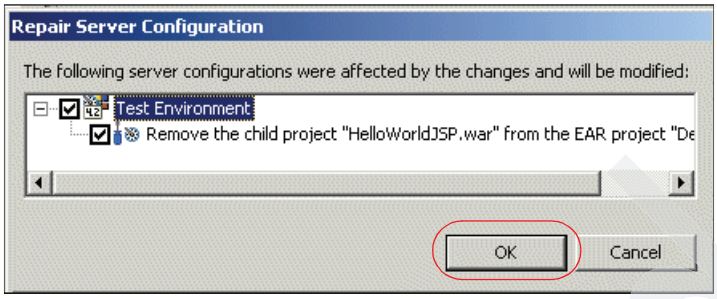


Figure 5-18 Configuration change dialog

16. The first step is to add a JavaServer Page (JSP) to your project. Right-click **ActionEvent**, then click **Web Content -> actionevent -> html**.

17. Click **New -> JSP File**.

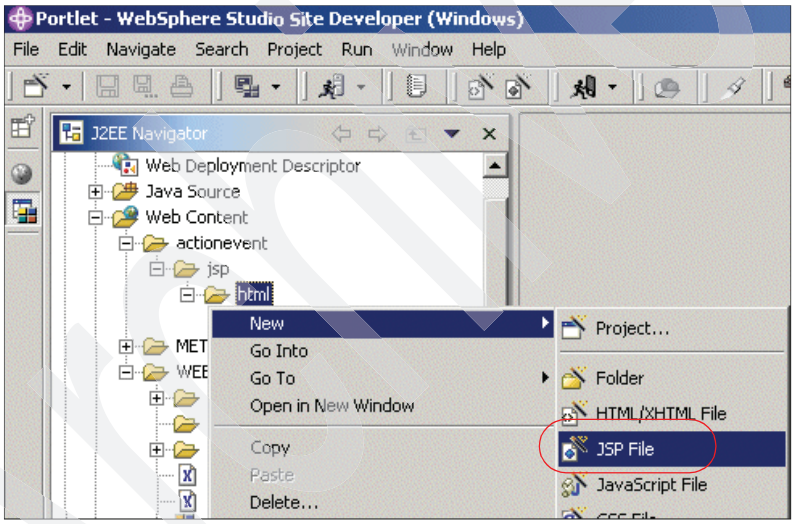


Figure 5-19 Adding a JSP

18. The Add JSP Wizard will run. On the first screen, enter `ActionEventPortletEdit.jsp` for the file name. Make sure that the **Create as JSP Fragment** option is checked. Click **Next**.

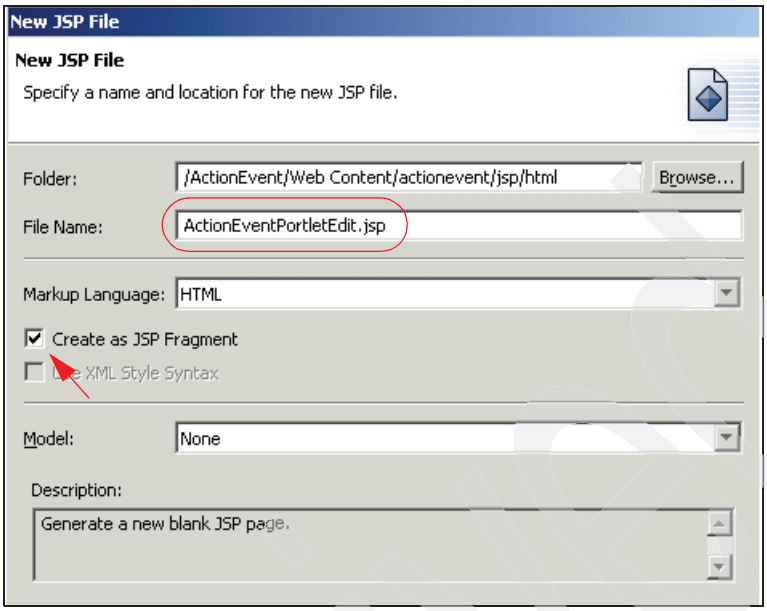


Figure 5-20 New JSP File Wizard

19. In the window that pops up, click the **Add Tag Library** button.

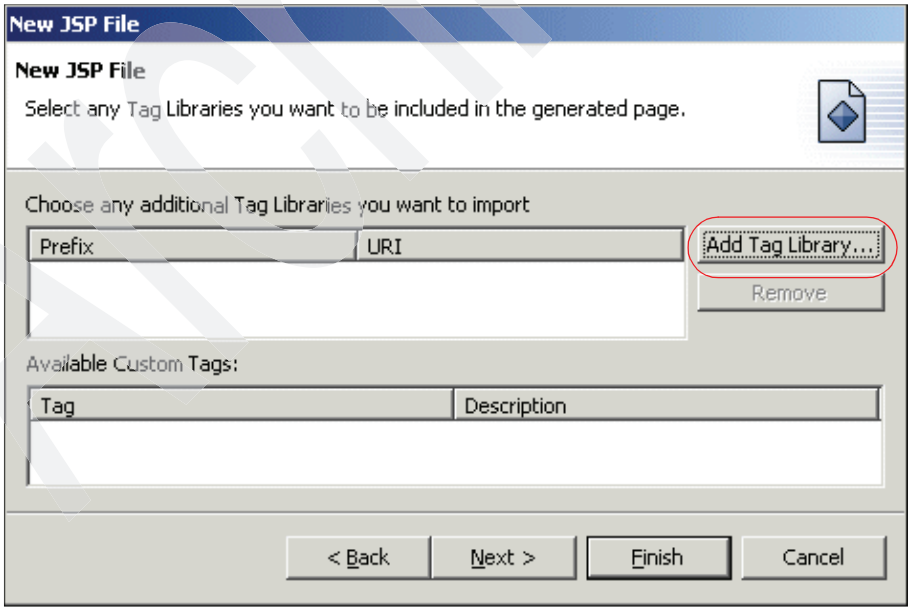


Figure 5-21 Add Tag Library window

20. Check the **/WEB-INF/tld/portlet.tld** tag library, then enter `portletAPI` as the Prefix. Click **OK**.

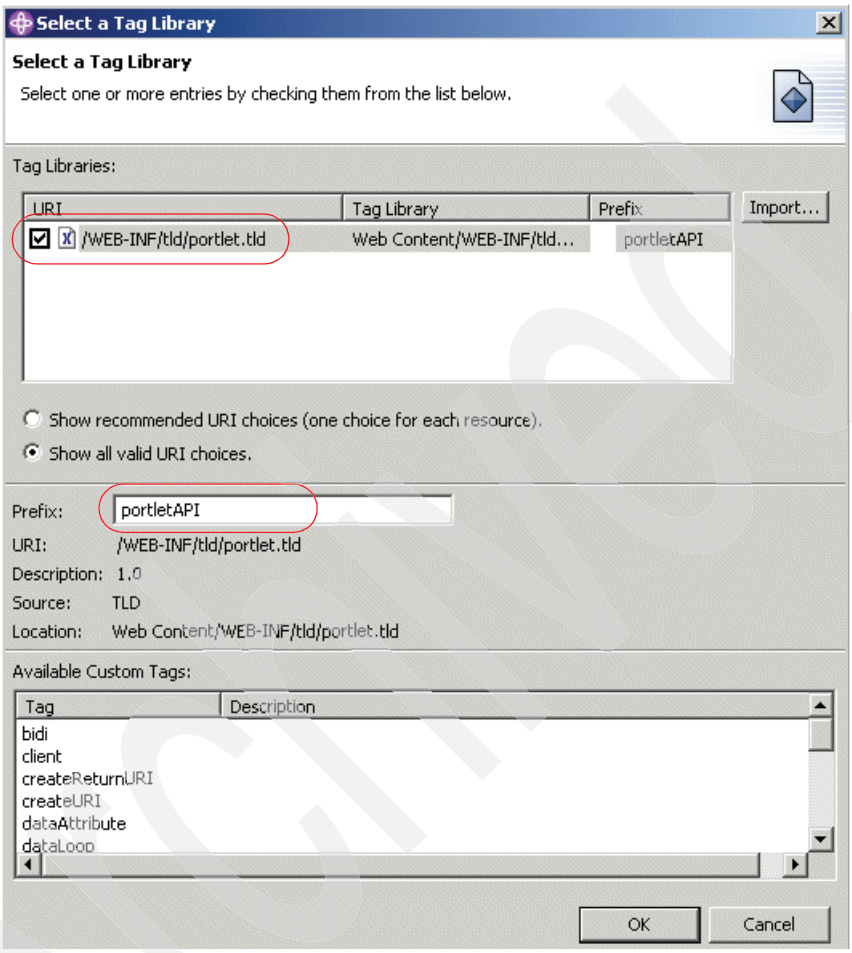


Figure 5-22 Selecting a tag library to add

21. Verify that the tag library has been added to the project. It should be listed in this window. Click **Next** to continue.

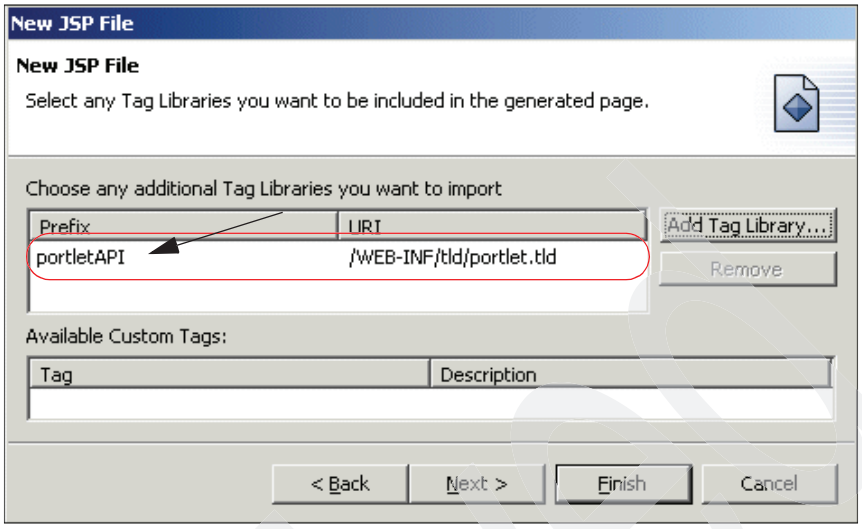


Figure 5-23 Tag libraries

22. On the next screen of the New JSP File wizard, make sure that **Generate a Page Directive** is unchecked; it is not needed. Click **Next** to continue to the last screen of the wizard.

Archive

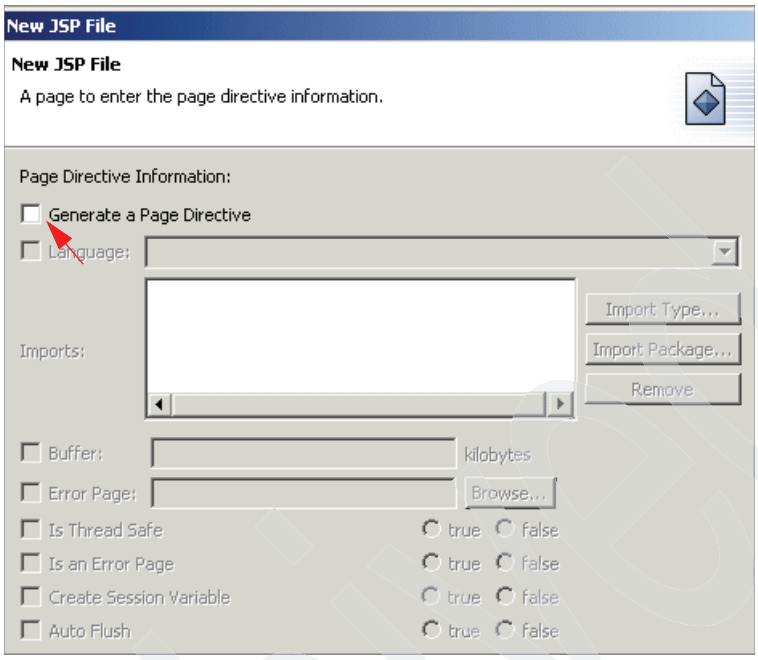


Figure 5-24 Page directive information

23. On the last screen of the wizard, accept the defaults and click **Finish**.

Archive

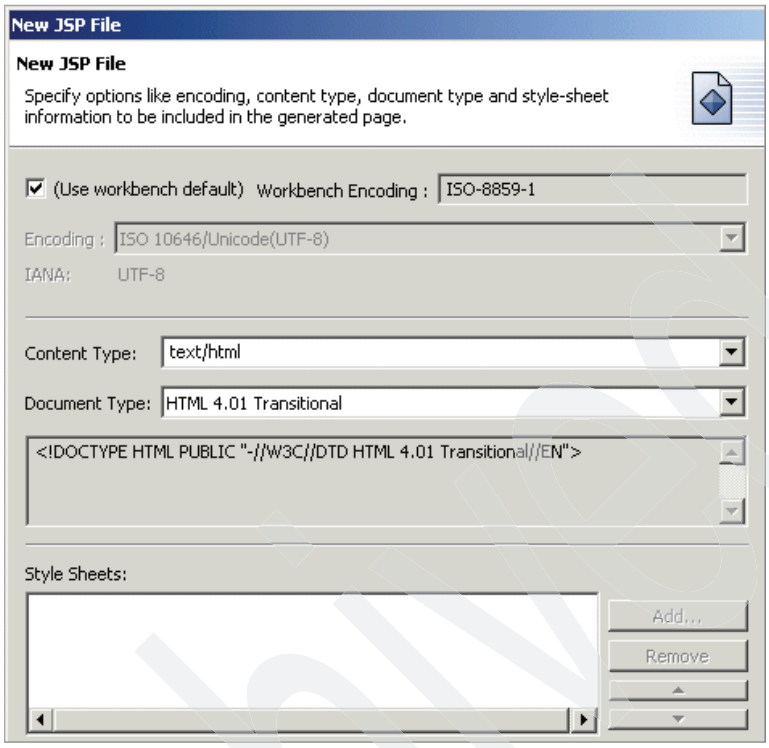


Figure 5-25 Encoding and content type

24. The ActionEventPortletEdit.jsp will now open on your screen. You will now edit this JSP and add two buttons for the user to click, corresponding to the two actions that they will be able to select when they run this portlet.

Example 5-7 ActionEventPortletEdit.jsp (Edit mode)

```
<%@ taglib uri="/WEB-INF/tld/portlet.tld" prefix="portletAPI" %>
<%@ page import="actionevent.*" %>

<TABLE class="Portlet" border="0">
  <TR>
    <TD>Please select an action:
    <FORM method='POST' action="<portletAPI:createReturnURI>
                                <portletAPI:URIAction

name='<%=ActionEventPortlet.ACTION_RED%>'/>
                                </portletAPI:createReturnURI>">
      <TABLE class="Portlet" border="0">
        <TR>
          <TD><INPUT type='submit' name='redButton' value='Red Action'></TD>
```

```
        </TR>
      </TABLE>
      </FORM>

    <FORM method='POST' action="<portletAPI:createReturnURI>
                                <portletAPI:URIAction

name='<%=ActionEventPortlet.ACTION_BLUE%>'/>
                                </portletAPI:createReturnURI>">
      <TABLE class="Portlet" border="0">
        <TR>
          <TD><INPUT type='submit' name='blueButton' value='Blue Action'></TD>
        </TR>
      </TABLE>
    </FORM>
    </TD>
  </TR>
</TABLE>
```

25. After making these updates, you will notice that the Tasks area of your screen indicates two errors. You will also see two small red Xs to the left of the JSP code. This is because the action values have not been defined at this time; they will be defined at a later time. For now, click **File -> Save ActionEventPortletEdit.jsp** and exit this file.



Figure 5-26 Editing ActionEventPortletEdit.jsp

26. Next, you will make changes to the ActionEventPortletView.jsp file. Open it by double-clicking the file which is located in the /Web Content/actionevent/jsp/html/ folder. Make the changes highlighted in the following example.

Example 5-8 ActionEventPortletView.jsp (View mode)

```
<%@ page contentType="text/html" import="java.util.*, actionevent.*"%>
<%@ taglib uri="/WEB-INF/tld/portlet.tld" prefix="portletAPI" %>
<portletAPI:init/>

<DIV style="margin: 6px">

<H3 style="margin-bottom: 3px">Welcome!</H3>
This is a sample <B>view mode</B> page. You have to edit this page to customize
it for your own use.<BR>
The source file for this page is "/Web
Content/actionevent/jsp/html/ActionEventPortletView.jsp".

<br>
<% if (request.getAttribute("value") == null) { %>
   <B>No action performed, select your action in Edit Mode</B>
<% } else { %>
   <B><%= request.getAttribute("value") %> ...was selected !</B>
<% } %>

</DIV>
```

27. Click **File -> Save ActionEventPortletView.jsp** to save your changes.
28. Next, you will make the changes to ActionEventPortlet.java. Open this file for editing by navigating to the /Java Source/actionevent/ folder and double-clicking it.

Archived

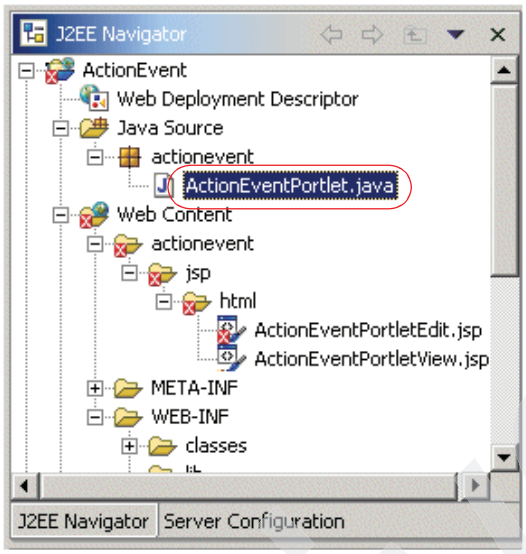


Figure 5-27 ActionEventPortlet.java

29. Several changes need to be made to ActionEventPortlet.java. Refer to the following examples to make these:

    a. First, you will add the variables `ACTION_RED` and `ACTION_BLUE` to hold the values for each of the two possible user actions.

Example 5-9 ActionEventPortlet.java

```
...
public static final String VIEW_JSP        =
"/actionevent/jsp/ActionEventPortletView.";

// Add strings corresponding to the actions
public static final String ACTION_RED       = "ACTION.RED";
public static final String ACTION_BLUE      = "ACTION.BLUE";
...
```

    b. Next, you will edit the doView method to send content to the JSP to render.

Example 5-10 ActionEventPortlet.java

```
...
public void doView(PortletRequest request, PortletResponse response) throws
                                                          PortletException,
IOException {
```

Archived

```
        // Create an instance of portlet data to store values
        PortletData portData = request.getData();

        // Extract value in portlet data into variable
        String value = (String) portData.getAttribute("value");

        // Store the extracted value in the request
        request.setAttribute("value", value);

          // Invoke the JSP to render

getPortletConfig().getContext().include(VIEW_JSP+getJspExtension(request),
request, response);
    }
...
```

c. Next, you will edit the actionPerformed method to process the action.

Example 5-11 ActionEventPortlet.java

```
...
public void actionPerformed(ActionEvent event) throws PortletException {
       if( getPortletLog().isDebugEnabled() )
          getPortletLog().debug("ActionListener - actionPerformed called");

       // ActionEvent handler
       String actionString = event.getActionString();

       // Add action string handler here

       if(actionString.equalsIgnoreCase(ACTION_RED)){

          // Create the string of HTML to be rendered
          String value = "Action <FONT color=\"#ff0000\">RED</FONT>";

          // Create a portlet request
          PortletRequest request = event.getRequest();

          // Create an instance of portlet data to store values
          PortletData portData = request.getData();

          try{
             // Save value into portlet data
             portData.setAttribute("value", value);
             portData.store();
          }
          catch (AccessDeniedException ade){
          }catch (IOException ioe){}
       }
```

Archived

```
        if(actionString.equalsIgnoreCase(ACTION_BLUE)){

            // Create the string of HTML to be rendered
            String value = "Action <FONT color=\"#0000ff\">BLUE</FONT>";

            // Create a portlet request
            PortletRequest request = event.getRequest();

            // Create an instance of portlet data to store values
            PortletData portData = request.getData();

            try{
                // Save value into portlet data
                portData.setAttribute("value", value);
                portData.store();
            }
            catch (AccessDeniedException ade){
            }catch (IOException ioe){}
        }
    }
...
```

d. Finally, you will add a doEdit method that will invoke the ActionEventPortletEdit.jsp when the user enters Edit mode. You can add this method wherever you like, as long as it is within the ActionEventPortlet class and not in any other method.

Example 5-12 ActionEventPortlet.java

```
...
// doEdit method called when user enters edit mode
public void doEdit(PortletRequest request, PortletResponse response) throws
                                                                PortletException,
IOException{

    // Invoke the JSP to render
    getPortletConfig().getContext().include("/actionevent/jsp/ActionEventPortle
    tEdit."+getJspExtension(request), request, response);

}
...
```

30. Click **File -> Save All**. Right-click the **ActionEvent** project in the navigator panel and click **Run Validation.** This should clean up the unresolved values in edit JSP.

Archived

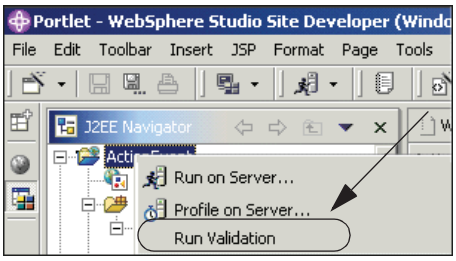


Figure 5-28 Run validation

5.4.3 Run the ActionEvent portlet application

1. Because you have created a new portlet project, you will need to restart the test environment to pick up these changes.
   a. Click the **Servers** tab at the bottom of your screen.
   b. Right-click **WebSphere Portal v5.0 Test Environment** or **Test Environment**.
   c. Click **Restart**.

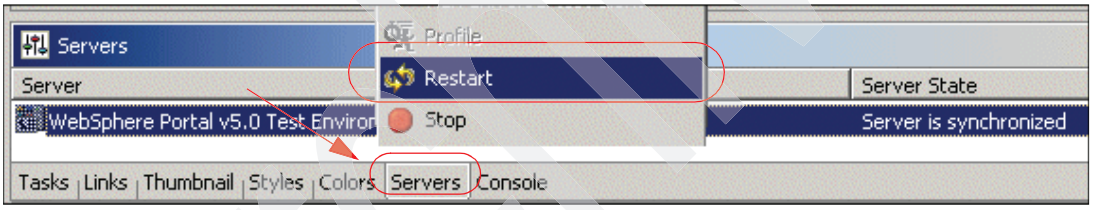


Figure 5-29 Restarting the server

2. It is now time to run your project on the server. Right-click the **ActionEvent** project in the navigator panel and click **Run on Server** using the test environment.

Archived

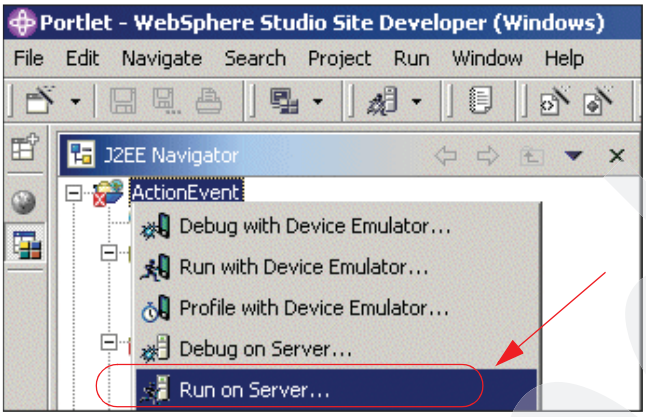


Figure 5-30 Run on Server

3. If prompted, click **Finish** in the Server Selection window to use the test environment to run your project.

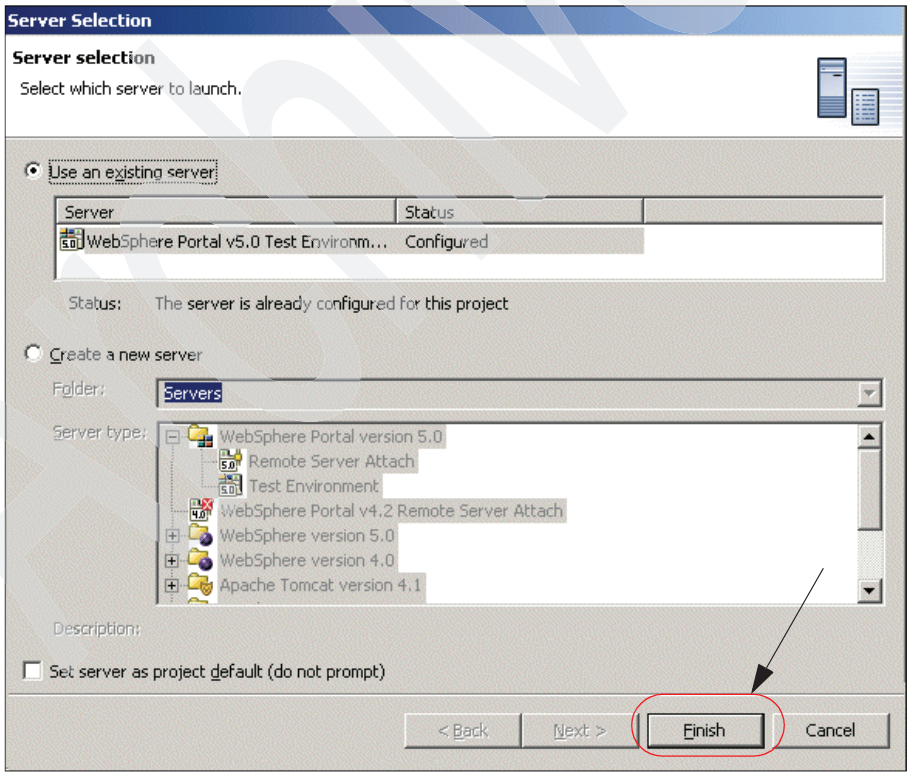


Figure 5-31 Server selection window

4. The portlet will display in the WebSphere Studio Site Developer integrated Web browser. Notice that the message `No action performed, select your action in Edit Mode` is displayed.

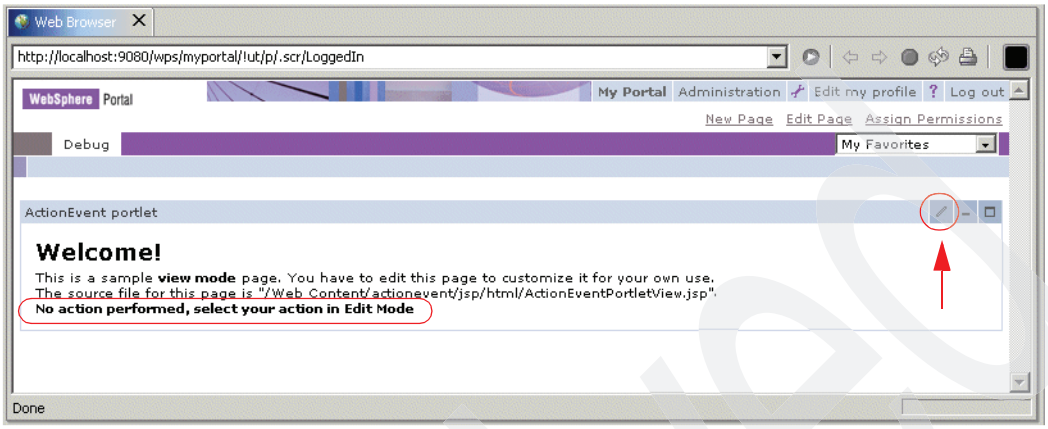


Figure 5-32 Portlet running in View mode

5. Click the icon highlighted in Figure 5-32 to enter the Edit mode. Once in Edit mode, click the **Red Action** button.

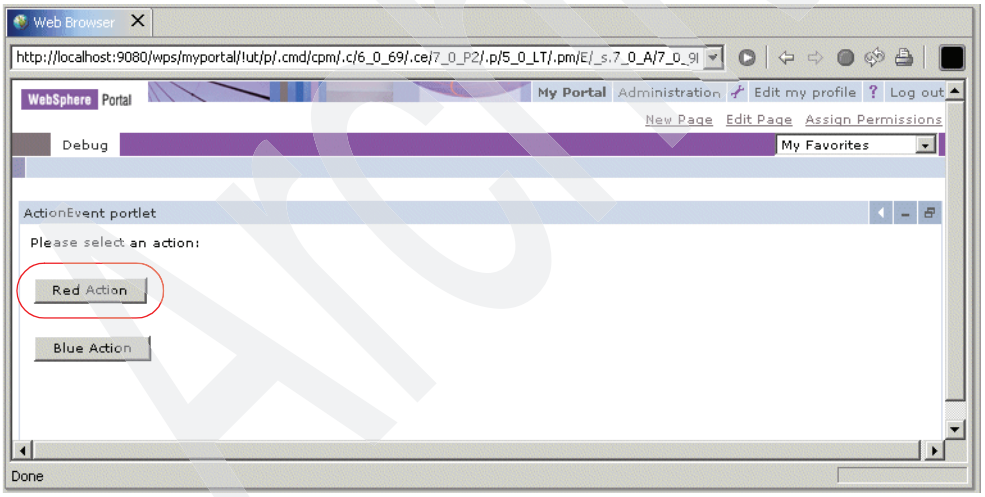


Figure 5-33 Edit mode

6. You are returned to View mode and the result of your action is displayed.

Archive

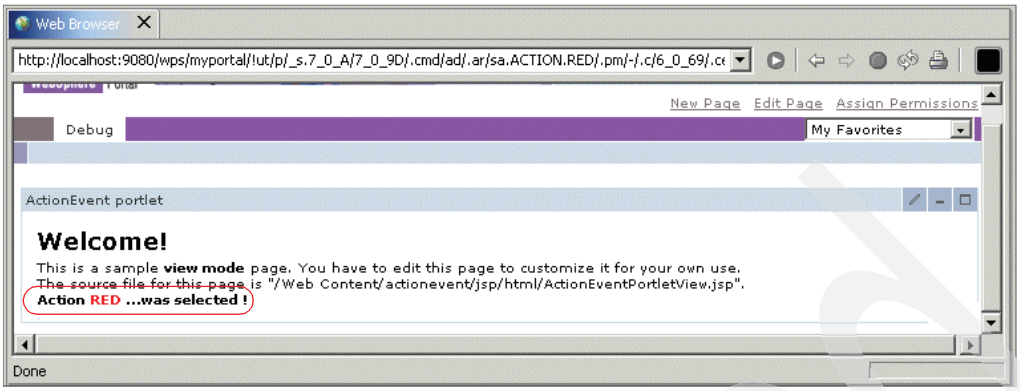


Figure 5-34 View mode showing Red action

7. Return to Edit mode and select the **Blue Action** button.

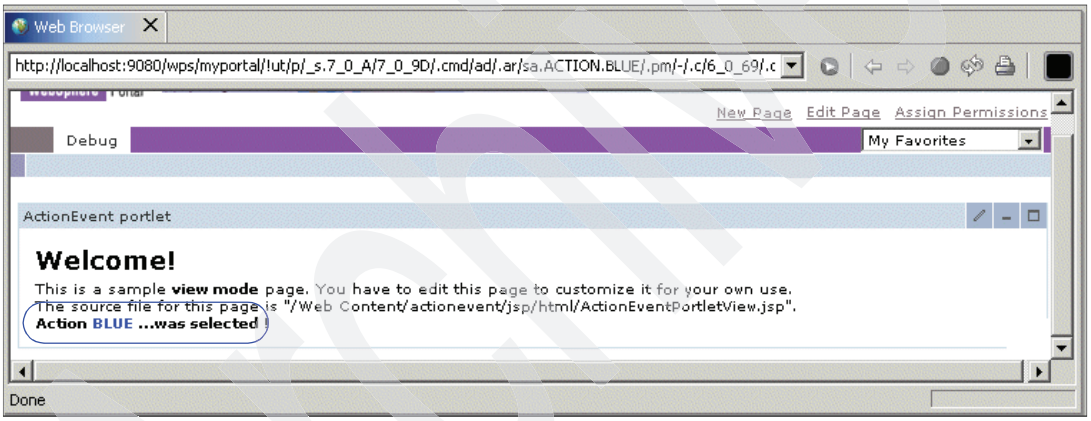


Figure 5-35 View mode showing Blue action

Archived

6

Portlet debugging

WebSphere Studio provides a powerful debugger for suspending launches, stepping through your code, and examining the contents of variables. This chapter gives you a brief introduction to the techniques used to debug portlets and discusses how to detect error during compile and runtime.

By the end of the chapter, you will be able to.

- Understand the value of debugging for portlet development.
- Fix compile errors using WebSphere Studio Toolkit.
- Set breakpoint and debug portlets in the Portlet perspective.

Archived

© Copyright IBM Corp. 2004. All rights reserved.

6.1 Overview

For software development, we can distinguish two different kinds of errors:

- Compile errors appear during compile and are thrown by the Java compiler. A typical example for this type of error is an improperly typed method or class name. This type of error can be found very easily because the compiler checks the code and presents a meaningful message.
- There are also runtime errors, which cannot be found by the compiler; thus they appear only during runtime. An example might be a loop stepping through an array with a size smaller than the loop variable. These kinds of errors are typically fixed using a debugger.

This chapter describes how to fix both types of errors using the validator and debugger tooling in WebSphere Studio.

6.2 Sample scenario

This section provides a sample scenario to illustrate how to debug portlets using the debug functions provided by WebSphere Studio Site Developer. You will use the portlet used in Chapter 5, “Action event handling” on page 181 as a base for this scenario. These activities will allow you to understand the techniques used to debug portlets.

6.2.1 Fixing compile errors

In this section, an example is provided to illustrate the use of the Java validator to validate Java code. An invalid character will be entered in the Java code to introduce an error and illustrate the correction process.

Note: WebSphere Studio provides different validators for different types of project resources, for example XML, HTML or Java files. In general, a validator is a process which validates a certain resource when you save it. After the validation process is finished, it displays the results of the validation in the Task view. To customize the validation settings, select **Properties** from the project context menu. In the Validation page, you can disable all or only certain validators.

To create a compile error, proceed as follows:

1. In the J2EE Navigator, select **ActionEvent -> Java Source -> actionevent -> ActionEventPortlet.java** as shown in Figure 6-1 on page 215. Double-click the Java file; it will open in the editor in the upper right-hand portion of the screen. If the file is already opened, scroll to the top.

Archived

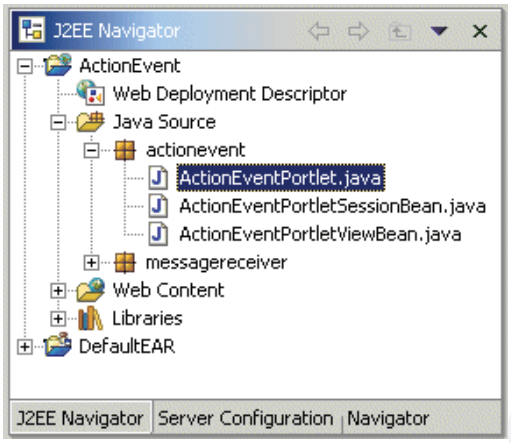


Figure 6-1 Double-click ActionEventPortlet.java to open

2. In the editor window, you should see a declarative statement:

   ```
   public class ActionEventPortlet extends PortletAdapter implements
   ActionListener
   ```

3. Place the letter `x` at the beginning of this statement to create an error. See Figure 6-2.

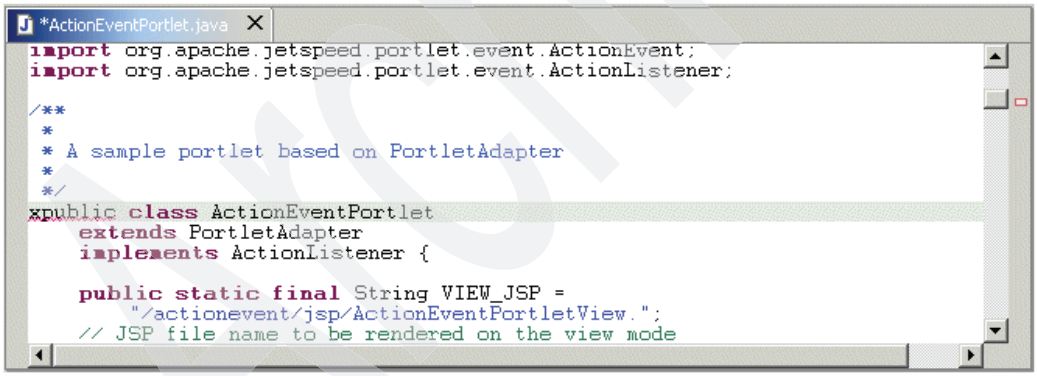


Figure 6-2 Incorrect public class declaration

4. Press **Ctrl-S** to save this file.
5. The compilation process fails due to the error you introduced. In the Tasks view in the lower right-hand portion of the screen, an error message appears, as shown in Figure 6-3 on page 216.

Archived

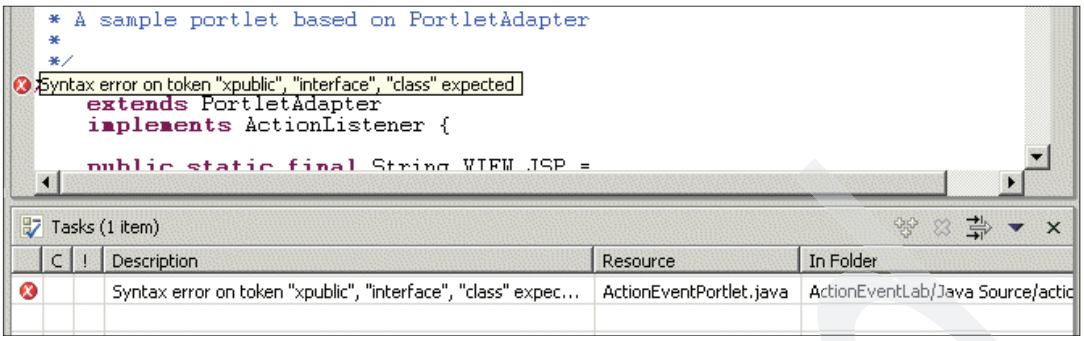


Figure 6-3 Result of saving an incorrect Java file

6. Double-click the red error icon in the Tasks window. The problem area in the code will be highlighted.

   Tip: If you cannot see the whole error message in the Task view because of its length, move over the red error symbol to the left of the Java editor. A small help window appears with the whole error message.

7. Remove the letter `x` before `public` to return the code to its original condition. Press **Crtl-S**. The code will be validated again, and the error message will disappear from the Task view.
8. Close the editor by clicking the **X** on the ActionPortlet.java tab.

6.2.2 Debugging a portlet application

When developing portlets, you often have to detect programming errors. One of the exciting features in WebSphere Studio is the integrated debugger for detecting errors during runtime. In this sample scenario, you will set a breakpoint, start WebSphere Portal in Debug mode and modify the value of a variable.

1. In the J2EE Navigator view, expand the ActionEvent project in the navigation panel.

Archive

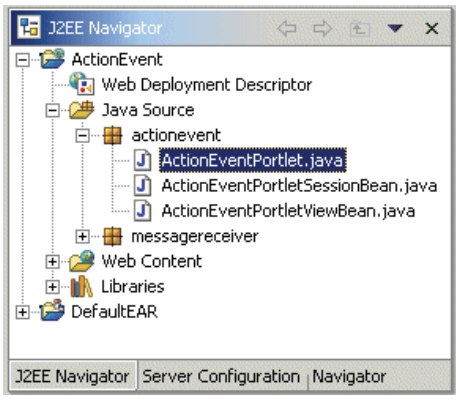


Figure 6-4 Select ActionEventPortlet.java to open it

2. Browse to the /Java Source/actionevent/ folder and double-click **ActionEventPortlet.java** to edit it.
3. The editor will open in the upper right-hand corner of the screen.
4. In this portlet class, there are five methods:
   - init
   - doView
   - actionPerformed
   - doEdit
   - getJspExtension
5. In the actionPerfomed method, you will set a breakpoint by placing the cursor on the `setAttribute` statement (this is the *if* clause that checks for ACTION_RED). Right-click the context bar to the left of the code, then select **Add Breakpoint** from the context menu as shown in Figure 6-5 on page 218.

Archived

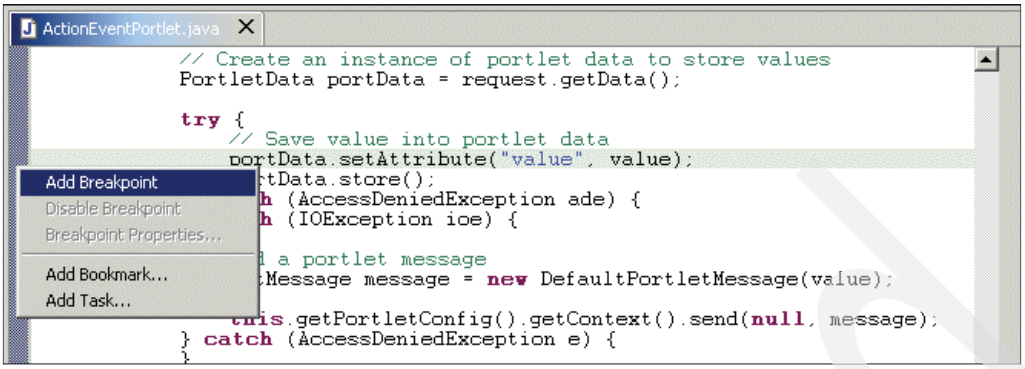


Figure 6-5 Adding a breakpoint

Note: It is not necessary to highlight the entire line.

6. After setting the breakpoint, you should see a dot of two possible colors in the context bar. If the breakpoint is blue, it is unverified, meaning that the containing class has not yet been loaded by the Java VM.

   If the breakpoint is green, it is verified, meaning that the containing class has been loaded by the Java VM, indicating that the breakpoint has been set.

   Note: This is the statement where the actionPerformed method has identified the action and set an attribute (setAttribute) in the request object; the attribute is to be rendered later in View mode.

7. To test this portlet, the Test Environment must be running in Debug mode. Click **Servers**. If the Test Environment is started, right-click and select **Stop** and wait until you see the message indicating that WebSphere Portal has stopped.
8. Once the Test Environment is stopped, right-click **ActionEvent** from the Navigator panel, then select **Debug on server**.

Archived

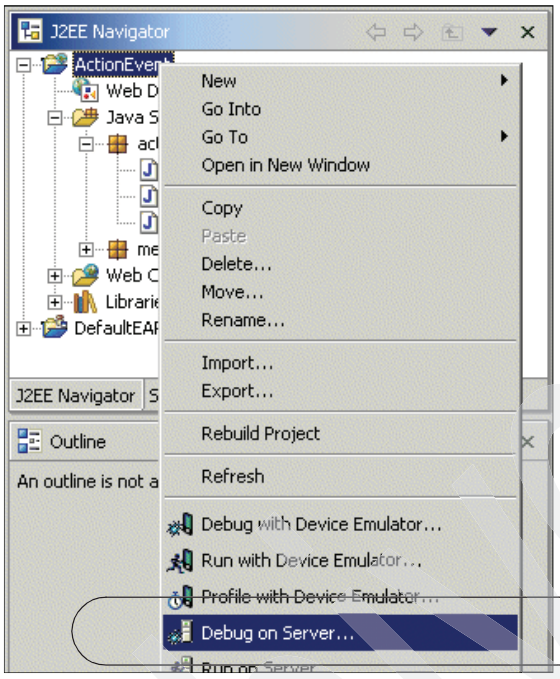


Figure 6-6 Start the server in debug mode

9. WebSphere Studio will change to the debug perspective. It will look as shown in Figure 6-7 on page 220.

   Note: Starting WebSphere Portal in debug mode will take a few minutes.

Archived

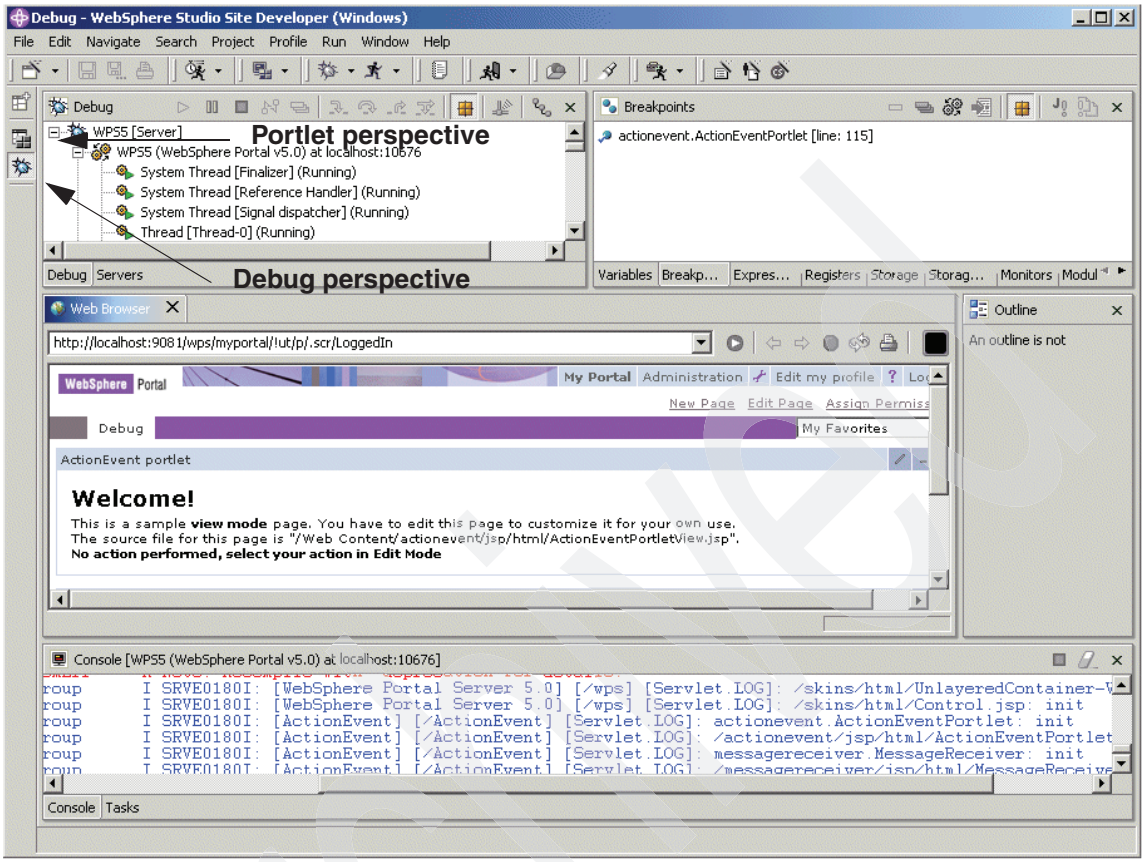


Figure 6-7 Debug perspective of Portal running the ActionEvent portlet

10. When the Step-by-Step Debug window appears, select **Skip** and **Disable step-by-step mode**, since you have already set a breakpoint, as shown in Figure 6-8 on page 221. Click **OK** and wait for execution of the portlet; the portlet will take extra time to execute in Debug mode.

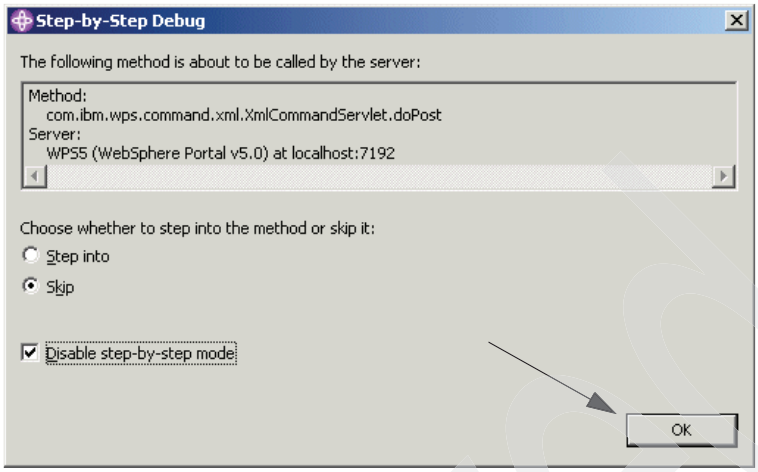


Figure 6-8 Disable the step-by-step mode

11. The portlet will run in the built-in browser shown in the middle left panel (in the Debug perspective) or the upper right panel (in the Portal perspective).
12. Now click **Edit mode** to select an action.
13. Select the **Red Action** button; remember that the breakpoint has been set in this action path (actionPerformed method).
14. The action (Red Action) will now execute up to the breakpoint you have previously set. When the breakpoint is reached, the Java editor displays the code and the statement with the breakpoint is highlighted.

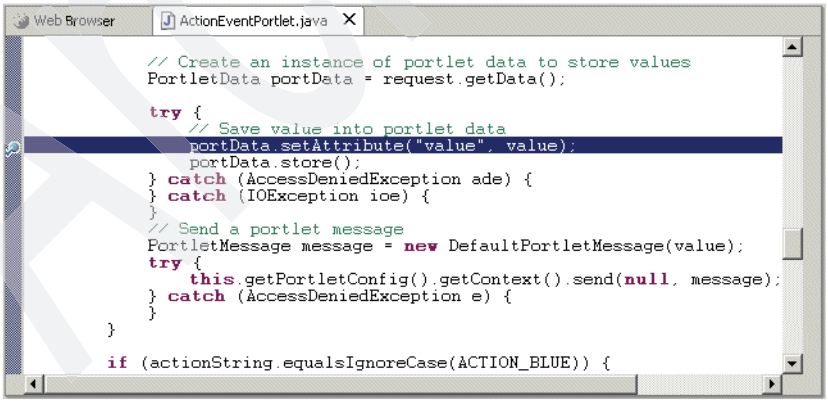


Figure 6-9 Debugger stops execution at the breakpoint

15. Place the cursor in the context bar where the breakpoint is located and right-click to select **Remove Breakpoint** from the context menu. Take a moment to examine the code before proceeding.
16. From the Debug perspective, select the **Variables** view.
17. Locate the variable *value* with a value of `Action <FONT color=\"#ff0000\">RED</FONT>`.
18. Select the **value** variable, then right-click it and select **Change Variable Value** from the context menu.

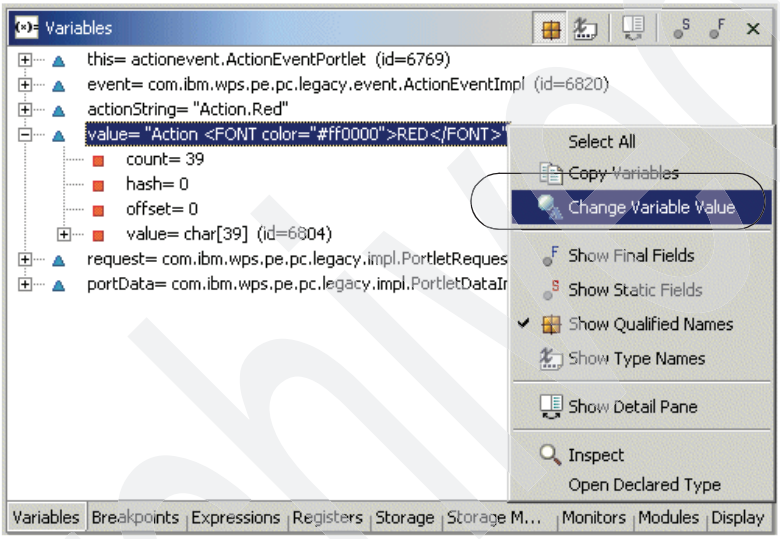


Figure 6-10 Changing the variable value in the Variables view

19. Enter `Action <FONT color=\"#00ff00\">GREEN</FONT>` as the new value. Press **Enter**.
20. Select the **Debug** view in the upper left. Click the **Resume** icon (green triangle on the left side of the toolbar icons).

Arcing

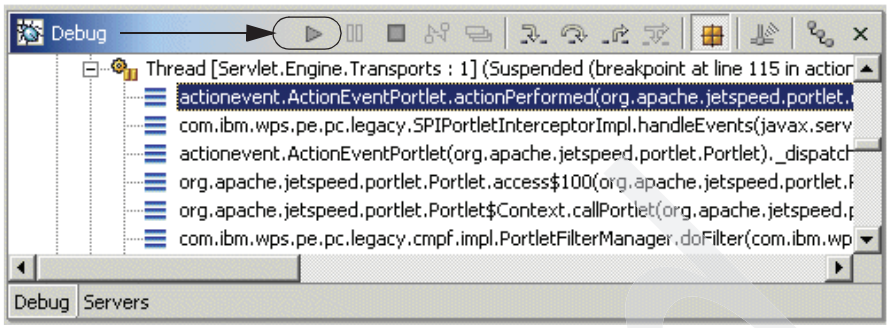


Figure 6-11 Click the Resume icon to resume the execution of the portlet

21. The portlet application will continue its execution. Select the **Web Browser** tab from the display toolbar in the middle left panel of the screen. The result of the action will be displayed. Note that the action displayed in View mode is now `Action GREEN ...was selected !` (and not `Action RED ...was selected !` as originally shown).

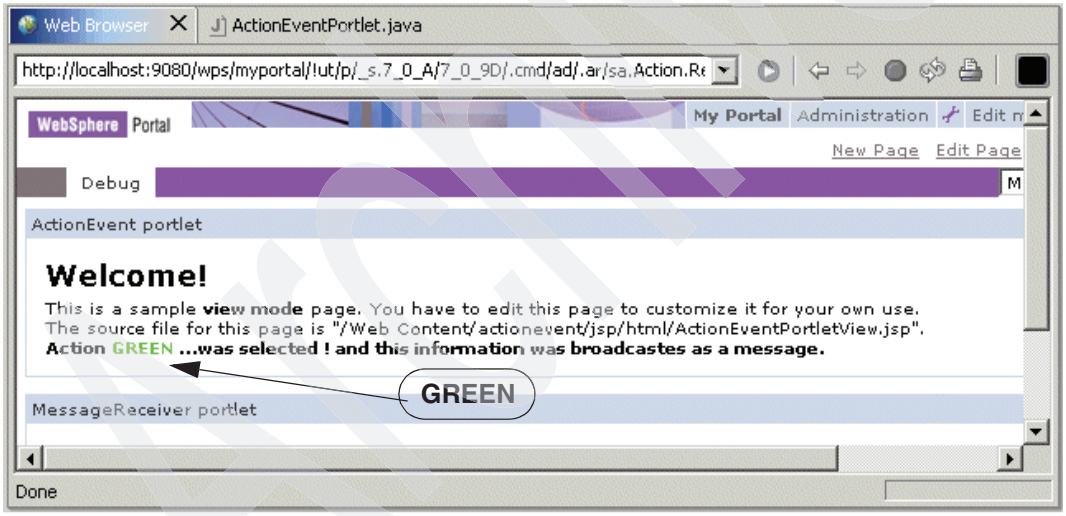


Figure 6-12 The ActionEvent portlet after changing a variable during debugging

Note: If you start another action in Edit mode, there will be no changes since the breakpoint was cleared and no other changes have been made at this time.

Archived

7

Portlet messaging

This chapter describes what portlet messaging is and the objects you will need to work with when messaging between portlets.

- MessageListener
- MessageEvent
- DefaultPortletMessage
- PortletMessage

Archived

© Copyright IBM Corp. 2004. All rights reserved.

7.1 Portlet messaging

One of the most significant advantages of the Portlet architecture is the portlets' ability to communicate with each other to create dynamic, interactive applications. Portlets can use messages to share information, notify each other of a user's actions or simply help better manage screen real estate.

Messages can be sent to all portlets on a page, to a specific, named portlet or to all portlets in a single portlet application. To send a message to all portlets on a page, you must send an instance of the DefaultPortletMessage.

In order to make full use of this potential, you need to adequately architect the entire portlet application, anticipating inter-portlet communication. Attempting to implement effective and meaningful message after significant portlet development will cause some difficulty and may require the entire application to be overhauled. This is true for several reasons. For example, access to certain storage objects, such as PortletData, is limited to certain modes. Therefore, if the initial design of an application makes significant use of the PorltetData object, implementing messaging later to share configuration information would require a considerable effort. Furthermore, in order to reduce or eliminate code, action event and message event functionality can be combined into a common method. However, to achieve this, it is necessary to consider the information passed via the action or message objects.

First, you must become familiar with the core objects used in the messaging architecture.

7.2 MessageListener

The org.apache.jetspeed.portlet.event.MessageListener interface must be implemented by the portlets you want the portal server to send messages to. The interface defines the single method listed in Example 7-1 on page 227. Since the portlet may be notified by more than one other portlet and therefore may receive different types of messages, it should validate the type of message received prior to working with the object. This is illustrated in Example 7-1 on page 227.

Archive

Example 7-1 Implementing the MessageListener interface

```
public void messageReceived(MessageEvent event) throws PortletException {

    PortletMessage msg = event.getMessage();

    if( msg instanceof DefaultPortletMessage ) {
        String messageText = ((DefaultPortletMessage)msg).getMessage();
        // Add DefaultPortletMessage handler here
    }
    .....
}
```

Be aware that when a portlet receives a message, it is not in Edit or Configure mode and therefore faces certain restrictions. For instance, portlets do not have write access to the PortletData object when they are not in Edit mode. Also, they cannot adjust the attributes stored in the PortletSettings object unless they are in configure mode. Attempts to store attributes in these object when not in the appropriate mode result in an AccessDeniedException.

Therefore, when attempting to share configuration or settings information between portlets, you need to choose your scope carefully or decide to persist to an outside resource.

7.3 MessageEvent

This object is sent to registered MessageListeners by the portlet container when a portlet executes the send method of the PortletContex object. There are two important methods available in this object

- getMessage: returns the message object sent with this event. Since this method returns a PortletMessage, the result must be cast to the appropriate type as illustrated in Example 7-1.
- getRequest: returns the current PortletRequest. The request can be used to access the PortletSession object or to store data to be used in the doView method.

7.4 DefaultPortletMessage

This object implements the PortletMessage interface and provides the basic functionality needed for sending string messages between portlets on the same page regardless of the portlet application.

Archive

Note: Since portlet messaging can be accomplished across portlets in different applications, this is the recommended way to implement portlet messaging.

If you broadcast a DefaultPortletMessage to null, it will be sent to all portlets on the page implementing the MessageListener interface. Example 7-2 illustrates sending a simple broadcast message to all portlets on the same page regardless of application affiliation.

Example 7-2 Broadcasting a message to all portlets on a page

```
PortletMessage msg = new DefaultPortletMessage("Some Message");
getPortletConfig().getContext().send(null, msg);
```

If you specify the portlet name, the message will be sent to all portlets and all their instances on the same page. The portlets with that name receive the message if they have implemented the appropriate listener. If the source and target portlet have the same name, the message will not be sent to avoid cyclic calls.

Example 7-3 Sending a message to a given portlet name on a page

```
PortletMessage msg = new DefaultPortletMessage("Some Message");
getPortletConfig().getContext().send("Portlet name", msg);
```

7.5 PortletMessage

This interface defines the message object that will be sent between portlets inside the same portlet application on the same page. Since it is a flag interface, it does not define any methods to be implemented. Therefore, you are free to create message objects that can store a wide variety of information. Example 7-4 illustrates a simple custom message used to carry a detail information about an entry of an agenda.

Example 7-4 Creating a custom message

```
import org.apache.jetspeed.portlet.*;
public class AgendaMessage implements PortletMessage {
   private AgendaBean entry;

   public AgendaBean getAgendaEntry() {
      return entry;
```


Archived

```
    }
    public void setAgendaEntry(AgendaBean newEntry) {
        this.entry = newEntry;
    }
}
```

If you simply need to send a string message between portlets, the DefaultPortletMessage provides this basic functionality. It is not possible to send a broadcast message using custom messages. Sending a custom message to null will only send the message to portlets implementing the MessageListener interface on the same page and deployed as part of the same portlet application. This is illustrated in Example 7-5; a message is sent with the information of an entry selected in other portlet in the same application.

Example 7-5 Sending a custom message

```
    public void actionPerformed(ActionEvent event) throws PortletException {
        if( getPortletLog().isDebugEnabled() )
            getPortletLog().debug("ActionListener - actionPerformed called");
        // ActionEvent handler
        String actionString = event.getActionString();
        // Add action string handler here
        PortletRequest request = event.getRequest();
        if ( actionString != null && actionString.equals(ACTION_DETAILS) ) {
            String opc = request.getParameter("option");
            int elem = Integer.valueOf(opc).intValue();
            Vector list = getSessionAgenda(request);
            AgendaBean entry = (AgendaBean)list.elementAt(elem);
            //send a message with this object
            AgendaMessage msg = new AgendaMessage();
            msg.setAgendaEntry(entry);
            getPortletConfig().getContext().send(null, msg);
        }
.....
```

If a portlet wants to receive this message, it has to implement the messageListener interface.

Example 7-6 Receiving a custom message

```
    public void messageReceived(MessageEvent event) throws PortletException {
        // MessageEvent handler
        PortletMessage msg = event.getMessage();
        // Add PortletMessage handler here
        if( msg instanceof AgendaMessage ) {
            AgendaBean detailEntry = ((AgendaMessage)msg).getAgendaEntry();
            // Add DefaultPortletMessage handler here
            PortletRequest request = event.getRequest();
```

```
        request.setAttribute("detailEntry", detailEntry);
        }
        else {
            // Add general PortletMessage handler here
        }
    }
```

Now you can see all the entries in one portlet and detailed information about an entry you selected previously in the other portlet. Figure 7-1 shows the result after selecting the third entry of the agenda.

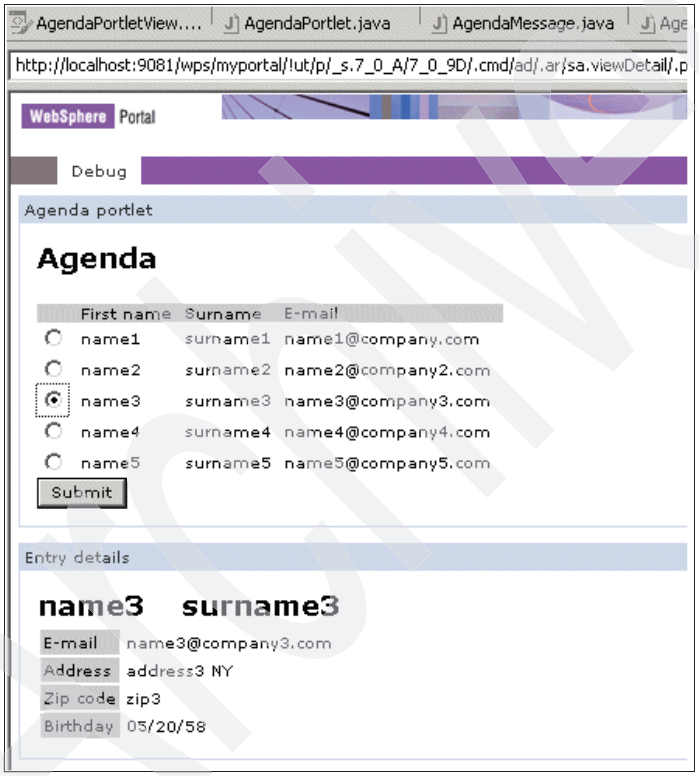


Figure 7-1 Receiving a custom message with an entry of the agenda

Archived

7.6 Sample scenario

Message events are a way for portlets to communicate with each other. This is accomplished through the familiar event-listener model. Portlets that need to listen for message events must implement a MessageListener interface, and portlets that need to send message events do so within the handling of their own Action Events, as you will see in this sample scenario. Message events can be sent to named portlets or broadcast to all portlets on the same page. All events are handled within the page's event-processing phase, after which comes the content generation phase.

For this sample scenario, the action event sample portlet application (see Chapter 5, "Action event handling" on page 181) will be modified to include message events so that you will see an example of how these can work together within portlet applications.

7.6.1 Description

In this scenario, you will enhance the ActionEvent portlet application to send messages to a new message receiver portlet as illustrated in Figure 7-2 on page 232.

Archived

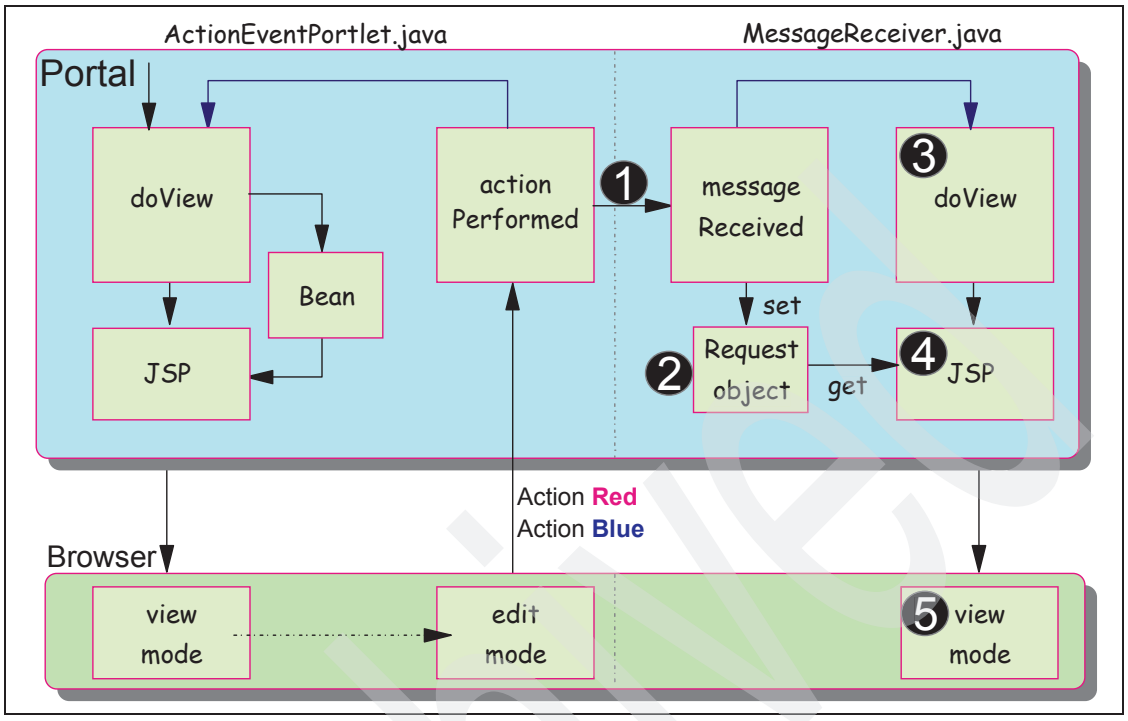


Figure 7-2 Message Event handling scenario

Figure 7-2 shows the flow for this scenario, as follows:

1. The actionPerformed() method in the ActionEventPortlet.java portlet will be extended to send a broadcast message event (DefaultPortletMessage) upon arrival of action events.
2. The MessageReceiver portlet, which will implement the MessageListener interface, receives the message in the new messageReceived method.
3. The received message is saved in the PortletRequest object.
4. The Portal invokes the doView method which in turn invokes the JSP (select).
5. The JSP retrieves the message from the request object and displays the message.

This scenario will be implemented using a broadcast style of message event rather than point-to-point messaging. In addition, the DefaultPortletMessage object will be used.

Archive

Note: While the DefaultPortletMessage object allows you to send messages to portlets in different applications, you can only send a String type message.

7.6.2 Sending a message

In this section, you will update ActionEventPortlet.java to send out a broadcast message from within its actionPerformed method. The message will be broadcast to all portlets implementing the MessageListener interface and using the DefaultPortletMessage object. Follow these steps:

1. If Studio is not running, start the IBM WebSphere Studio Site Developer by clicking **Start -> Programs -> IBM WebSphere Studio -> Site Developer 5.0**.
2. If the Test Environment is still running in Debug mode, stop the server by invoking **Servers** (make sure you switch to the portlet perspective), right-click **Test Environment** (started in Debug mode) and click **Stop**.
3. Next, you will update the actionPerformed() method to instantiate a DefaultPortletMessage object (the parameter `value` contains the message to be included in the object) and send the message (the parameter `null` indicates that this is a *broadcast* message). Add the *highlighted* code to the actionPerformed method in ActionEventPortlet.java (located in the /Java Source/actionevent/ folder) as illustrated in Example 7-7.

Example 7-7 Modifying implementation of actionPerformed() method

```
...
   public void actionPerformed(ActionEvent event) throws PortletException {
      if( getPortletLog().isDebugEnabled() )
         getPortletLog().debug("ActionListener - actionPerformed called");

      // ActionEvent handler
      String actionString = event.getActionString();

      // Add action string handler here

      if(actionString.equalsIgnoreCase(ACTION_RED)){

         // Create the string of HTML to be rendered
         String value = "Action <FONT color=\"#ff0000\">RED</FONT>";

         // Create a portlet request
         PortletRequest request = event.getRequest();

         // Create an instance of portlet data to store values
```

```
            PortletData portData = request.getData();

            try{
               // Save value into portlet data
               portData.setAttribute("value", value);
               portData.store();
            }
            catch (AccessDeniedException ade){
            }catch (IOException ioe){}

            // Send a portlet message
            PortletMessage message = new DefaultPortletMessage(value);
            try{
               this.getPortletConfig().getContext().send(null,message);
            }catch (AccessDeniedException e){}

         }

         if(actionString.equalsIgnoreCase(ACTION_BLUE)){

            // Create the string of HTML to be rendered
            String value = "Action <FONT color=\"#0000ff\">BLUE</FONT>";

            // Create a portlet request
            PortletRequest request = event.getRequest();

            // Create an instance of portlet data to store values
            PortletData portData = request.getData();

            try{
               // Save value into portlet data
               portData.setAttribute("value", value);
               portData.store();
            }
            catch (AccessDeniedException ade){
            }catch (IOException ioe){}

            // Send a portlet message
            PortletMessage message = new DefaultPortletMessage(value);
            try{
               this.getPortletConfig().getContext().send(null,message);
            }catch (AccessDeniedException e){}
      }
   }
   ...
```

4. Save and close the ActionEventPortlet.java file.

Archived

5. Next, you will slightly update the ActionEventPortletView.jsp page to notify that you are now sending a message. Double-click the **ActionEventPortletView.jsp under the /Web Content/actionevent/jsp/html/** directory.

   Note: Make sure ActionEventPortletView.jsp is in the html directory.

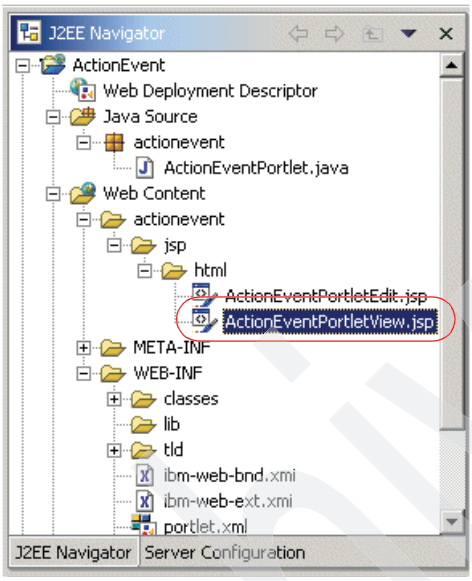


Figure 7-3 Edit tView.jsp

6. Insert the text highlighted in Example 7-8.

Example 7-8 Adding the code to broadcast PortletMessage

```
<%@ page contentType="text/html" import="java.util.*, actionevent.*"%>
<%@ taglib uri="/WEB-INF/tld/portlet.tld" prefix="portletAPI" %>

<portletAPI:init/>

<DIV style="margin: 6px">

<H3 style="margin-bottom: 3px">Welcome!</H3>
This is a sample <B>view mode</B> page. You have to edit this page to customize
it for your own use.<BR>
The source file for this page is "/Web
Content/actionevent/jsp/html/ActionEventPortletView.jsp".

<br>
<% if (request.getAttribute("value") == null) { %>
   <B>No action performed, select your action in Edit Mode</B>
```

Archived

```
<% } else { %>
   <B><%= request.getAttribute("value") %> ...was selected ! and this
information was broadcasted as a message.</B>
<% } %>

</DIV>
```

7. Save and close the ActionEventPortletView.jsp file.

   Note: At this point, you have implemented all the required logic in ActionEventPortlet to be able to send a broadcast message from within its actionPerformed() method.

7.6.3 Creating the target portlet

In this section, you will use a wizard to create the target portlet to receive the message sent by ActionEventPortlet.java.

1. Click **File -> New -> Other**.
2. Select **Portlet Development** from the left column and **Portlet** from the right. Click **Next** to continue.

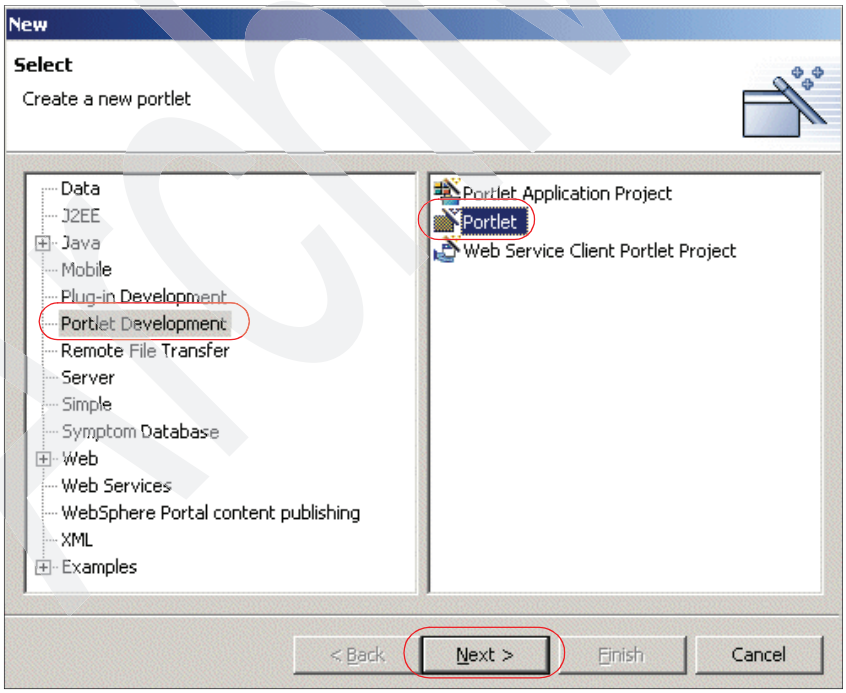


Figure 7-4 Add a portlet wizard

Archived

3. If required, select the **ActionEvent** project.
4. Examine and accept the default values. The wizard will add a basic portlet to the ActionEvent project.

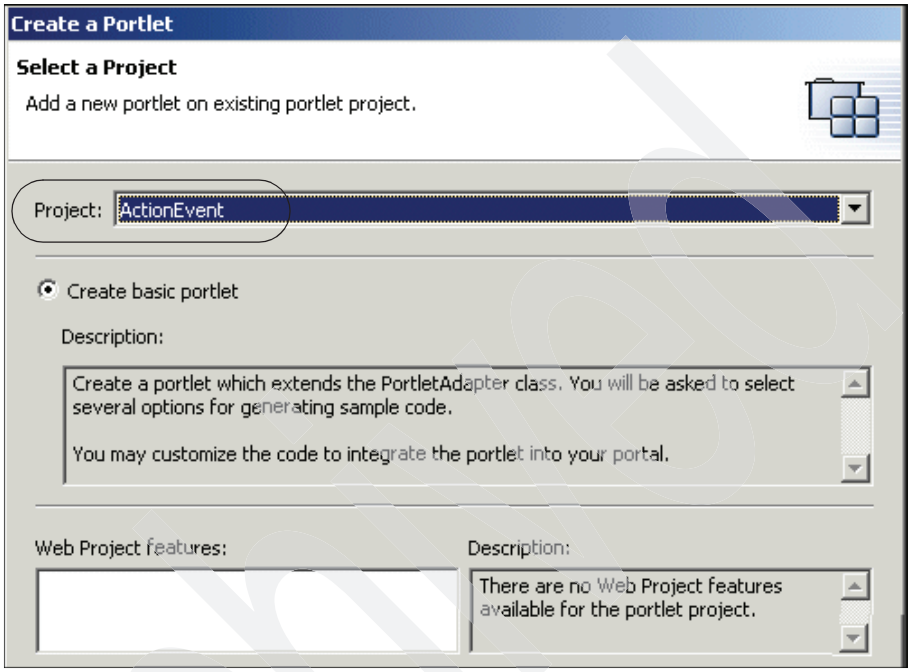


Figure 7-5 Adding a portlet

5. Click **Next**.
6. In the next window, change the portlet settings information to be relevant to the portlet you are adding. Enter the following information:
   a. Application name: `MessageReceiver application`
   b. Portlet name: `MessageReceiver portlet`
   c. Portlet title: `MessageReceiver portlet`
   d. Check the **Change code generation options** box to enter the following information:
      i. Package prefix: `messagereceiver`

         Important: Use lowercase for this prefix to follow naming conventions.
      ii. Class prefix: `MessageReceiver`

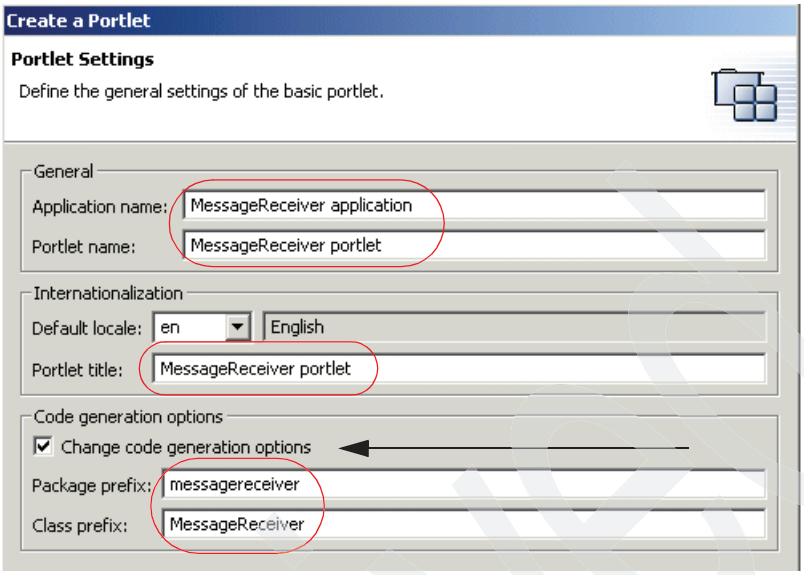


Figure 7-6 Portlet settings

7. Click **Next**.
8. Uncheck the **Add form sample** and the **Add action listener** boxes. Check the **Add message listener** box to add the messageReceived method. Click **Next**.

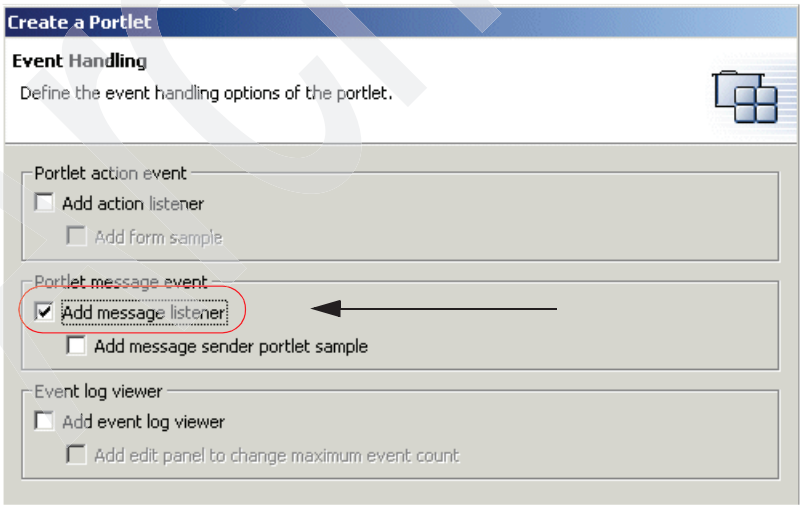


Figure 7-7 Adding a message listener

9. Do not check the **Add credential vault handling** box (not required in this sample scenario). Click **Next**.

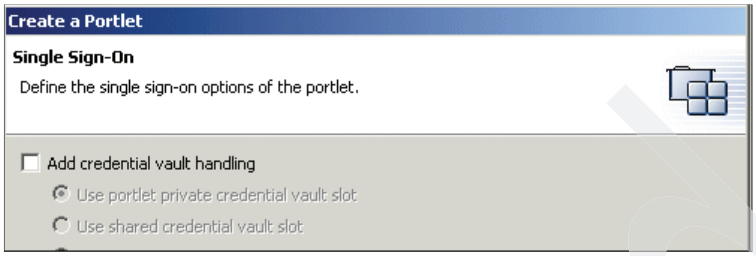


Figure 7-8 Credential vault handling is not required

10. Leave the options for markups and modes unchecked. Click **Finish** to add the portlet to your project.

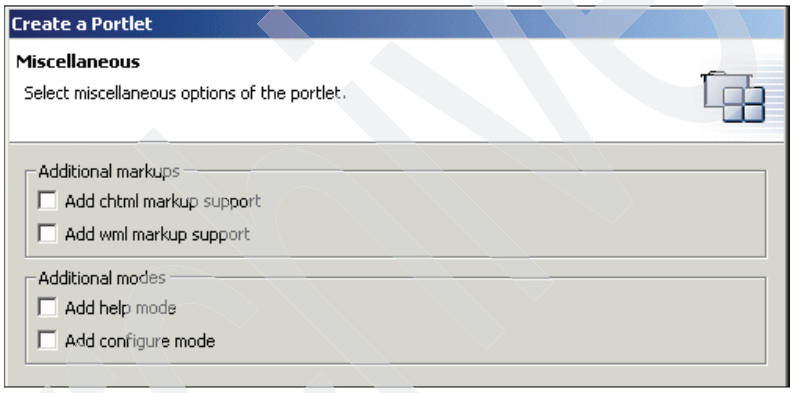


Figure 7-9 Markups and modes

11. You will now see the new portlet files in the Navigator panel.

Archived

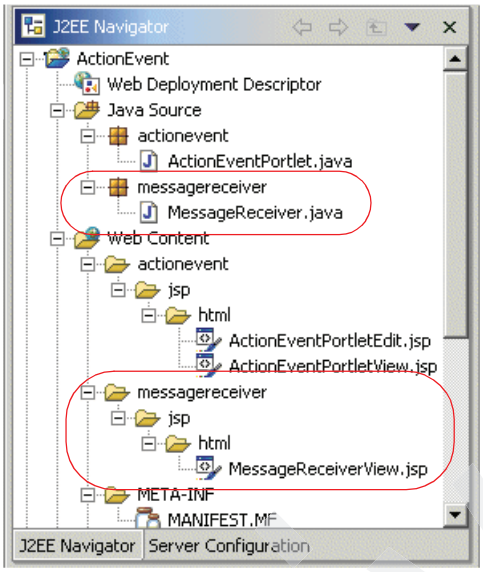


Figure 7-10 Navigator panel

12. Open the MessageReceiver.java file located in the /Java Source/messagereceiver/ folder by double-clicking it.

13. Add the following highlighted code to this file to receive the broadcast Portlet Message in the messageReceived() method.

    Note: The messageReceived () method implements the logic to receive the PortletMessage. In this example, you only need to check for messages of type DefaultPortletMessage, which is the type of message sent by ActionEventPortlet. Then the message is extracted via the getMessage() method, and you set the text of this message into a portlet request as an attribute with name MyMessage.

Archived

Example 7-9 MessageReceiver.java

```
...
public void messageReceived(MessageEvent event) throws PortletException {
       if( getPortletLog().isDebugEnabled() )
           getPortletLog().debug("MessageListener - messageReceived called");
       // MessageEvent handler
       PortletMessage msg = event.getMessage();
       // Add PortletMessage handler here
       if( msg instanceof DefaultPortletMessage ) {
            String messageText = ((DefaultPortletMessage)msg).getMessage();
            // Add DefaultPortletMessage handler here
            PortletRequest request = event.getRequest();
            request.setAttribute("MyMessage", messageText);
       }
       else {
           // Add general PortletMessage handler here
       }
   }
...
```

14. When you are done, save the file and exit.

15. Now that you can receive the message, you will need to modify the MessageReceiverView.jsp to display the message to the user. To edit this file, open the /Web Content/messagereceiver/jsp/html/ folder and double-click the file.

    Note: The Java code inside the scriptlet checks the value of the portlet request attribute MyMessage. If null, no message has been received yet, and it displays that it is ready to receive a message. If not null, the message is displayed with HTML markup. Make the following changes.

Example 7-10 MessageReceiverView.jsp file (message receiver portlet)

```
<%@ page contentType="text/html" import="java.util.*, messagereceiver.*"%>
<%@ taglib uri="/WEB-INF/tld/portlet.tld" prefix="portletAPI" %>
<portletAPI:init/>

<DIV style="margin: 6px">

<H3 style="margin-bottom: 3px">Welcome!</H3>
This is a sample <B>view mode</B> page. You have to edit this page to customize
it for your own use.<BR>
The source file for this page is "/Web
Content/messagereceiver/jsp/html/MessageReceiverView.jsp".

<br>
<% if (request.getAttribute("MyMessage") == null) { %>
```

Archived

```
   <B>Ready to receive message...</B>
<% } else { %>
   <B>Received a message:</B>
   <B><%= request.getAttribute("MyMessage") %></B>
<% } %>

</DIV>
```

16. When you are done, save the file and exit.

    You have now implemented the code to receive and display a broadcast portlet message to the user.

7.6.4 Running the portlet application

In this section, you will run the portlet application you have developed to send a message from the message sender portlet (ActionEventPortlet.java) to the message receiver portlet (MessageReceiver.java). Follow these steps:

1. Right-click the **ActionEvent** project and choose **Run on Server**. The project will be published and then started.

   Note: You will see that the internal Web browser brings up the two portlets on your screen, as shown in Figure 7-11. Notice that the ActionEvent portlet indicates that no action has been performed and the MessageReceiver portlet indicates that it is ready to receive a message.

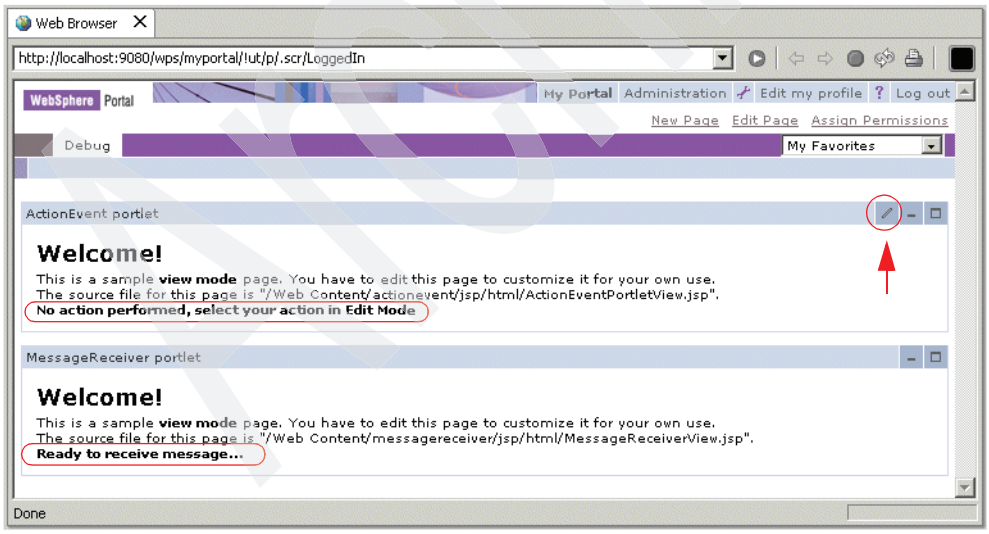


Figure 7-11 Running the messaging project in the Portal Server Test Environment

2. Choose the Edit mode of ActionPortlet as indicated with the arrow in Figure 7-11 on page 242. Click the **Red Action** button. This will both (a) create an action that you will see the action in ActionPortlet, and (b) broadcast a message which will be sent and shown in MessageReceiver.

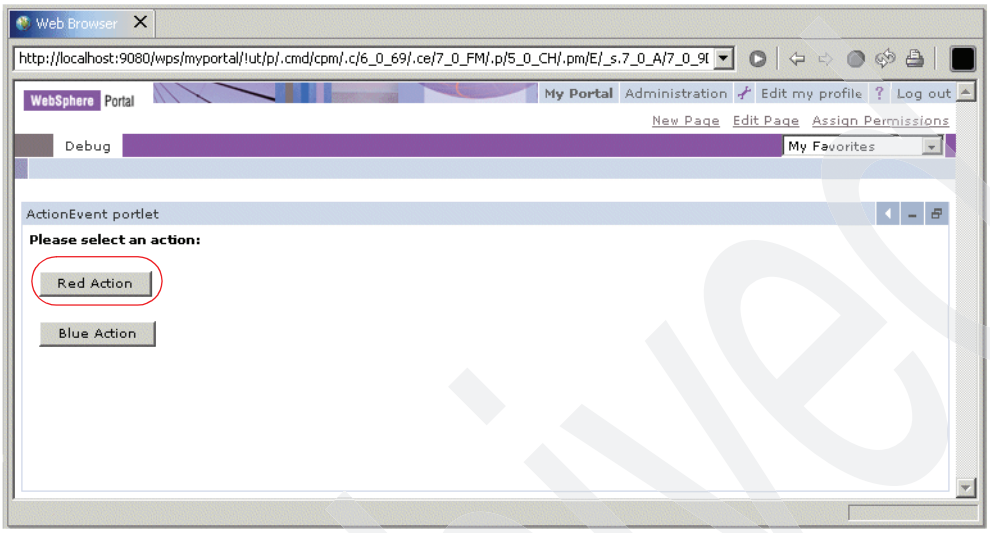


Figure 7-12 Creating an action and broadcasting the message

3. You will see the value shown in both the ActionEvent portlet and the MessageReceiver portlet.

   Note: In summary, you have seen how the portlet API implements message events which can be useful for passing data between portlets that need to be notified of other portlet's actions and events. This is a very useful feature of the API when building portlet applications that contain multiple portlets.

Archive

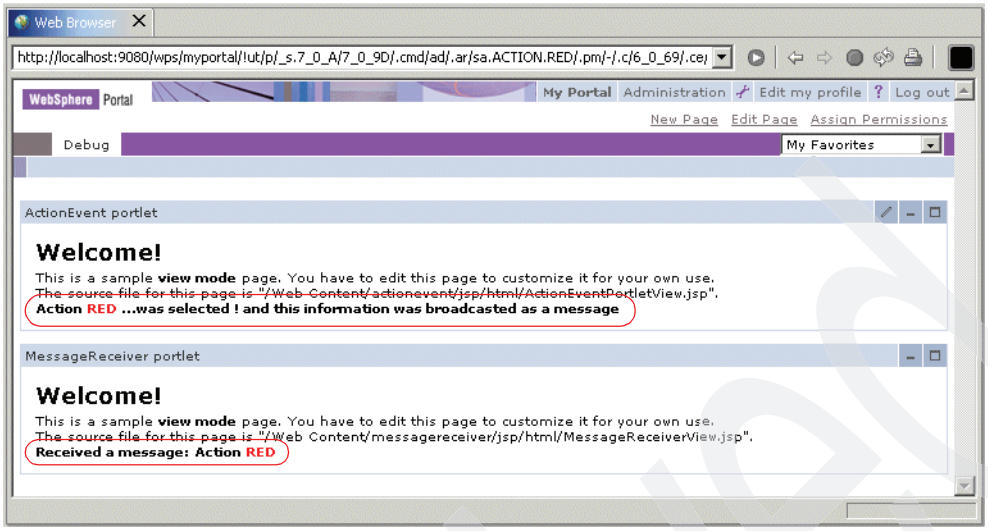


Figure 7-13 Red action and Red message broadcast

4. You can enter Edit mode again and select the **Blue Action** button. The results will again be displayed accordingly.

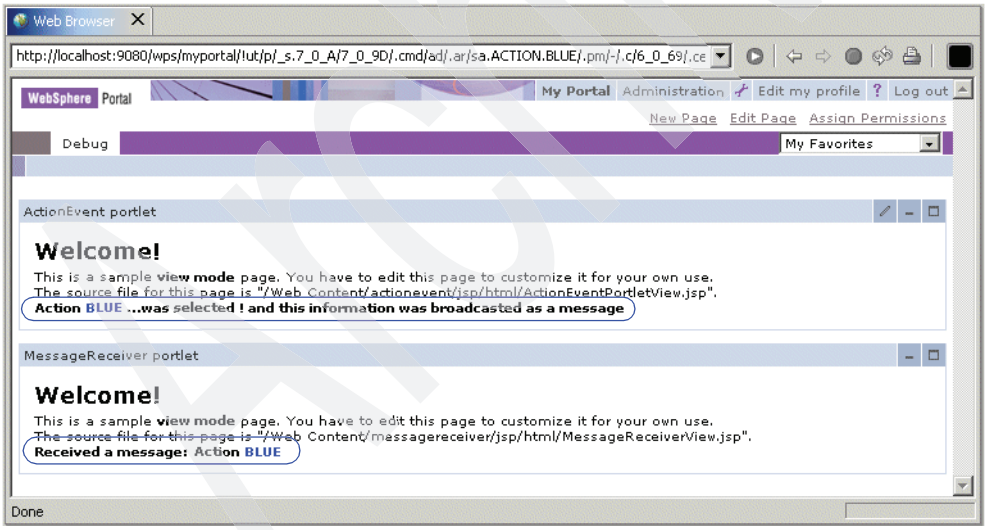


Figure 7-14 Blue action and message broadcast

7.7 Broadcasting messages

In this section, we show how to send a broadcast message to all portlets on a page that have implemented the MessageListener interface. For example follow these steps:

1. Create a new portlet application project:
   a. Select **File -> New -> Portlet Application Project**.
   b. Enter `Message` as the project name and in the event handling page, check only the option **Add message listener**.
   c. Click **Finish**.
   d. The new project has been added to DefaultEAR project; to be sure, open the application.xml file located in /DefaultEAR/META-INF/ folder. The Module tab should include the ActionEvent.war and Message.war. If there are more applications, remove them.
2. Open the MessagePortlet.java file located in the /Java Source/message/ folder and modify the messageReceived() method to receive the broadcasted message. You only need to check for a message of type DefaultPortletMessage, which is the type of message sent by ActionEventPortlet. Once the message is extracted, it will set the text of this message as an attribute into the portlet request.

Example 7-11 MessagePortlet.java

```
public void messageReceived(MessageEvent event) throws PortletException {
    if( getPortletLog().isDebugEnabled() )
        getPortletLog().debug("MessageListener - messageReceived called");
    // MessageEvent handler
    PortletMessage msg = event.getMessage();
    // Add PortletMessage handler here
    if( msg instanceof DefaultPortletMessage ) {
        String messageText = ((DefaultPortletMessage)msg).getMessage();
        // Add DefaultPortletMessage handler here
        PortletRequest request = event.getRequest();
        request.setAttribute("message", messageText);
    }
    else {
        // Add general PortletMessage handler here
    }
}
```

3. . Modify the MessagePortletView.jsp to display the message to the user.

Example 7-12 MessagePortletView.jsp

```
.....
<H3 style="margin-bottom: 3px">New application.</H3>
<br>
<% if ( request.getAttribute("message") == null ) { %>
   <B>No message has been received from another portlet application.</B>
<% } else { %>
   <B>Message received: <%= (String)request.getAttribute("message") %></B>
<% } %>
.....
```

4. Because you have added a new application, you have to restart the server. Restart the server and then right-click the **ActionEvent** project or **Message** project and choose **Run on Server** to test the application.
5. You will see that the internal Web browser brings up the tree portlets on your screen, as shown in Figure 7-15.

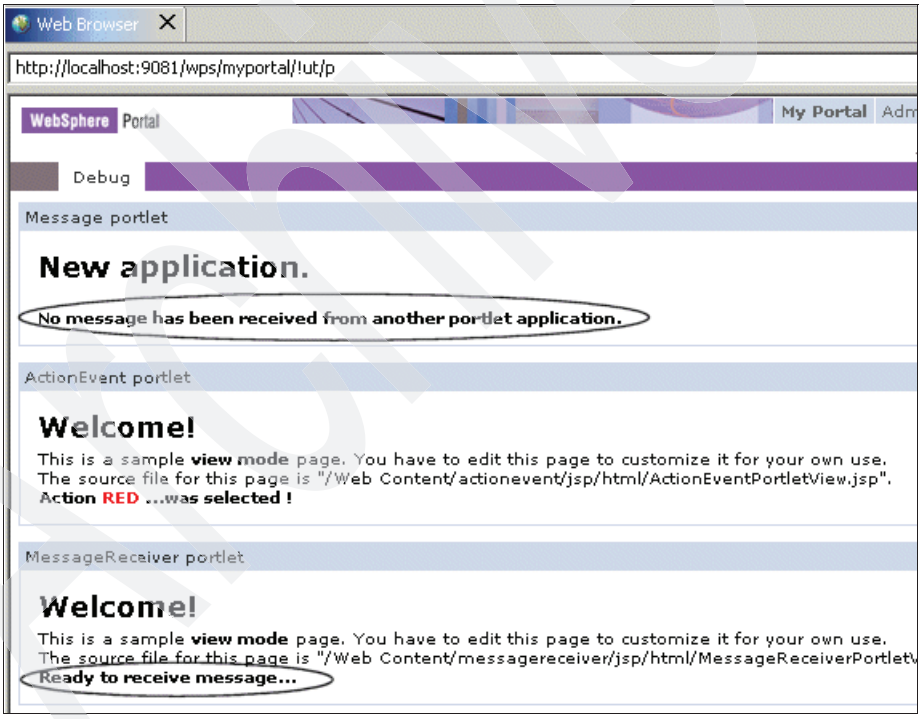


Figure 7-15 Before sending a broadcast message to all portlets

6. Go to the Edit mode of the ActionEvent portlet and click the **Red action** button; now both portlets (the portlet in the same application of

ActionEventPortlet and the portlet in the other application) display the message.

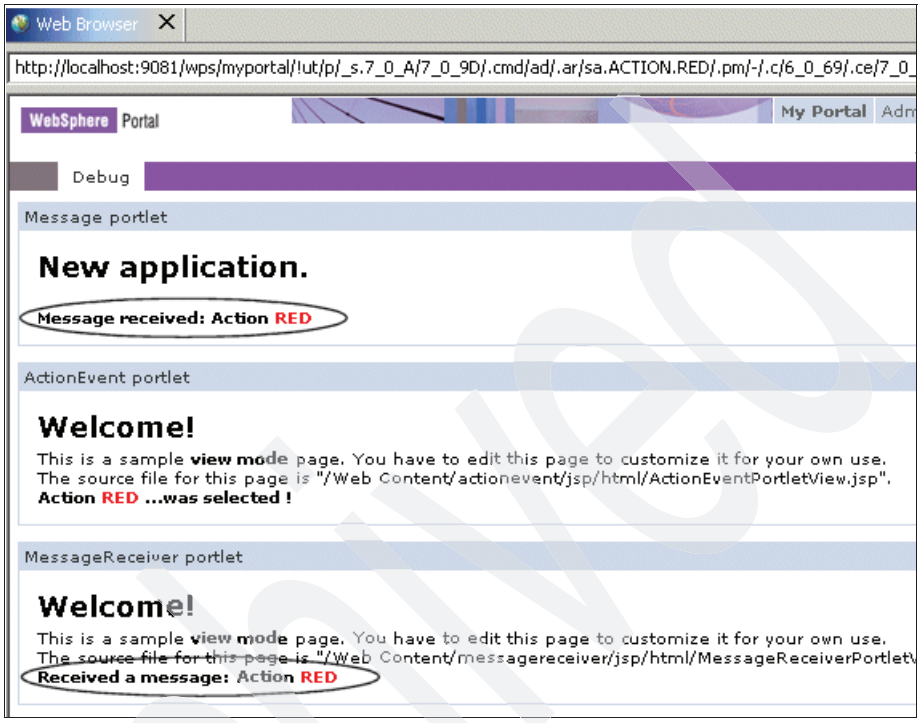


Figure 7-16 After sending a broadcast message

Sending a message to a portlet in a different application

Now modify the actionPerformed() method of ActionEventPortlet.java class to send a message to a specific portlet in a different application project.

1. Open the ActionEventPortlet.java file located in the /Java Source/actionevent/ folder of ActionEvent project. In the actionPerformed() method, when the action received is `ACTION_RED`, send the message only to the portlet called Message portlet.

Archived

Example 7-13 actionPerformed() method - ActionEventPortlet.java

```
        if(actionString.equalsIgnoreCase(ACTION_RED)){
           .......
           // Send a portlet message
           PortletMessage message = new DefaultPortletMessage(value);
           try{
           this.getPortletConfig().getContext().send("Message portlet",message);
           }catch (AccessDeniedException e){}
.....
```

2. Save the change and run the application. Now it is not necessary to restart the server; you only have to close the browser and run the project. Go to the Edit mode of the ActionEvent portlet and click the **Red action** button; now only the portlet called Message portlet receives the message.

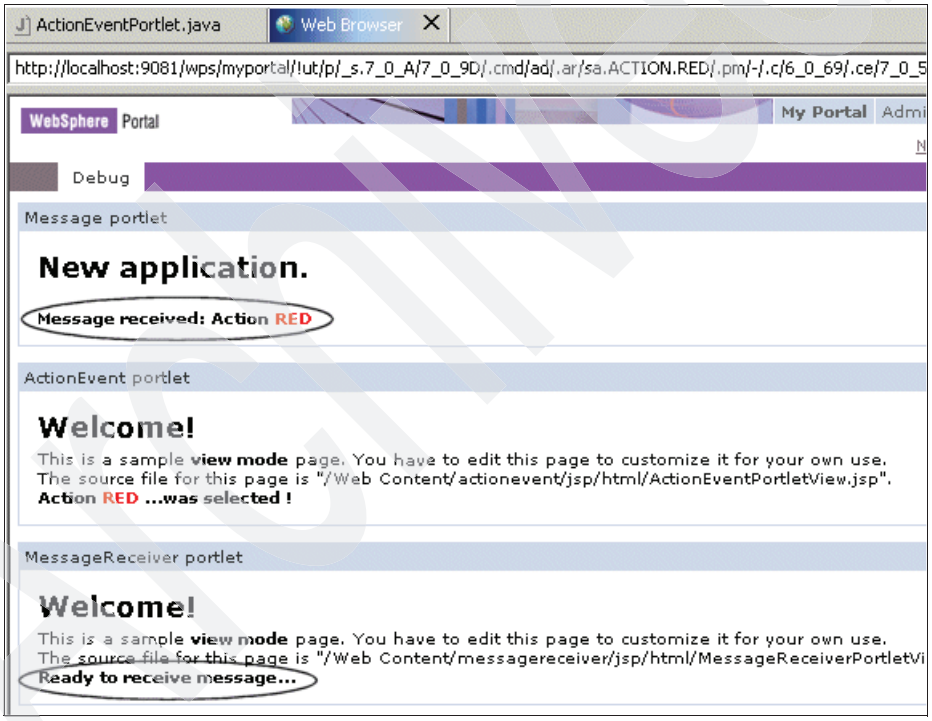


Figure 7-17 After sending a message to a specific portlet

Archived

8

National Language Support (NLS)

In order to make a portal accessible and attractive to a wider audience, it is necessary to provide the portal and its components in multiple languages. At a minimum, you should consider supporting the group 1 languages listed on Table 8-1 on page 251.

The WebSphere Portal architecture makes it easy and efficient to provide an NLS enabled portal. The enablement can be performed during development, deployment or runtime with the proper design decisions up front.

This chapter will guide you through several approaches to NLS enablement.

- Using resource bundles
- Translating whole resources
- NLS administration
- Working with characters
- NLS best practices

© Copyright IBM Corp. 2004. All rights reserved.

8.1 Resource bundles

A resource bundle is a simple text file that contains key-value pairs. The key is used by a Java class to retrieve a locale-specific value. To provide support for a new locale, you need only create a new resource bundle with the same key names and translated values.

Example 8-1 demonstrates a base resource bundle. Example 8-2 demonstrates the resource bundle translated for Spanish. Notice the key names do not change, only the value is translated.

Example 8-1 NLSExample.properties resource bundle

```
welcome = hello
goodbye = goodbye
message = This is the NLSExample portlet
```

Example 8-2 NLSExample_es.properties resource bundle

```
welcome = hola
goodbye = adiós
message = éste es el portlet NLSExample
```

The file name of the resource bundle is very important. The file must of type *properties*. All translated copies of the default resource bundle must include the locale in their title. This is illustrated in Figure 8-1.

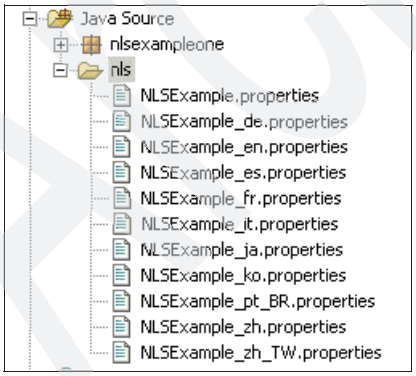


Figure 8-1 Translated resource bundles

The name is important because the Portal will locate the appropriate bundle for you based on the locale you provide. You need only provide the base name of

Archived

the bundle and it will append the appropriate locale. The Group 1 locales are listed in Table 8-1.

Table 8-1 Group 1 languages

Locale Code	Language
ar	Arabic
cs	Czech
da	Danish
de	German
el	Greek
en	English
ru	Russian
sv	Swedish
es	Spanish
tr	Turkish
fi	Finnish
fr	French
zh	Simplified Chinese
zh_TW	Traditional Chinese
hu	Hungarian
it	Italian
iw	Hebrew
ja	Japanese
ko	Korean
nl	Dutch
no	Norwegian
pl	Polish
pt	Portuguese
pt_BR	Brazilian Portuguese

Archived

8.1.1 Creating resource bundles in WebSphere Studio

The resource bundles need to be created in the Java Source directory as illustrated in Figure 8-1 on page 250. Though not required, as a matter of good practice, you should place the files in a dedicated directory such as nls. To create a new resource bundle in WebSphere Studio, open the navigator view in the portlet perspective. Locate the Java Source directory of the portlet you are enabling and right-click. From the context menu, select **New -> Other**.

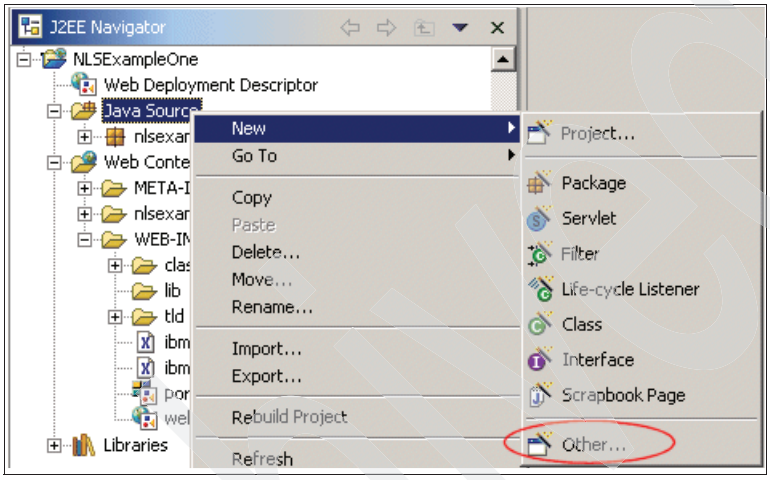


Figure 8-2 Creating new folder.

Select **Simple** in the left panel and **Folder** in the right panel and click **Next**.

Archived

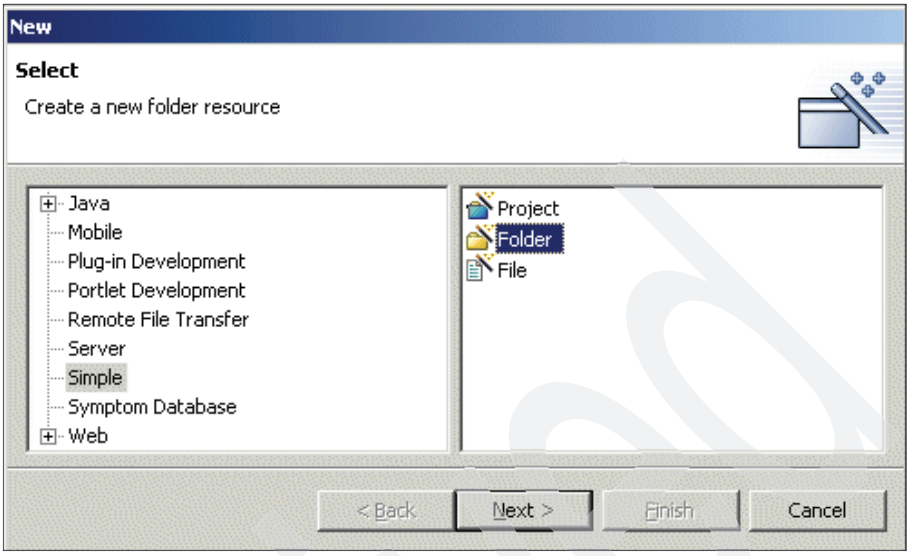


Figure 8-3 Creating nls folder

Enter the name of the new folder, typically `nls` as shown in Figure 8-4.

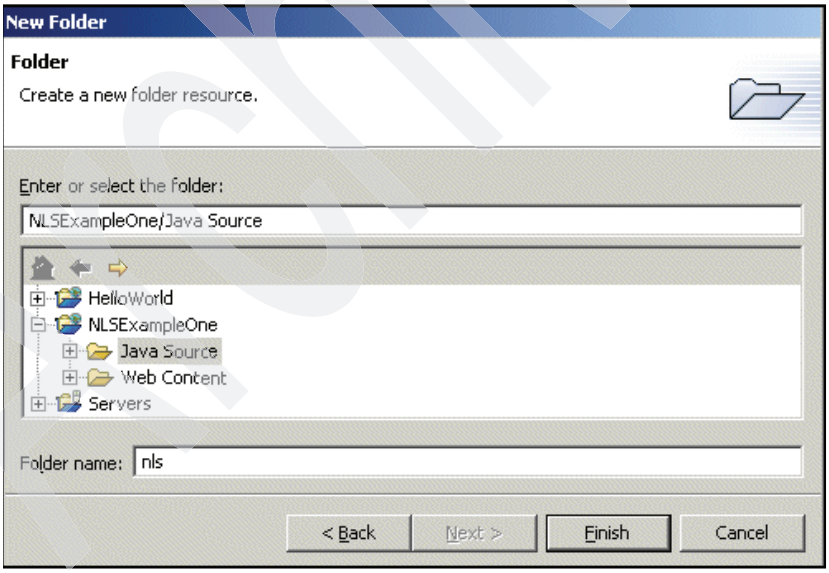


Figure 8-4 Creating the nls folder in WebSphere Studio Application Developer

In the nls folder, you need to create the default properties files. Select the **nls** folder and right-click. From the context menu, select **New -> Other -> Simple ->**

File. Be sure the correct directory is selected and enter the name of the default properties file as illustrated in Figure 8-5. Do not include any language codes in the name, or include any spaces in the name of the resource bundle.

When you are done, double-click the properties file in the navigator to open the simple text editor. Using the text editor, define your keys and the default values, such as those shown in Figure 8-1 on page 250. Use **CTRL-S** to save the file.

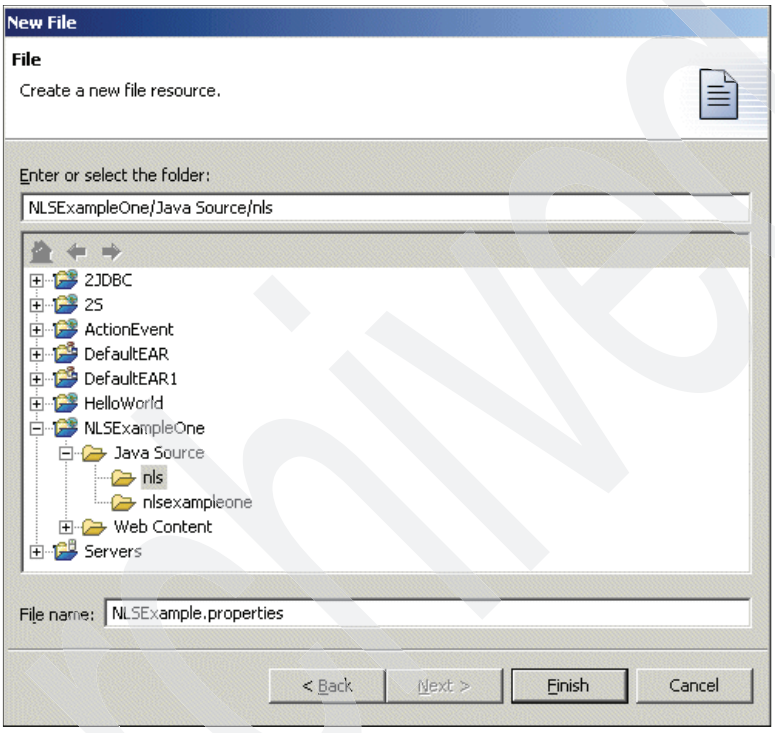


Figure 8-5 Creating the default properties bundle

Tip: If you create your nls folder directly in the Java Source subdirectory, WebSphere Studio Site Developer deletes this folder when it rebuilds the project.

8.1.2 Translating resource bundles

Once you have defined your default resource bundle with all the keys that will be used by the portlets and JSPs in your application, you must provide translations. It is possible to use the copy functionality in WebSphere Studio. However, there are several reasons you may choose not to. It is a cumbersome process in that

simple CTRL-C commands are not recognized when copying whole files. Also, renaming requires a selection from the context menu. While these may seem trivial issues, when creating dozens of resources bundles, they can be frustrating.

Instead, you may find it easier to work directly with the source files on the file system. Locate the directory containing the current workspace. You can obtain this path by right-clicking the portlet application and selecting **Properties** from the context menu. The Info option will display the file system location of the application. This is illustrated in Figure 8-6.

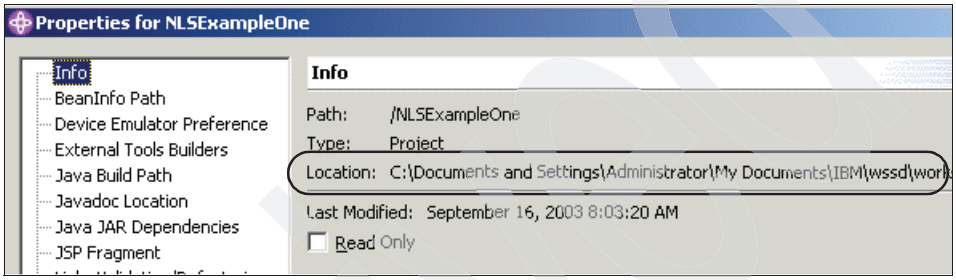


Figure 8-6 Locating the application on the file system

Open the directory containing the application and use the normal system copy/paste and rename functionality to create the new resource bundles. Each new bundle should have a unique locale appended. In practice, you may at development time only have the default and English properties files. This same approach can later be used to import translated files received from an outside source.

Once you have created the bundles you want, you need to make them available in the WebSphere Studio environment. To do this, simply select the **nls** folder, right-click and select **Refresh** as shown in Figure 8-7 on page 256.

Archive

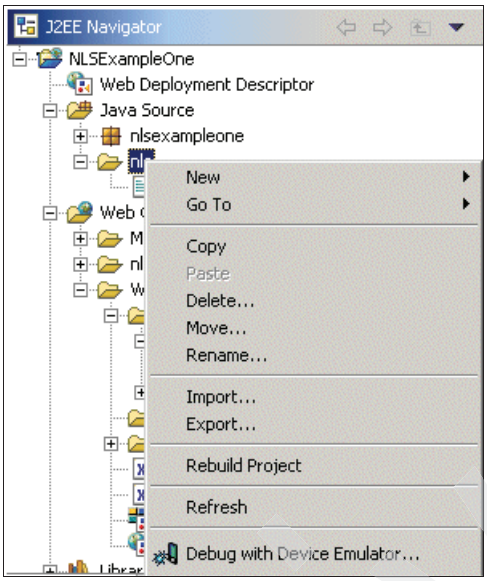


Figure 8-7 Loading resource bundles into WebSphere Studio Site Developer

When you are done, the folder should appear as in Example 8-1 on page 250, depending of course on the number of languages you choose to support.

8.1.3 Accessing resource bundles in portlets

If you are printing out content directly from the portlet, you can use the portlet API to access the resource bundles quite easily. Most of your development will adhere to a good MVC approach; you can use this approach for setting the title, predefining parameters in a PortletURI or if you are providing some content via the beginPage or endPage methods.

The resource bundle is accessed via the PortletContext object's getText method as displayed in Example 8-3.

Example 8-3 getText API

```
PortletContext.getText("Bundle Base Name", "Key", Locale)
```

- `Bundle Base Name`: the first parameter indicates the base name of the resource bundle. The name includes the path relative to the classes directory as shown in Example 8-4 on page 257. The name does not specify the locale suffix or the properties file type. If the base file name cannot be found, or the key is not present in the properties file, a PortletException will be thrown.

Archived

- `Key`: this parameter maps to a key value in the properties file. If the key is not found, a PortletException is thrown.
- `Locale`: this is used by the Portal to create the complete resource bundle name. You are free to use any locale you like but to ensure the user's locale is returned, the code in Example 8-4 works well. The getLocale method returns the preferred locale for the user. The Portal Server determines the locale by first retrieving the user's preferred language set during registration. If the preferred language is not set, the locale is retrieved from the accept-language header supplied by the client.

Example 8-4 Accessing resource bundles via the API

```
getPortletConfig().getContext().getText("nls.NLSExample", "welcome",
request.getLocale());
```

8.1.4 Accessing resource bundles in JSPs

When you employ a well designed MVC approach to your portlet development, the vast majority of NLS enablement work will need to take place in the view space. This section will guide you through providing locale-specific strings in a JSP. Section 8.2, "Translating whole resources" on page 258 will guide you through providing a unique JSP for each locale you choose to support.

To access resource bundles in JSP, you need to include the JSP Standard Tag Library. Right-click in your portlet application, select **Properties** and then **Web**. In Available Web project features, check the **JSP Standard Tag Library** and click **OK**. Your JSP files can access resource bundles in two ways, as shown in the following examples.

Example 8-5 Accessing resource bundles in a portlet JSP

```
<%@ taglib uri="http://java.sun.com/jstl/fmt" prefix="fmt" %>
<fmt:setBundle basename="nls.NLSExample"/>
<fmt:message key="message"/>
```

Example 8-6 Accessing resource bundles in a portlet JSP

```
<%@ taglib uri="http://java.sun.com/jstl/fmt" prefix="fmt" %>
<fmt:bundle basename="nls.NLSExample">
<fmt:message key="message"/>
</fmt:bundle>
```

As with specifying bundles in portlet code, the bundle name here must include the package name relative to the classes directory.

If the key cannot be located in the properties file, you will see the key written between question marks.



Figure 8-8 Key not found in a properties file

8.2 Translating whole resources

If the entirety of your JSP requires translation, you may find programmatically accessing resource bundles impractical. In practice, you will find that help JSPs, for example, contain little or no code and as such can be completely translated without incurring the runtime expense of NLS enablement.

WebSphere Portal facilitates this approach by allowing you to organize translated files in a predictable directory structure. Portal will then take responsibility for locating the correct file at runtime. This facility is also available for multiple markup support. Your directory structure should reflect Figure 8-9 on page 259.

Archived

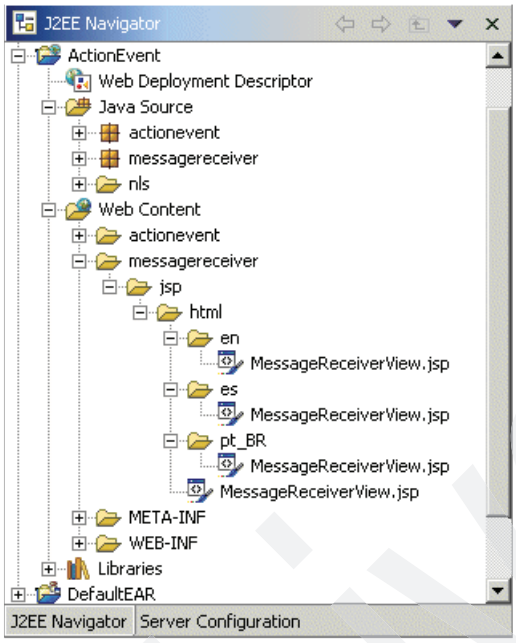


Figure 8-9 NLS directory structure

Each locale you support must have its own folder and contain whatever fully translated resources you want the portal to serve. If the portal cannot find the requested resource in the appropriate directory, it will attempt to locate the default by searching one level higher. If no default resource is located up the directory tree, an exception is thrown.

To retrieve the translated resource, simply use the include method of the PortletContext object. Do not specify the NLS directory structure. This code is illustrated in Example 8-9. You should notice that there is in fact nothing unique about calling a translated JSP and calling the simple JSPs. All the work is done by the portal.

Example 8-7 Including translated JSP files

```
getPortletConfig().getContext().include("/messagereceiver/jsp/MessageReceiverVi
ew.jsp", request, response);
```

Archived

8.3 NLS administration

Certain aspects of NLS enablement can be configured via the Administration windowin WebSphere Portal. While this section will guide you through these features, bear in mind that the administration does not replace the developer's responsibility for designing and incorporating NLS enablement.

8.3.1 Portlet NLS administration

The Manage Portlets page allows you to set locale-specific titles for portlets. You cannot add support for new locales. Only locales specified by the portlet.xml deployment descriptor can be adjusted. Furthermore, you cannot change the default locale specified by the portlet.xml. To access the titles, log in as the administrator and navigate to the Manage Portlets page in the Administration window, as illustrated in Figure 8-10.

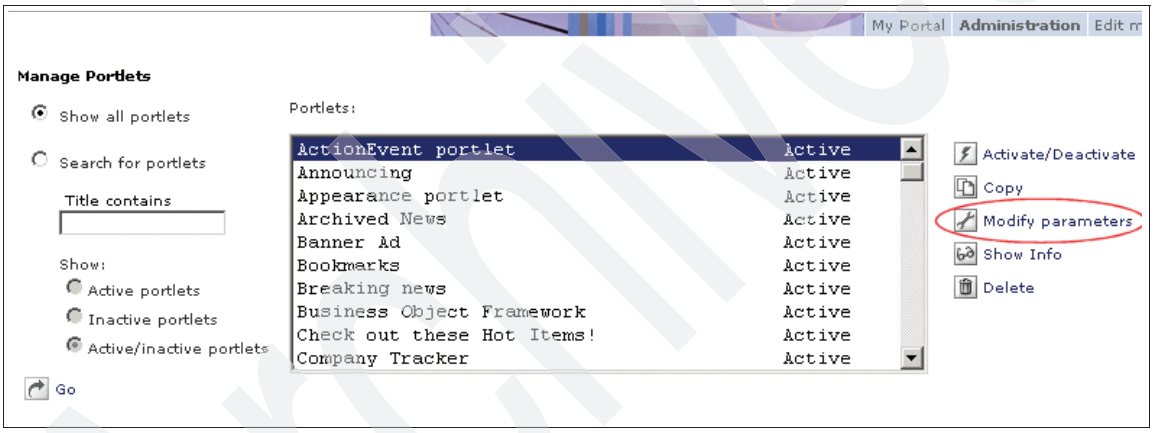


Figure 8-10 Working with portlets

Locate the portlet you want to work with and select the **Modify parameters** link. By default, the resulting window, shown in Figure 8-11 on page 261, will only display the Group 1 languages indicated for support by the portlet.xml. To add new languages in the administration space, you need to add that language to the file language.properties and to the language table in the Web Member Services database if you use the Member Subsystem. Then you have to insert resource bundles, with an appropriate name, in the directory located at /wp_root/shared/nls. The directory where you have to store the JSP will depend on how the portal locates your JSP for rendering its content.

Archived

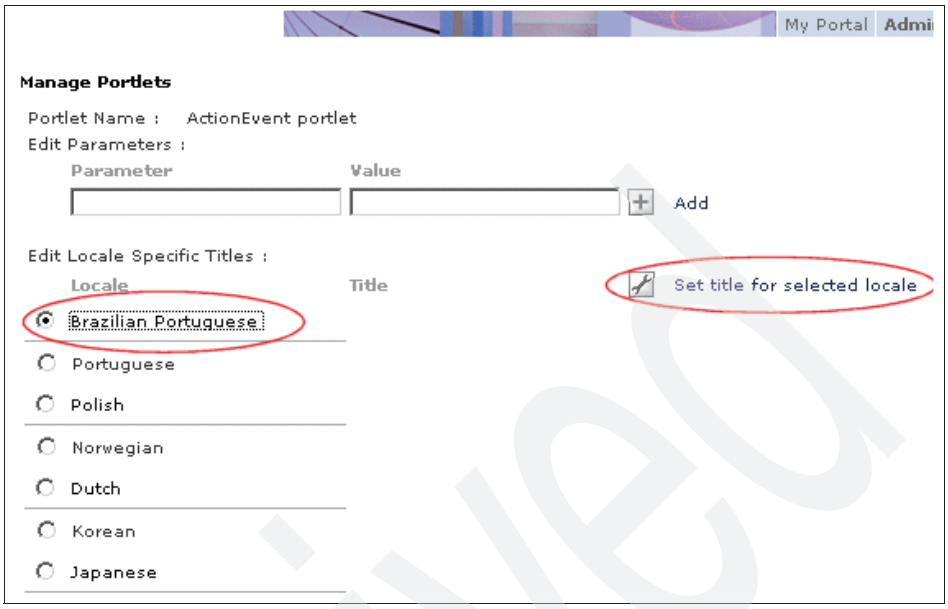


Figure 8-11 Supported languages of the selected portlet

To change the title specified by the portlet.xml, select the portlet's associated radio button and select the **Set title for selected locale** link as shown in Figure 8-11. Enter the new title in the resulting window shown in Figure 8-12.

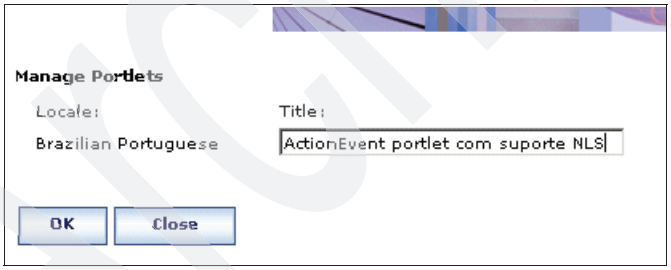


Figure 8-12 Setting the locale-specific title

You may have noticed that only the title can be adjusted via the Administration window.

Again, it is important to stress that by default, the administrator will only be able to set the title for Group 1 locales explicitly specified in the portlet.xml. To add support for a locale in the portlet.xml in WebSphere Studio, open the portlet.xml editor and select the concrete portlet you wish to work with. This is illustrated in Figure 8-13 on page 262.

Archived

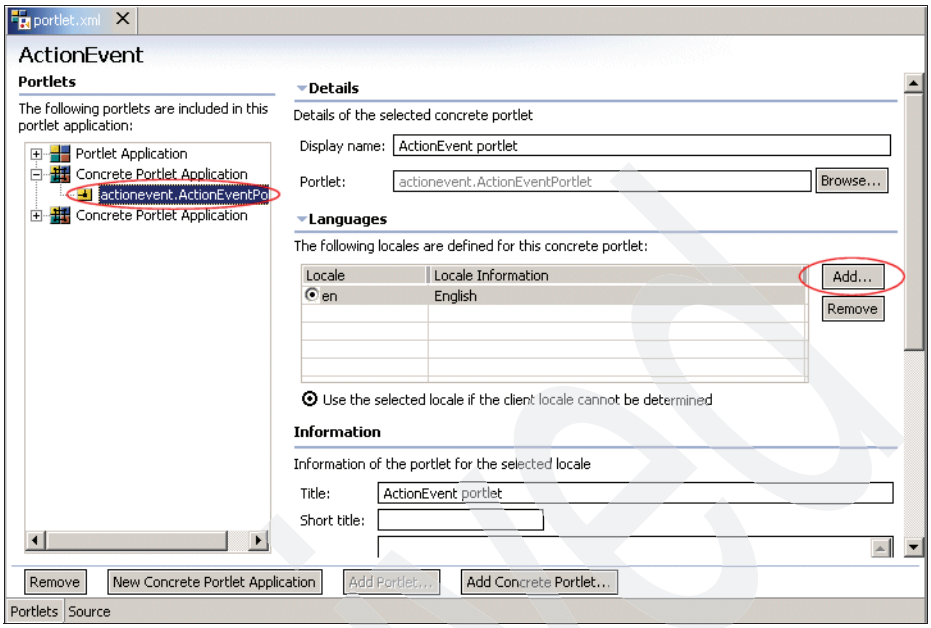


Figure 8-13 Adding locale support to a concrete portlet

To add a new locale, select the **Add** button in the locale section as shown in Figure 8-13. In the resulting dialog, you can select the locale from the drop-down list or enter the country code if you know it. This is illustrated in Figure 8-14. If the locale you choose is already defined in the portlet.xml, you will be prevented from adding it again.

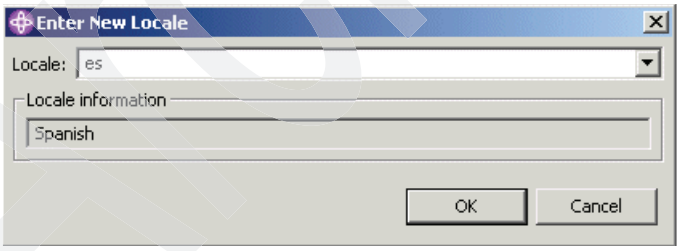


Figure 8-14 Specifying a new locale

All portlets must have a default language specified in the deployment description, otherwise the portlet cannot be installed.

8.3.2 Portal NLS administration

While it is the developer's responsibility to carefully consider the NLS support portlet applications will bring, the administrator is responsible for ensuring the Portal itself properly supports multiple languages.

Some of the basic settings for NLS enablement include setting page, theme and skin names properly, configuring or maintaining property files and incorporating support for new languages.

8.3.3 Setting NLS titles

To set locale-specific titles for a page, navigate to **Administration -> Portal User Interface -> Manage Pages**, then locate your page and click the **Edit Page Properties** icon. Display the advanced options as illustrated in Example 8-15.

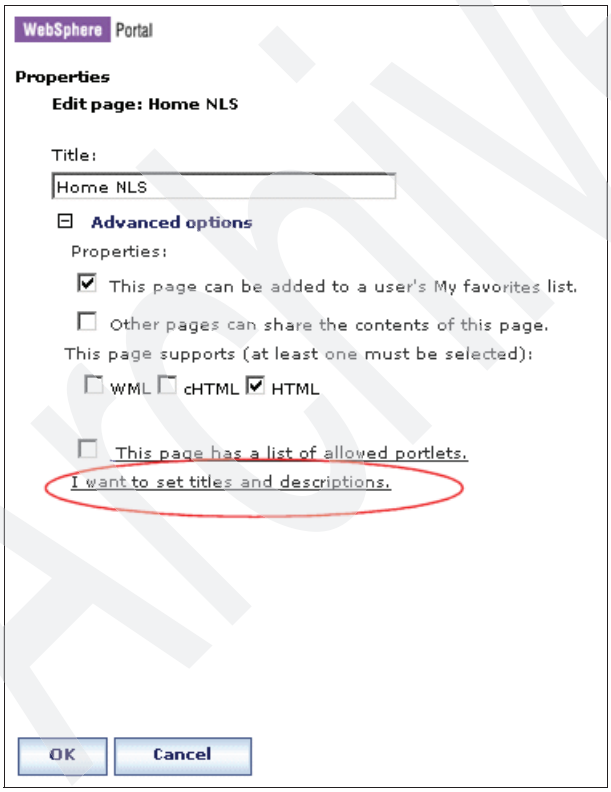


Figure 8-15 Editing page properties

Archived

In the resulting window, select the **Edit** icon of the language for which you want to set a title and/or description. Enter the new title and/or description and select **OK**.

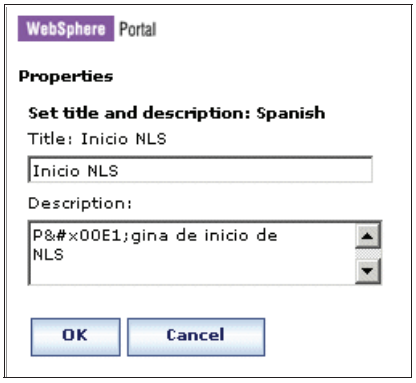


Figure 8-16 Using unicode values

Now you will see the new title and description in the list.

If you need to enter characters that you cannot generate with your keyboard, such as Japanese characters, written accents, etc., you will need to discover the unicode values and enter them using entity references as illustrated in Figure 8-16. The four character unicode values can be found at `http://www.unicode.org/charts`. The entity reference syntax is `&#x`***`0000`***`;` where ***`0000`*** represents the unicode character you want to display. If you have access to the interpreted unicode character, you can copy and paste it in the text field as well.\

8.3.4 Adjusting Portal resource bundles

There are several resource bundles that are used by the portal server to present locale-specific messages. Be aware that changes to a property file are not recognized until the portal is restarted. All the properties files listed below can be found in /wp_root/shared/app/nls.

- button.properties
- commonAdmin.properties
- problem.properties
- field.properties
- engine.properties
- CSRes.properties (not available in Enable)

Archived

- titlebar.properties
- registration.properties
- LocaleNames.properties
- pbruntime.properties
- virtual_principals.properties

8.4 Working with characters

Typically, it will not be the developer's responsibility to provide the translations necessary to provide NLS enablement. Once your base resource bundles and/or static files have been created, the translation process should be completed by a language expert. However, during development, you may need to include some characters that cannot be created by the typical English keyboard. To include these characters, make use of the unicode mappings. Refer to `http://www.unicode.org` for the current character mappings.

Example 8-8 Using unicode in NLSExample_es.properties resource bundle

```
welcome = hola
goodbye = adi\u00F3s
message = \u00E9ste es el portlet NLSExample
```

8.5 NLS best practices

- Make the decision to use the API or translated resources early. This decision will play a large role in the design and development of your View components.
- Do not commit entirely to one approach. For example, it may make sense to translate your View JSPs at runtime and have your Help JSPs fully translated since they are simple text.
- Plan for NLS enablement from the beginning. Though you may not have access to the translated values during development, building the default resource bundle as you iterate through the development will make future NLS enablement much easier and virtually painless.
- Be conscious of character-locale ratios. If you are developing in English, be aware that translations into other languages such as German or Spanish may require more screen real estate. A number of API facilities are available for you to determine the current locale.

Archived

- Do not rely on an administration implementation of NLS. The NLS enablement facility for portlets is limited and there is no guarantee or check system. To implement dynamic NLS titles, consider implementing the PortletTitleListener interface and generating the title content via JSP or HTML files.
- Leave translations to language experts. With proper design, planning and construction of your portlet applications, there should be little to no effort involved in incorporating support for new languages.

8.6 Sample scenario: NLS bundles

In this sample scenario, NLS bundles will be created to support multiple languages. Once you have done this using WebSphere Studio, the JSP that delivers markup content in View mode for portlet MessageReceiver will be enhanced to access the NLS bundles.

The Portlet Messaging application sample scenario from Chapter 7, “Portlet messaging” on page 225 will be used as a base application to add NLS support. The scenario is illustrated in Figure 8-17 on page 267.

Archived

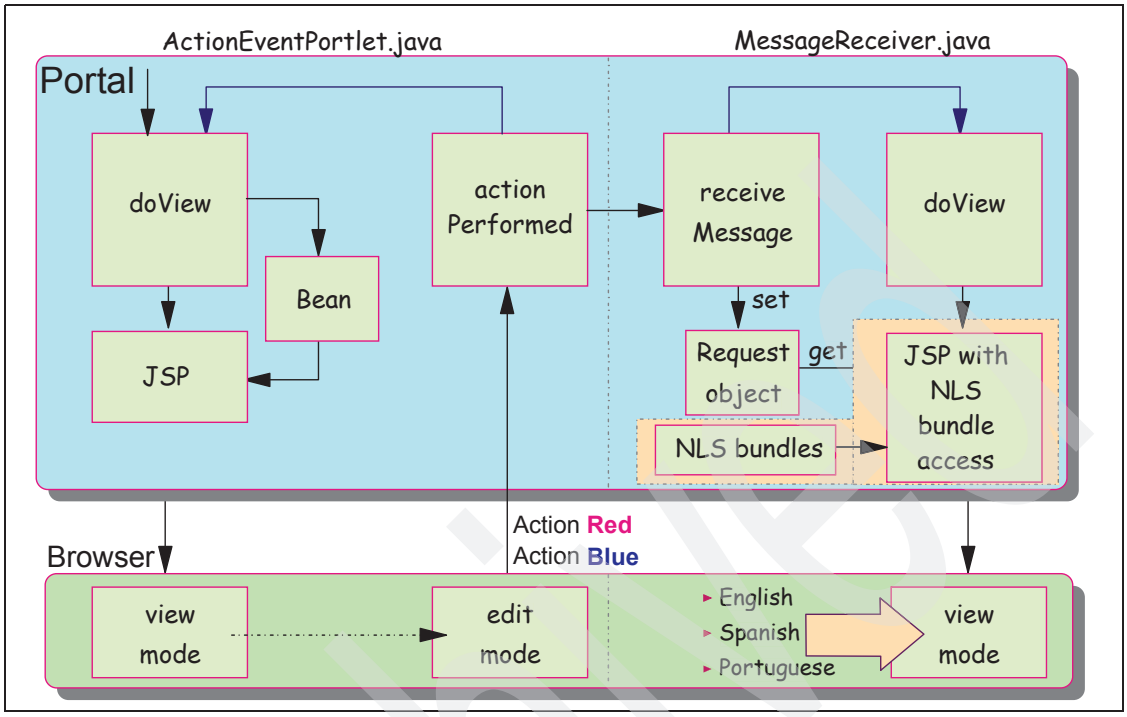


Figure 8-17 National Language Support (NLS) scenario

The resource bundle is accessed via the PortletContext object's getText method and you will need to provide the following:

- Bundle Base Name: the first parameter indicates the base name of the resource bundle. The name includes the path relative to the classes directory. The name does not specify the locale suffix or the properties file type. If the base file name cannot be found, or the key is not present in the properties file, a PortletException is thrown.
- Key: this parameter maps to a key value in the properties file. If the key is not found, a PortletException is thrown.

In addition, the locale is used by the Portal to select the proper language bundle. However, you cannot set this value when invoking NLS bundles from JSPs.

8.6.1 NLS bundles

In this section, you will use the sample scenario from Chapter 7, “Portlet messaging” on page 225. The portlet application will be enhanced to support NLS. Follow these steps:

1. If needed, start the IBM WebSphere Studio Site Developer. Click **Start -> Programs -> IBM WebSphere Studio -> Site Developer 5.0**.
2. You will create a new folder with the name nls to store the resource bundles. The following resource bundles will be imported into this folder:
   - NLSLab.properties (default)
   - NLSLab_en.properties (English)
   - NLSLab_es.properties (Spanish)
   - NLSLab_pt_BR.properties (Brazilian Portuguese)
3. Select your **ActionEvent/Java Source** folder.

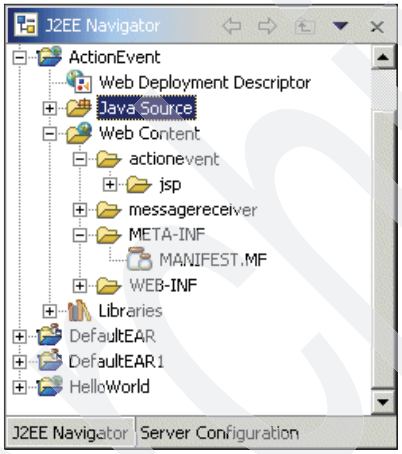


Figure 8-18 Select Java source to create an nls folder

4. Select **File -> New -> Folder**.

Archive

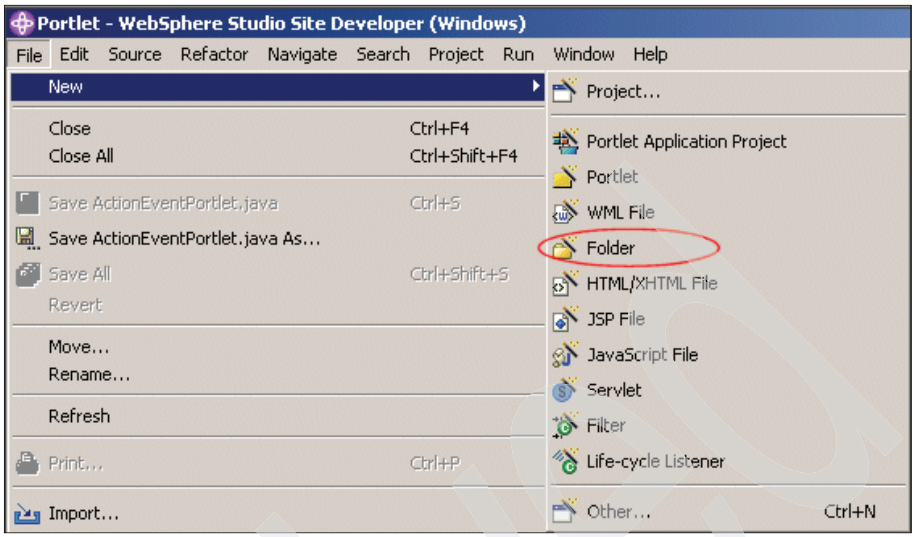


Figure 8-19 Select a new folder

5. Enter `nls` for the Folder name field then click **Finish**.

Archive

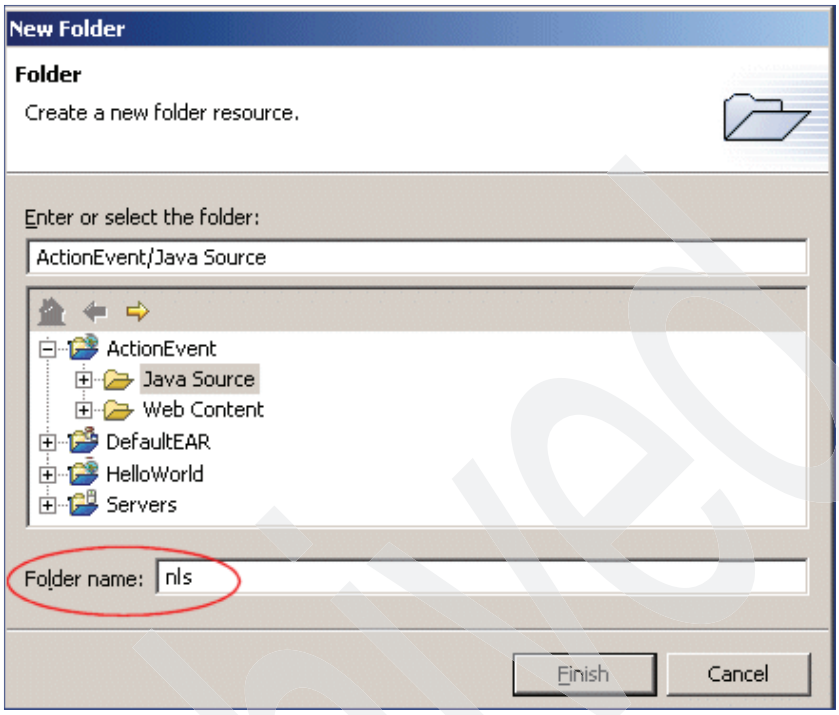


Figure 8-20 Enter a name for the new folder (nls)

6. Your directory structure should now look as shown in Figure 8-21.

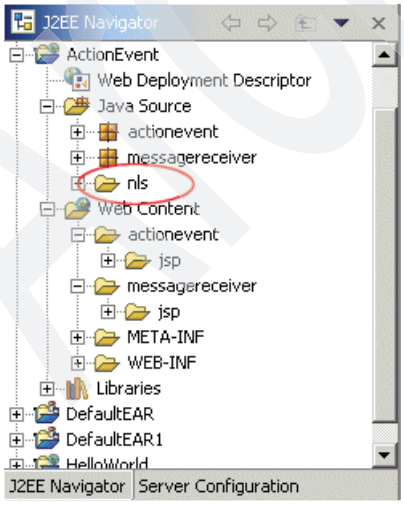


Figure 8-21 New nls folder

Archived

7. Select the new **ActionEvent/Java Source/nls** folder.
8. Click **File -> Import**.
9. Select **File System** and click **Next**. Browse to C:\LabFiles\NLSLab\Bundles.

   Note: The sample scenario included in this chapter requires that you download the sample code available as additional materials. See Appendix C, “Additional material” on page 543.
10. Select all four properties files and click **Finish** to import.

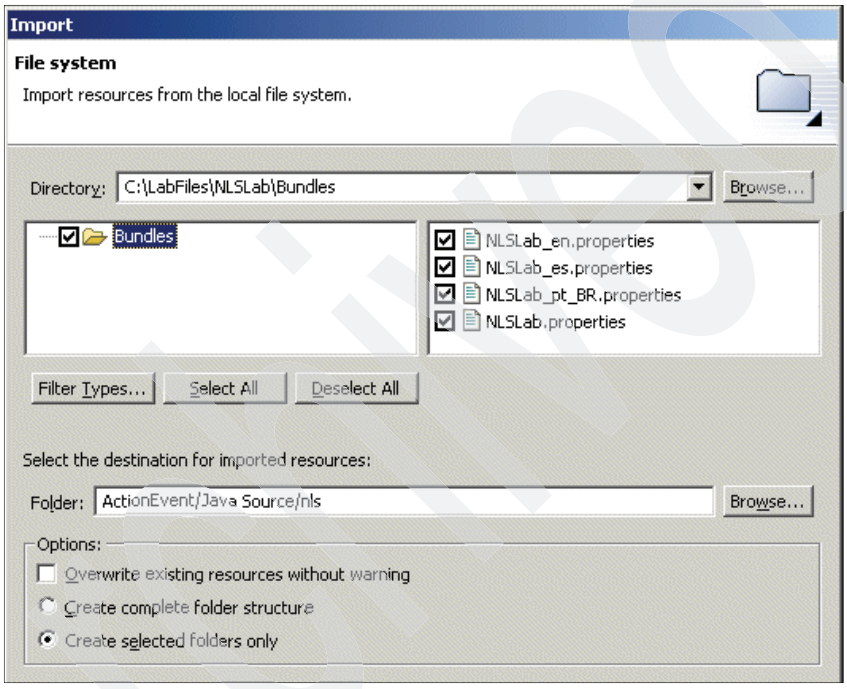


Figure 8-22 Importing the bundles

11. View the files in the nls folder by double-clicking them. Notice how they are structured.

Example 8-9 NLSLab.properties (default)

```
readystatus=Ready to receive message
receivedstatus=Received a message
viewmode=Operating in View mode
redColor=RED
blueColor=BLUE
```

Example 8-10 NLSLab_en.properties (English)

```
readystatus=Ready to receive message
receivedstatus=Received a message
viewmode=Operating in View mode
redColor=RED
blueColor=BLUE
```

Example 8-11 NLSLab_es.properties (Spanish)

```
readystatus=Listo para recibir mensaje
receivedstatus=Mensaje recibido
viewmode=Operando en modo de visualizacion
redColor=ROJO
blueColor=AZUL
```

Example 8-12 NLSLab_pt_BR.properties (Brazilian Portuguese)

```
readystatus=Pronto para receber mensagem
receivedstatus=Mensagem recebida
viewmode=Operando em modo de Visualização
redColor=VERMELHA
blueColor=AZUL
```

8.6.2 Accessing NLS bundles from JSPs

In this section, you will update the JSP MessageReceiverView.jsp to display NLS content based on the locale value (English, Spanish or Brazilian Portuguese). In general, modifications to the JSPs are necessary to allow them to display language-specific content. Follow these steps:

1. First you have to include the JSP standard tag library in your project as follows:
   a. Right-click the **ActionEvent** project.
   b. Select **Properties**.
   c. Select the **Web** option from the list and in the available Web project features.
   d. Check the **Include the JSP Standard Tag library** box.
   e. Click **OK**.

Archived

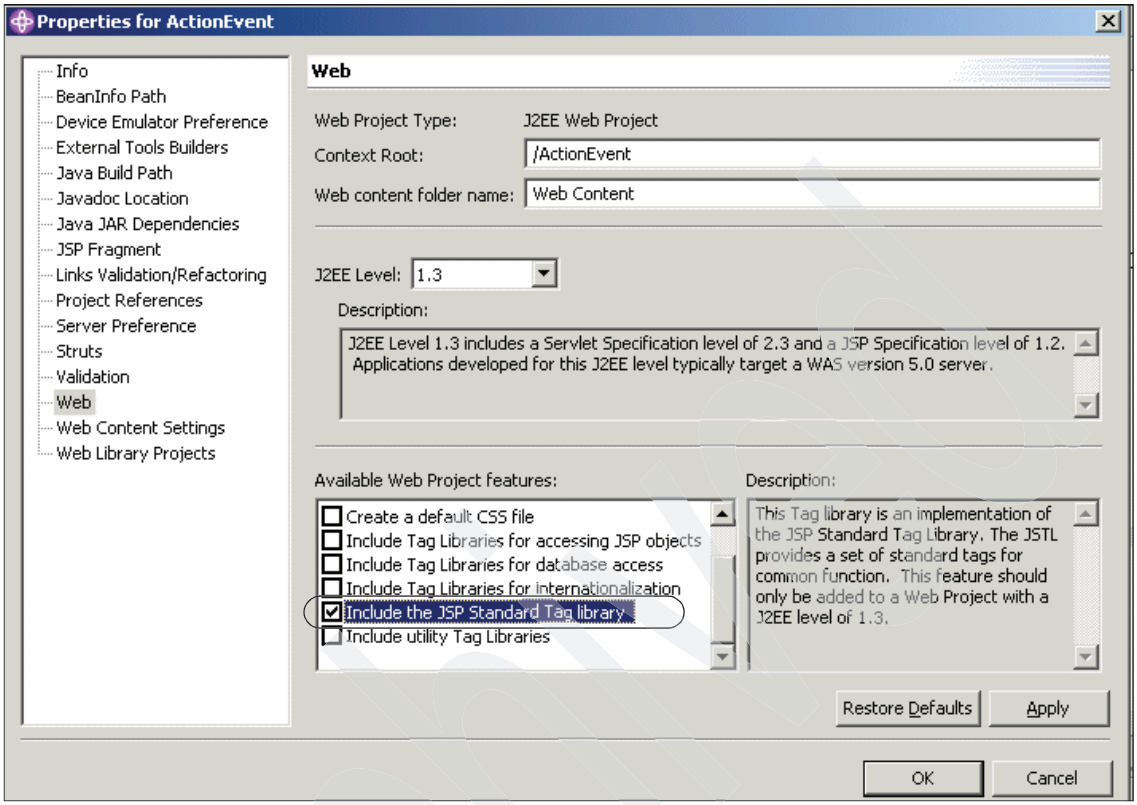


Figure 8-23 Include JSP Standard Tag Library

2. Open the MessageReceiverView.jsp file in WebSphere Studio Site Developer. This file is located in the /Web Content/messagereceiver/jsp/html/ directory.

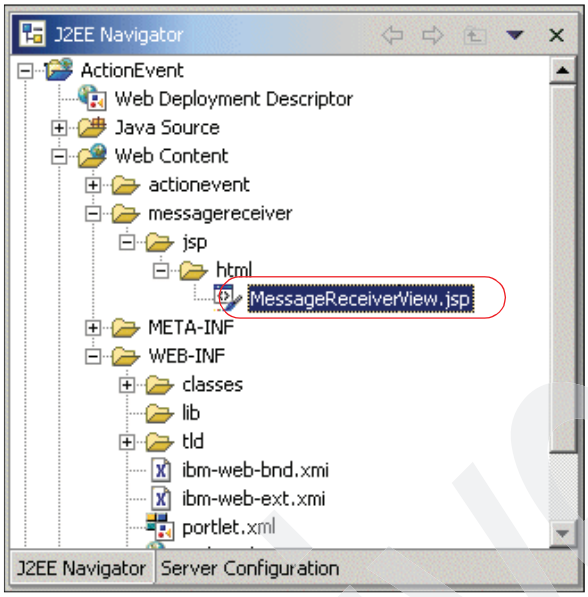


Figure 8-24 MessageReceiverView.jsp

3. In this JSP, you have the following text with static information:
   - `Ready to receive message`
   - `Received a message`
4. Add logic to display messages in the proper language. Updates to this JSP are highlighted in bold in Example 8-13.

Example 8-13 MessageReceiverView.jsp supporting NLS with bundles

```
<%@ page contentType="text/html" import="java.util.*, messagereceiver.*"%>
<%@ taglib uri="/WEB-INF/tld/portlet.tld" prefix="portletAPI" %>
<%@ taglib uri="http://java.sun.com/jstl/fmt" prefix="fmt" %>
<portletAPI:init/>

<DIV style="margin: 6px">

<H3 style="margin-bottom: 3px">Welcome!</H3>
This is a sample <B>view mode</B> page. You have to edit this page to customize
it for your own use.<BR>
The source file for this page is "/Web
Content/messagereceiver/jsp/html/MessageReceiverView.jsp".

<fmt:setBundle basename="nls.NLSLab"/>

<br>
```

Archived

```
<% if (request.getAttribute("MyMessage") == null) { %>
   <B><fmt:message key="readystatus"/> ...</B>
<% } else { %>
   <B><fmt:message key="receivedstatus"/>:</B>
   <B><%= request.getAttribute("MyMessage") %></B>
<% } %>

</DIV>
```

5. Select **File -> Save All** to save all your changes to the project.

8.6.3 Running the NLS scenario

In this section, you will run the portlet application messaging scenario now enabled for NLS.

1. Stop the Test Environment server so that next time, the new properties files will be used.
2. Right-click **ActionEvent** and select **Run on Server**.

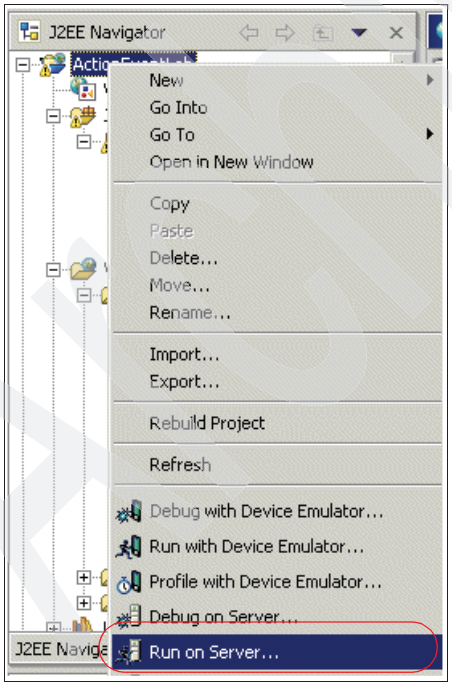


Figure 8-25 Running the NLS project

3. The message receiver portlet will now display its markup using NLS.

Archived

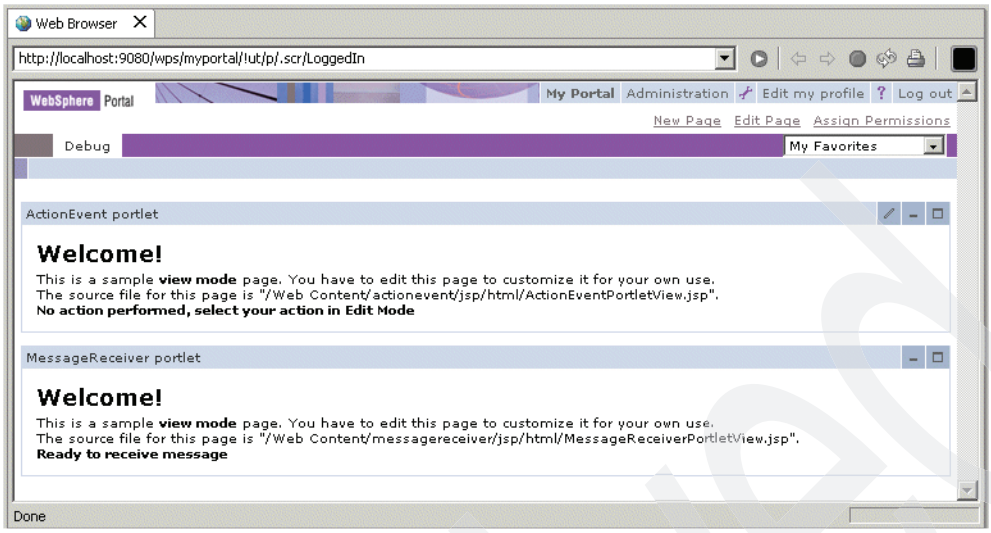


Figure 8-26 ActionEvent Portlet with no preferred language selected

4. Select a new locale value by clicking the **Edit my profile** icon to select a preferred language, as shown in Figure 8-27.

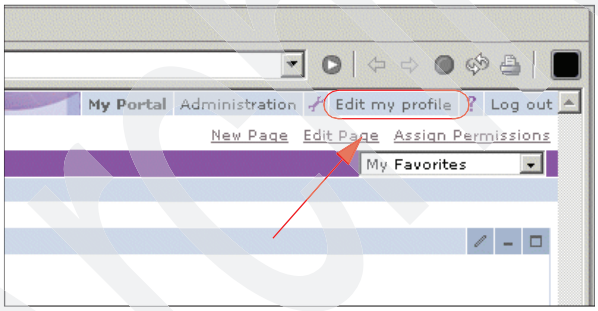


Figure 8-27 Edit my profile icon

For example, select **Brazilian Portuguese** as the preferred language for the wpsadmin user (default user for portlet development environment). It may be necessary to enter a first and last name before you can continue. Enter `wps` for both if this happens.

Abridged

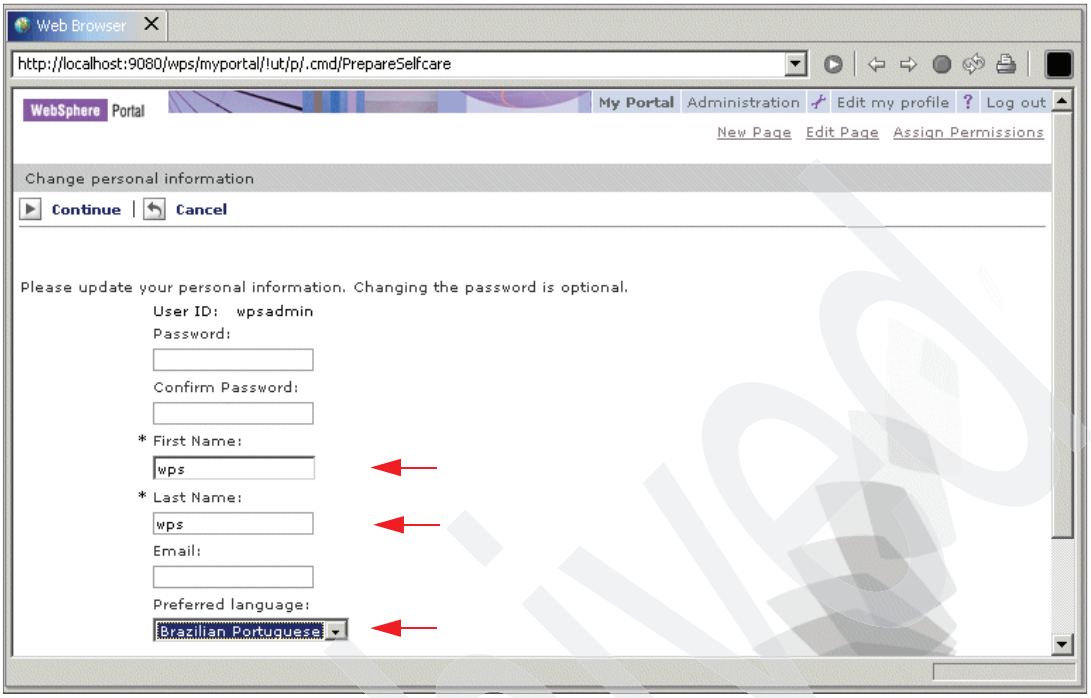


Figure 8-28 User profile

5. Click **Continue**. You will need to click **Continue** again to confirm your change and view the information displayed in the MessageReceiver portlet in Brazilian Portuguese.

   Note: You should also notice that when the language locale changes, the text of the WebSphere Portal menu at the top of the page also changes.

Archived

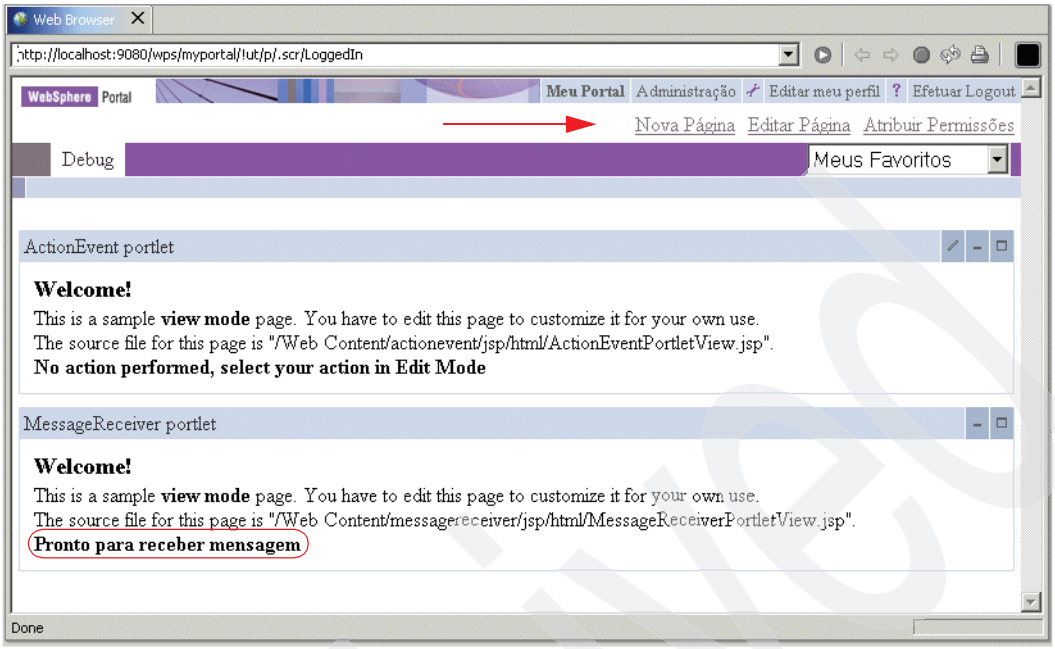


Figure 8-29 ActionEvent Portlet in View mode after action selected

6. Edit the user profile again and try **Spanish** as the new locale.

Archived

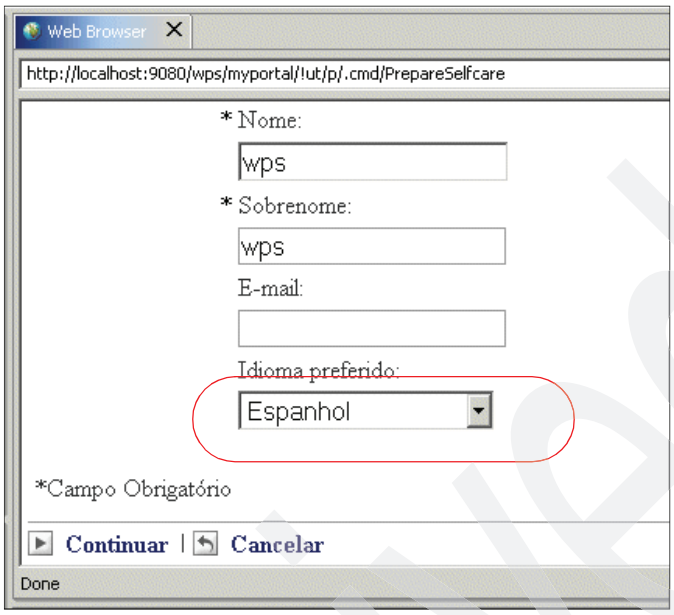


Figure 8-30 Selecting Spanish

The content you specified will display in Spanish.

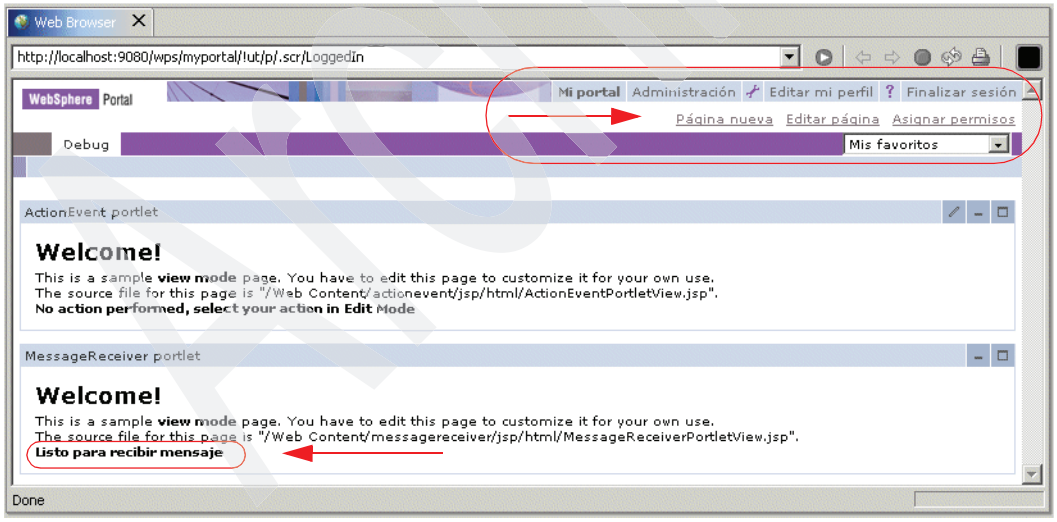


Figure 8-31 Content in Spanish

7. Edit the user profile again and try **French** (**Frances** in Spanish) as the new locale.

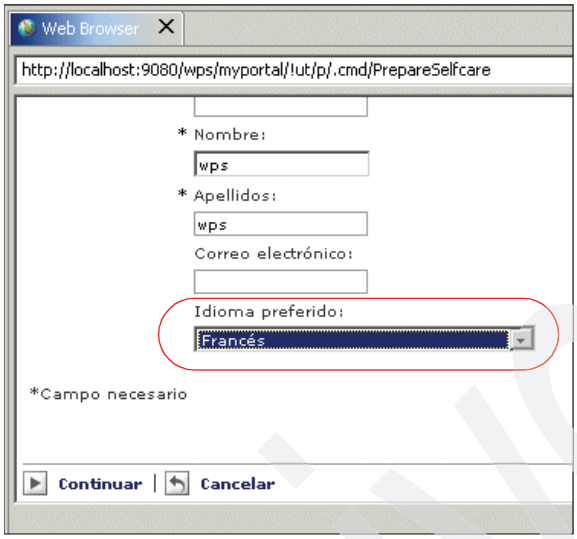


Figure 8-32 Selecting French as the language

8. Since French has not been enabled, Portal will use the default bundle. Your message will display in English (as is specified in the default bundle), but the WebSphere Portal menu at the top will display in French.

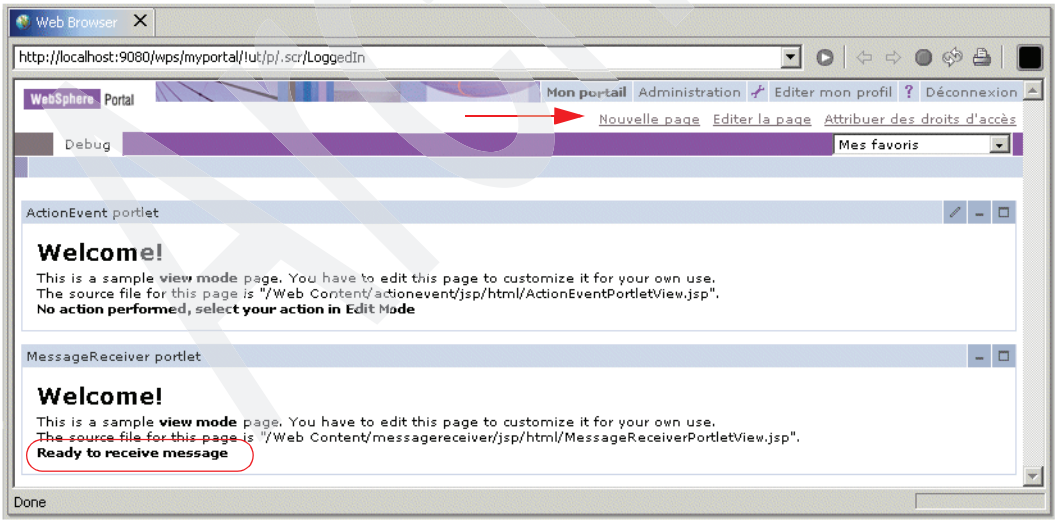


Figure 8-33 In French

9. To change your language back to English before you exit, click **Edit my profile** and select **English** (**Anglais** in French) as your language.

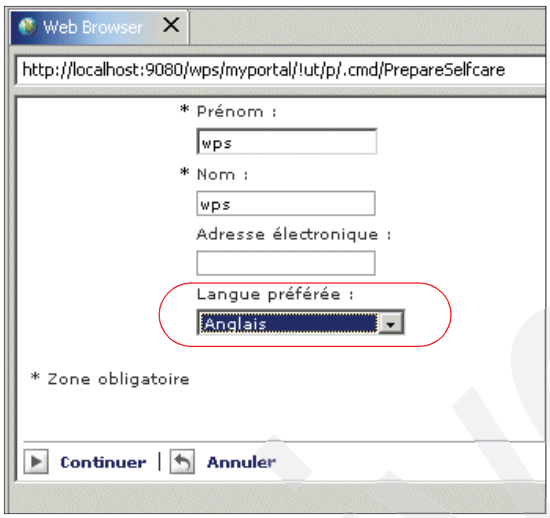


Figure 8-34 Back to English

8.6.4 Accessing NLS bundles in Java portlets

In this section, you will update the ActionEventPortlet.java file to display NLS content based on the locale value (English, Spanish or Brazilian Portuguese). You also need to add the new key-value pairs in the associated property file.

1. Open the file ActionEventPortlet.java for editing by double-clicking it. Next, you will update the code to display the color in the preferred language selected by the user. The resource bundle is accessed via the PortletContext object's getText() method. This method receives three parameters:
   a. Base name of the resource bundle, including the path relative to the classes directory and without the locale suffix or the properties file type.
   b. Key specified in the properties file.
   c. Locale.
2. Make the following highlighted updates in the actionPerformed() method.

Example 8-14 ActionEventPortlet.java

```
......
import java.io.IOException;
import java.util.Locale;

......
```

Archived

```
   public void actionPerformed(ActionEvent event) throws PortletException {
      if( getPortletLog().isDebugEnabled() )
         getPortletLog().debug("ActionListener - actionPerformed called");
      // ActionEvent handler
      String actionString = event.getActionString();

      //get the preferred locale for the user
      Locale loc = event.getRequest().getLocale();

      // Add action string handler here
      if (actionString.equalsIgnoreCase(ACTION_RED)) {
         //access the resource bundle via the PortletContext object's getText
method
         String red = getPortletConfig().getContext().getText("nls.NLSLab",
"redColor", loc);

         //create the string of HTML to be rendered
         String value = "Action <FONT color=\"#ff0000\">" + red + "</FONT>";

         //create a portlet request
         PortletRequest request = event.getRequest();
         ......................

      if (actionString.equalsIgnoreCase(ACTION_BLUE)) {
         //access the resource bundle via the PortletContext object's getText
method
         String blue = getPortletConfig().getContext().getText("nls.NLSLab",
"blueColor", loc);

         //create the string of HTML to be rendered
         String value = "Action <FONT color=\"#0000ff\">" + blue + "</FONT>";

         //create a portlet request
         PortletRequest request = event.getRequest();
.......
```

3. Open the resource bundles located in the Java Source/nls folder and make sure the following key-value pairs required for this scenario have been included in the properties files.

Example 8-15 NLSLab.properties (default)

```
redColor=RED
blueColor=BLUE
```

Archived

Example 8-16 NLSLab_en.properties (English)

```
redColor=RED
blueColor=BLUE
```

Example 8-17 NLSLab_es.properties (Spanish)

```
redColor=ROJO
blueColor=AZUL
```

Example 8-18 NLSLab_pt_BR.properties (Brazilian Portuguese)

```
redColor=VERMELHA
blueColor=AZUL
```

4. Select **File -> Save All** to save all your changes to the project.
5. Close the browser.
6. Click **Run on Server** to test your changes.
7. Click **Edit my profile** to change the preferred languages and execute the application again to check that portlets display the word `BLUE` or `RED` in the language you have selected.

   Note: For simplicity, not all text in this sample scenario has been enabled for NLS.

8.7 Sample scenario: translating whole resources

Another way to accomplish internationalization is by translating and maintaining separate JSPs within a predictable directory structure. The Portal will take responsability for locating the correct file at runtime, depending on the preferred language selected by the user.

Archived

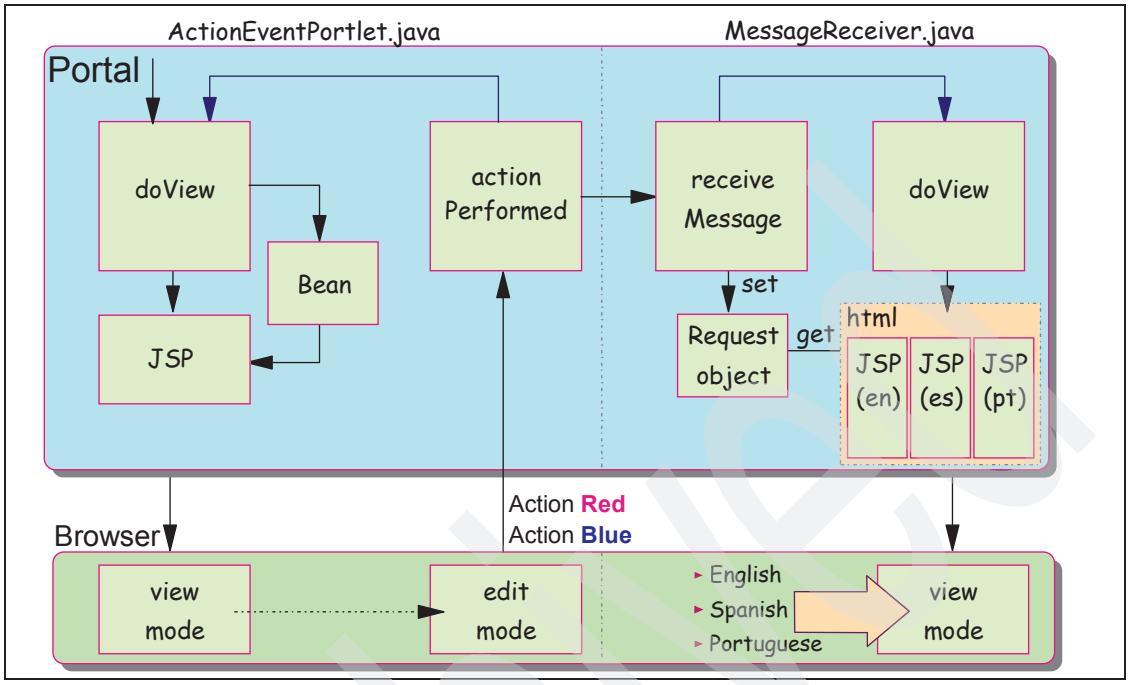


Figure 8-35 Sample scenario

1. Open the Web Content/messagereceiver/jsp/html/MessageReceiverView.jsp page to delete fmt tags and return to static information. Your code should look as shown in Example 8-19.

Example 8-19 MessageReceiverView.jsp

```
<%@ page contentType="text/html" import="java.util.*, messagereceiver.*"%>
<%@ taglib uri="/WEB-INF/tld/portlet.tld" prefix="portletAPI" %>
<portletAPI:init/>

<DIV style="margin: 6px">

<H3 style="margin-bottom: 3px">Welcome!</H3>
This is a sample <B>view mode</B> page. You have to edit this page to customize
it for your own use.<BR>
The source file for this page is "/Web
Content/messagereceiver/jsp/html/MessageReceiverView.jsp".

<br>
<% if (request.getAttribute("MyMessage") == null) { %>
   <B>Ready to receive message ... </B>
<% } else { %>
```

```
   <B>Received a message:</B>
   <B><%= request.getAttribute("MyMessage") %></B>
<% } %>

</DIV>
```

2. Select your **Web Content/messagereceiver/jsp/html** folder, right-click it and select **New -> Folder**. Type en for the Folder name field and click **Finish**.
3. Right-click **Web Content/messagereceiver/jsp/html/MessageReceiverView.jsp** and select **Copy**.

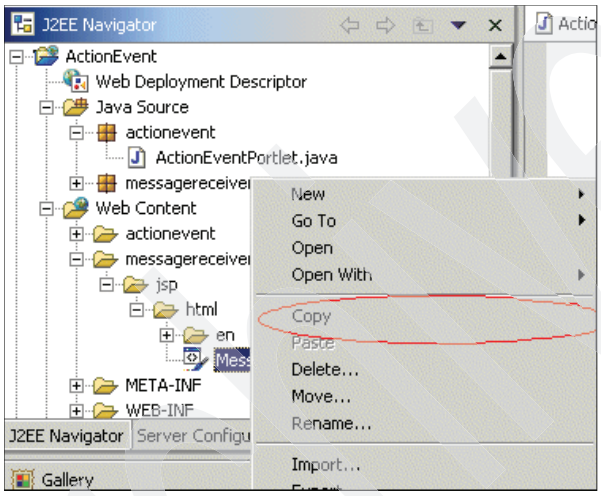


Figure 8-36 Copy a JSP page

4. Now select the **Web Content/messagereceiver/jsp/html/en** folder, right-click it and select **Paste**.

Archived

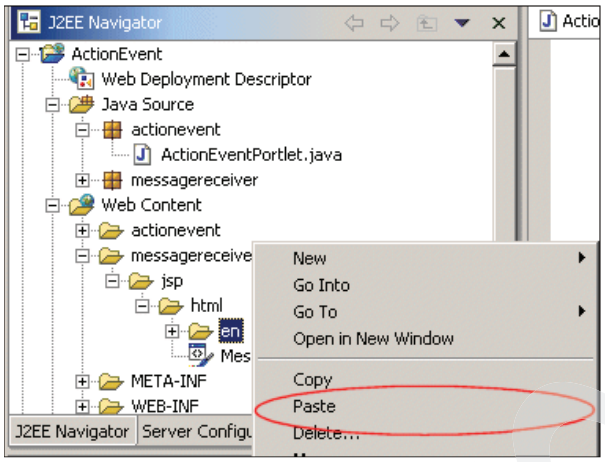


Figure 8-37 Paste a JSP page

5. Open the page Web Content/messagereceiver/jsp/html/en/MessageReceiverView.jsp to update the text indicating the location of the source file page. This is not required but it is recommended for clarity.

Archived

Example 8-20 MessageReceiverView.jsp (English)

```
<%@ page contentType="text/html" import="java.util.*, messagereceiver.*"%>
<%@ taglib uri="/WEB-INF/tld/portlet.tld" prefix="portletAPI" %>
<portletAPI:init/>

<DIV style="margin: 6px">

<H3 style="margin-bottom: 3px">Welcome!</H3>
This is a sample <B>view mode</B> page. You have to edit this page to customize
it for your own use.<BR>
The source file for this page is "/Web
Content/messagereceiver/jsp/html/en/MessageReceiverView.jsp".


<br>
<% if (request.getAttribute("MyMessage") == null) { %>
   <B>Ready to receive message ...</B>
<% } else { %>
   <B>Received a message:</B>
   <B><%= request.getAttribute("MyMessage") %></B>
<% } %>

</DIV>
```

6. Optionally, repeat the steps to create the folders and JSPs for other languages such as Spanish (es) and Brazilian Portuguese (pt_BR). Your directory structure should be as illustrated in Figure 8-38 on page 288.

Archived

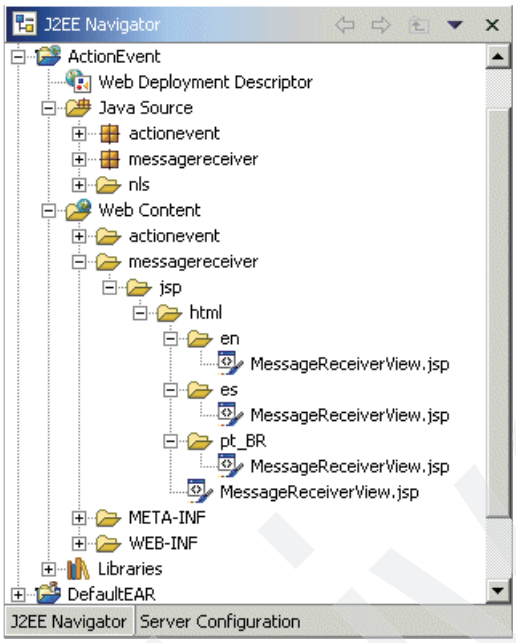


Figure 8-38 Directory structure

7. Modify the JSP pages to display a message in the proper language. Also change the directory of the source file pages. For example, create a folder with a JSP for Spanish (es).

Example 8-21 MessageReceiverView.jsp (Spanish)

```
<%@ page contentType="text/html" import="java.util.*, messagereceiver.*"%>
<%@ taglib uri="/WEB-INF/tld/portlet.tld" prefix="portletAPI" %>
<portletAPI:init/>

<DIV style="margin: 6px">

<H3 style="margin-bottom: 3px">Welcome!</H3>
This is a sample <B>view mode</B> page. You have to edit this page to customize
it for your own use.<BR>
The source file for this page is "/Web
Content/messagereceiver/jsp/html/es/MessageReceiverView.jsp".


<br>
<% if (request.getAttribute("MyMessage") == null) { %>
   <B>Listo para recibir mensajes ...</B>
<% } else { %>
   <B>Mensaje recibido:</B>
```

Archived

```
   <B><%= request.getAttribute("MyMessage") %></B>
<% } %>

</DIV>
```

Example 8-22 MessageReceiverView.jsp (Br Portuguese)

```
<%@ page contentType="text/html" import="java.util.*, messagereceiver.*"%>
<%@ taglib uri="/WEB-INF/tld/portlet.tld" prefix="portletAPI" %>
<portletAPI:init/>

<DIV style="margin: 6px">

<H3 style="margin-bottom: 3px">Welcome!</H3>
This is a sample <B>view mode</B> page. You have to edit this page to customize
it for your own use.<BR>
The source file for this page is "/Web
Content/messagereceiver/jsp/html/pt_BR/MessageReceiverView.jsp".


<br>
<% if (request.getAttribute("MyMessage") == null) { %>
   <B>Pronto para receber mensagem ...</B>
<% } else { %>
   <B>Mensagem recebida:</B>
   <B><%= request.getAttribute("MyMessage") %></B>
<% } %>

</DIV>
```

8. Run the scenario (click **Run on Server**) and verify your results in multiple languages.
9. Change the locale in the user profile as before and try other supported languages.

Archived

Archived

9

Accessing Web Services

This chapter provides an overview and a sample scenario for creating a sample portlet project that will work as a Web Service client to interact with a Web Service. The Web Service client portlet is created using the wizard provided by the Portal Toolkit. The sample scenario in this chapter will allow you to understand the techniques used to develop portlets that retrieve information using Web Services.

Note: The sample scenario included in this chapter requires that Web Services be available. You can download the sample code available as additional materials. See Appendix C, “Additional material” on page 543.

© Copyright IBM Corp. 2004. All rights reserved.

9.1 Overview

In this section, we show the most important steps required to develop a Web Service client portlet accessing a local or remote Web Service. In most cases, the following tasks will be executed:

1. Develop a sample Web Service from a JavaBean. This Web Service will be used to test and run the Web Service client portlet. The following tasks are required:
   a. Create a sample Web project and import an existing JavaBean class.
   b. Using the available wizards in WebSphere Studio Site Developer, transform this JavaBean into a Web Service so it can be accessed from portlet applications.
2. Using the wizards provided by WebSphere Studio Site Developer, create a Web Services client portlet project to access the sample Web Service.

The scenario illustrated in Figure 9-1 shows how portlet applications can be easily integrated with existent Web Services without the need to write extra code.

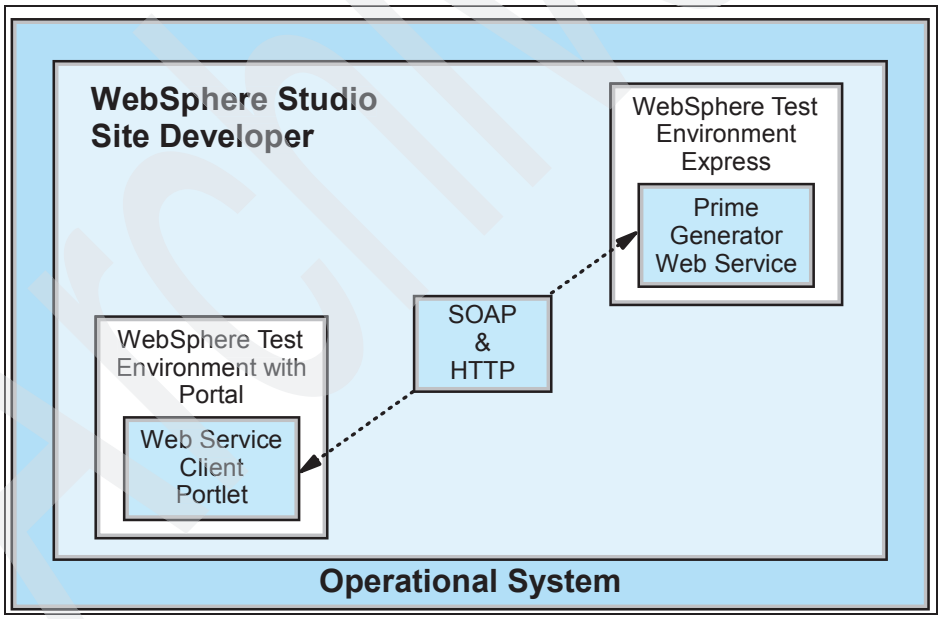


Figure 9-1 Web Services client portlet scenario

The development workstation used to create the sample application is illustrated in Figure 9-2 on page 293.

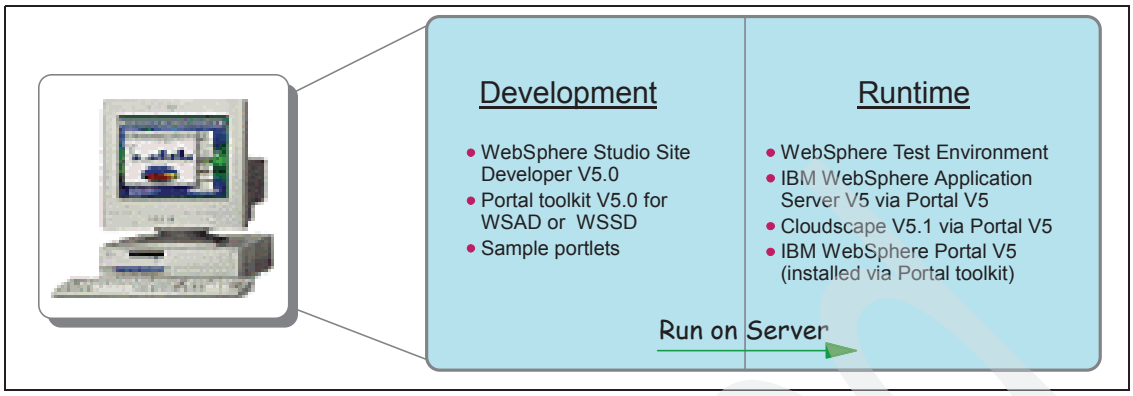


Figure 9-2 Development workstation

9.2 A simple Web Service project

The project will be created using the Web Project wizard. In this section, you create a Web project with the name *PrimesWebService*. You will import a JavaBean that generates prime numbers into this project. The Web project will also be published and executed in a WebSphere Test Environment Express.

1. If not already running, start the IBM WebSphere Studio Site Developer; click **Start -> Programs -> IBM WebSphere Studio -> Site Developer 5.0**.
2. Select **File -> New -> Other**.

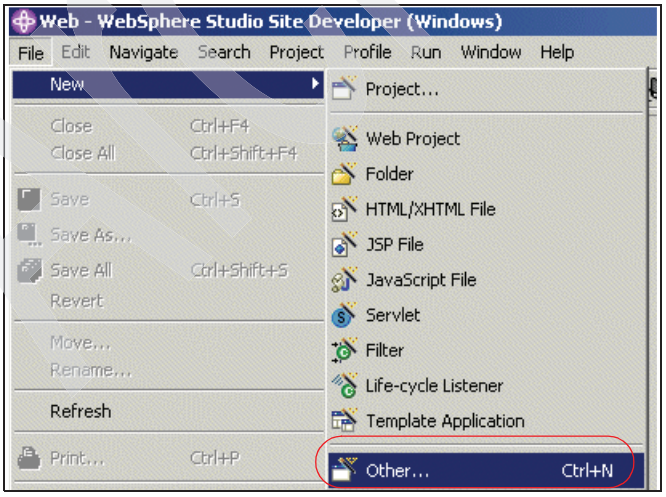


Figure 9-3 New project creation

3. Select **Web** -> **Web Project** and click **Next**.

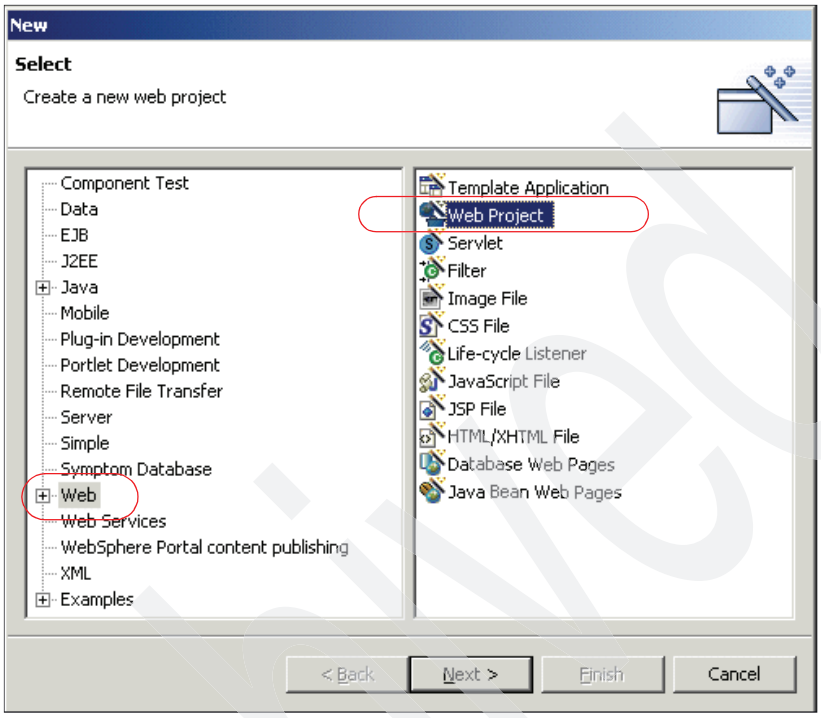


Figure 9-4 Selection of Web Project

4. Enter `PrimesWebService` for the Project name. Click **Next**.

Archived

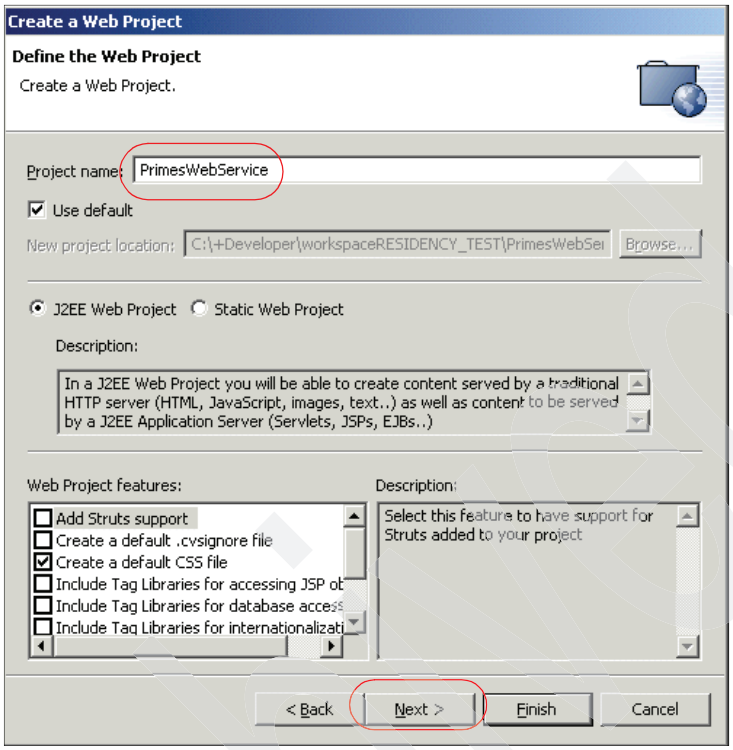


Figure 9-5 New Project wizard

5. Select **New** for the Enterprise application project and enter `PrimesWebServiceEAR` (was DefaultEAR) in the New project name field. Click **Finish**.

Archive

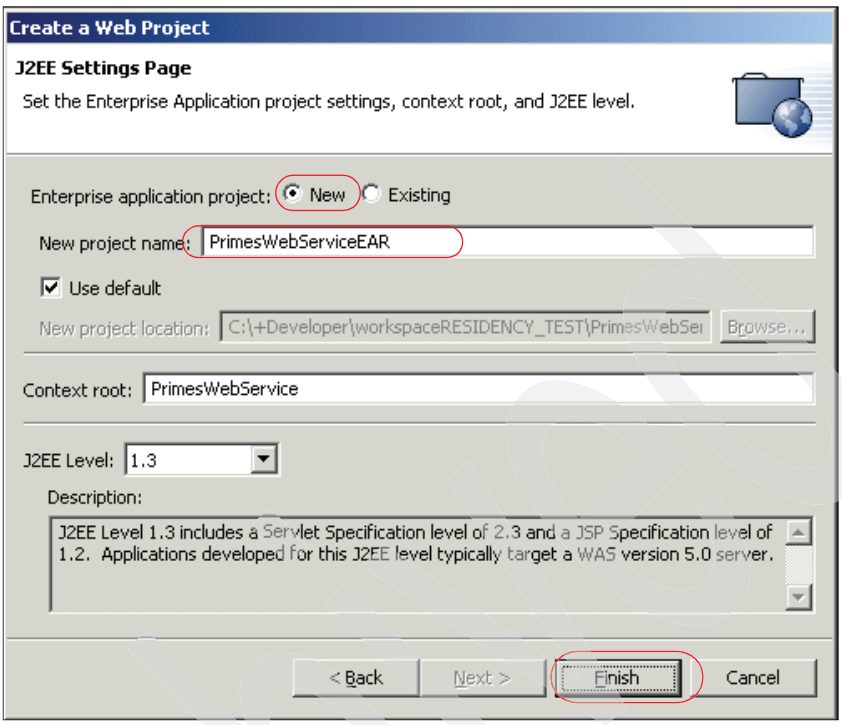


Figure 9-6 New Project wizard

6. A class file is provided for this sample scenario (Prime.java); follow these steps to import the Java file:

   a. Import the file by selecting **File -> Import**, select **File system** and click **Next**.

Archive

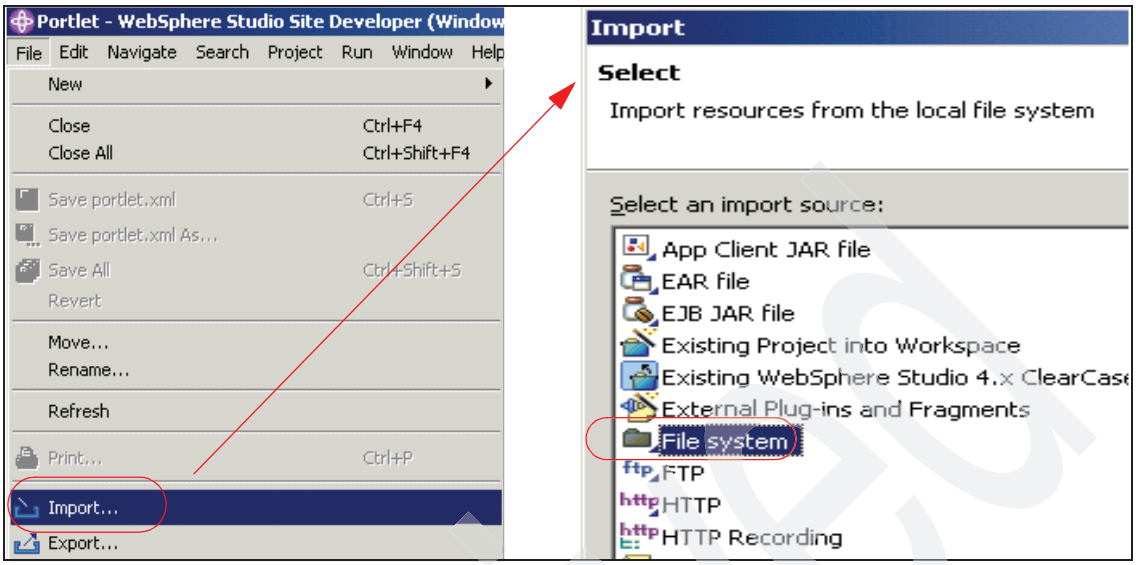


Figure 9-7 Importing a file

b. In the Import File system window, enter the following information:

   i. Directory: browse to `C:\LabFiles\WebServicesClient\Java Source`, then select **Primes.java**.

      Note: You can also download the sample code available as additional materials. See Appendix C, “Additional material” on page 543.

   ii. For the destination, browse to the folder: `PrimesWebService/Java Source`. Click **Finish**.

Archived

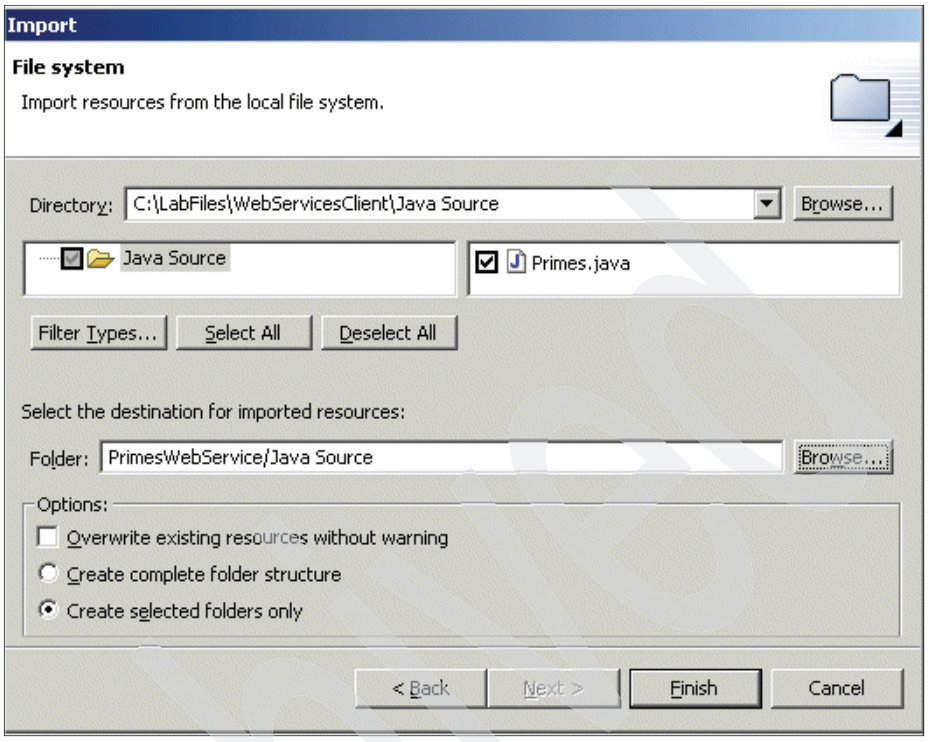


Figure 9-8 Importing the provided Java file

9.2.1 A sample Web Service

In this section of the sample scenario, you will review the Primes.java code. For example, to view the source code to *Primes.java*, double-click the file name. This file is located in the /Java Source/ folder.

- The only important method for you in this code is the getPrime() method. This method receive a number of digits and returns a prime number with the specified length of digits.
- When executing this method, try generating prime numbers of 20 or fewer digits to avoid long computations. A short version of the prime number generator is shown in Example 9-1.

Example 9-1 Primes.java - getPrime() method

```
import java.math.BigInteger;
import java.util.Vector;
```

```
public class Primes {

   private static final BigInteger ZERO = new BigInteger("0");
   private static final BigInteger ONE = new BigInteger("1");
   private static final BigInteger TWO = new BigInteger("2");
   private String prime = "";

   ......
   ......
   ......

   public static String getPrime(int numDigits) {

      BigInteger start = random(numDigits);
      start = nextPrime(start);

      return start.toString();
   }
}
```

Creating a Web Service

In this section, you create a Web Service in the PrimesWebService project that you just created. You will use the wizards provided by WebSphere Studio Site Developer to do this. Once created, the Web Service will be published and invoked to be executed in a WebSphere Test Environment.

1. Select **File -> New -> Other**.

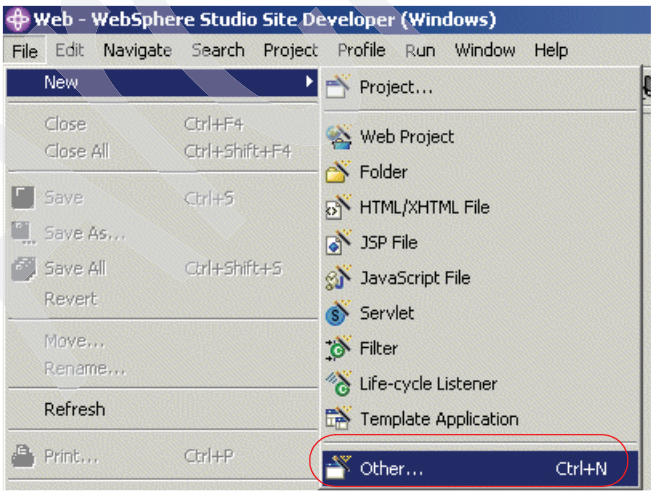


Figure 9-9 Starting creation of Portlet project

Archived

2. Select **Web Services** -> **Web Service**.

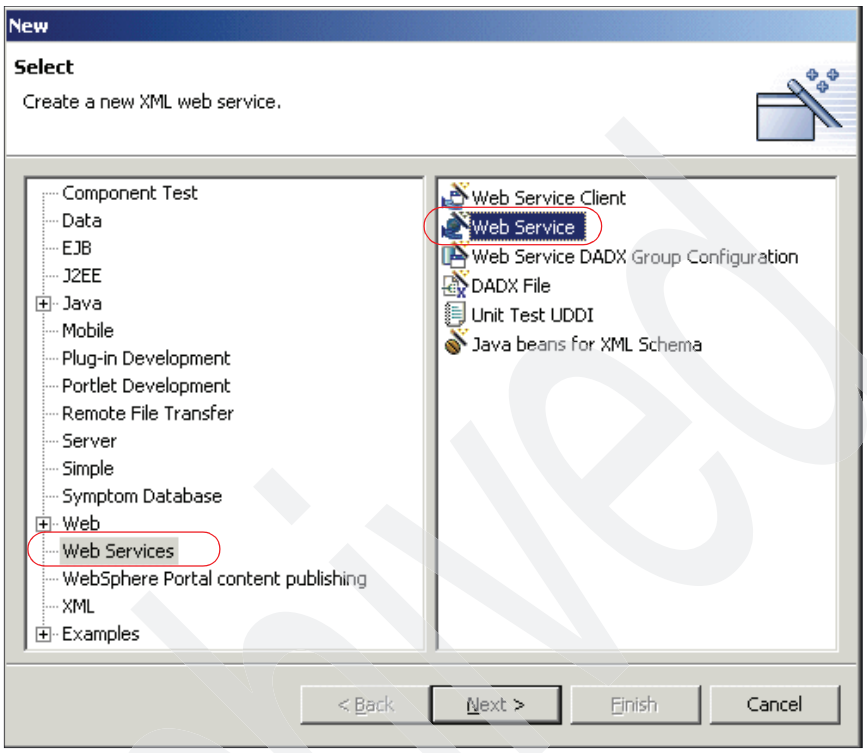


Figure 9-10 Selection of Web Service

3. Click **Next**.
4. Enter the following information in the Web Services window:
   a. Web Service type: `Java bean Web Service`.
   b. Check the box **Generate a proxy**.
   c. Client proxy type: `Java proxy`.
   d. Check the box **Test the generate proxy**.

   Click **Next**.

Archived

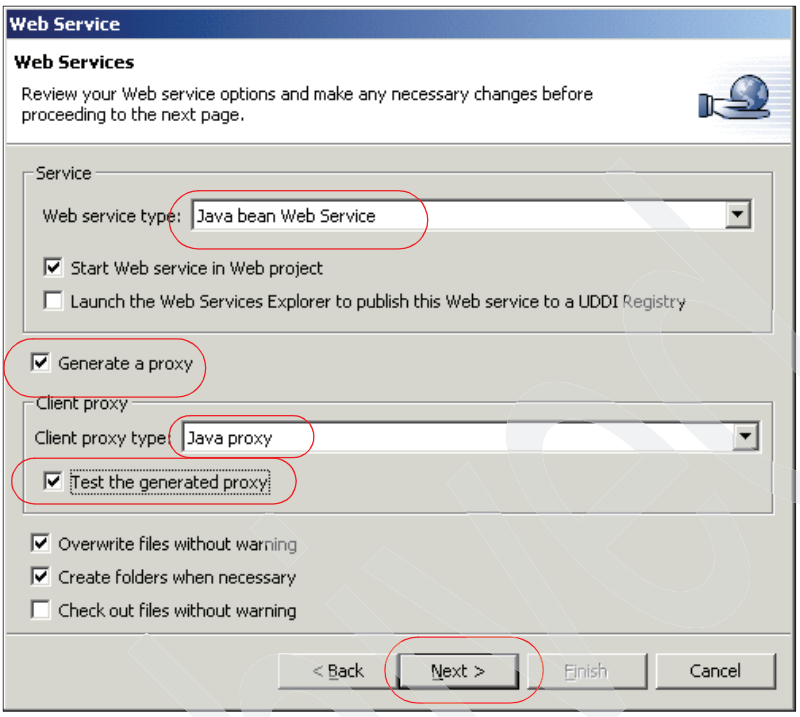


Figure 9-11 New Project wizard

5. In Web Service Deployment Settings, examine and accept default values. Click **Next**.

Archive

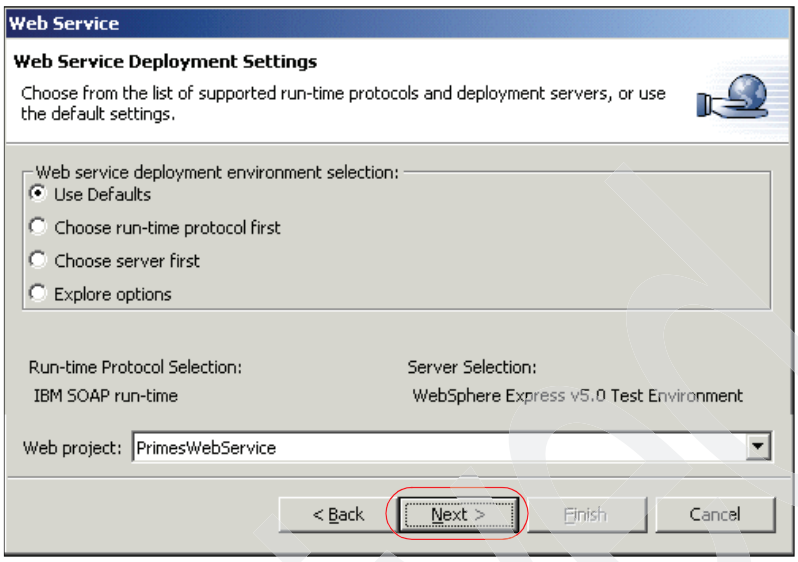


Figure 9-12 Web Service Deployment Settings

6. In the *Web Service Java Bean Selection* window, click **Browse Files**.
7. Select the **Primes.java** class. Click **OK**.

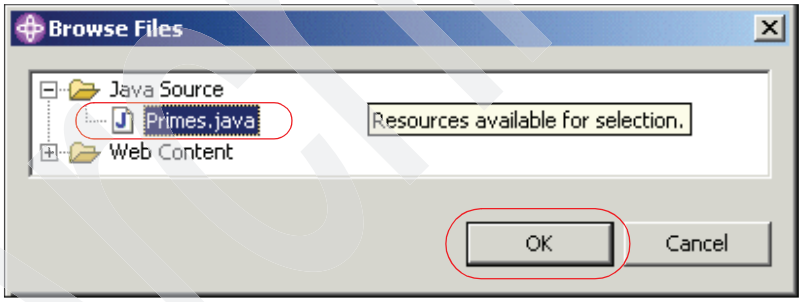


Figure 9-13 Primes.java selection

8. Click **Next** to generate the Web Service.
9. In the Web Service Bean Identity, examine and accept the default values. Click **Next**.

Archive

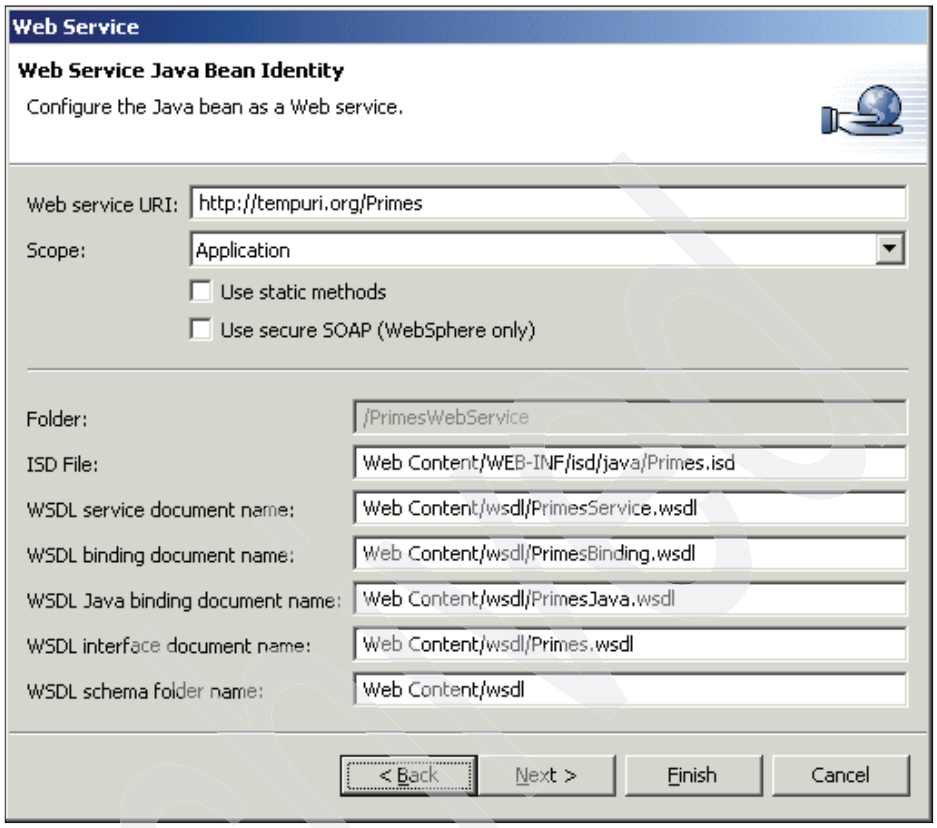


Figure 9-14 Web Service Java Bean Identity

10. In the Web Service Java Bean Methods:
    a. Select the **getPrime(int)** method.
    b. Deselect other methods as shown in Figure 9-15 on page 304.
    c. Click **Next**.

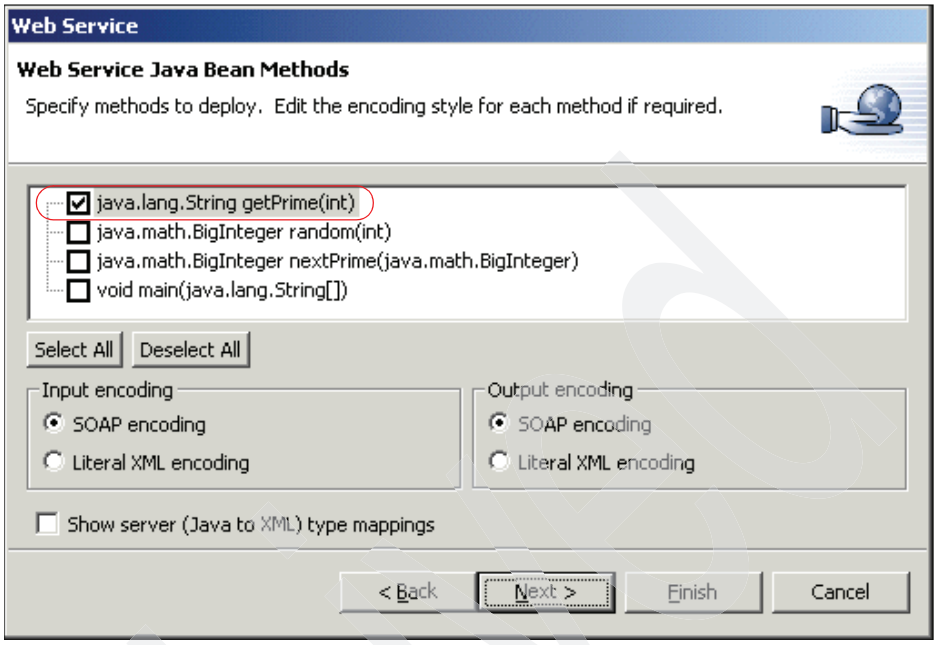


Figure 9-15 Web Service Java Bean Methods

11. In the Web Service Bind Proxy Generation window, accept the defaults and click **Next** to generate the proxy.

Archived

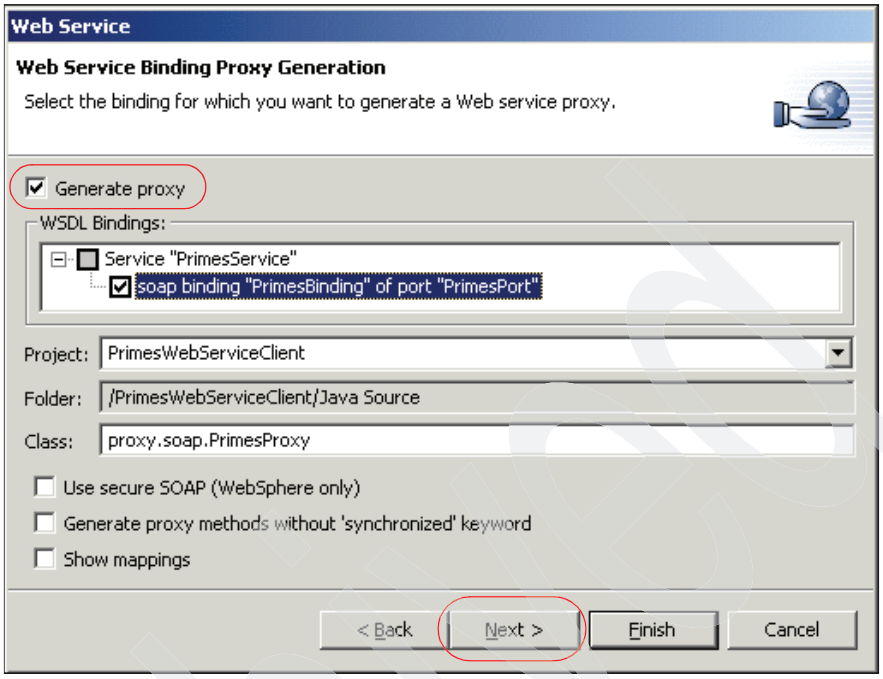


Figure 9-16 Web Service Binding Proxy Generation

12.I In the Web Service Test window, make sure you select the **Run test on Server** option. Click **Next**.

Archive

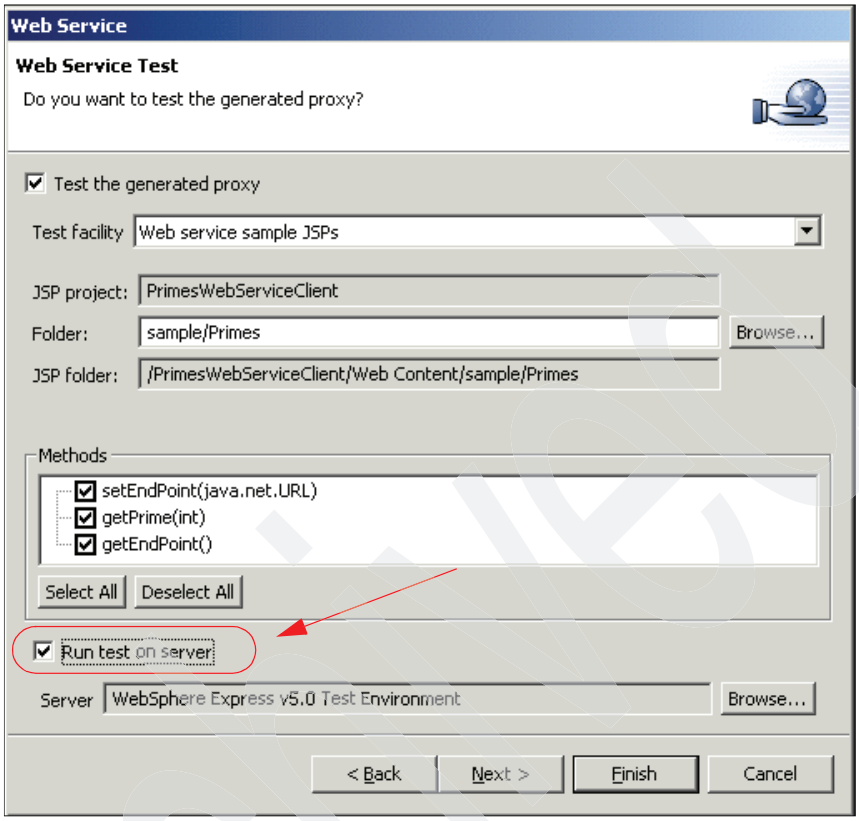


Figure 9-17 Web Service Test

13. In the Web Server Publication window, accept the defaults and click **Finish** to publish the Web Service.

Archive

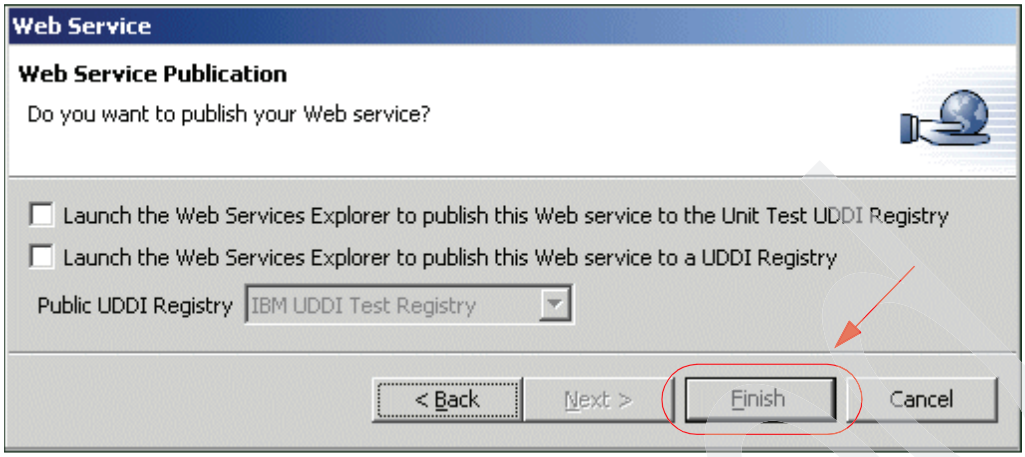


Figure 9-18 Web Service Publication

14. You will be presented with the available methods in the built-in browser. It is a Web test client interface to access and verify that the Web Service is running as expected.

15. Click the method **getPrime(int)**.

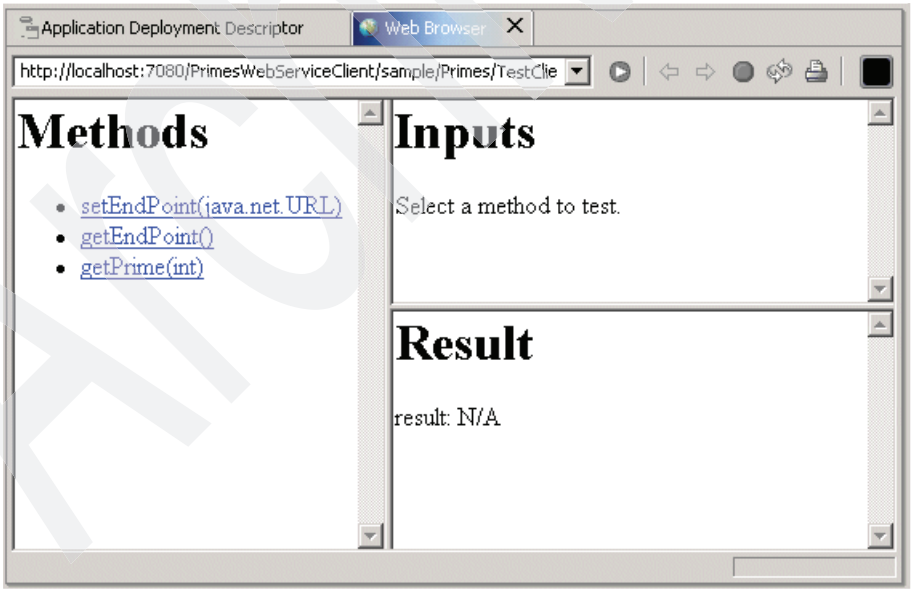


Figure 9-19 Available methods

16. Enter the number of digits you want the prime number to be. Do not try long numbers. It will be time consuming.

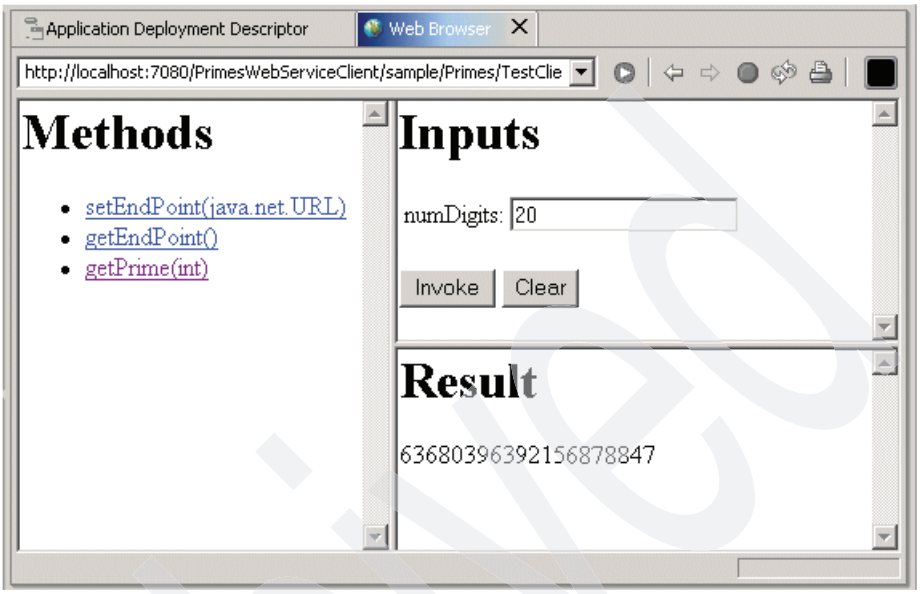


Figure 9-20 Accessing a Web Service to generate prime numbers

17. Every time you invoke this Web Service to generate a prime number, you will probably get a different result.

9.3 Creating a Web Services client portlet

In this section of the sample scenario, you will create a Web Services client portlet to access the prime number generator Web Service.

1. Select **File -> New -> Project**.

Archive

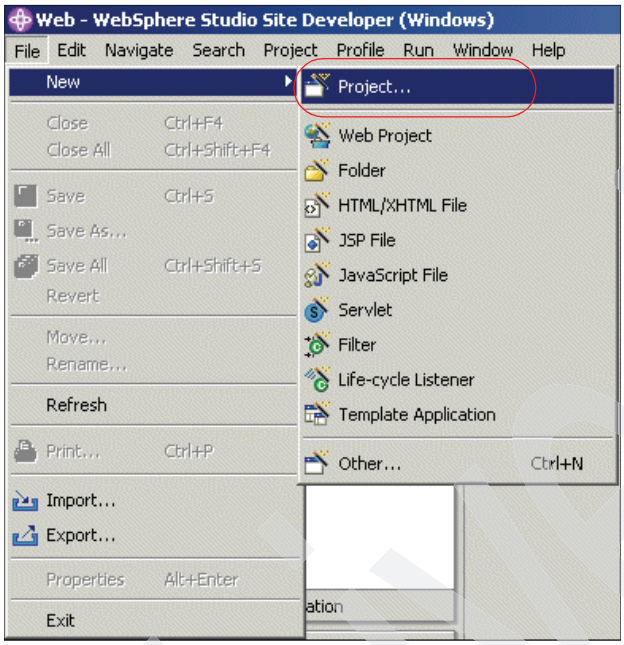


Figure 9-21 Creating a new project

2. Select **Portlet Development** from the left panel and **Web Service Client Portlet Project** from the right panel. Then click **Next**.

Archived

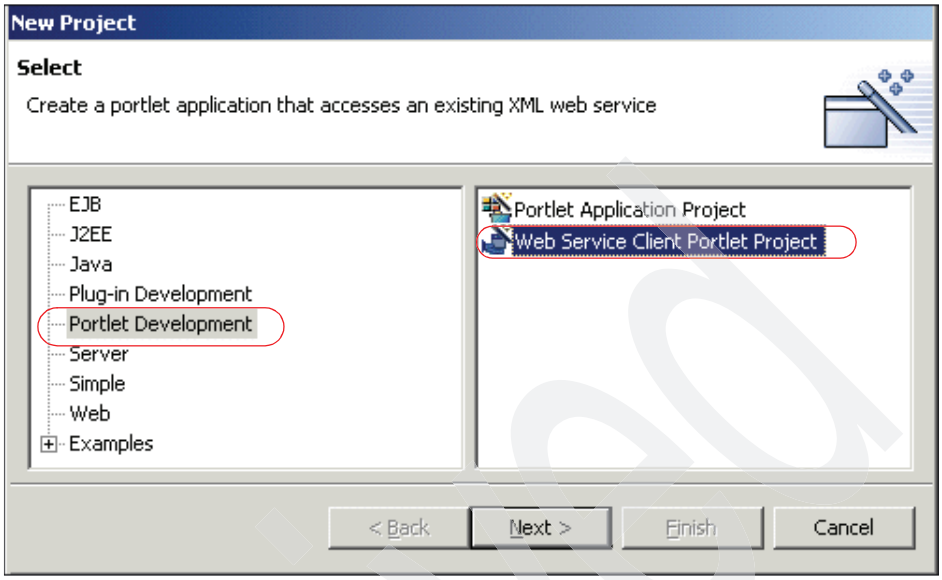


Figure 9-22 Selecting Web Service Client Portlet Project

3. In the Define the Portlet Project window, enter `WSClientPortlet` as the Project Name.
4. Select **New** for the Enterprise application project.
5. Enter `WSClientPortletEAR` for the new project name.
6. Click **Next** to continue.

Archive

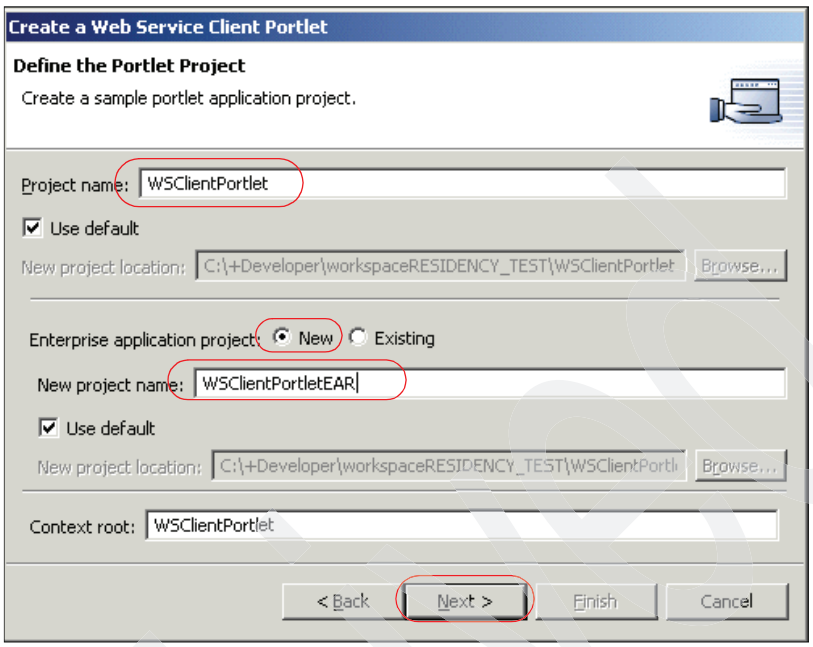


Figure 9-23 Defining the portlet project

7. In the Web Service Portlet Parameters window, click **Browse**.

Archive

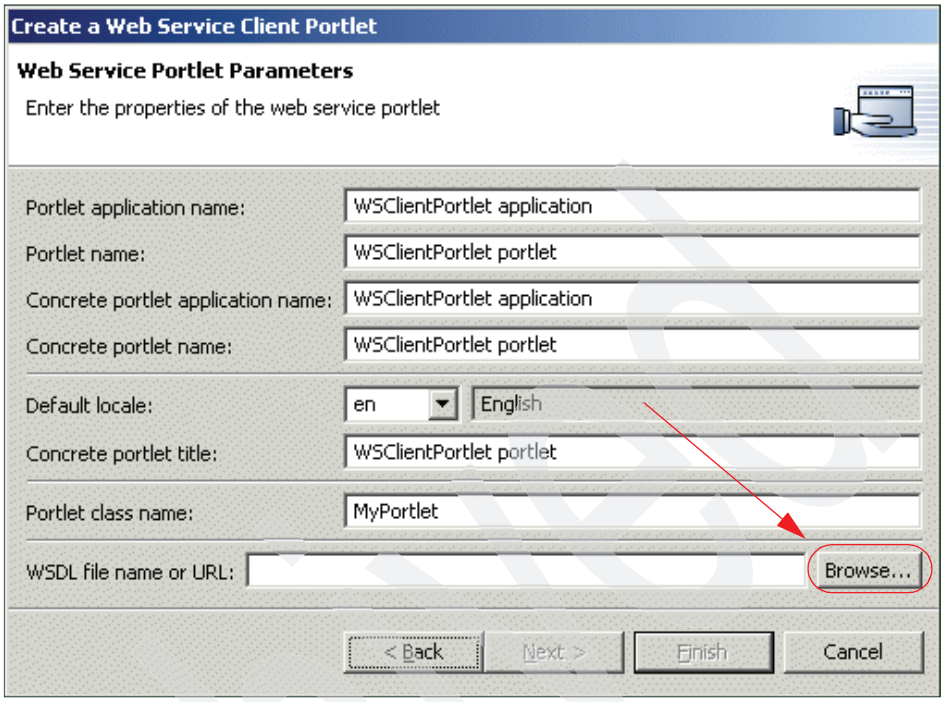


Figure 9-24 Web Service Portlet Parameters

8. Select **PrimesService.wsdl**. Click **OK**.

Archived

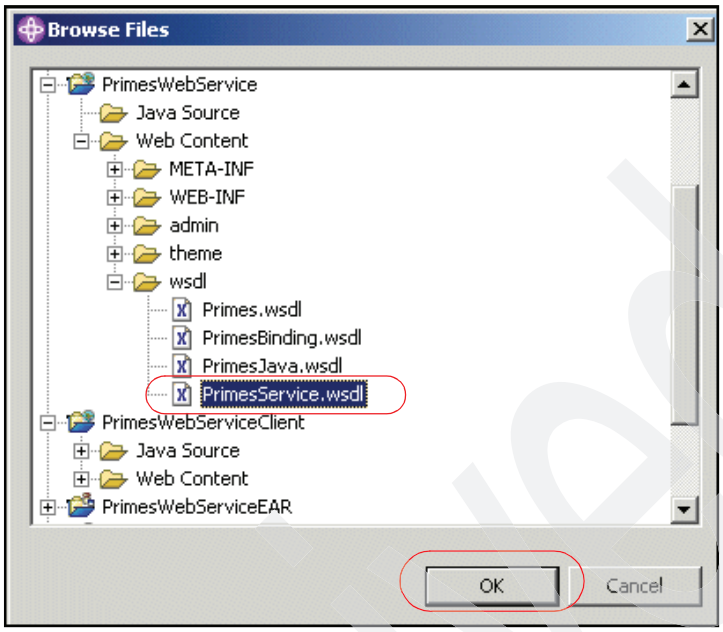


Figure 9-25 Selecting PrimesService.wsdl

9. Click **Finish**.

Archived

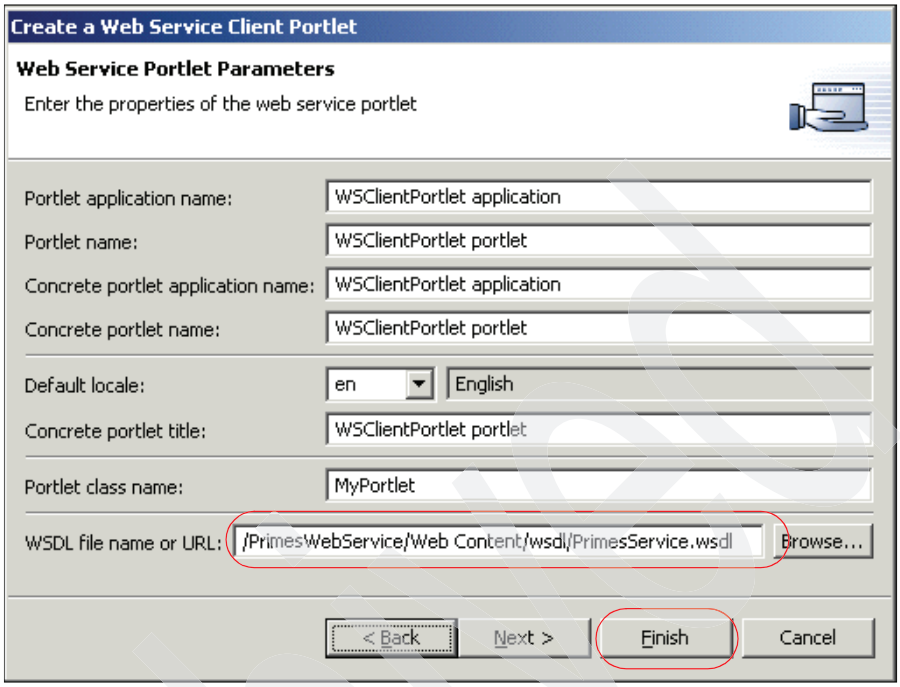


Figure 9-26 Web Service portlet properties

9.4 Run the WSClientPortlet application

In this section, you will run the Web Services client (WSClientPortlet) to access the prime number generator Web Service. Execute the following steps:

1. Run the WSClientPortlet portlet application by right-clicking **WSClientPortlet** in the Navigator panel and selecting **Run on Server**.

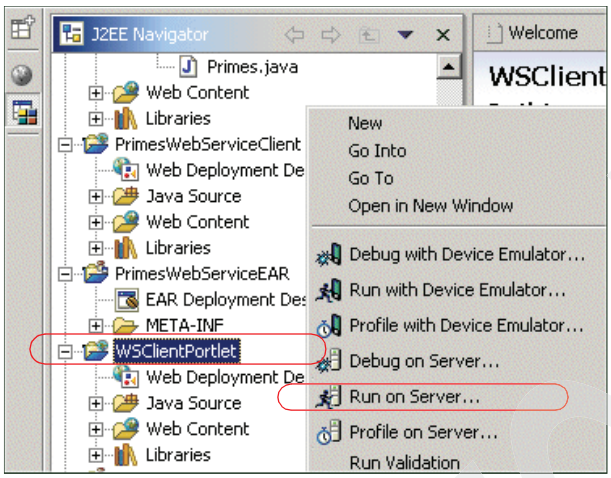


Figure 9-27 Running WSClientPortlet

2. Select the option to **Create a new server**.

Archived

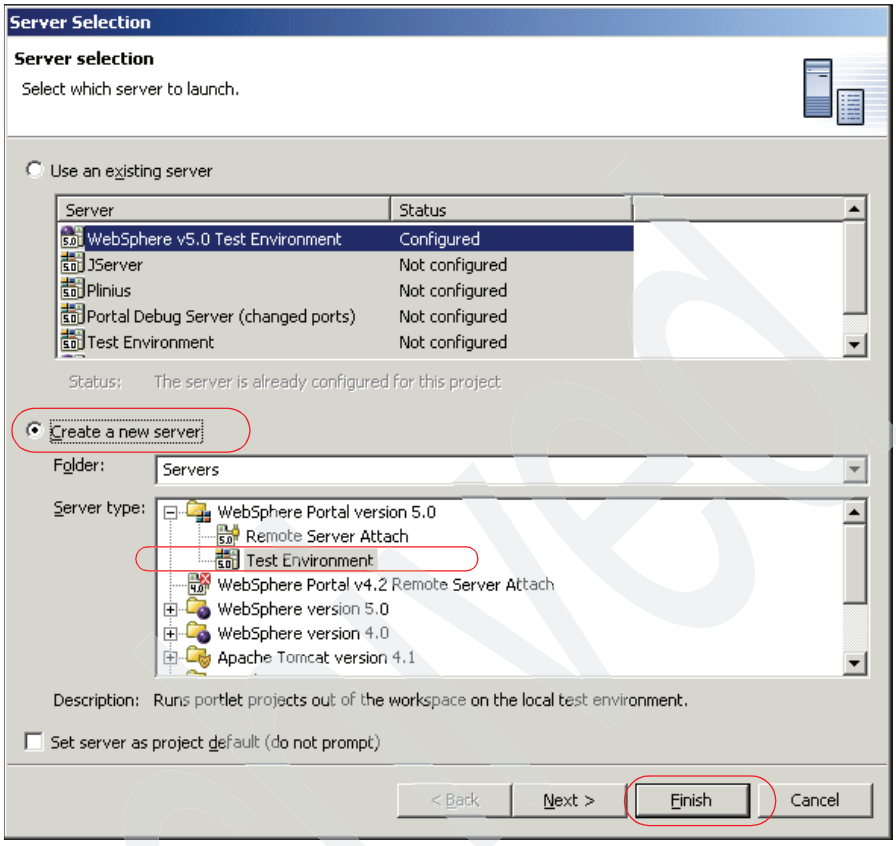


Figure 9-28 Server selection

3. Click **Finish** (Figure 9-28) and wait a few minutes for the server. The portlet will run and will be seen in the built-in browser. Click the method **getPrime(int)** to access the Web Service and generate a prime number.

Archived

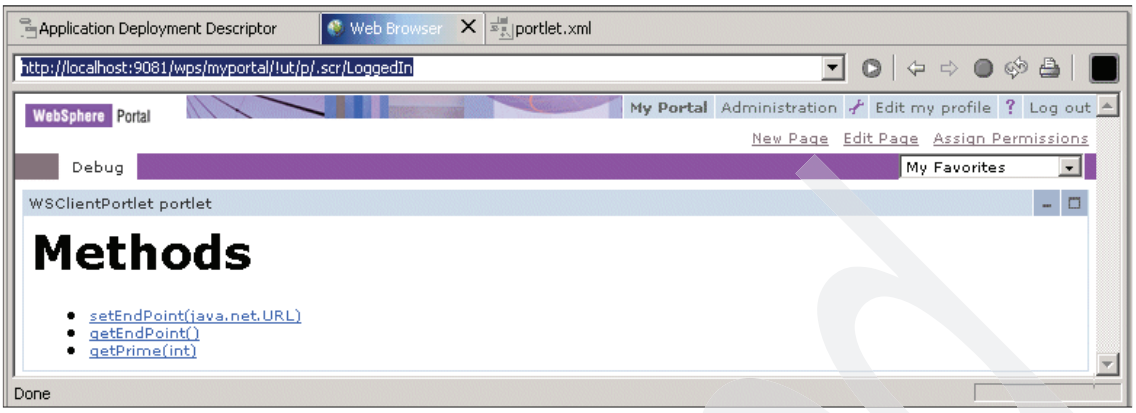


Figure 9-29 Available methods

4. Enter a number of digits, for example 20, and click **Invoke**.

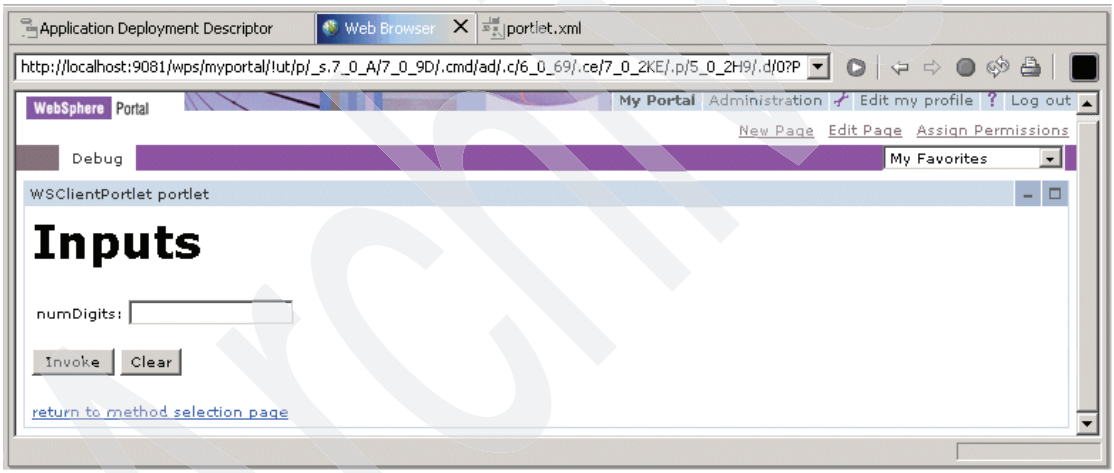


Figure 9-30 Invoking Web Services

Archived

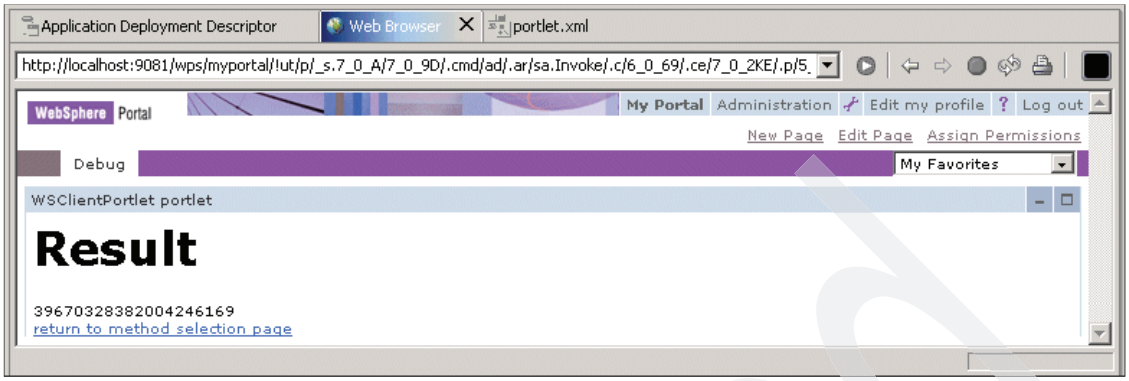


Figure 9-31 Results from the Web Service

Archived

10

Using the Credential Vault

This chapter provides an overview and a sample scenario for creating a sample portlet application that uses Credential Vault to log in and interact with back-end systems. You will create, deploy and run portlet applications. The sample scenario will allow you to understand the techniques used to develop portlets using the Credential Vault provided by IBM WebSphere Portal.

The development workstation for the sample scenario included in this chapter is illustrated in Figure 10-1.

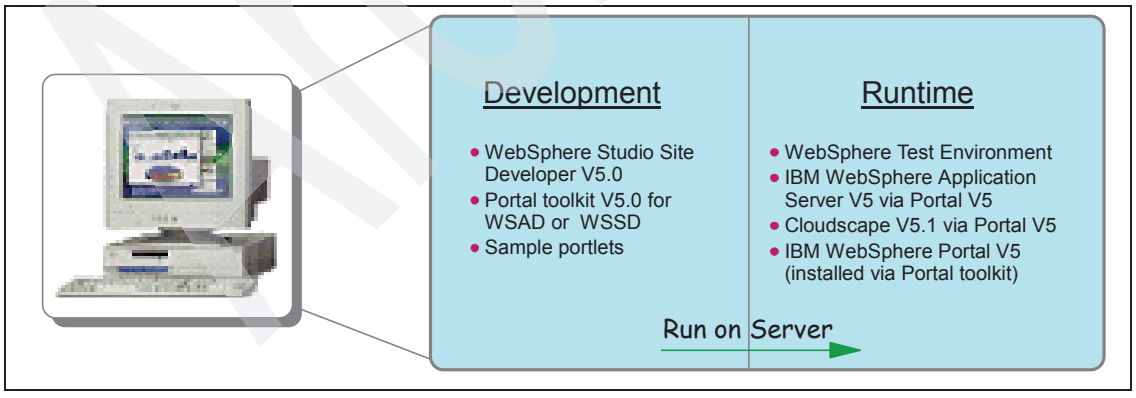


Figure 10-1 Development workstation

© Copyright IBM Corp. 2004. All rights reserved.

10.1 Overview

When integrating different back-end systems, portlets often need to provide some type of authentication to access these back-end systems. WebSphere Portal provides the use of a Credential Vault to store and retrieve user credentials. By using Credential Vault portlets, you can provide a single sign-on experience to the user.

After reading this chapter, you will be able to:

- Understand the value of Credential Vault for portlet development
- Identify the different components of Credential Vault
- Build portlet applications using Credential Vault technology and active and passive objects

Portlets running on WebSphere Portal may need to access remote applications that require some form of authentication by using appropriate credentials. In this section, we provide an overview of the Credential Vault components.

Credentials

Examples of credentials are user IDs and passwords, SSL client certificates and private keys. In order to provide a single sign-on user experience, portlets should not ask the user for the credentials of individual applications each time the user starts a new portal session. Instead, they must be able to store and retrieve user credentials for their particular associated application and use those credentials to log in on behalf of the user. The Portal back-end secure access is illustrated in Figure 10-2 on page 321.

Archived

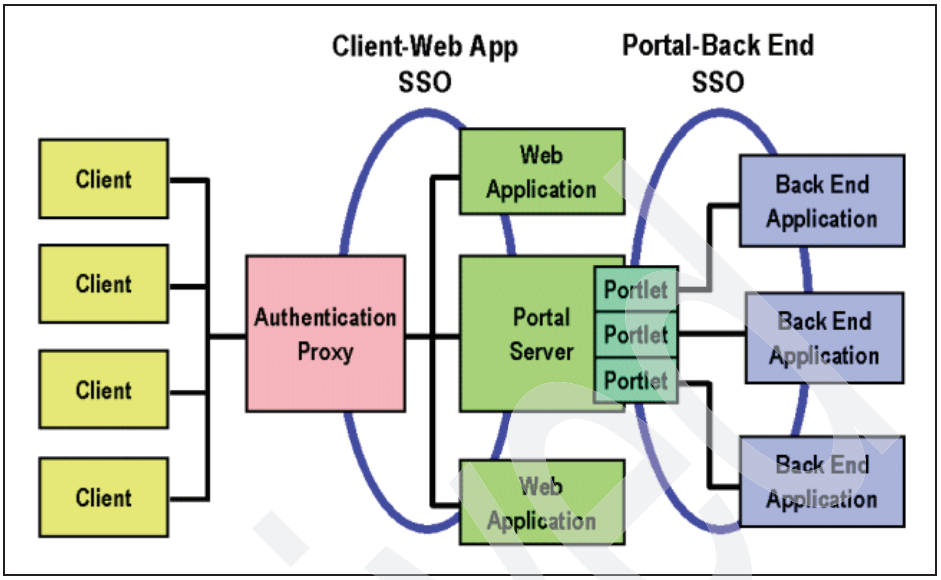


Figure 10-2 Credential Vault in action

The Credential Vault provides this functionality and portlets can use it through the Credential Vault Portlet Service.

Components of the Credential Vault organization

The organization of Credential Vault in WebSphere Portal consists of vault segments and credential slots. Figure 10-3 on page 322 shows an overview of these components.

Vault segments

The Credential Vault is partitioned into segments and a vault segment contains one or more credential slots.

There are two different types of vault segments:

- Administrator-managed segments: in this type of vault segment, the creation of new slots is restricted to the portlet administrator.
- User-managed segments: in this type of vault segment, portlets can also create new slots on behalf of the user.

Note: Setting and retrieving credentials can be performed by portlets for both types of vault segments.

Archive

Vault implementations are the actual locations where the credentials are stored. This can be for example the default database of WebSphere Portal or the Tivoli Access Manager lock box.

Credential slots

As mentioned previously, every vault segment contains one or more credential slots. Slots are “drawers” where portlets store and retrieve a user’s credentials. Each slot holds one credential and links to a resource in a vault implementation. There are four different types of slots:

- A system slot stores system credentials where the actual secret is shared among all users and portlets.
- An administrative slot allows each user to store a secret for an administrator-defined resource (for example, Lotus Notes).
- A shared slot stores user credentials that are shared among the user's portlets.
- A portlet private slot stores user credentials that are not shared among portlets.

Note: In the sample scenario included in this chapter, only private slots will be used.

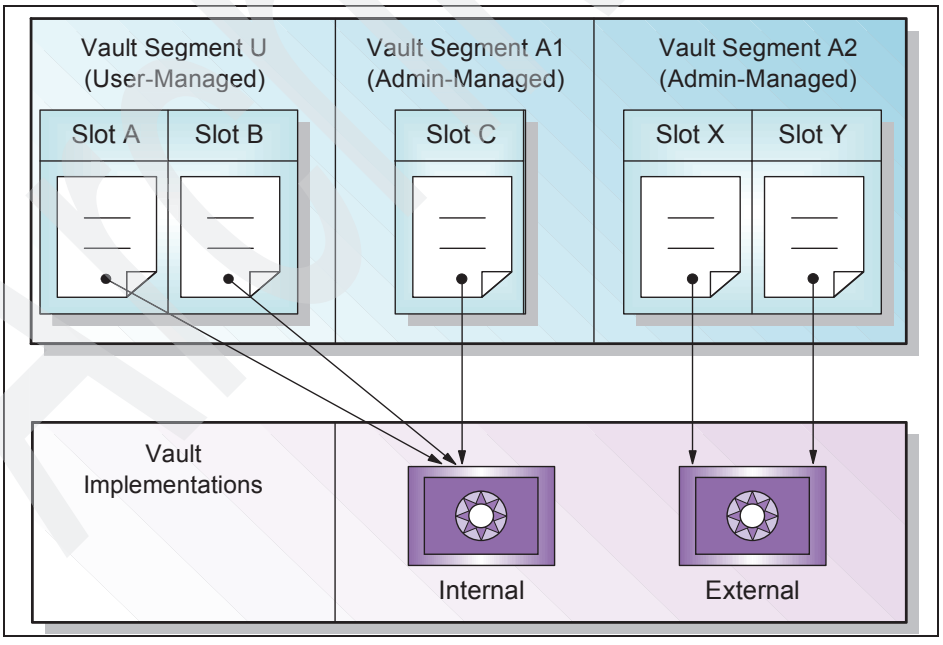


Figure 10-3 Credential Vault organization

Archived

Credentials objects

WebSphere Portal differentiates between passive and active credential objects:

- **Passive credential objects** are containers for the credential's secret. Portlets that use passive credentials need to extract the secret out of the credential and do all the authentication communication with the back-end resource. The following passive credential support is provided with WebSphere Portal:
  - UserPasswordPassive, which stores secrets in the form of user ID/password pairs
  - SimplePassive, which stores secrets in the form of serializable Java objects
  - JaasSubjectPassive (Java Authentication and Authorization Service), which stores secrets in form of javax.security.auth.Subject objects

  Currently, the vault service in WebSphere Portal only supports UserPasswordPassive.
- **Active credential objects** hide the credential's secret from the portlet; there is no way of extracting it out of the credential. In return, active credential objects offer business methods that take care of all the authentication. The following active credential support is provided with WebSphere Portal:
  - HttpBasicAuth
  - HttpFormBasedAuth
  - JavaMail
  - LtpaToken
  - SiteMinderToken
  - WebSealToken

Note: When using active credentials, portlets never get in touch with the credential secrets and thus there is no risk a portlet could violate any security rules such as, for example, storing the secret on the portlet session. While there might not always be an appropriate active credential class available, this is the preferred type of credential objects to use.

Sample scenario

In this sample scenario, you will create a sample portlet based on a Basic portlet type using the Portlet Wizard. You will also use this wizard to enable Credential Vault to interact with back-end resources.

Archived

In this scenario, the protected back-end resource is a servlet and requires a user ID and password credentials to log in to the Web application (servlet). The servlet application has been secured with HTTP Basic Authentication.

The sample scenario illustrates the following:

- How the Credential Vault with active credentials is used
- How the Credential Vault with passive credentials is used
- How to store credentials
- How to retrieve credentials
- How to log in to the Web application
- How to retrieve the Web application content in the portlet's View mode

In the first part of this scenario, active credentials are used to access a secure Web application using HTTP Basic Authentication, as shown in Figure 10-4.

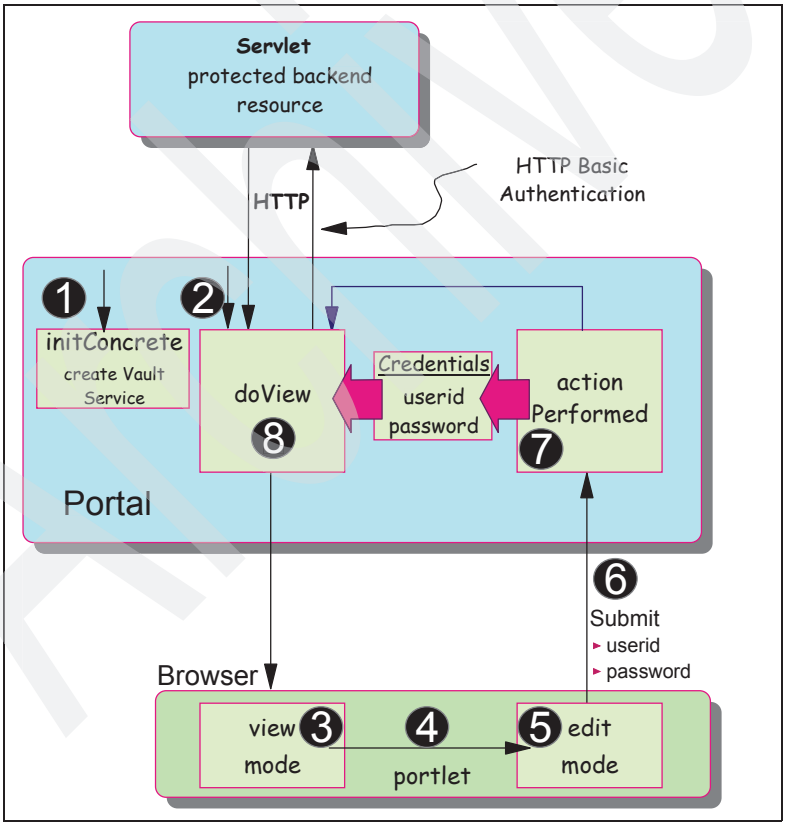


Figure 10-4 Credential Vault sample scenario

The sequence flow for this scenario is as follows:

1. The initConcrete method is used to initialize the Credential Vault Service.
2. Portal invokes the portlet doView method. Since initially, no credentials have been stored, a message is written indicating that a user ID and password must be entered in Edit mode.
3. In the portlet View mode, a message is shown directing the user to use the Edit mode to enter credentials.
4. The user clicks **Edit** to go into Edit mode.
5. The Edit mode screen is displayed, that is, the doEdit method is executed and a JSP displays a form to enter credentials and submit the action.
6. The user enters a user ID and password and selects **Submit**.
7. The actionPerformed method is executed to process the action. It creates a slot and stores the user ID/password information.
8. The doView method is executed to complete the cycle. The following tasks are executed in this mode:
   a. An HttpBasicAuth active credential object is retrieved from the credential service. Because authentication is done in this object, we never get in touch with the real credentials.
   b. The authorization header is set in the request HTTP header.
   c. The connection to the back-end resource (protected servlet in this scenario) is invoked.
   d. The user is authenticated and the servlet executes.
   e. The received content is rendered in View mode.

10.2 Importing a protected servlet application

In this section, you will import a previously created Treasure servlet. This servlet will be the back-end secure resource you will access using Credential Vault. The servlet displays a simple image and is only accessible via HTTP basic authentication.

The Treasure servlet application is in c:\LabFiles\CredentialVault\SecureServlet\CredentialVault_TreasureWeb.war.

Note: The sample scenario included in this chapter requires that you download the sample code available as additional materials. See Appendix C, “Additional material” on page 543.

Archived

Follow these steps to import the secure servlet:

1. If required, start WebSphere Studio Site Developer.
2. Switch to the Web perspective.
3. From the main menu, select **File -> Import...**.
4. Select **WAR file** and click **Next**.

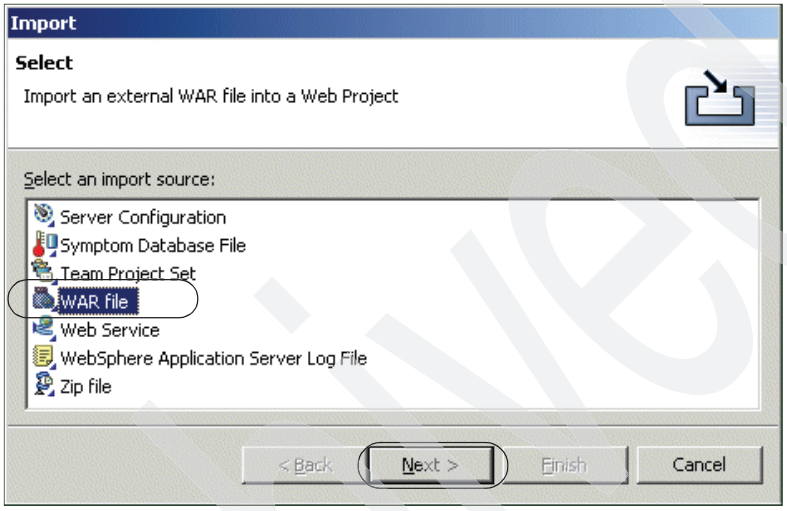


Figure 10-5 Importing a WAR file

5. Browse to the location of the TreasureWeb.war file in c:\LabFiles\CredentialVault\SecureServlet\CredentialVault_TreasureWeb.war.

   Note: The sample scenario included in this chapter requires that you download the sample code available as additional materials. See Appendix C, “Additional material” on page 543.

6. Enter a new Web project name of `TreasureWeb` and a new enterprise project name of `TreasureEAR`.

Archived

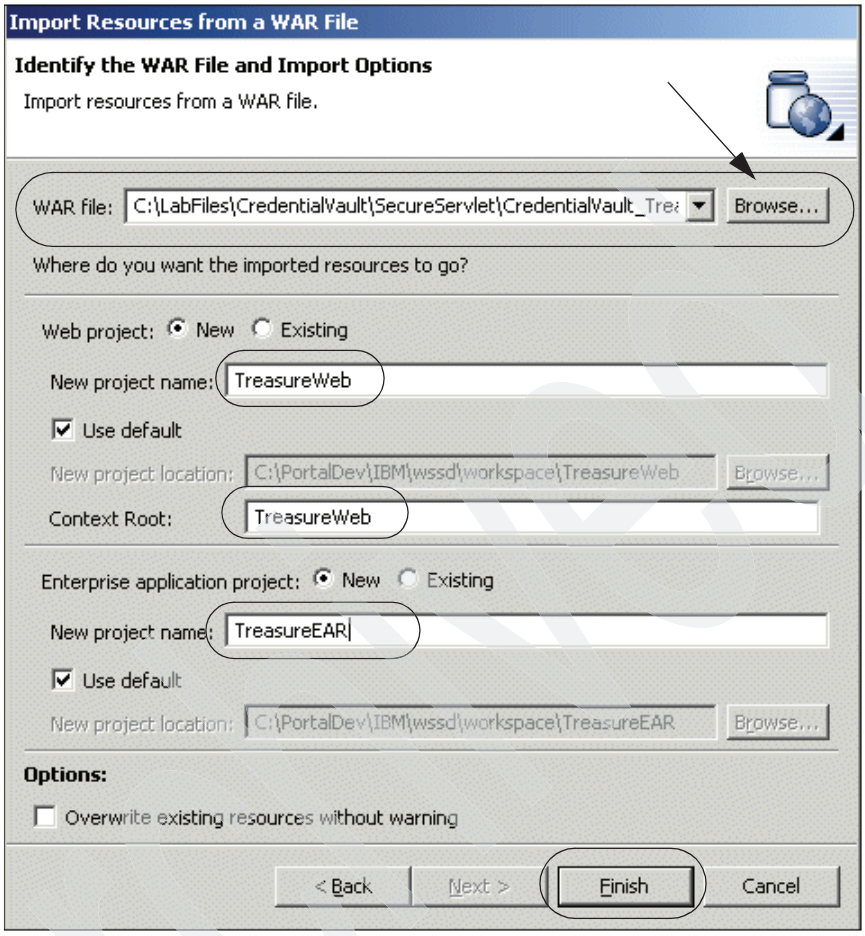


Figure 10-6 Import the Treasure WAR file

7. Click **Finish** to import the application (secure servlet WAR file).

After importing the WAR file, a new Web and enterprise project exist in your project. Now you can test the servlet to check that it is running properly.

To test the servlet, proceed as follows:

1. In the J2EE Navigator view, expand the TreasureWeb/Java Source/treasure package.
2. Right-click **TreasurePage.java** and select **Run on server...** from the context menu.

3. Click **OK** to create a new server using Test Environment and wait for a few minutes until the portal starts for e-business and the servlet is executed.

   Note: If you are not prompted to create a new server, you can always create a new server as follows:

   a. In the Server Selection dialog, click **Advanced...**.

   b. In the Advanced Server Selection dialog, check **Create a New Server** and select **WebSphere Version 5.0 Test Environment** as the server type. Click **Finish**. This is illustrated in Figure 10-7.

   Note: Because you are only using the Web container of the application server, you can also select a **WebSphere Version 5.0 Express Test Environment** if it is available.

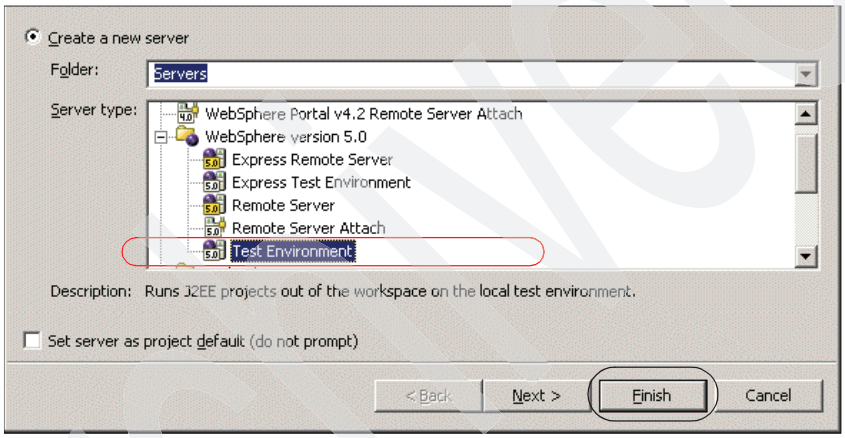


Figure 10-7 Select a WebSphere V5.0 Test Environment to run the servlet

4. The internal Web browser opens. Because this servlet is secured, you have to enter a user name and password. Enter `user1` as the user name and `password1` as the password.

Archived

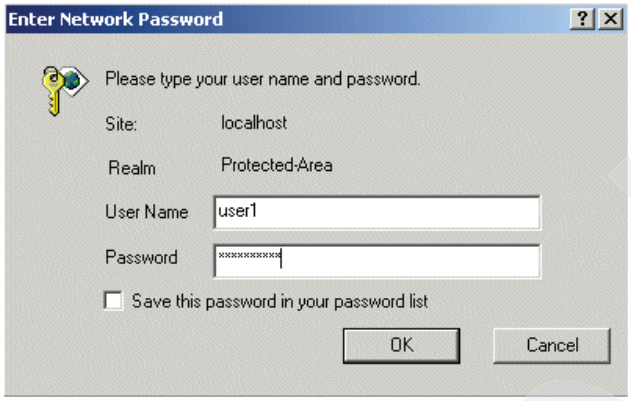


Figure 10-8 Basic authentication

5. Click **OK**. Now the browser should show the treasure servlet. See Figure 10-9.

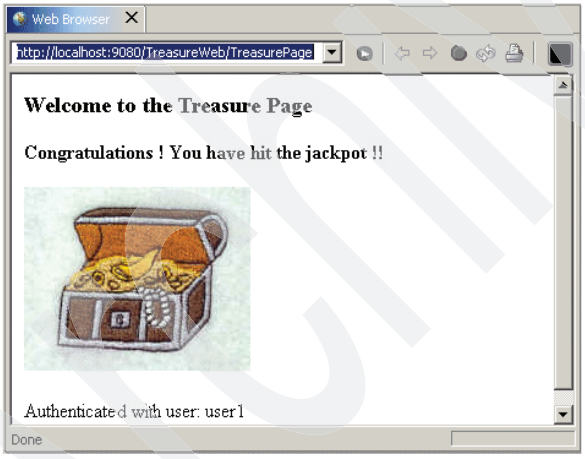


Figure 10-9 Running the secured servlet

6. From the Servers view on the bottom of WebSphere Studio, select the **WebSphere V5 Test Environment** server and click the red **Stop** button to stop the server.

Archived

10.3 Using active credentials

After importing and testing the protected servlet, you will build a portlet application accessing the Treasure servlet and using active credential objects.The portlet will be created based on the Basic portlet type and will demonstrate the use of credentials. Once the project is created, you will run it in the WebSphere Portal Test Environment to view it.

Creating the Credential Vault portlet application

To create the new portlet project, follow these steps:

1. Switch to the Portlet perspective (**Window -> Open Perspective**).
2. Select **File -> New -> Other**.

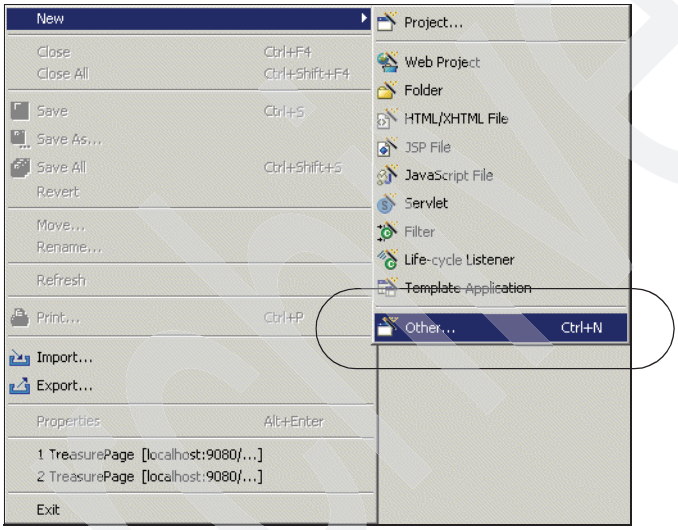


Figure 10-10 Invoking New Project wizard

3. Select **Portlet development** -> **Portlet application project.** Click **Next**.

Abstract

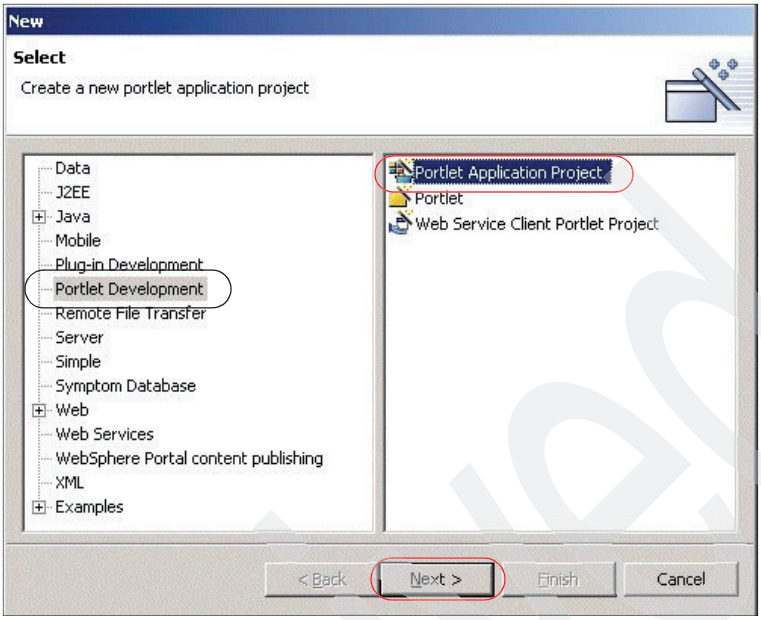


Figure 10-11 Creating a new portlet application

4. In the Define the Portlet Project page, enter a project name of `CredVaultBasicAuth` and click **Next**.

Archived

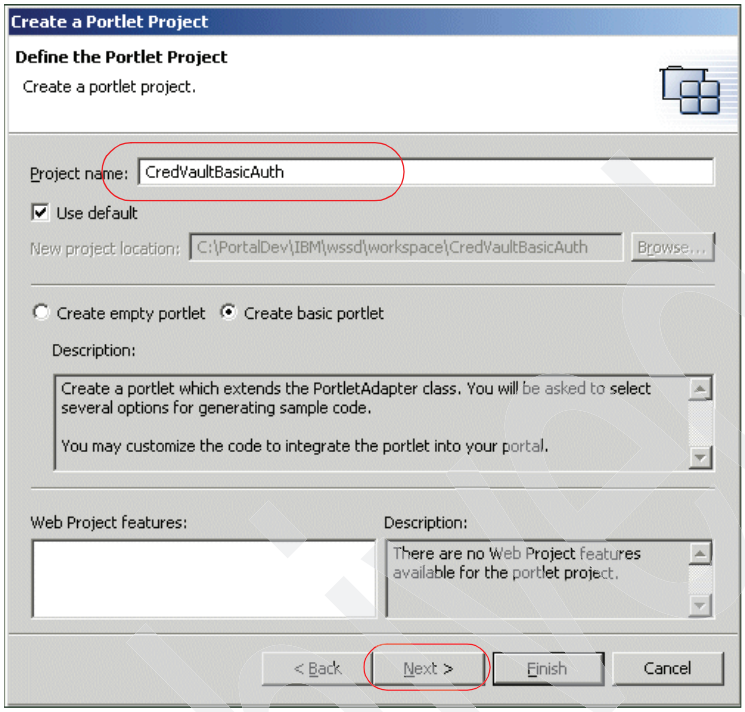


Figure 10-12 Define the Portlet Project

5. In the J2EE Settings Page, leave the defaults and click **Next**.
6. In Portlet Settings, accept all values. Click **Next**.
7. In Event Handling, uncheck **Add form sample** so that only **Add action listener** is checked. Click **Next**.

Archive

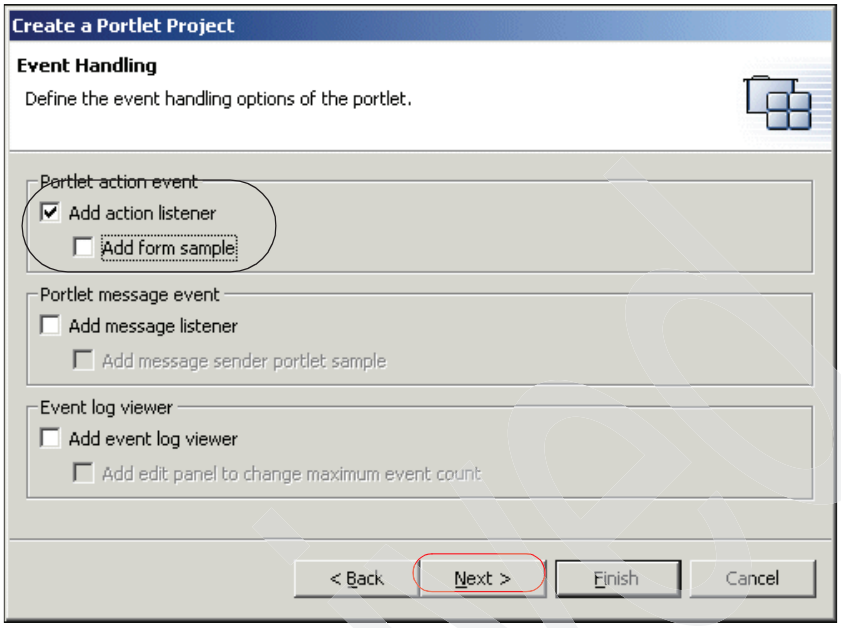


Figure 10-13 Event Handling page of New Portlet Application wizard

8. In the Single Sign-On page, check **Add credential vault handling** and enter a slot name of `TreasureCredentialSlot`. Click **Next**.

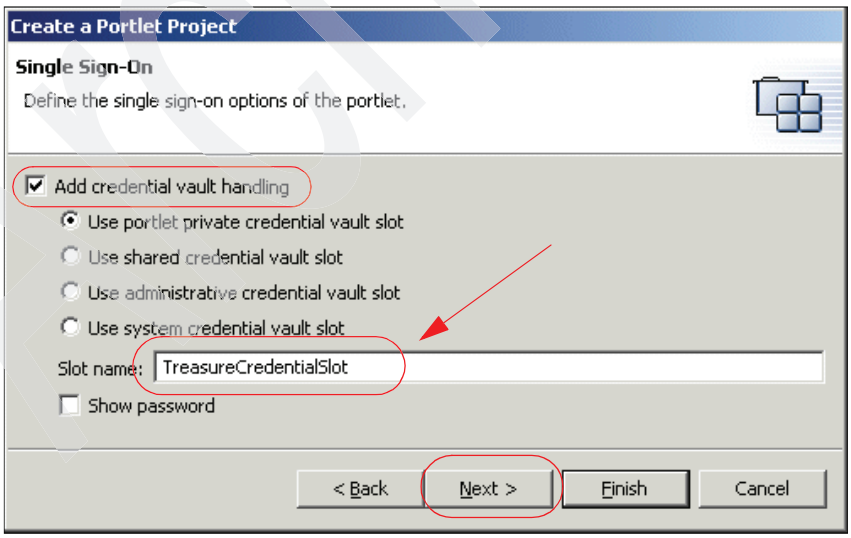


Figure 10-14 Single Sign-On page of the new portlet application wizard

9. Since no additional markups and no additional modes will be supported in this scenario, click **Finish** to generate the portlet. After a few minutes, the portlet deployment descriptor of the new portlet application opens.

Reviewing the generated code

Before the portlet code is modified to access the secure portlet, let's examine the wizard generated code.

If you expand the credvaultbasicauth package in the Source folder of the new project, you can see a CredVaultBasicAuthSecurityManager class in addition to the portlet and bean classes. This class is responsible for initializing the Credential Vault service and administering the credentials.

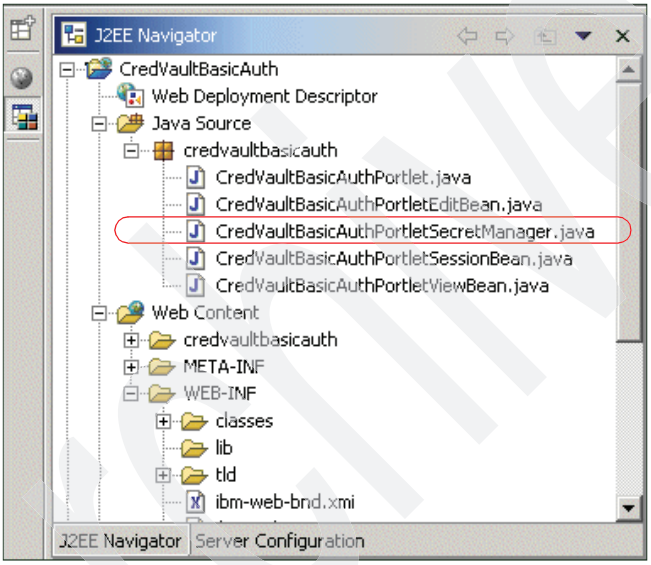


Figure 10-15 Reviewing CredVaultBasicAuthSecurityManager class

The following methods are provided in this class to handle Credential Vault issues:

- The init method of this class initializes the vaultService data member.
- getCredential returns the user name and password by using a string buffer.
- setCredential sets the user name and password.
- getSlotId returns the ID of the slot. Depending on the type of slot, this method uses PortletData or VaultService to get the ID.
- New slots are created in the createNewSlot method.

- getPrincipalFromSubject retrieves the specified Principal from the provided subject.
- isWritable checks whether the password can be saved.

The wizard has also created an input form for a user ID and password in the CredVaultBasicAuthPortletEdit.jsp. As previously described, when clicking the **Save** button, the actionPerformed() method in the portlet class is called. This method retrieves the user ID and password from the form and uses the security manager class to set the credentials.

The current version of the doView method retrieves the user credentials from the security manager and displays them in the JSP. Because we want to include the content of the secured Treasure servlet, we will replace this method in the next section of this scenario.

10.3.1 Updating the generated portlet

Modify the portlet application as follows:

1. Open CredVaultBasicAuthPortletSecretManager from the credvaultbasicauth package.
2. Using the Java editor, add the method shown in Example 10-1 to the class. The method can be found in the c:\LabFiles\CredentialVault\Snippets folder.

   Note: The sample scenario included in this chapter requires that you download the sample code available as additional materials. See Appendix C, “Additional material” on page 543.

   You may want to use WordPad to edit getConnectionUsingActiveObject.java and then copy and paste.

   Note: The getConnectionUsingActiveObject method returns an http connection.

Example 10-1 getConnectionUsingActiveObject method (active credentials)

```
public static HttpURLConnection getConnectionUsingActiveObject(
    PortletRequest portletRequest,
    CredVaultBasicAuthPortletSessionBean sessionBean,
    String host, String port, String path) {
    HttpURLConnection connection=null;
    try {
      URL urlSpec =
         new URL("http://" + host + ":" + port + path);
      String slotId = getSlotId(portletRequest, sessionBean, false);
      if (slotId != null) {
         HttpBasicAuthCredential credential =
            (HttpBasicAuthCredential) vaultService.getCredential(
```

```
                    slotId,
                    "HttpBasicAuth",
                    new HashMap(),
                    portletRequest);

            connection = credential.getAuthenticatedConnection(urlSpec);
         }
      } catch (Exception e) {
         e.printStackTrace();
      }
      return connection;
   }
```

3. Some code errors appear because the required import statements are missing. To fix these errors, right-click the Java editor and select **Source -> Organize Imports**.
4. In the Organize Imports dialog, choose
   a. **java.net.HttpURLConnection**
   b. **select java.net.URL**

   Click **Finish** to close the Organize Imports dialog.

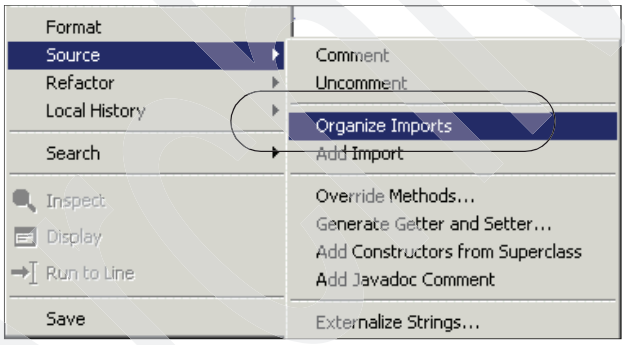


Figure 10-16 Importing missing import statements using Organize Imports tool

5. Save and close the Java file.
6. Open the class CredVaultBasicAuthPortlet from the credvaultbasicauth package.
7. Replace the doView method so it looks as shown in Example 10-2 on page 337. You may want to copy and paste from c:\LabFiles\CredentialVault\Snippets\doView.java.

   Note: The sample scenario included in this chapter requires that you download the sample code available as additional materials. See Appendix C, “Additional material” on page 543.

Example 10-2 The doView method uses a Http connection from the SecretManager class

```
   public void doView(PortletRequest request, PortletResponse response)
      throws PortletException, IOException {
      // Check if portlet session exists
      CredVaultBasicAuthPortletSessionBean sessionBean =
         getSessionBean(request);
      if (sessionBean == null) {
         response.getWriter().println("<b>NO PORTLET SESSION YET</b>");
         return;
      }
      // get output stream to write the results
      PrintWriter writer = response.getWriter();
      // get the CredentialVault PortletService
      PortletContext context = this.getPortletConfig().getContext();
      try {
         String host = request.getServerName();
         //String host = request.getRemoteHost();
         String port = String.valueOf(request.getServerPort());
         String path = "/TreasureWeb/TreasurePage";
         HttpURLConnection connection =

CredVaultBasicAuthPortletSecretManager.getConnectionUsingActiveObject(
               request, sessionBean, host, port, path );
         if (connection != null) {
            connection.connect();
            String responseMessage = connection.getResponseMessage();
            int responseCode = connection.getResponseCode();
            // Were we successful?
            if (HttpURLConnection.HTTP_OK == responseCode) {
               writer.println("<P>Successfully connected!</P>");
            } else {
               writer.println(
                  "<P>Unable to successfully connect to back end."
                     + ", HTTP Response Code = " + responseCode
                     + ", HTTP Response Message = \"" + responseMessage
                     + "\"</P>");
            }
            BufferedReader br =
               new BufferedReader(
                  new InputStreamReader(connection.getInputStream()));
            String line;
            while ((line = br.readLine()) != null)
               writer.println(line + "\n");
         } else {
            writer.println(
               "<h2>Credential not found. Please set it in the edit mode!
</h2>");
            return;
```

```
            }
        } catch (IOException exc) {
            writer.println(
                "<h2>Single-sign-on error, login at back-end failed! </h2>");
            return;
        }
    }
```

8. Organize the import statements as you did before.
9. Save and close the Java file.

Important: If you get a message indicating that getConnectionUsingActiveObject() is undefined, try making a small modification to the file and enabling the save option. Save the file again. This procedure should resolve any pending undefined issues.

10.3.2 Running the portlet

In this section, you will run the portlet using active credentials to access the back-end resource, a protected servlet in this case.

1. Close any open browser viewers.
2. Switch to the Portlet perspective.
3. In the Server Configuration view, right-click the **Servers** folder and choose **New -> Server and Server Configuration.**

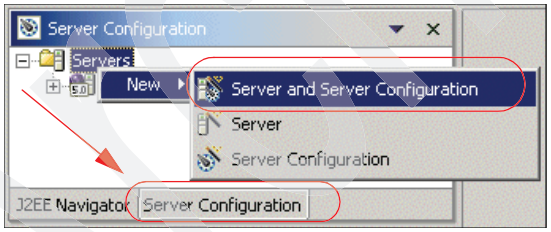


Figure 10-17 Creation of a new server

4. In the Server Selection dialog, choose a server of the WebSphere Portal V5.0 Test Environment and enter a server name of `WPS 5.0`. Click **Finish** to add the new server.

Archived

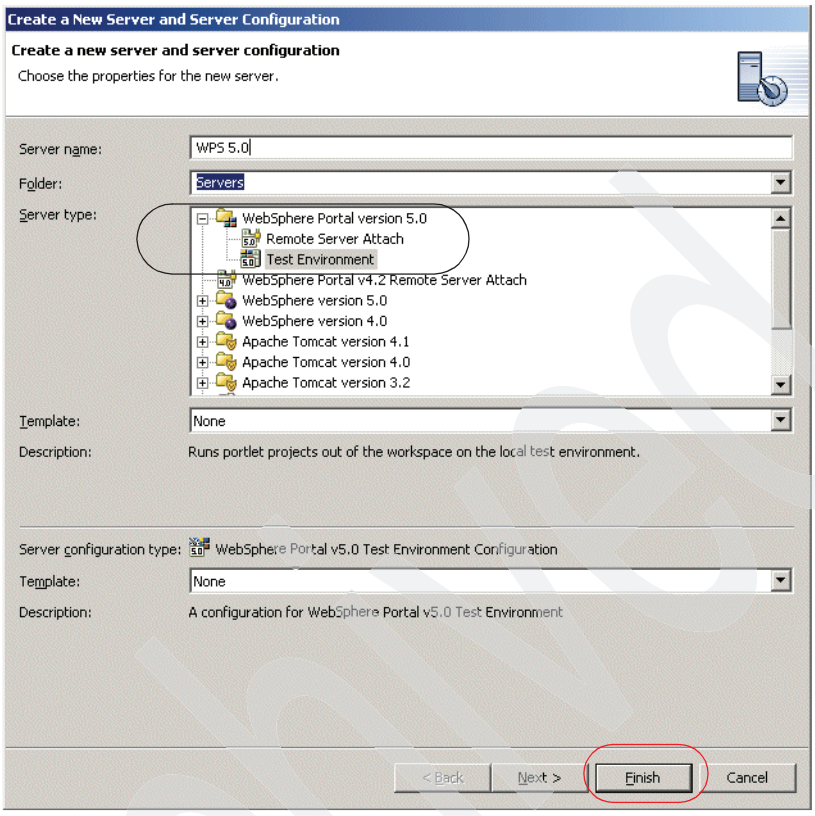


Figure 10-18 Create a new WebSphere Portal Test Environment

5. Add the Treasure servlet to the portal test environment, right-click the **WPS5.0** server and choose **Add -> TreasureEAR**.

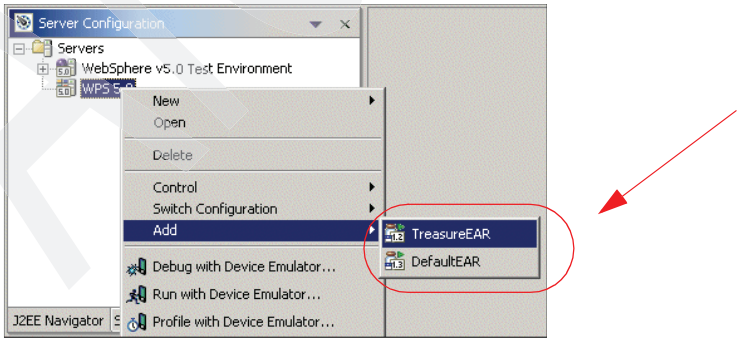


Figure 10-19 Add Treasure servlet and CredVault portlet to portal test environment

6. Repeat the previous step to add the DefaultEAR to the portal Test Environment. This will also add the CredVaultBasicAuth portlet to the server.
7. In the J2EE Navigator view select **CredVaultBasicAuth**, choose **Run on server** and wait a few minutes for the Portal server to open for e-business. This will start the server and will also open a browser displaying the portlet.

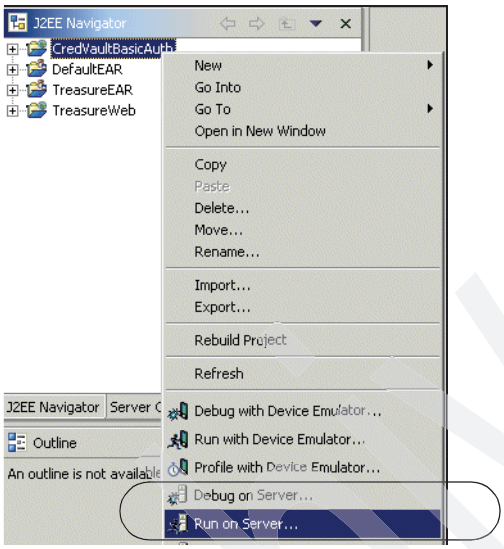


Figure 10-20 Selecting Run on Server... to test the portlet.

8. The portlet will execute the initConcrete method to initialize the Credential Vault Service and the doView method. Since there are no credentials yet, a message is displayed.
9. Switch to the Edit mode and enter the following information:
   - User ID: `user1`
   - Password: `password1`
10. Submit the action. This will generate an action that will be checked by the actionPerformed method in the CredVaultBasicAuthPortlet class. The portlet returns to View mode, showing the contents of the Treasure Servlet.
11. In the Servers view, stop the running portal server.

Archived

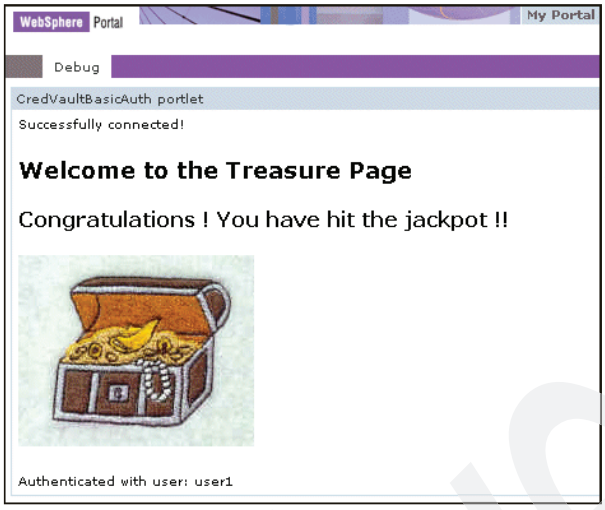


Figure 10-21 The CredentialVault portlet in action

10.4 Using passive credentials

In the previous section, a portlet using an active credential object was built. Although this is the preferred type of credential object, there are certain cases where you have to use passive credential objects, for example when an appropriate active credential class is not available.

In this sample scenario, the portlet will be changed to use a passive credential object.

To modify the portlet application, proceed as follows:

1. Open the class CredVaultBasicAuthPortletSecretManager from the credvaultbasicauth project.
2. In the Java editor, add the following method to the class.

Example 10-3 The getConnectionUsingPassiveObject method (passive credentials)

```
public static HttpURLConnection getConnectionUsingPassiveObject(
        PortletRequest portletRequest,
        CredVaultBasicAuthPortletSessionBean sessionBean,
        String host, String port, String path) {
        StringBuffer userid = new StringBuffer("");
        StringBuffer password = new StringBuffer("");
        HttpURLConnection connection = null;
        try {
```

```
            getCredential(portletRequest, sessionBean, userid, password);
            if (!userid.toString().equals("")) {
               String userAndPassword =
                  new String(userid.toString() + ":" + password.toString());
               byte[] userAndPasswordBytes = userAndPassword.getBytes();
               BASE64Encoder encoder = new BASE64Encoder();
               String basicAuth =
                  new String(encoder.encode(userAndPasswordBytes));
               basicAuth = "Basic " + basicAuth;
               URL url = new URL("http://" + host + ":" + port + path);
               connection = (HttpURLConnection) url.openConnection();
               connection.setRequestProperty("authorization", basicAuth);
            }
         } catch (Exception e) {
            e.printStackTrace();
         }
         return connection;
      }
```

3. Organize the import statements as you did in the previous scenario.
4. Save and close the Java file.
5. Open the class CredVaultBasicAuthPortlet from the credvaultbasicauth package.

   In the doView method, change the line:

   ```
   HttpURLConnection connection =
   CredVaultBasicAuthPortletSecretManager.getConnectionUsingActiveObject(reque
   st,   sessionBean,    host, port, path );
   ```

   to

   ```
   HttpURLConnection connection =
   CredVaultBasicAuthPortletSecretManager.getConnectionUsing Passive Object(requ
   est,  sessionBean,   host, port, path );
   ```

6. Save and close the Java file.
7. Start the server and test the servlet as described in 10.3.2, “Running the portlet” on page 338. Now the portlet runs exactly the same way as it did in the previous section when it was using active credentials. This time, the getConnectionUsingPassiveObject method has access to the credentials.

Note: Having access to credentials could be a security risk, so when possible, use always active credential objects.

Archived

11

Accessing back-end JDBC databases

In this chapter, we introduce the process of gaining access to back-end JDBC databases from portlet applications. A simple portlet application project is included to show how portlet applications access relational databases using the JDBC interface.

This chapter discusses the following topics:

- How to create a connection using WebSphere Studio Site Developer
- An example to access a database from a portlet application

© Copyright IBM Corp. 2004. All rights reserved.

11.1 Creating a database connection

In this section, we describe the process of creating a database connection and generating the Java classes similar to the classes used in the sample scenario included in this chapter.

In section 11.2, “Sample scenario” on page 353, a portlet application project to access a JDBC database is created from a WAR file; in this implementation, the JDBC connection is already included. This section explains how to create the database connection using the wizard provided by WebSphere Studio Site Developer.

Once a portlet application project has been created (for example, we called this project HRPortlet), open the data perspective by selecting **Window -> Open Perspective -> Data**.

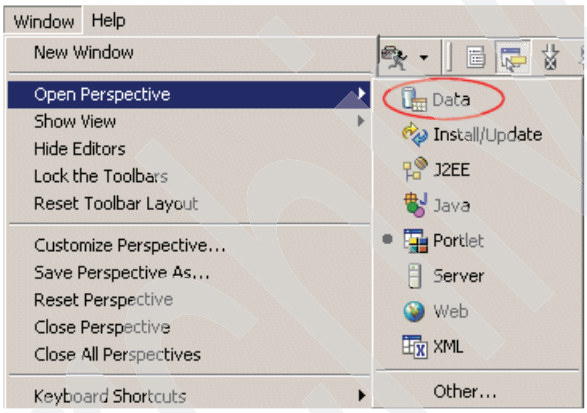


Figure 11-1 Open data perspective

11.1.1 Creating a new connection

In the DB Servers view, select **New Connection** from the context menu to create a new connection. Enter the following values:

- Connection name: for example `ConnHR`
- Database: sample scenario uses `WSSAMPLE`
- User ID: sample scenario uses `db2admin`
- Password: sample scenario uses `db2admin`
- Database vendor type: `Cloudscape, V5.0`
- JDBC driver: `Cloudscape Embedded JDBC driver`

- Database location: `C:\LabFiles\Cloudscape\WSSAMPLE`

  Note: The sample scenario included in this chapter requires that you download the sample code available as additional materials. See Appendix C, “Additional material” on page 543.

- Class location: the directory where db2.jar file is located

Click **Finish**.

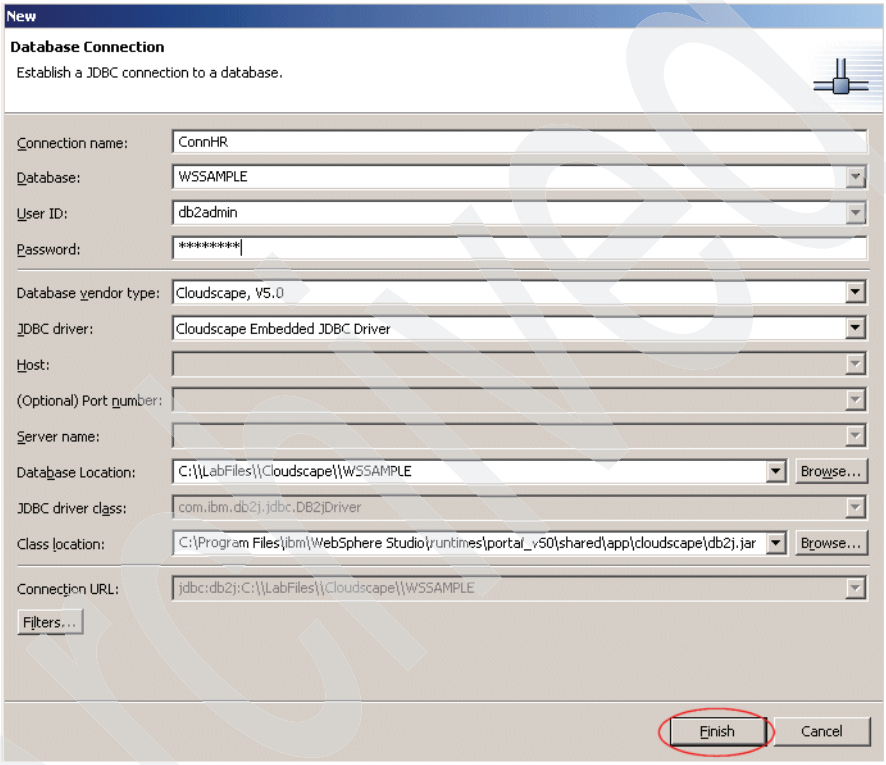


Figure 11-2 Create new connection

Note that we assume that the database has been created in your system. The new database entry is added to the DB Servers view; you can expand it to see the schemas and tables but no editor will open because this view cannot be used for editing.

11.1.2 Importing to a folder

Now you have to import the tables to use them in the application. Select the connection, right-click it and select **Import to Folder**.

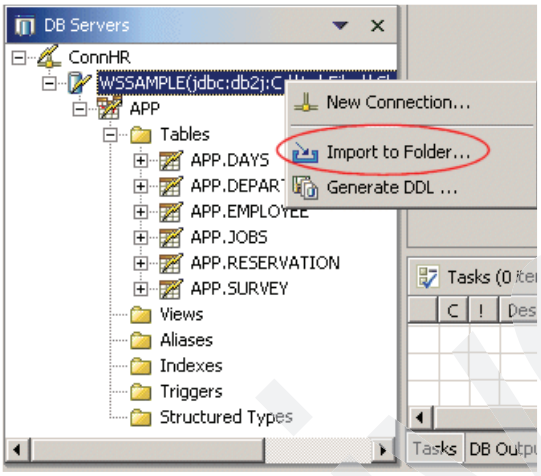


Figure 11-3 Import the database into a project.

In the Import to Folder window, click **Browse** to select the portlet application project where you want to gain access to the back-end system. Click **Finish**.

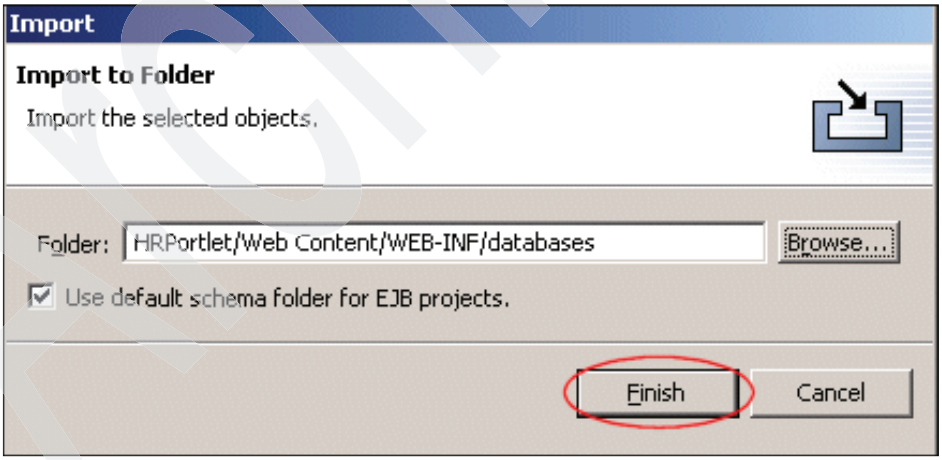


Figure 11-4 Import to folder

If you are prompted to create a databases folder, click **Yes**. Now, in the Data Definition view, you can see the schemas and tables as in DB Servers view, but in this view you can update schemas and tables definitions by double-clicking

Archived

them. In the Navigator view, XML files have been created for the database objects.

11.1.3 Creating an SQL statement

A simple SQL statement will be created. In the Data Definition view, select the statement folder under the sample WSSAMPLE database, right-click and select **New -> Select Statement.** Enter for example SQLUtility for the Statement Name and click **OK**.

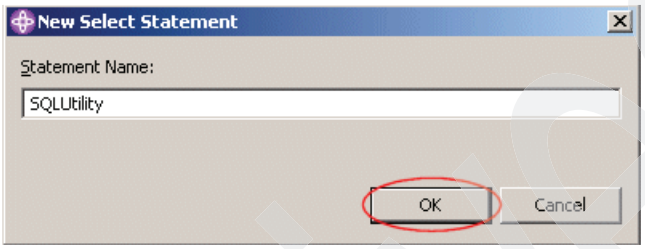


Figure 11-5 Create a new statement

Note: If you cannot see the statement folder, close the Data perspective and open it again.

When the statement has been created, the editor opens and you can select a table to make a query, for example the Employee table. There are different ways to create a SQL statement, with the wizard or just by writing your query using the editor. For details, see for example the IBM Redbook *WebSphere Studio Application Developer V5 Programming Guide*, SG24-6585.

11.1.4 Generating Java classes

In the code provided in 11.2, “Sample scenario” on page 353, the SQLUtilities.java class provides methods that execute the SQL statement, returns a DBSelect reference and returns an array of objects representing the rows in the result set. To create this class, right-click the **SQLUtility** statement and select **Generate Java Bean**.

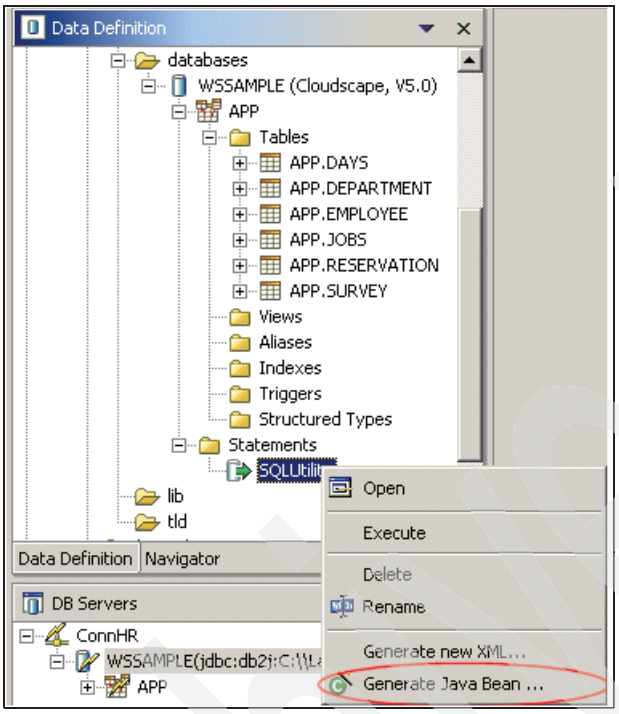


Figure 11-6 Generate Java Bean

In the first screen of the Generate Java Bean option, enter the following information:

- Source Folder: enter the Java Source folder of your portlet application project.
- Package: `utilities`
- Name: `SQLUtilities`

Click **Next**.

Archived

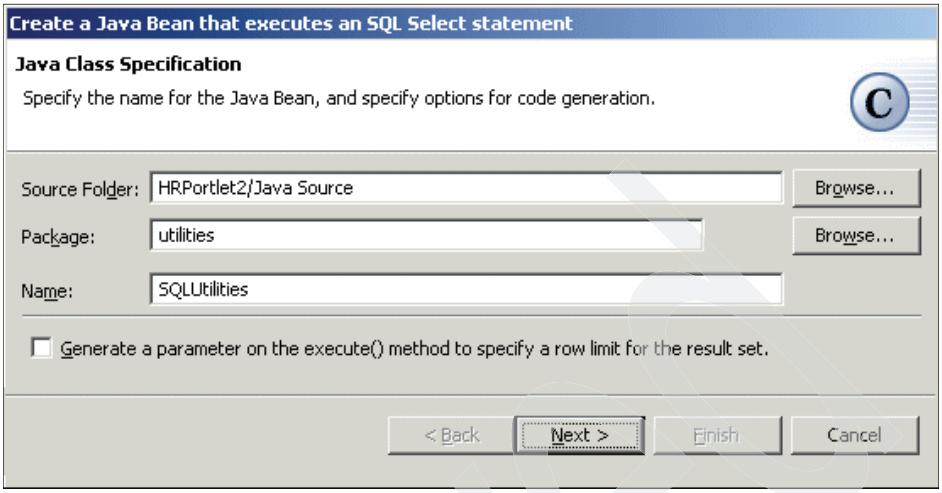


Figure 11-7 First screen of Generate Java Bean

In the next window, enter the following options:

- Use Driver Manager Connection:
  - Driver name: `com.ibm.db2j.jdbc.DB2jDriver`
  - URL: `jdbc:db2j:C:\\LabFiles\\Cloudscape\\WSSAMPLE`

  Note: The sample scenario included in this chapter requires that you download the sample code available as additional materials. See Appendix C, "Additional material" on page 543.
- For the option *How will user authentication be provided* select **By the execute() method's caller**.

Click **Next**.

Archive

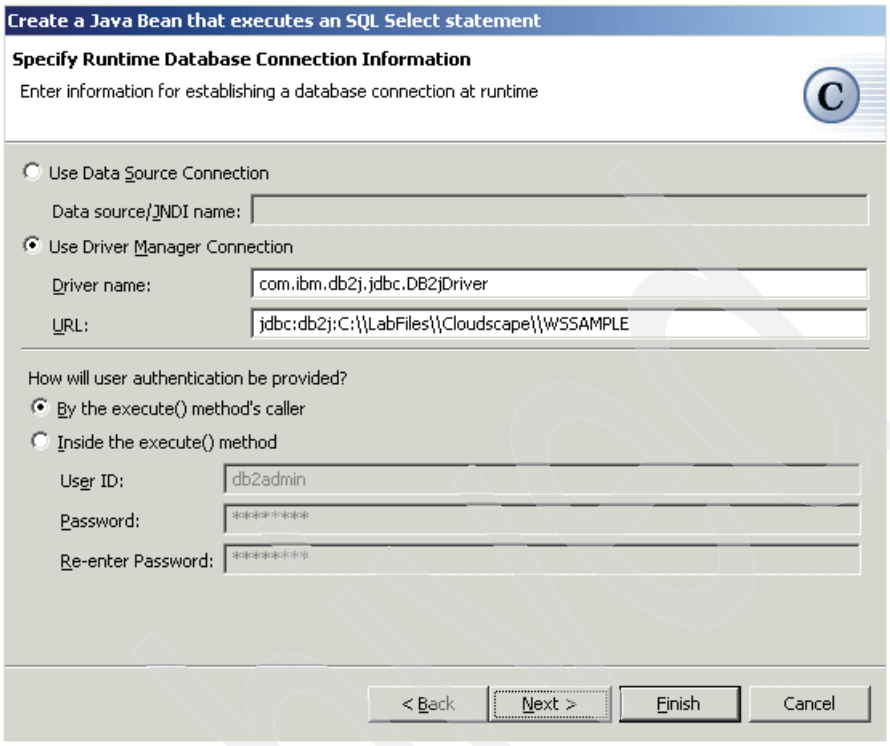


Figure 11-8 Second screen of Generate Java Bean

Figure 11-9 on page 351 shows the methods which will be created. Click **Finish**.

Archive

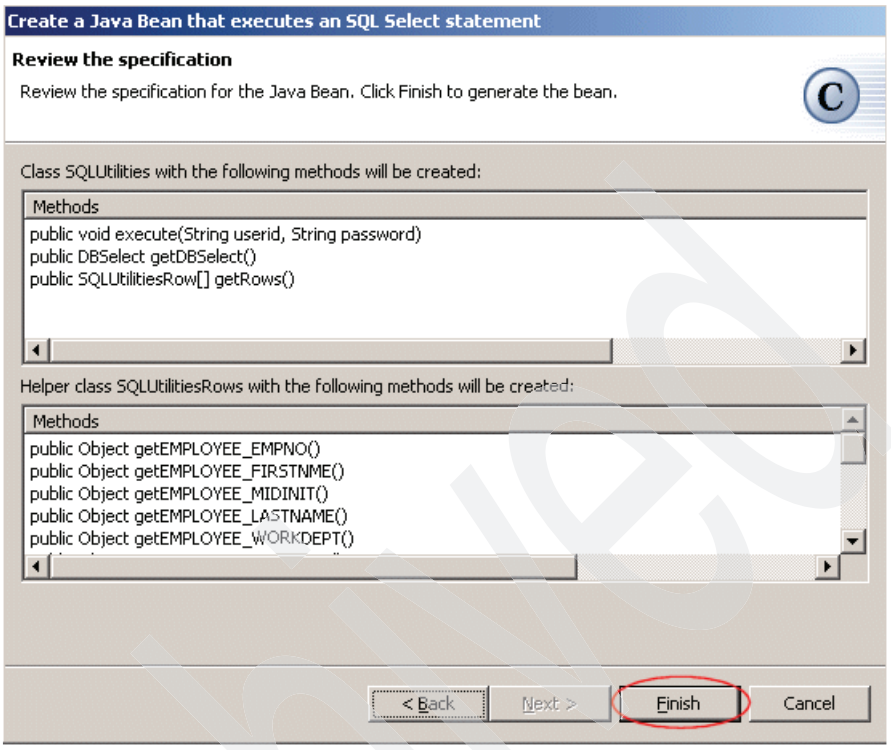


Figure 11-9 Review the specification window

11.1.5 Running the SQL statement

In the Data Definition view, right-click the SQL statement and select **Execute** as shown in Figure 11-10 on page 352.

Archived

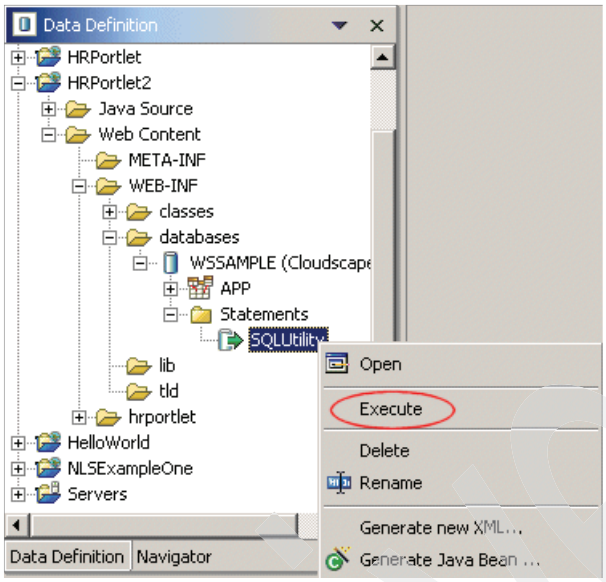


Figure 11-10 Execute a SQL statement

The results appear in the DB Output view.

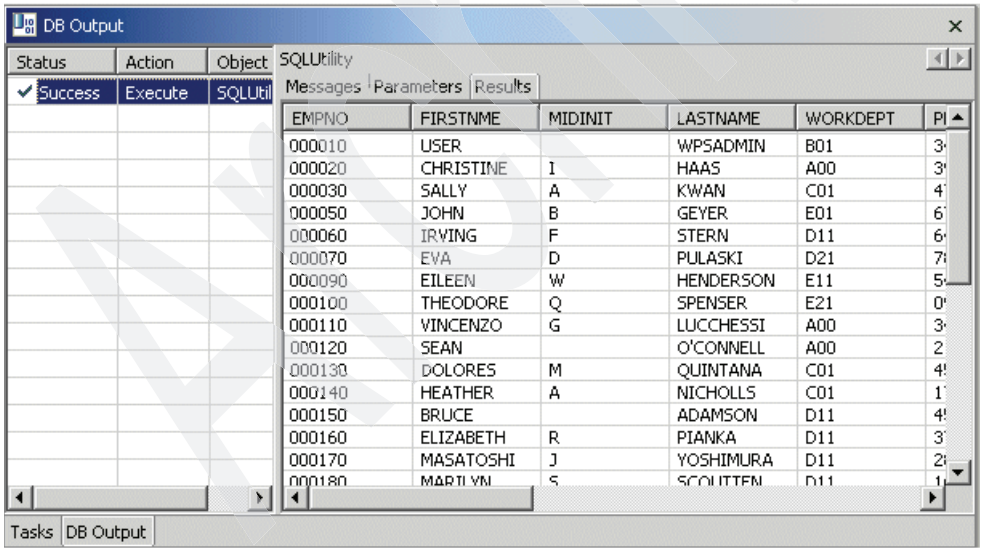


Figure 11-11 Results of the execution

11.2 Sample scenario

This section provides a sample scenario for creating a sample portlet project that uses the JDBC interface to interact with relational database back end systems. You will create, deploy and run this portlet application. This sample scenario will allow you understand the techniques used to develop portlets that retrieve information from databases using JDBC.

The development workstation has already been created for you and its components can be seen in Figure 11-12.

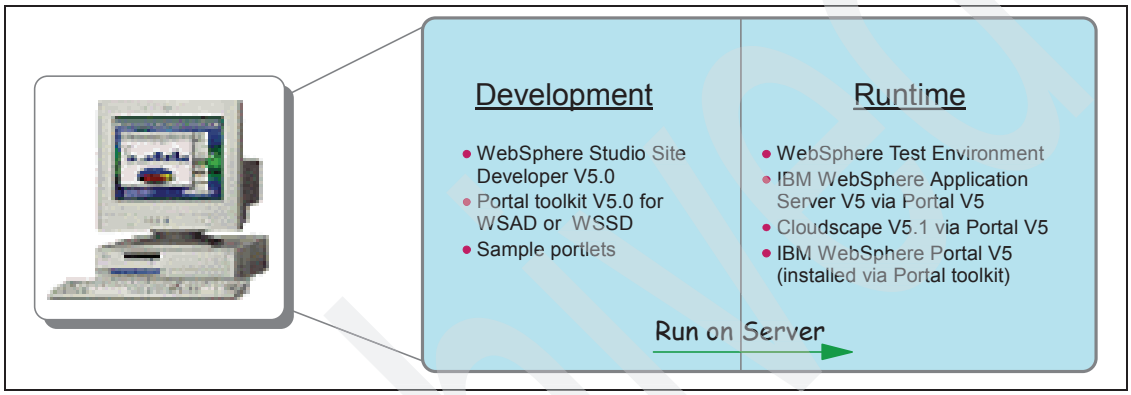


Figure 11-12 Development workstation

11.2.1 Overview

In this section, you will review and understand the sample scenario. You will create a sample portlet based on a Basic portlet type. You will then import the code to use JDBC to interact with databases. In this sample scenario, it is assumed that the database has already been created and populated.

This example shows how the JDBC interface is used to read information from a Cloudscape sample table. In your portlet Edit mode, you will provide the information needed to establish a connection with the database in order to retrieve the content in View mode. The sample scenario is illustrated in Figure 11-13 on page 354.

Archive

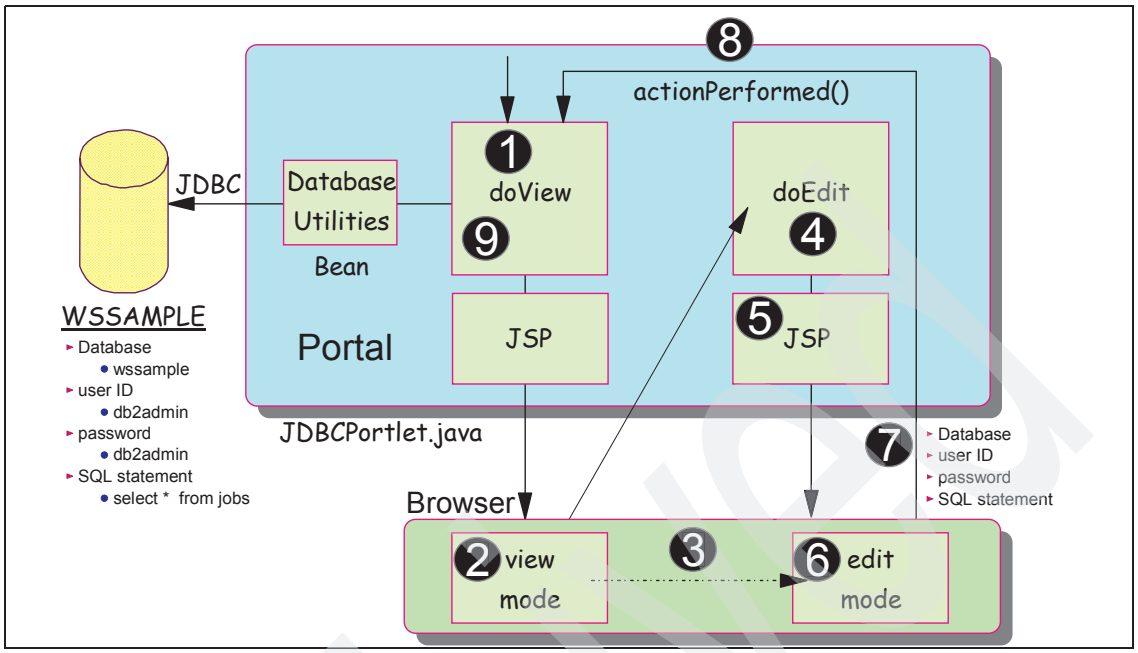


Figure 11-13 JDBC scenario

The sequence flow for this sample scenario is as follows:

1. Initially, the doView method is executed.
2. A JSP is invoked in View mode to render the initial screen containing a welcome message telling the user to enter the database inquiry values in Edit mode.
3. The user clicks **Edit** to go into Edit mode.
4. The Edit mode method executes and invokes a JSP (include).
5. The JSP renders the form.
6. The user enters the database name, user ID, password and SQL statement.

   Note: A user ID and password have not been implemented for this version of Cloudscape, therefore any user ID and password can be used.
7. The user submits the request (post) to the previous mode (View mode). An action event is generated.
8. The actionPerformed() method executes and the following processes take place:

   a. The database, user ID, password, and SQL statement are extracted and sent to the JDBCPortletResults Bean.

b. A connection object is created.

c. A DBResults object is created. This object encapsulates the database inquiry.

d. Database Utilities (another bean) is invoked to execute the actual database inquiry.

e. Results are stored as a String in the request object.

9. The doView method executes again:

a. Results are obtained in View mode.

b. Results are rendered directly in View mode.

11.2.2 Creating HRPortlet

The portlet will be created based on a Basic portlet type using the wizard. In this section you create a portlet application with name **HRPortlet.** The portlet application will also be published and executed in the Portal Test Environment.

1. If not already running, start the IBM WebSphere Studio Site Developer and click **Start -> Programs -> IBM WebSphere Studio -> Site Developer 5.0**.

2. Select **File -> New -> Portlet Application Project**.

   Note: If you do not see this option, select **File -> New -> Other** and then **Portlet Development** and **Portlet Application Project**.

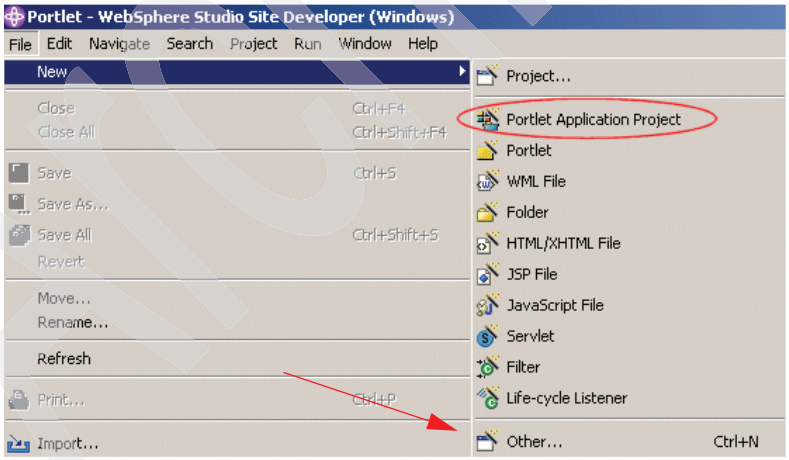


Figure 11-14 Starting creation of Portlet project

3. In Define Portlet Project, enter `HRPortlet` for the project name. Click **Next**.

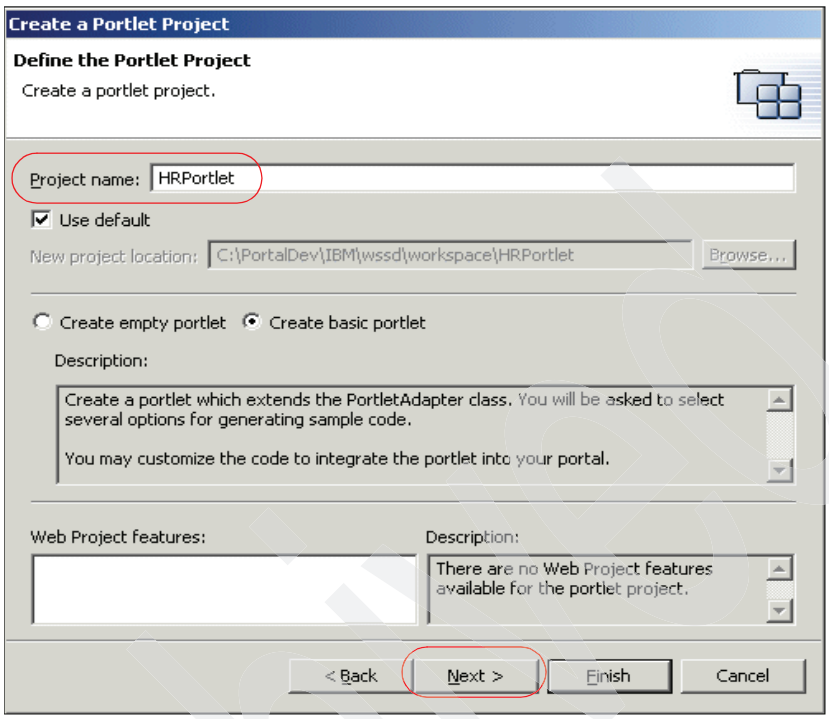


Figure 11-15 Define the Portlet Project

4. In the J2EE Settings Page, click **Next** to accept default values.

Archive

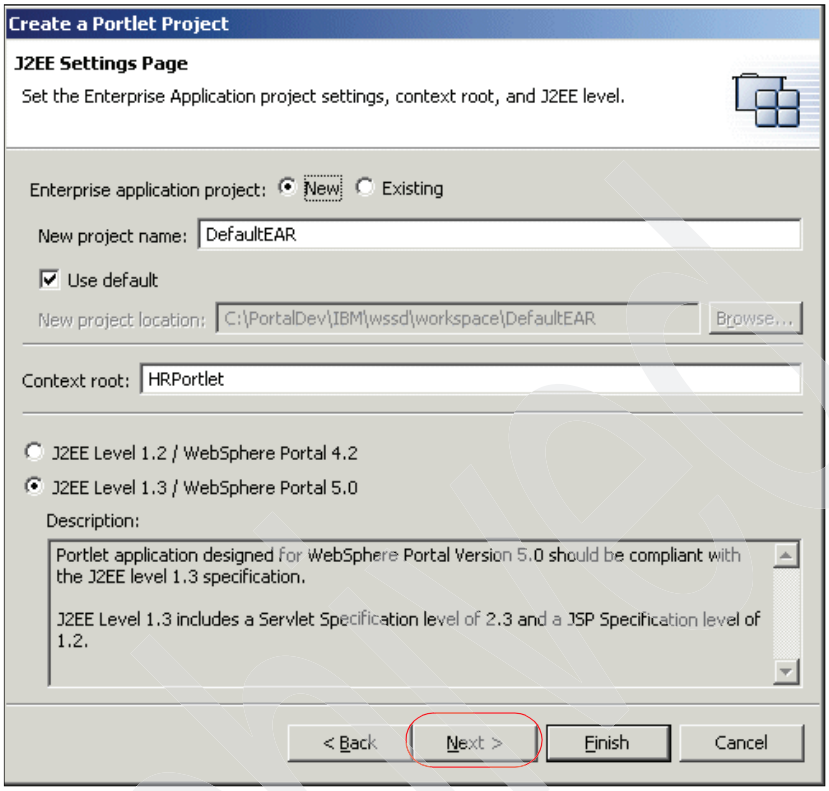


Figure 11-16 J2EE Settings Page

5. In Portlet Settings, check **Change code generation options** and enter `HRPortlet` for the Class prefix. Click **Finish** to generate the framework for your project.

Archive

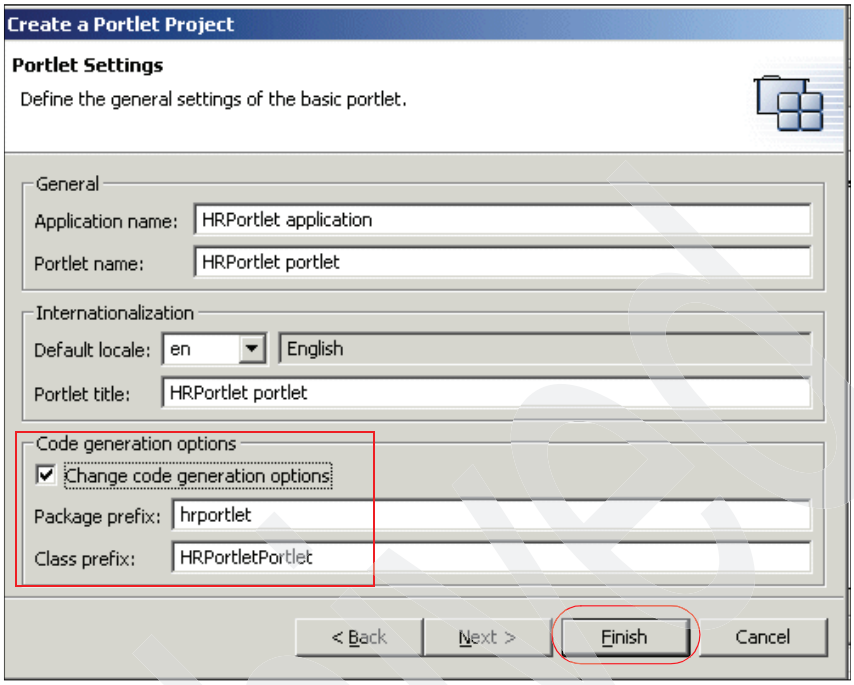


Figure 11-17 Portlet Settings

6. If you have any portlets in the DefaultEAR project, remove them at this time. For example:
   a. Open the DefaultEAR / META-INF folder.
   b. Double-click **om application.xml** file.
   c. Select **Module**.
   d. Remove all WAR modules except for HRPortlet.
   e. In the workspace, click **File -> Save All**.

Archive

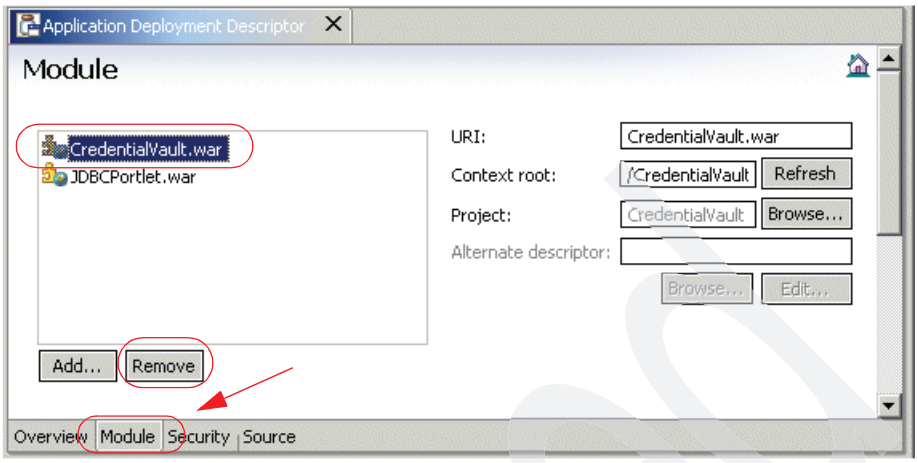


Figure 11-18 Removing a WAR module

11.2.3 Importing the WAR file

A WAR file is provided for this sample scenario. By importing this WAR file into your project, you will replace the original files previously created by the wizard. For example:

- HRPortlet.java (Portlet class)
- HRPortletSessionBean.java (Session bean)
- HRPortletEditBean.java (Bean)
- HRPortletEdit.jsp (Edit Mode)
- HRPortletView.jsp (View Mode)
- portlet.xml (Portlet deployment descriptor)
- web.xml (Web deployment descriptor)
- A new class SQLUtilities.java (Utility) to set the DBSelect properties values and create methods to execute SQL statements
- A new class SQLUtilitiesRow.java (Utility) to retrieve each row of the result set
- A new class HRPortletViewBean.java (Bean) to store the DBResult object

Follow these steps to import the WAR file:

1. Import the WAR file by selecting **File -> Import**.

Archive

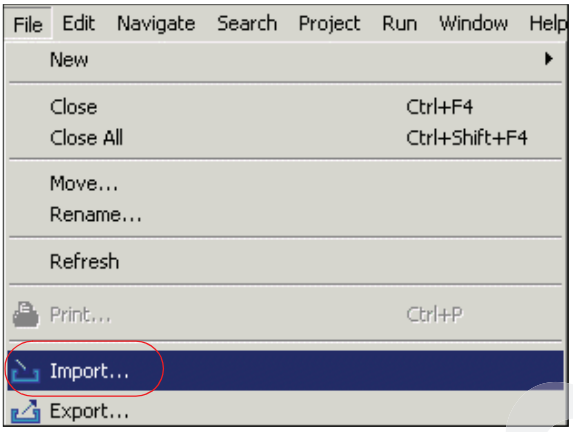


Figure 11-19 Importing a WAR file

2. Select **WAR file** and click **Next**.

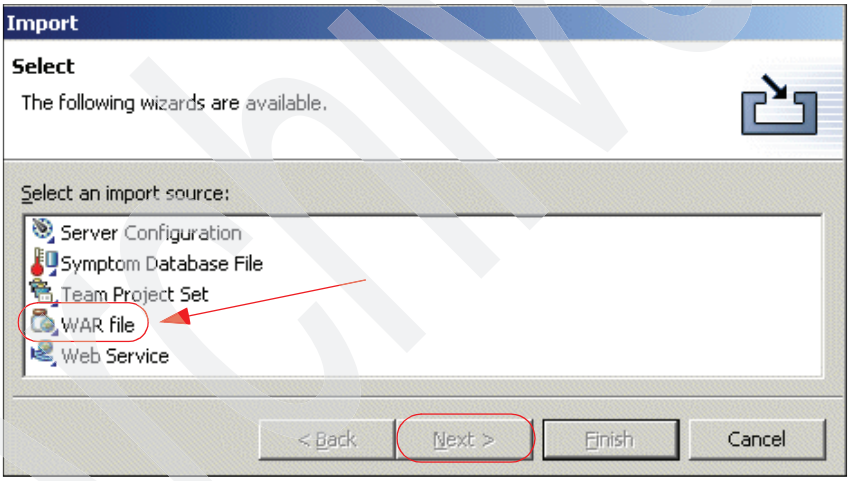


Figure 11-20 Select WAR file

3. In the *Import Resources from a WAR file* window, enter the following information:
   a. WAR file: browse to `C:\LabFiles\JDBC\HR\HRPortlet.war`

      Note: The sample scenario included in this chapter requires that you download the sample code available as additional materials. See Appendix C, “Additional material” on page 543.

Archived

b. Web project: select **Existing**. In the box that pops up, select **HRPortlet** and click **OK**.

c. Context root: this will change to `/HRPortlet`.

d. Select in Options: **Overwrite existent resources without warning**.

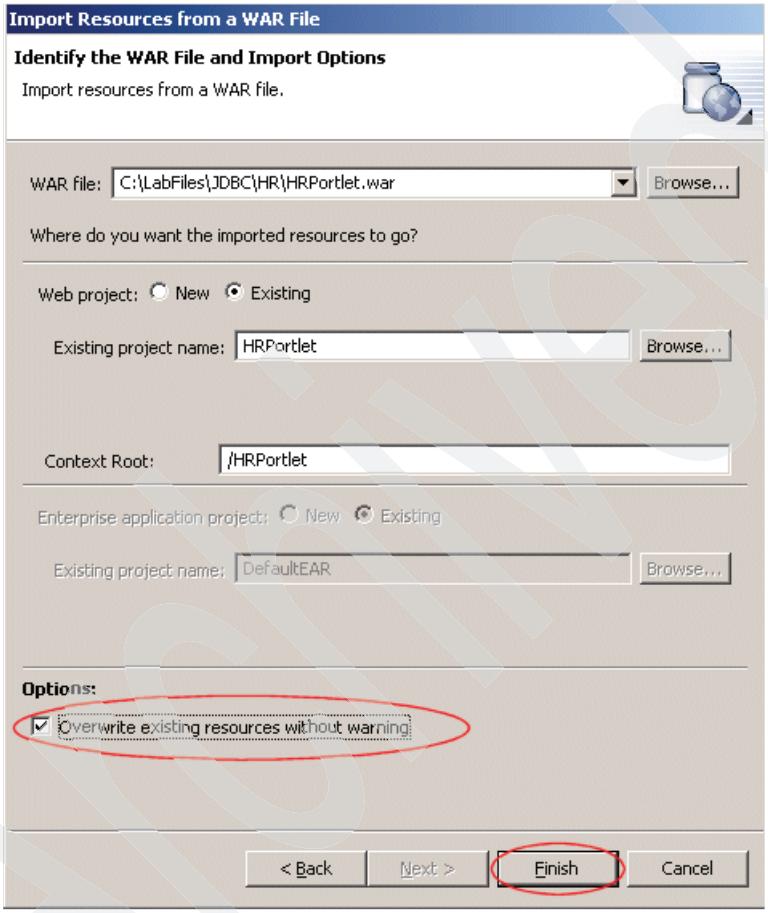


Figure 11-21 Importing the provided WAR file

4. Click **Finish**.

11.2.4 Reviewing the portlet code

In this section, you will review the portlet code used in this sample scenario. For example, to view the source code to HRPortlet.java, double-click the file name. This file is located in the /Java Source/hrportlet/ folder.

1. The actionPerformed() method in this portlet does the following (see Example 11-1) when an edit action occurs.
   a. It gets the parameters (database, user ID, password and SQL statement) from the request (PortletRequest).
   b. It sets these values in the HRPortletSessionBean.java bean.

Example 11-1 HRPortlet.java - actionPerformed() method

```
.....
      // ActionEvent handler
      String actionString = event.getActionString();
      // Add action string handler here
      PortletRequest request = event.getRequest();

      HRPortletSessionBean sessionBean = getSessionBean(request);

      if (EDIT_ACTION.equals(actionString)) {
         String dbname = (String) request.getParameter("dbname");
         String userid = (String) request.getParameter("userid");
         String password = (String) request.getParameter("password");
         String sqlstring = (String) request.getParameter("sqlstring");
         sessionBean.setDbName(dbname);
         sessionBean.setUserId(userid);
         sessionBean.setPassword(password);
         sessionBean.setSqlString(sqlstring);

      }

      if (FORM_ACTION.equals(actionString)) {
         // Set form text in the session bean
         sessionBean.setFormText(request.getParameter(TEXT));
      }
   }
.....
```

2. The doEdit() method in the portlet does the following:
   a. It creates an instance of HRPorteltEditBean bean.
   b. It adds the action which will be executed when the form is submitted and sets this value in the edit mode bean.
   c. When the database, user ID, password and SQL statement values exist for the session, it will also store these values in the edit mode bean.
   d. The edit mode bean is passed in the request to the HRPortletEdit.jsp.

Archived

Example 11-2 HRPortlet.java - doEdit() method

```
.....
      HRPortletEditBean editBean = new HRPortletEditBean();

      PortletURI formActionURI = response.createReturnURI();
      formActionURI.addAction(EDIT_ACTION);
      editBean.setFormActionURI(formActionURI.toString());
      HRPortletSessionBean sessionBean = getSessionBean(request);
      String sqlstring = sessionBean.getSqlString();
      if (sqlstring != null) {
         String dbname = sessionBean.getDbName();
         String userid = sessionBean.getUserId();
         String password = sessionBean.getPassword();

         editBean.setDbname(dbname);
         editBean.setPassword(password);
         editBean.setUserid(userid);
         editBean.setSqlstring(sqlstring);
      }
      request.setAttribute(EDIT_BEAN, editBean);

      // Invoke the JSP to render
      getPortletConfig().getContext().include(EDIT_JSP +
getJspExtension(request),
         request, response);
.....
```

3. The doView() method does the following:
   a. It checks to see whether there is a HRPortletSessionBean in the session. The first time this method is invoked, the session bean will be null.
   b. If there is an HRPortletSessionBean in session, it gets the database, user ID, password and SQL sentence values stored in it and creates an instance of HRPortletViewBean.
   c. The execute() method of SQLUtilities class is called to execute the SQL statement and the result is stored in the view mode bean by calling populateData() method in the SQLUtilities class.
   d. The view mode bean is passed in the request to HRPortletView.jsp.

Example 11-3 HRPortlet.java - doView() method

```
.....
      // Check if portlet session exists
      HRPortletSessionBean sessionBean = getSessionBean(request);
      if (sessionBean == null) {
         response.getWriter().println("<b>NO PORTLET SESSION YET</b>");
         return;
```

Archived

```
        }

        String sqlstring = sessionBean.getSqlString();
        if (sqlstring != null && !sqlstring.equals("")) {
           String dbname = sessionBean.getDbName();
           String userid = sessionBean.getUserId();
           String password = sessionBean.getPassword();

           // Make a view mode bean
           HRPortletViewBean viewBean = new HRPortletViewBean();

           SQLUtilities sqlUtility = new SQLUtilities();
           try {
              sqlUtility.execute(userid, password, dbname, sqlstring);
           } catch (SQLException e) {
              e.printStackTrace();
           }
           sqlUtility.populateData(viewBean);
           request.setAttribute(VIEW_BEAN, viewBean);

           // Set actionURI in the view mode bean
           PortletURI formActionURI = response.createURI();
           formActionURI.addAction(FORM_ACTION);
           viewBean.setFormActionURI(formActionURI.toString());
        }

        // Invoke the JSP to render
        getPortletConfig().getContext().include(VIEW_JSP +
getJspExtension(request),
            request, response);
.....
```

4. The HRPortletEdit.jsp is used to prompt for the database, user ID, password and SQL statement parameters. Double-click this file (located in /Web Content/hrportlet/jsp/html/ folder) to view its source code.

   a. Notice that the database name, user ID, password and SQL fields all have the HTML tag `value="<%=editBean.getXXX()%>"`. This allows the JSP to display the persistent data stored in the bean.

Example 11-4 HRPortletEdit.jsp (Edit mode sample code)

```
.....
<% HRPortletEditBean editBean =
(HRPortletEditBean)portletRequest.getAttribute("hrportlet.HRPortletEditBean");
%>

<HTML>
<BODY>
```

```
<h4>Complete with your database information and click Submit</h4>

<FORM method="post"
action="<portletAPI:createReturnURI><portletAPI:URIAction
name='<%=editBean.getFormActionURI()%>'/></portletAPI:createReturnURI>"">
<TABLE border="0">
   <TBODY>
      <TR>
         <TD>
         <TABLE border="0">
            <TBODY>
               <TR>
                  <TD>Database :</TD>
                  <TD><INPUT type="text" value="<%=editBean.getDbname()%>"
                     name='<portletAPI:encodeNamespace value="dbname"/>'
size="20"></TD>

               </TR>
               <TR>
.....
```

5. Once the database parameters are collected, the request is processed by the actionPerformed() and doView() methods. The results are displayed by HRPortletView.jsp. Double-click this file to view its source code.
   a. This JSP tests to see whether there is a HRPortletViewBean.java bean in the request. If there isn't then displays a message to go to Edit mode and configure and SQL statement. If the bean exists then the results of the query are displayed.

Example 11-5 HRPortletView.jsp (View mode sample code)

```
.....
<%
   HRPortletViewBean viewBean =
(HRPortletViewBean)portletRequest.getAttribute(HRPortlet.VIEW_BEAN);
%>

<% if (viewBean == null) { %>
<HTML>
<BODY>
<B>This is the JDBC Sample Portlet. Go to Edit mode and configure a SQL
query</B>
<% } else {
   com.ibm.db.beans.DBSelect results = viewBean.getResultFromDatabase();
%>
.....
```

Archived

6. The classes SQLUtilities.java and SQLUtilitiesRow.java have been generated in the data perspective. In the data perspective, you can create a connection to database, import the tables, create SQL statements and generate Java beans for the statements. These classes contain the methods to execute and retrieve information from database. The execute() method is called by the doView() method of HRPortlet.java when there is a statement in the session and the information retrieved is stored in the view mode bean by calling the populateData() method of SQLUtilities.java.

Example 11-6 SQLUtilities.java

```
   public void execute(String userid,String password,String url,String
command)
       throws SQLException {
       try {
          select.setUrl(url);
          select.setCommand(command);
          select.setUsername(userid);
          select.setPassword(password);
          select.execute();
       }

       // Free resources of select object.
       finally {
          select.close();
       }
   }

   public void populateData(HRPortletViewBean viewBean) {
      viewBean.setResultFromDatabase(select);
   }
```

11.2.5 Running the HRPortlet application

1. Before you run the portlet, run the batch file to populate the test database to be used in this sample scenario. Click **c:\LabFiles\Cloudscape\CreateCloudTable.bat** to do this.

   Note: The sample scenario included in this chapter requires that you download the sample code available as additional materials. See Appendix C, “Additional material” on page 543.

Archived

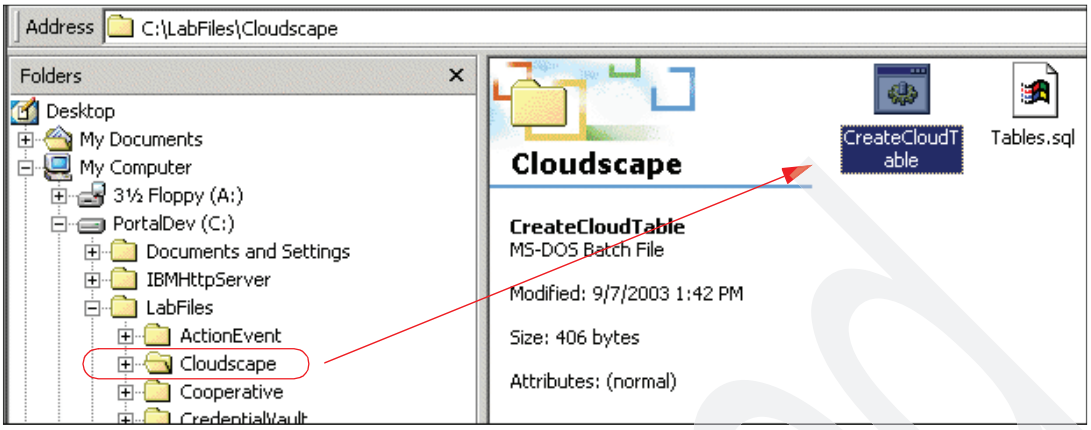


Figure 11-22 Populating the test database

2. Run the HRPortlet portlet application by right-clicking **HRPortlet** in the Navigator panel and selecting **Run on server**. Wait for Portal to start and run your portlet.

   Note: Click **OK** if you are prompted to use the Test Environment.

3. The portlet will run and you will see it in the built-in browser.

   The View mode is shown with a message indicating that you have to provide an SQL query; you will also need to switch the portlet into Edit mode (as indicated in Figure 11-23) so you can enter these values.

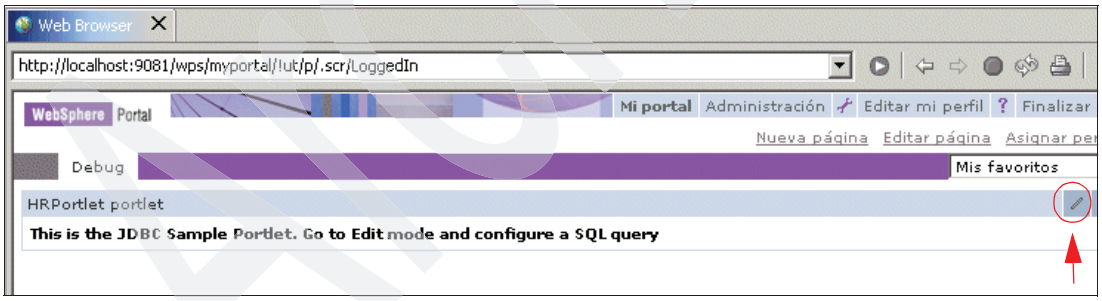


Figure 11-23 HRPortlet in View mode before query

4. When in Edit mode, the JSP for this mode renders the form requesting the database parameters.

Archived

5. For the first example, enter the following information and select **Submit**:
   - Database: `jdbc:db2j:C:\LabFiles\Cloudscape\WSSAMPLE`

     Note: The sample scenario included in this chapter requires that you download the sample code available as additional materials. See Appendix C, “Additional material” on page 543.
   - User: `db2admin`
   - Password: `db2admin`
   - SQL statement: `select * from jobs`

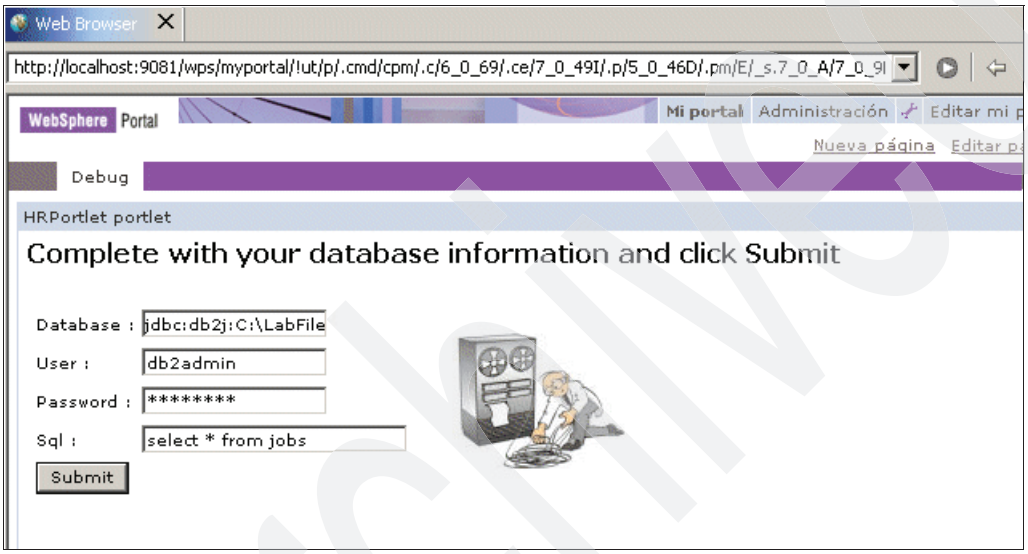
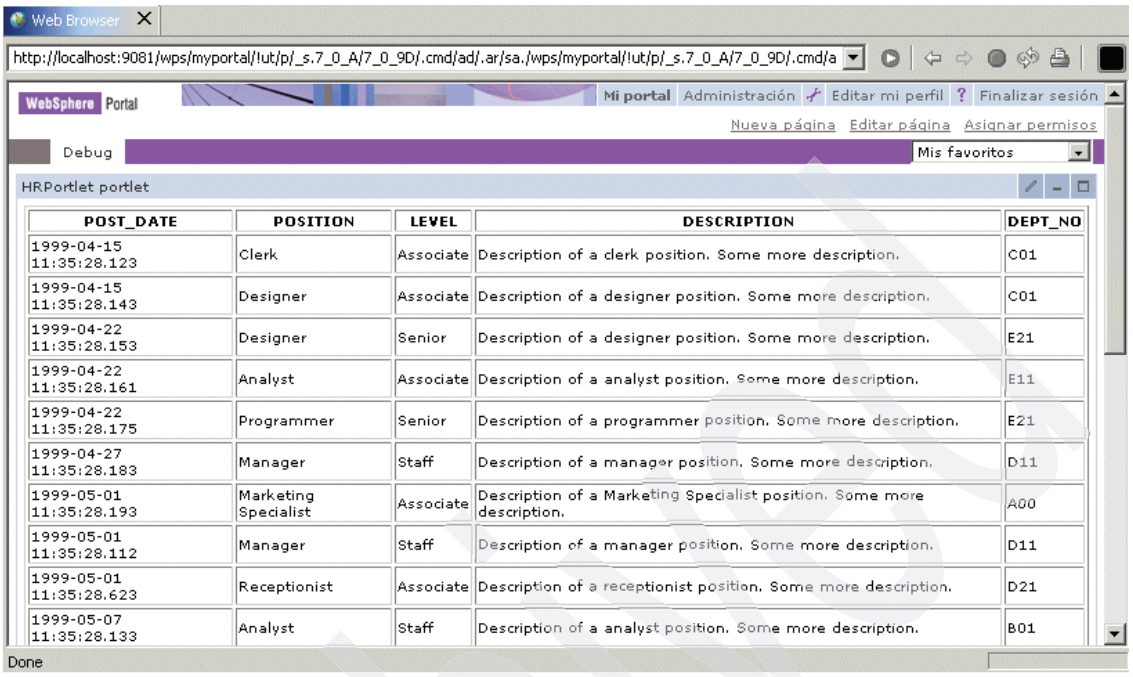


Figure 11-24 HRPortlet portlet in Edit mode

6. Clicking **Submit** generates a createReturnURI and the portlet will return to View mode showing the results of your query against the WSSAMPLE database.

Archived



POST_DATE	POSITION	LEVEL	DESCRIPTION	DEPT_NO
1999-04-15 11:35:28.123	Clerk	Associate	Description of a clerk position. Some more description.	C01
1999-04-15 11:35:28.143	Designer	Associate	Description of a designer position. Some more description.	C01
1999-04-22 11:35:28.153	Designer	Senior	Description of a designer position. Some more description.	E21
1999-04-22 11:35:28.161	Analyst	Associate	Description of a analyst position. Some more description.	E11
1999-04-22 11:35:28.175	Programmer	Senior	Description of a programmer position. Some more description.	E21
1999-04-27 11:35:28.183	Manager	Staff	Description of a manager position. Some more description.	D11
1999-05-01 11:35:28.193	Marketing Specialist	Associate	Description of a Marketing Specialist position. Some more description.	A00
1999-05-01 11:35:28.112	Manager	Staff	Description of a manager position. Some more description.	D11
1999-05-01 11:35:28.623	Receptionist	Associate	Description of a receptionist position. Some more description.	D21
1999-05-07 11:35:28.133	Analyst	Staff	Description of a analyst position. Some more description.	B01

Figure 11-25 HRPortlet in View mode with the query results in View mode

7. Enter Edit mode again. Notice that the database name, user ID, password and SQL statement which were stored in Session are persistent.

Archived

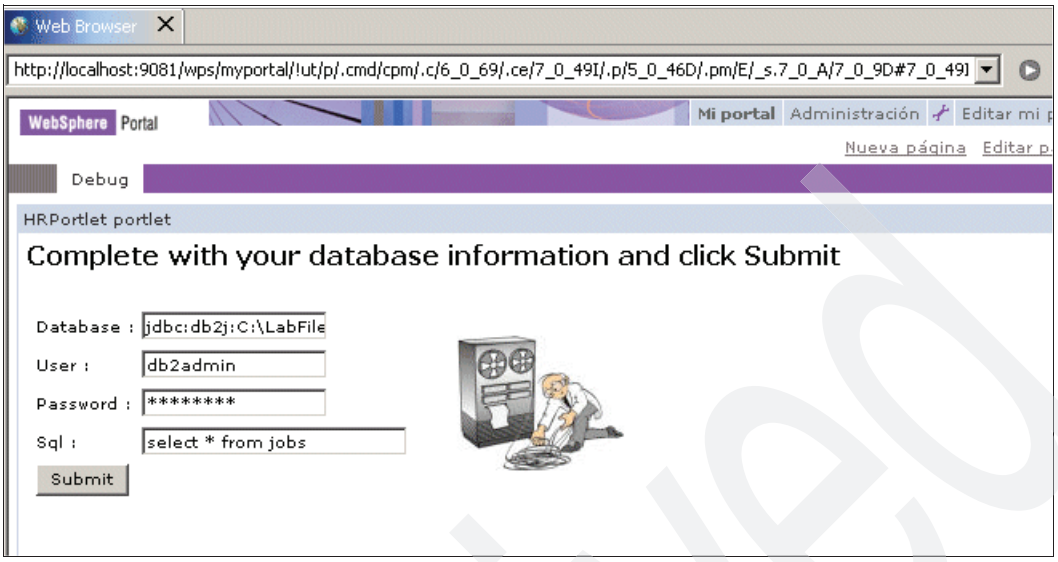


Figure 11-26 Persistence of data

8. Enter a new query, for example `select * from survey`, and click **Submit**. You will be presented with the results of your new query.

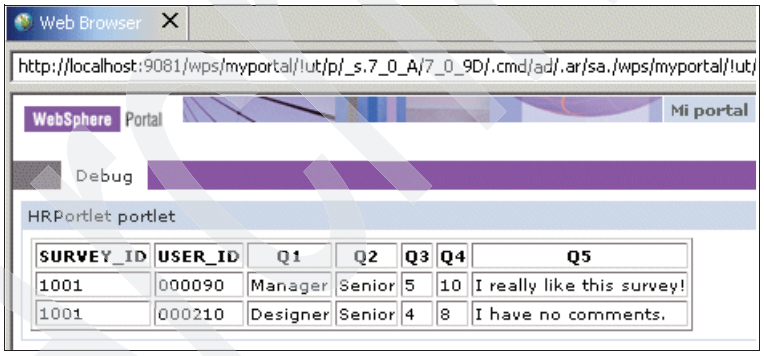


Figure 11-27 Results of the second query

Archived

12

Cooperative portlets

This chapter describes the architecture and development of cooperative portlets, formerly known as Click-to-Action portlets. Cooperative portlets placed on a portal page can be developed independently but they interact with one another and share the same information. This enables an advanced user experience scenario where portlets automatically react to events and actions originated from other portlets.

After reading this chapter, you will be able to:

- Understand the architecture and value of cooperative portlets
- Develop source cooperative portlets
- Develop target cooperative portlets

In addition, at the end of this chapter, you will find information and solutions to common problems in cooperative portlet development.

Note: The sample scenarios included in this chapter require that you have completed Chapter 11, “Accessing back-end JDBC databases” on page 343. You can also download the sample code available as additional materials. See Appendix C, “Additional material” on page 543.

© Copyright IBM Corp. 2004. All rights reserved.

12.1 Overview

Cooperative portlets exchange information, react in a coordinated manner and provide menus to share information by selecting an action. Selecting such a menu item results in the execution of the actionPerformed() method on a target cooperative portlet. Figure 12-1 illustrates an example of such a cooperative portlet menu.

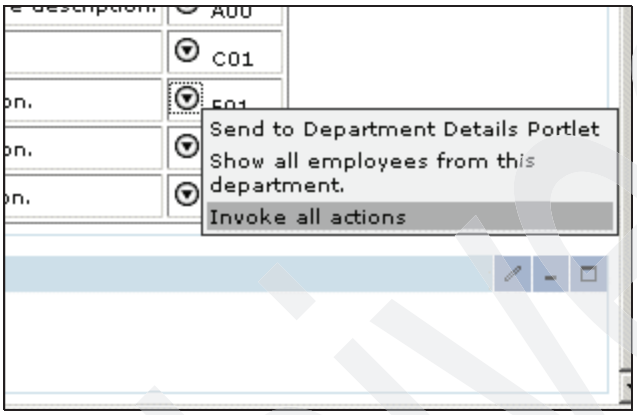


Figure 12-1 An example of a cooperative portlet menu

In Figure 12-1, one piece of information, for example a department number, can be sent to a target portlet displaying all employees from the selected department number or to a portlet displaying detailed information of the selected department. In addition, the cooperative portlet technology also enables the broadcast of data to multiple portlets by sending multiple property values with only one click.

The transfer of properties can be saved as wires using the **Ctrl** key. So the next time the user clicks the icon, the saved menu selection is used without prompting the user.

Portlet messaging versus cooperative portlets

In general, both portlet messaging and cooperative portlets can be used to share data between two or more portlets. The most important difference is that cooperative portlets are more loosely coupled than portlets using messaging. Cooperative portlets do not have to know the name of the target portlet even if they do not broadcast data. The matching of source and target portlets is done at runtime based on registered properties and actions. Cooperative portlets can also include a menu with a list of executable portlet actions. For this menu, no additional programming is needed because it is part of the C2A tag library.

Archived

12.1.1 The WebSphere Portal property broker

During runtime, the WebSphere Portal property broker is responsible for enabling cooperative portlets. This is done by matching the data type of output properties from a source portlet with the data type of input properties from one or more target portlets. Figure 12-2 shows the relationships between the two portlets and properties.

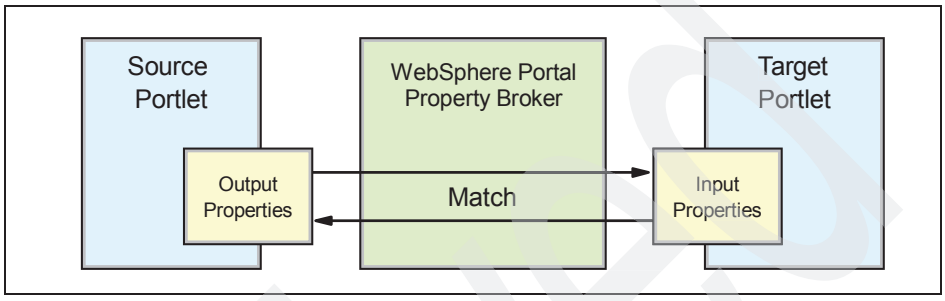


Figure 12-2 The property broker matches input and output properties

Target portlets optionally provide actions to process the properties that it receives. There is no difference between an action initiated by the portlet itself as mentioned in Chapter 5, “Action event handling” on page 181 and an action initiated by a source cooperative portlet.

Cooperative portlets can be source portlets, target portlets, or both.

- Source portlets identify to the property broker properties which they are able to share with other portlets.
- Target portlets identify to the property broker actions which are able to process properties contributed by other portlets.

12.1.2 Programming model

To enable your portlet for cooperation as well as a broker component, you have to wrap your portlet class with a generic wrapper portlet. This wrapper intercepts calls and interfaces with the broker. The wrapper is packaged in each cooperative portlet’s WAR file.

Cooperative portlets can use a declarative or programmatic approach, or a combination, to register and publish properties to the property broker. The programmatic approach to publish properties is discussed in Chapter 13, “Advanced cooperative portlets” on page 413. The declarative approach is simpler. Few changes need to be made to existing portlets to enable them to interact with other cooperative portlets on the page. Existing portlets that already

Archived

use action processing simply declare their actions to the property broker using WSDL. Figure 12-3 presents how to develop source cooperative portlets.

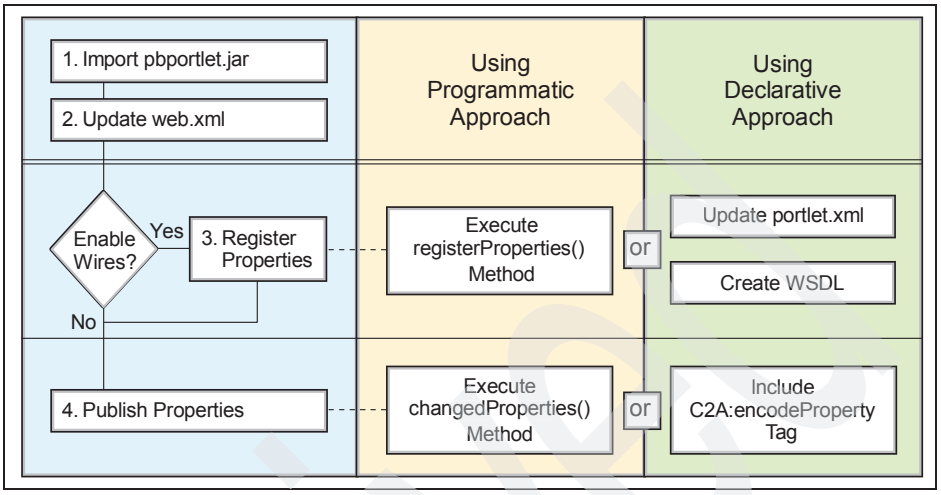


Figure 12-3 Steps to program a source cooperative portlet

In general terms, there are two steps at runtime to establish cooperative portlet communication:

1. All properties must be registered with the cooperative broker. This can be done by using the declarative approach which includes the creation of a Web Service Description Language (WSDL) file and the configuration of the portlet deployment descriptor. To register properties programmatically, you can use the property broker API. Please note that registration of properties can only be done during the event phase of the request-response cycle.
2. Notify the property broker about property changes. The easiest way to achieve this is to include the *encodeProperty* tag in your JavaServer Page. As an alternative, in the programmatic approach you will use the *changedProperties()* method to publish properties.

   Also, in the programmatic approach you have to configure a wire before you publish properties. Figure 12-4 on page 375 shows how to develop target portlets.

Archive

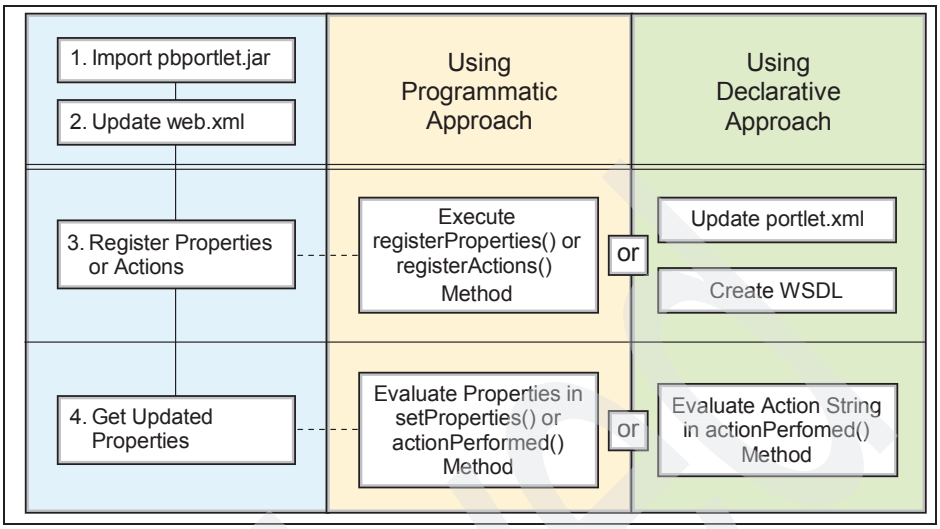


Figure 12-4 Steps to create a target cooperative portlet

To get information about changed properties, target portlets register properties and, optionally, actions with the property broker. When using the declarative approach both properties and actions are always registered. When using the programmatic approach, you can register properties without any actions.

In addition, during runtime process such programmatic target portlets are notified about property changes by using the *setProperties* method from the *PropertyListener* interface instead of the actionPerformed method.

12.1.3 Registering and publishing properties

For a portlet to be a source of data, programmers can use a custom JSP tag library to flag sharable data on their output pages. The tags require a data type to be specified as well as a specific value corresponding to an instance of this type. If you want to use wires source portlets, you must register properties by using a declarative or programmatic approach.

Target portlets associate their actions with an input property which has been declared as an XML type. The actions are declared using WSDL, with a custom binding extension which specifies the mapping from the abstract action declaration to the actual action implementation. Associated with each action is a single input parameter described by an XML type and zero or more output parameters, each described by an XML type. Each input or output parameter encapsulates exactly one property. The input property's type is used for matching the action to sources, and its value is filled in when the end user triggers the

action using Click-to-Action. The output parameters, if specified, are used to automatically trigger other compatible actions (ones which can consume the same type) on other wired portlets every time the action executes (this may be used to trigger chains of related actions).

Note: The location of the WSDL file is configured as a portlet parameter.

12.2 Sample scenario

The sample application shown in this section is based on the HRPortlet from Chapter 11, “Accessing back-end JDBC databases” on page 343. Two different versions of the HRPortlet will be used, as follows:

- The source cooperative portlet HRPortlet displays a list of jobs.
- The target cooperative portlet Employee Details Portlet displays a list of employees working in the same department.

Using the cooperative portlet technology, users can select a department number from the source cooperative portlet. After that, WebSphere Portal updates the target portlet displaying all employees from the selected department.

12.2.1 Development workstation

This sample scenario provides step-by-step exercises to enable the JDBC portlet project (HRPortlet) to work as a Click-to-Action target portlet. You will also enable this portlet to act as a source Click-to-Action portlet. You will create, deploy and run the portlet application. This exercise will allow you to understand the techniques used to develop portlets with Click-to-Action features using the C2A declarative approach.

The development workstation and its components can be seen in Figure 12-5 on page 377.

Archived

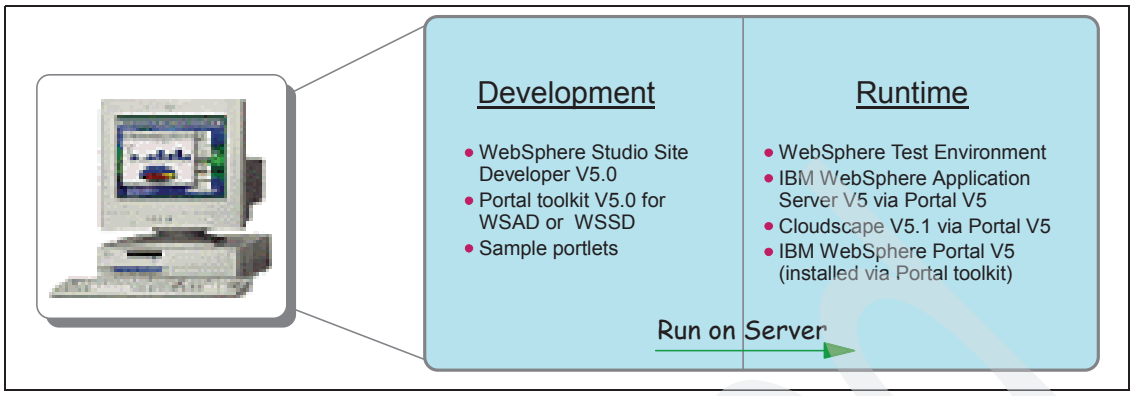


Figure 12-5 Development workstation

12.2.2 Description

Cooperative portlets subscribe to a model for declaring, publishing, and sharing information with each other using the WebSphere Portal property broker. Portlets subscribe to the broker by publishing typed data items, or properties, that they can share, either as a provider or as a recipient.

- The portlet that provides a property is called the source portlet.
- The properties that the source portlet publishes are called output properties.
- The portlet that receives a property is called the target portlet.
- The properties that are received by the target are called input properties.

The target portlets optionally provide actions to process the properties that they receive. Action processing in target portlets does not need to distinguish between an action initiated within its own portlet area and an action initiated by the transfer of a portlet property value. Each action is associated with a single input parameter and zero or more output parameters, which provide information to the action about the objects in which the property value should be bound, such as the request or the session. Each parameter is associated with exactly one property. Parameters associated with input properties are called input parameters, while those associated with output properties are called output parameters. Instead of actions, the target portlet can receive property changes directly through the *PropertyListener* interface.

Archive

At runtime, the property broker matches the data type of output properties from a source portlet with the data type of input properties from one or more target portlets. If a match is determined, the portlets are capable of sharing the property. The actual transfer of the property can be initiated by one of the following methods:

- A user launches a Click-to-Action event from an icon on the source portlet. The icon presents a pop-up menu containing the list of targets for the action. After the user selects a specific target, the property broker delivers the data to the target in the form of the corresponding portlet action. Using the Click-to-Action delivery method, users can transfer data with a simple click from a source portlet to one or more target portlets, causing the target to react to the action and display a new view with the results. The user can also broadcast the property to all portlets on the page that have declared an action associated with a matching input property.
- A user holds down the **Ctrl** key while clicking an action and chooses to have the selection saved persistently as a connection between two portlets, called a wire. If a wire is present the next time the user clicks the icon, no selection menu is shown. Instead, the wired action(s) is/are automatically fired. Subsequent updates to that property are transferred without further deliberate user choice.
- The source portlet can perform a programmatic publish of properties to the broker when it determines that property values have changed. Such property values are transferred to the target(s) only if wires have been created.

Cooperative portlets can be source portlets, target portlets, or both.

- Source portlets identify to the property broker properties which they are able to share with other portlets.
- Target portlets identify to the property broker actions which are able to process properties contributed by other portlets.

Archive

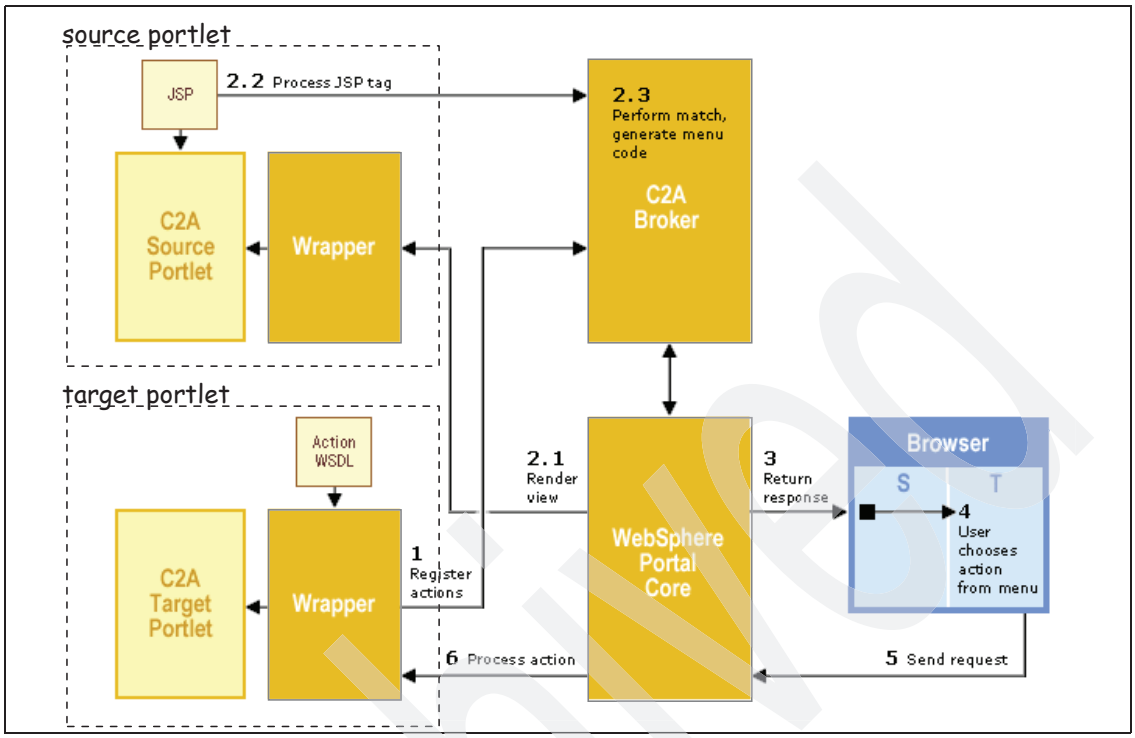


Figure 12-6 Click-to-Action architecture

The sequence flow for this sample scenario is as follows:

1. At portlet initialization time, the C2A wrapper processes any action WSDL file associated with the application portlet and registers the actions with the C2A broker.
2. During the render phase of a request cycle, JSPs associated with C2A source portlets are processed. The custom C2A tags produce calls to the C2A broker, which examines the type information to determine matching actions. The broker generates additional code to create an icon to be used to display a pop-up menu of actions, and adds code to dispatch actions on portlets upon user selection from the menu.
3. After all render phase portlet callbacks are complete, the WebSphere Portal core assembles the response page and returns it to the client (for example, a browser).
4. When the user clicks the C2A icon for a source, he or she sees a menu of compatible actions (on the page) and selects one.

Arrowed

5. The client (for example, a browser) generates a new request containing the chosen source and action information and sends it to the WebSphere Portal Server.
6. The WebSphere Portal Core delivers the action to the target portlet. The action is intercepted by the wrapper, which may interact with the broker to further process the request before delivering the action to the target.

All portlet actions are intercepted by the wrapper; however, actions which are invoked through direct interaction with the portlet (as opposed to interaction through the C2A menus) are passed through transparently to the portlet. In more advanced scenarios, such as the broadcast and scatter mentioned earlier, there will be more interactions between the wrappers and the broker to determine the appropriate target set and deliver the right data to the targets.

12.2.3 Source cooperative portlet

In this section, you will be required to execute the following tasks to enable a portlet to act as a Click-to-Action source portlet using the declarative approach:

1. You will import the original portlet if it is not in your workspace. In this scenario, the JDBC portlet from the previous sample scenario will be used (HRPortlet). See Chapter 11, “Accessing back-end JDBC databases” on page 343.
2. You will import the property broker jar file (pbportlet.jar).
3. You will update web.xml to refer to the property broker classes.
   a. The servlet class entry should specify the `com.ibm.wps.pb.wrapper.PortletWrapper` class in the property broker:

      ```
      <servlet-class>com.ibm.wps.pb.wrapper.PortletWrapper</servlet-class>
      ```

   b. The original portlet application class should also be specified in the c2a-application-portlet-class initialization parameter. For example:

      ```
      <init-param>
      <param-name>c2a-application-portlet-class</param-name>
      <param-value>hrportlet.HRPortlet</param-value>
      </init-param>
      ```

4. You will update the JSP for View mode to include Click-to-Action menus.

Figure 12-7 on page 381 illustrates the source cooperative portlet for this sample scenario.

Archived

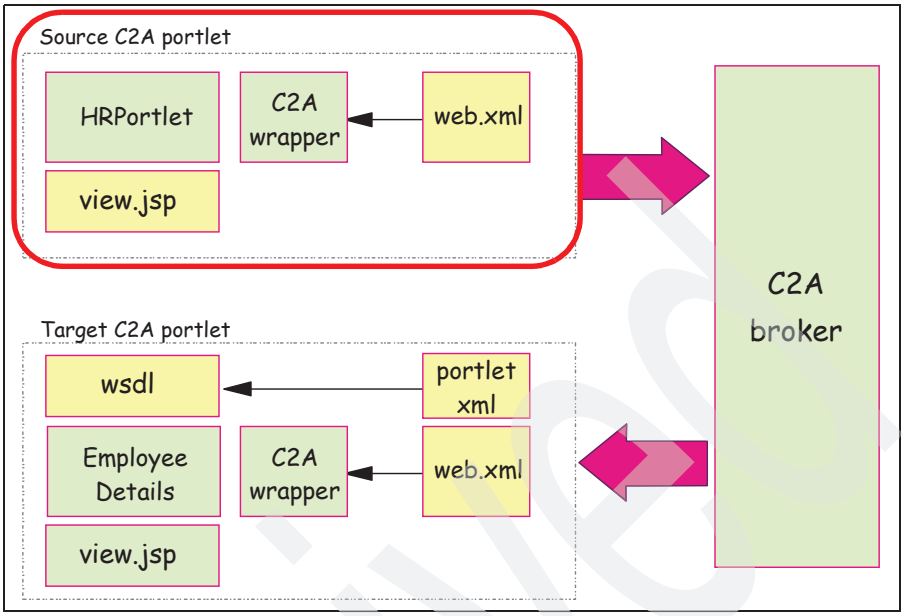


Figure 12-7 Cooperative portlets - sample scenario

Importing the original portlet

The HRPortlet portlet from the JDBC chapter (see Chapter 11, “Accessing back-end JDBC databases” on page 343) will be used as a base for this sample scenario. This portlet will be enabled to act as a source cooperative portlet in this scenario. You will need to import this portlet if it is not in your workspace.

Follow these steps to import this portlet:

1. Import the WAR file by selecting **File -> Import**.
2. Select **WAR file** and click **Next**.
3. In the *Import Resources from a WAR File* window, enter the following information:
   a. WAR file: browse to `C:\LabFiles\C2A\HRPortlet.war`.

      Note: You can also download the sample code available as additional materials. See Appendix C, “Additional material” on page 543.

   b. Web project: select **Existing**. In the box that pops up, select **HRPortlet** and click **OK**.
   c. Context root: this will change to `/HRPortlet`.
   d. In Options, select the **Overwrite existent resources without warning** check box.

Archived

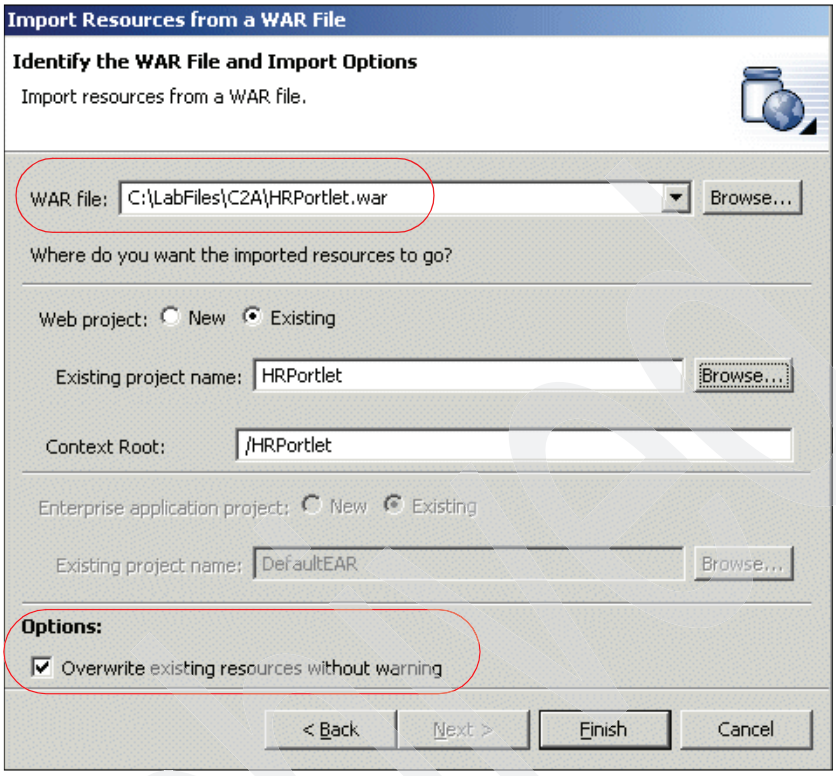


Figure 12-8 Importing the original portlet HRPortlet

Importing the property broker file (pbportlet.jar)

You will need to import the property broker jar file (pbportlet.jar). Follow these steps:

1. Switch to the portlet perspective.
2. Import the property broker jar file (pbportlet.jar):
   a. In the J2EE Navigator view, select the **HRPortlet\Web Content\WEB-INF\lib** folder.
   b. Select **File -> Import**.
   c. Select **File system** and click **Next**.

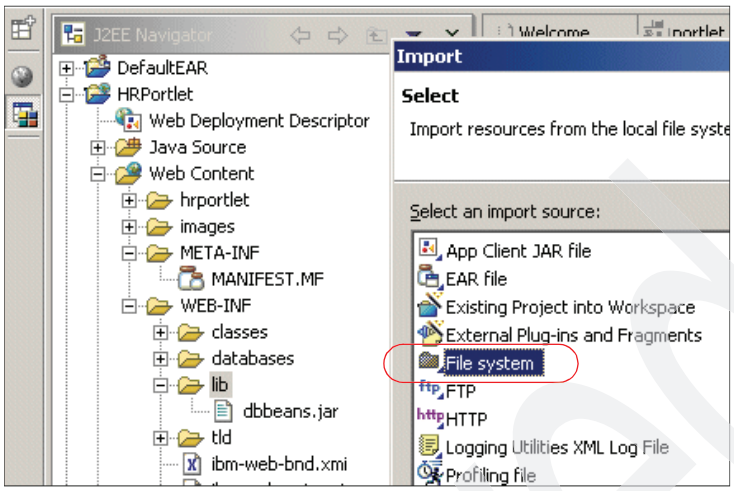


Figure 12-9 Select File System

d. In the *Import File system* window, enter the following information:

- For the directory, browse to:

```
C:\Program Files\ibm\WebSphere Studio\runtimes\portal_v50\pb\lib
```

Note: You can also download the sample code available as additional materials. See Appendix C, “Additional material” on page 543.

- Select **pbportlet.jar**.
- For the destination, select the folder **HRPortlet/Web Content/WEB-INF/lib**.

Archived

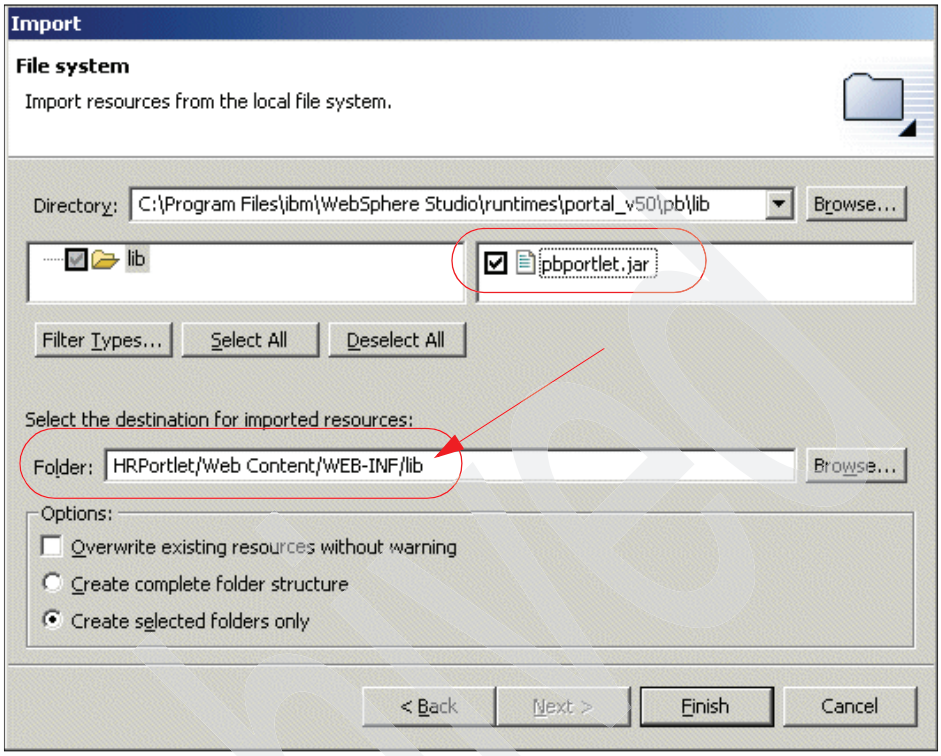


Figure 12-10 Select pbportlet.jar to import the jar file

e. Click **Finish**.

Important: Make sure pbportlet.jar is in the lib folder.

Archive

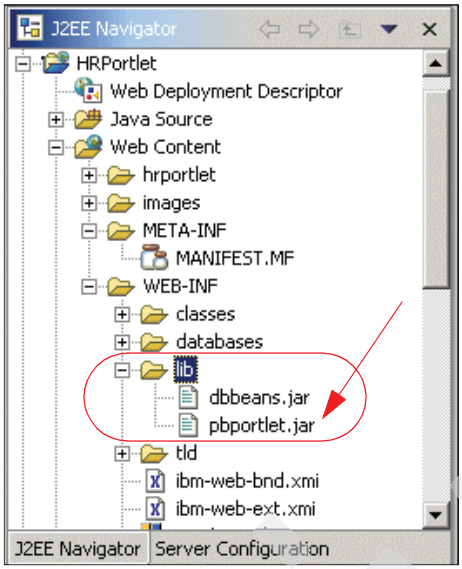


Figure 12-11 pbportlet.jar

3. Update the Web Deployment Descriptor as follows:
   a. In the J2EE Navigator view, expand HRPortlet and double-click **Web Deployment Descriptor**.
   b. Switch to the Servlets tab and select the **hrportlet.HRPortlet** servlet (Figure 12-9 on page 383).
   c. In the Details area, click the **Browse...** button to change the servlet class.
   d. In the Servlet selection dialog, select the **PortletWrapper** class from the com.ibm.wps.pb.wrapper package and click **OK**.

Archived

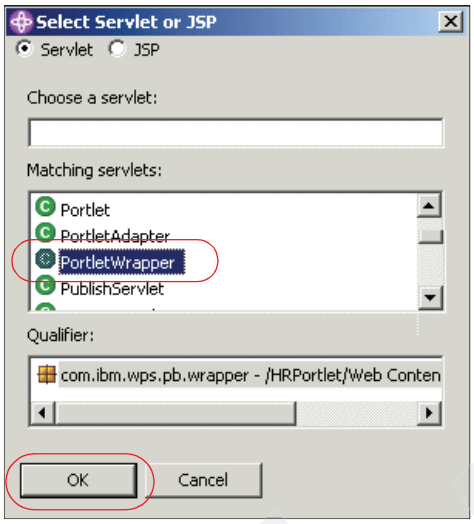


Figure 12-12 Adding PortletWrapper

e. In the Initialization area, click the **Add...** button to add a new parameter.

f. Enter a parameter name of `c2a-application-portlet-class` and a parameter value of `hrportlet.HRPortlet`. The final editor should look as shown in Figure 12-13.

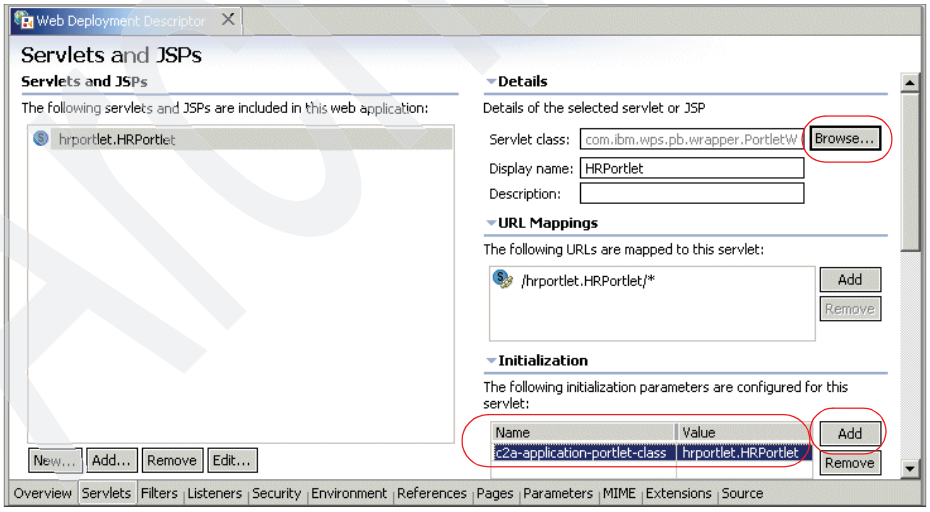


Figure 12-13 Web Deployment Descriptor for the source cooperative portlet

Archived

g. Save your files (or press **Ctrl-S** to save the file) and close the deployment descriptor editor.

4. Import the Click-to-Action tag library (c2a.tld).

   Note: This step in included here as a reference for previous releases only, the c2a.tld library is now part of WebSphere Portal V5 and does not need to be packaged in the portlet WAR file.

   a. Import the file by selecting **File -> Import**.

   b. Select **File system** and click **Next**.

   c. In the Import File system window, enter the following information:

      - For the directory, browse to:

        ```
        C:\Program Files\ibm\WebSphere
        Studio\runtimes\portal_v50\shared\app\WEB-INF\tld
        ```

        Note: You can also download the sample code available as additional materials. See Appendix C, “Additional material” on page 543.

      - Check the **c2a.tld** box.

      - For the destination, enter the folder:

        ```
        HRPortlet/WebContent/WEB-INF/tld
        ```

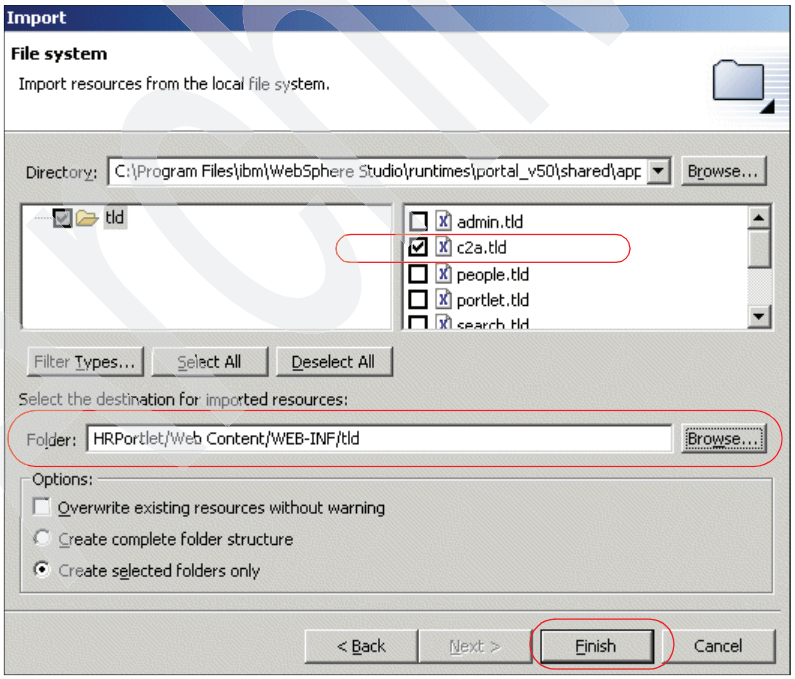


Figure 12-14 Importing c2a tag library (c2a.tld)

Archived

d. Click **Finish**.

5. The final step is to update the HRPortletView.jsp with c2a tags.

   a. In the J2EE Navigator, expand HRPortlet/Web Content/hrportlet/jsp/html and double-click **HRPortletView.jsp**.

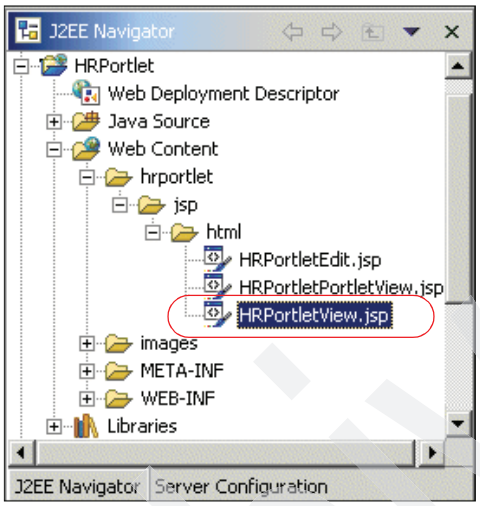


Figure 12-15 Selecting HRPortalView.jsp

   b. In the JSP editor, switch to the Source tab.

      Note: Source portlets can publish their output properties by inserting tags from a custom JSP library in their JSPs. A JSP tag library is provided to allow source properties to be identified in JSPs.

   c. In the third line of this JSP, include the following line:

```
<%@ taglib uri="/WEB-INF/tld/c2a.tld" prefix="C2A" %>
```

      You can also copy and paste the JSP from c:\LabFiles\C2A\snippets\.

      Note: You can also download the sample code available as additional materials. See Appendix C, “Additional material” on page 543.

      For example, see Figure 12-16.

HRPortletView.jsp

```
<%@ page contentType="text/html" import="java.util.*, hrportlet.*"%>
<%@ taglib uri="/WEB-INF/tld/portlet.tld" prefix="portletAPI" %>
<%@ taglib uri="/WEB-INF/tld/c2a.tld" prefix="C2A" %>
<portletAPI:init />
```

Figure 12-16 Adding tag library c2a.tld

Archived

d. Two JSP tags can be used to declare output properties in the source portlet:

- **<c2a:encodeProperty/>**

  Uses source data and type information to insert markup that displays the icon, generating a pop-up menu.

- **<c2a:encodeProperties/>**

  Used to enclose normal HTML markup and one or more encodeProperty tags with the markup. This tag is provided to support the scatter scenario, where a user can optionally send more than one unit of data to target portlets.

You will now declare the output properties. Scroll down to the following lines:

```
<TD>
        <P><%=results.getCacheValueAt(row, col)%></P>
</TD>
```

Change the line so it looks as follows:

```
<P>
   <C2A:encodeProperty
name="<%=results.getColumnName(col).toString()+\"Param\"%>"
   namespace="http://www.ibm.com/wps/c2a/examples/hrdetails"
   type="<%=results.getColumnName(col)%>"
   value="<%=results.getCacheValueAt(row, col).toString()%>" />
   <%=results.getCacheValueAt(row, col)%>
</P>
```

You can also copy and paste the JSP from c:\LabFiles\C2A\snippets\.

Note: You can also download the sample code available as additional materials. See Appendix C, “Additional material” on page 543.

For example, see Figure 12-17 on page 390.

Archived

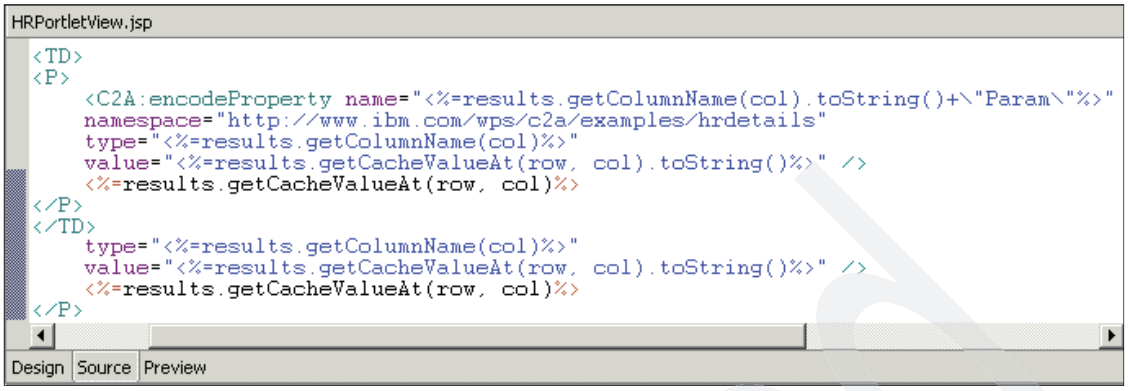


Figure 12-17 Adding C2A:encodeProperty to JSP (view mode)

Note: As you can see, the table column name is used as an output property type. Therefore, a target portlet using the specified namespace should provide an inbound property with this name.

e. Save all your files (you can use **Ctlr-S**).

12.2.4 Target cooperative portlet

In this section, you will import a second copy of the HRPortlet and update it to support Click-to Action as a target cooperative portlet using the declarative approach. This target portlet will execute a fixed SQL statement with a variable *where* clause.

The code for the target portlet class must meet the following requirements:

- The action must be implemented either as a portlet action or a Struts action. For portlet actions, you should use the simple action Strings rather than the deprecated PortletAction class.
- Portlet actions must accept a single parameter. The parameter may appear as a request parameter, a request attribute, a session attribute, or an action attribute (deprecated), as specified in the action declaration or registration.

The HRPortlet is already prepared for this situation, so only a few changes are needed in the portlet class code.

Figure 12-18 on page 391 illustrates the target cooperative portlet for this sample scenario.

Archive

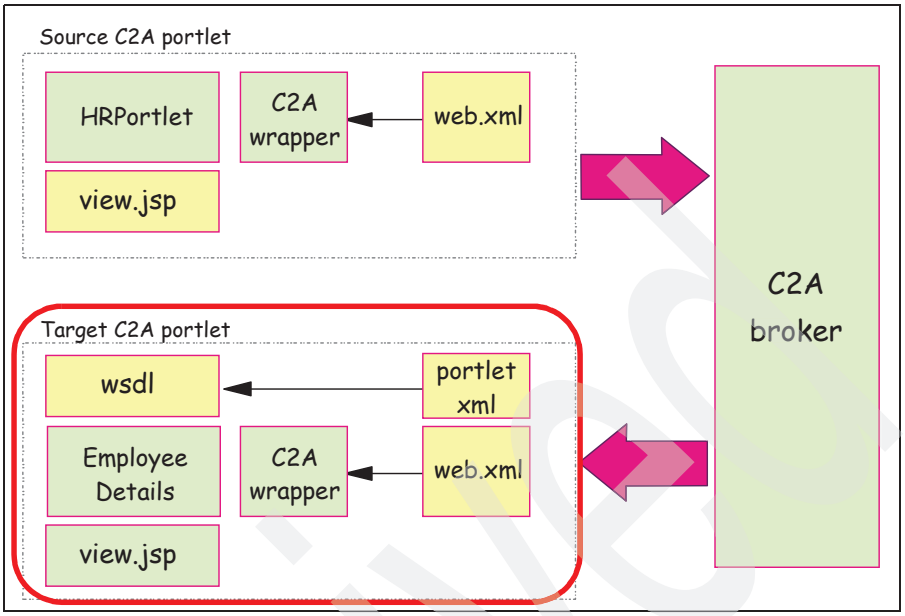


Figure 12-18 Cooperative portlets - sample scenario

To import a second version of the HRPortlet, proceed as follows:

1. In the Portlet perspective, choose **File -> New -> Portlet Application Project** from the main menu.
2. In the Create a Portlet Project window, enter a project name of `EmployeeDetailsPortlet` and check **Create empty portlet**. Click **Next**.

Archive

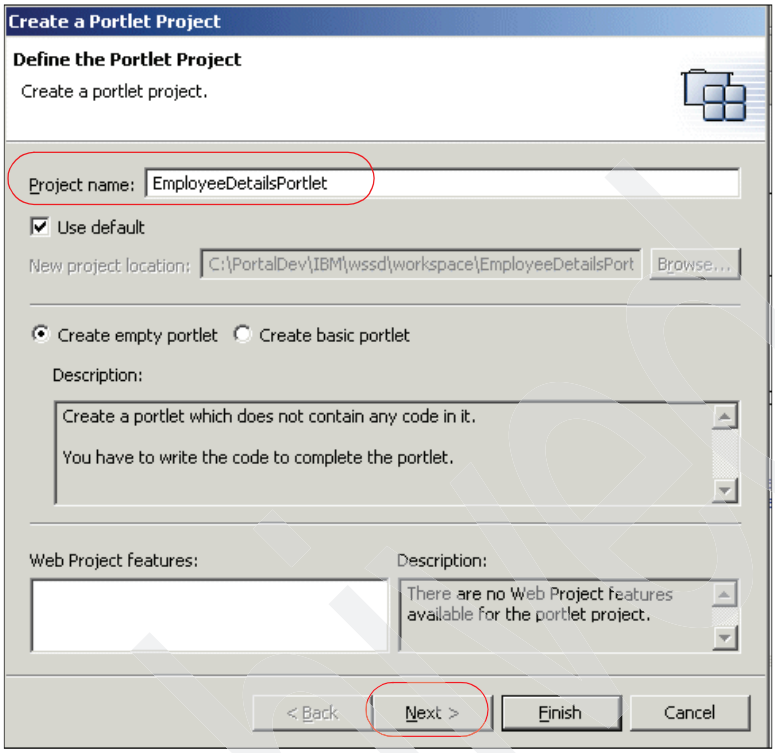


Figure 12-19 Create portlet project EmployeeDetailsPortlet

3. In the J2EE Settings Page, check the **Existing** radio button for the Enterprise application project and enter `DefaultEAR`. Click **Finish**.

Archive

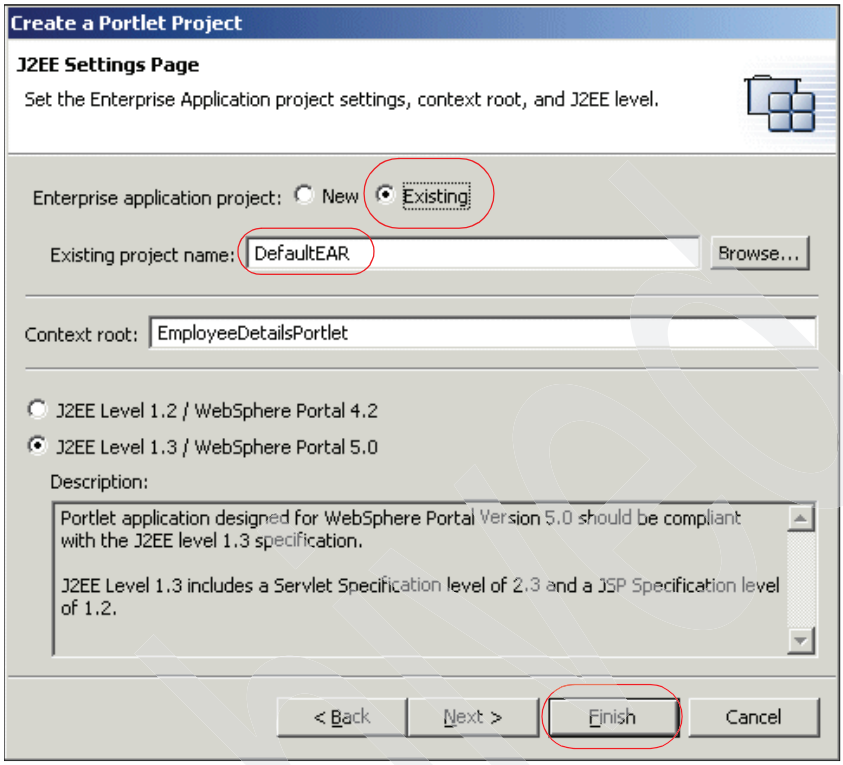


Figure 12-20 J2EE Settings Page

4. Click **OK** if you receive the Repair Service Configuration message indicating that the project will be added to DefaultEAR.

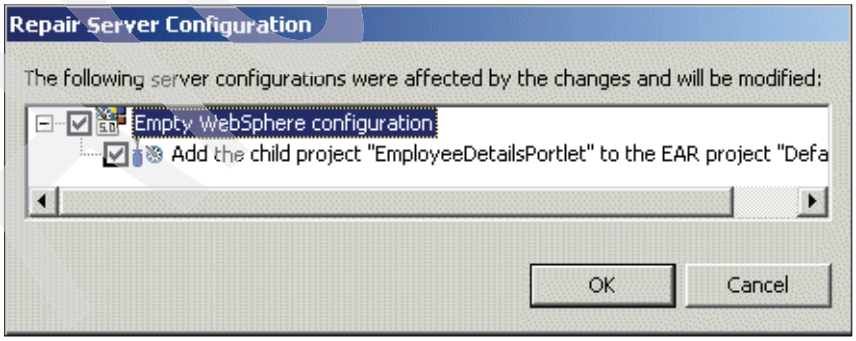


Figure 12-21 Repair Server Configuration message

5. From the main menu, select **File -> Import** to import the original HRPortlet.

6. Choose **WAR file**, click **Next** and configure as follows:
   a. Browse to the location of the HRPortlet.war file in c:\LabFiles\C2A\HRPortlet.war.

      Note: You can also download the sample code available as additional materials. See Appendix C, “Additional material” on page 543.

   b. As the Web Project, select the existing Web project **EmployeeDetailsPortlet**.
   c. Check **Overwrite existing resources without warning** and click **Finish**.

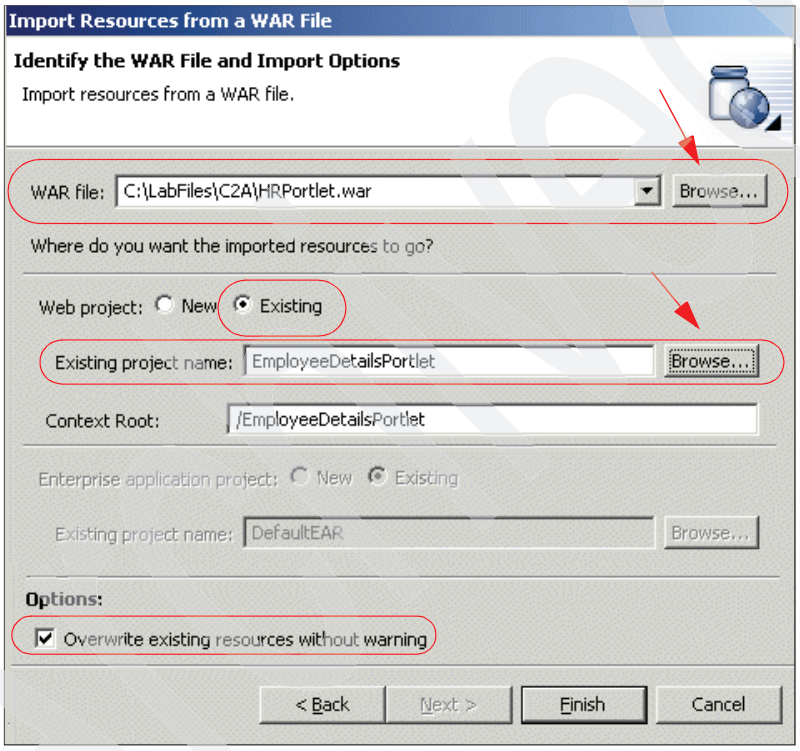


Figure 12-22 Import a second version of HRPortlet (target c2a portlet)

7. Since the HRPortlet and EmployeeDetailsPortlet portlet applications use the same UID, a warning message will appear in the task pane.

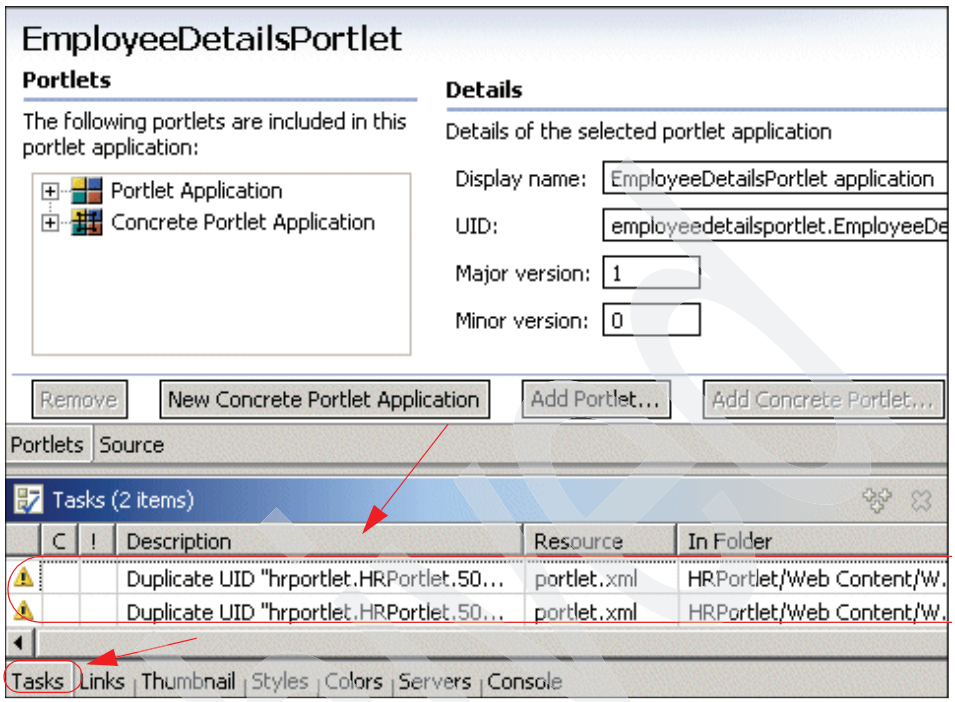


Figure 12-23 Duplicate UID messages

8. To fix this problem, expand EmployeeDetailsPortlet/Web Content/WEB-INF in the J2EE Navigator view. Double-click **portlet.xml**.

Archived

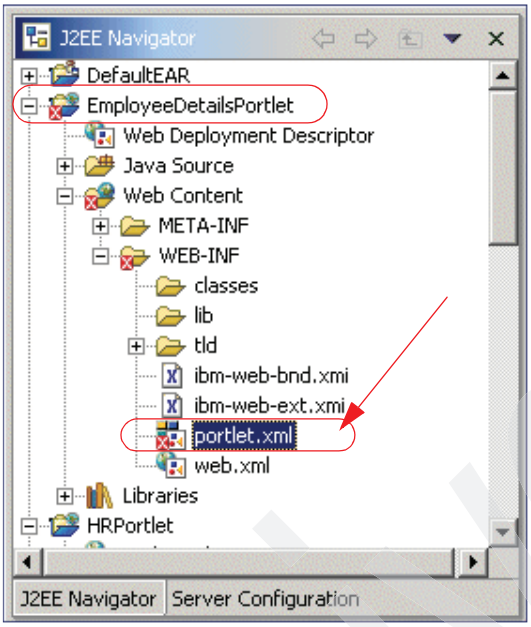


Figure 12-24 Selecting portlet.xml

9. In the portlet deployment descriptor editor, select **Portlet Application** and change the last digit of the UID for this portlet application. For example, in this sample scenario, the last digit was 6 and it was changed to 7.

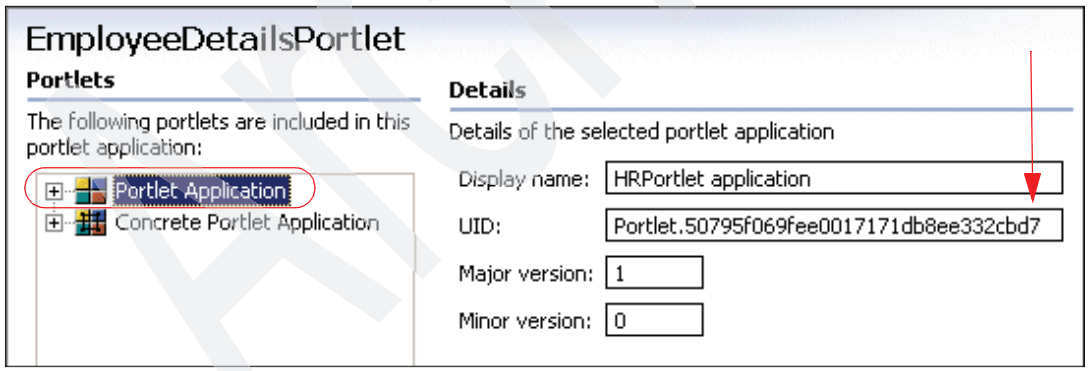


Figure 12-25 Changing a digit in portlet application UID

10. In a similar way, select **Concrete Portlet Application** and change the last digit before the last dot of the UID to make it the same as in the previous step.

Archived

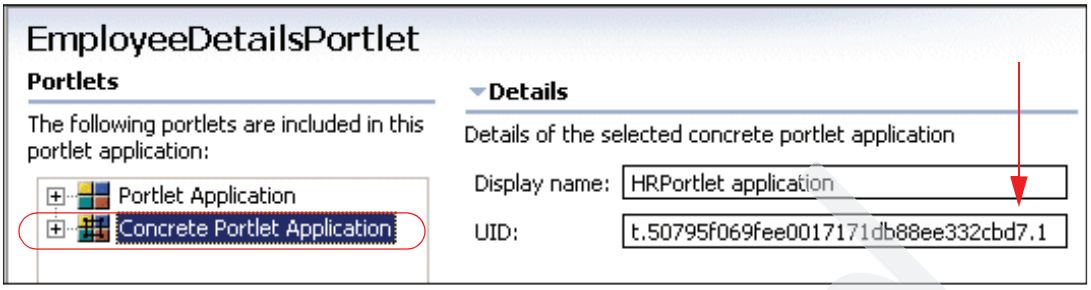


Figure 12-26 Changing digit in concrete portlet application UID

11. Save your changes. The warning messages in the Tasks view should disappear.

The following updates are needed in order to enable the EmployeeDetailsPortlet to work as a target cooperative portlet.

- Import the property broker jar file (pbportlet.jar).
- Update web.xml to refer to the property broker classes. The servlet class entry should specify the com.ibm.wps.pb.wrapper.PortletWrapper class in the property broker. The original portlet application class should also be specified using the c2a-application-portlet-class initialization parameter.
- Update portlet.xml to add a configuration parameter to each concrete portlet that exposes actions to the property broker through the WSDL file. The configuration parameter, c2a-action-descriptor, must specify a URL that points to the WSDL file that declares actions.
- Create a WSDL file that will declare all Portlets actions which can accept data transferred using the property broker.

Execute the following steps:

1. Import the property broker (pbportlet.jar) file into the EmployeeDetailsPortlet project as you did for the c2a source portlet. For example:
   a. Select **File -> Import -> File system**.
   b. For the directory, browse to:

      C:\Program Files\ibm\WebSphere Studio\runtimes\portal_v50\pb\lib

      Note: You can also download the sample code available as additional materials. See Appendix C, “Additional material” on page 543.

   c. Select **pbportlet.jar**.
   d. For the destination, select the folder **EmployeeDetailsPortlet/Web Content/WEB-INF/lib**.

Archive

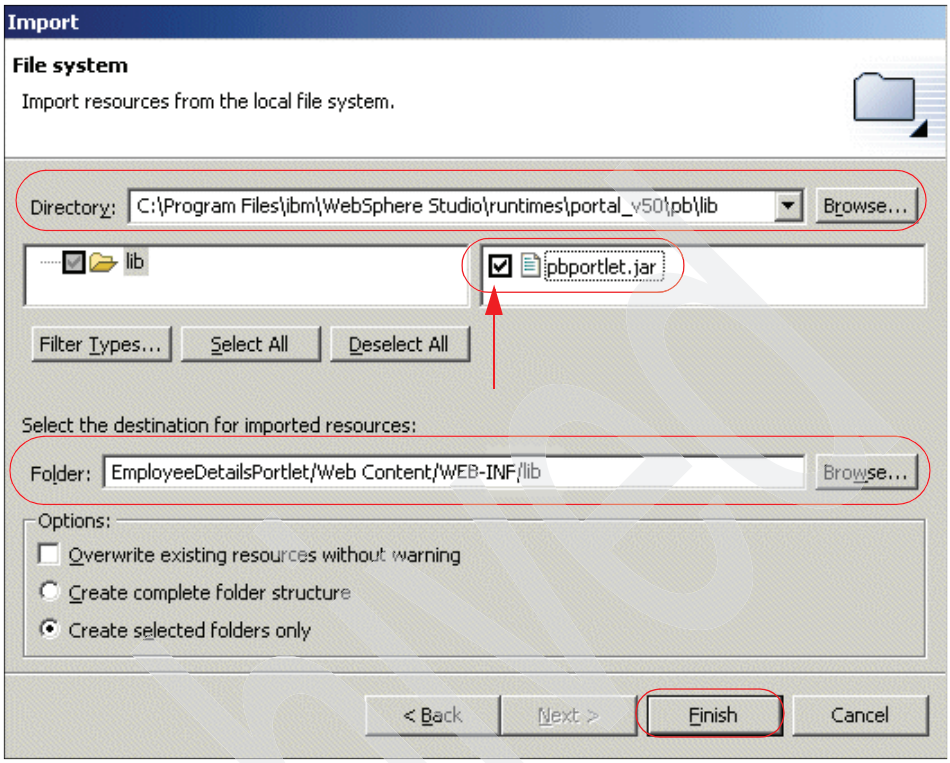


Figure 12-27 Import the property broker jar file (pbportlet.jar)

e. Click **Finish**. Make sure the pbportlet.jar file is in the \WEB-INF\lib folder.

2. Update the Web deployment descriptor by changing the servlet class to `PortletWrapper` and including the `c2a-application-portlet-class` parameter.

   a. In the J2EE Navigator view, expand HRPortlet and double-click **Web Deployment Descriptor**.

   b. Switch to the Servlets tab and select the **hrportlet.HRPortlet** servlet (Figure 12-29 on page 399).

   c. In the Details area, click the **Browse...** button to change the servlet class (Figure 12-29 on page 399).

   d. In the Servlet selection dialog, select the **PortletWrapper** class from the com.ibm.wps.pb.wrapper package and click **OK**.

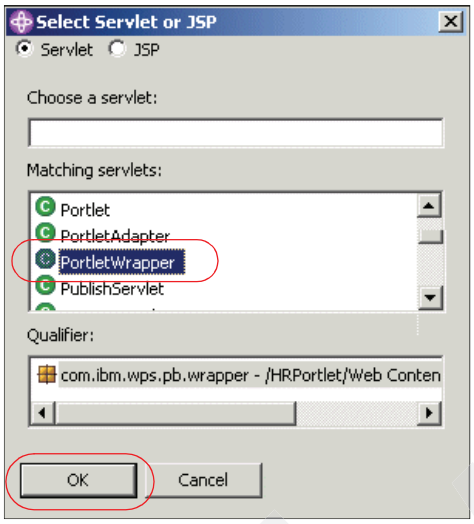


Figure 12-28 Adding PortletWrapper

e. In the Initialization area, click the **Add...** button to add a new parameter.

f. Enter a parameter name of `c2a-application-portlet-class` and a parameter value of `hrportlet.HRPortlet`. The final editor should look as shown in Figure 12-29.

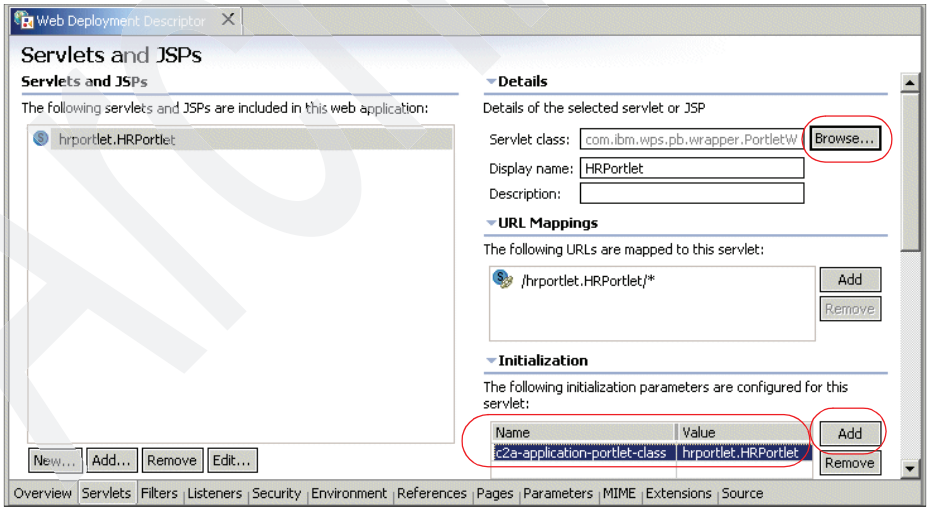


Figure 12-29 Web Deployment Descriptor for the target cooperative portlet

Archived

g. Save your files (or press **Ctrl-S** to save the file) and close the deployment descriptor editor.

3. A WSDL file is needed (EmployeeDetailsPortletC2A.wsdl); follow these steps:

   a. Create a new folder with the name wsdl to store the WSDL files. Select your **EmployeeDetailsPortlet\Web Content** folder.

   b. Right-click the **Web Content** folder and select **New -> Folder**.

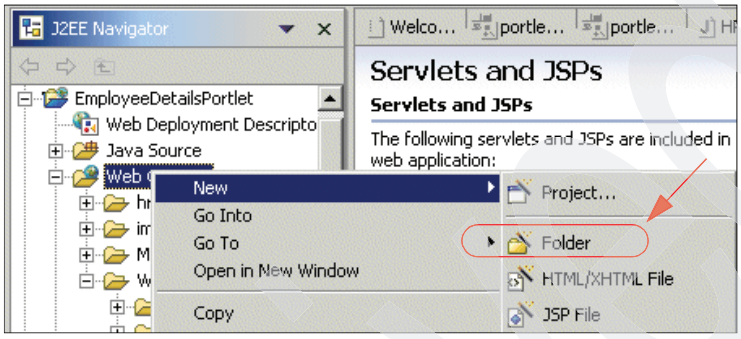


Figure 12-30 Create a new folder in Web Content directory

   c. Type `wsdl` for the Folder name field then click **Finish**.

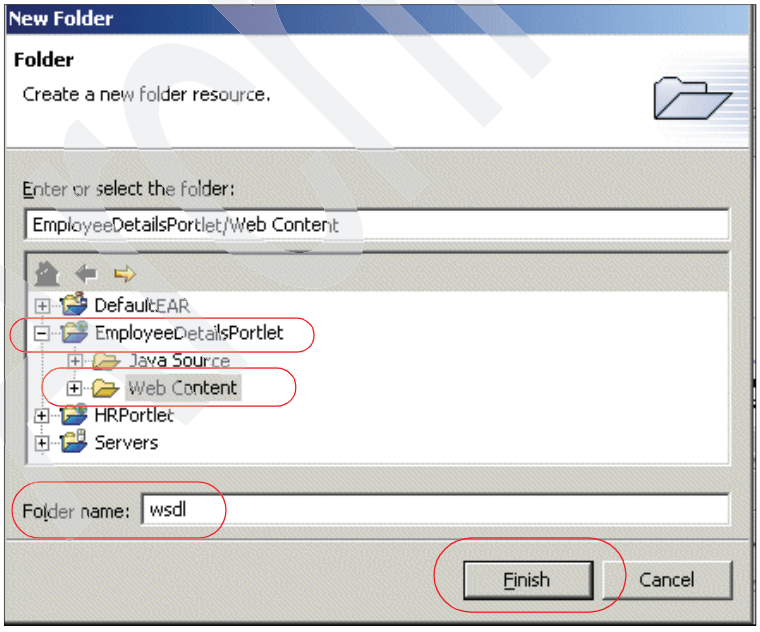


Figure 12-31 Create new folder wsdl

d. The directory structure should now look as illustrated in Figure 12-32.

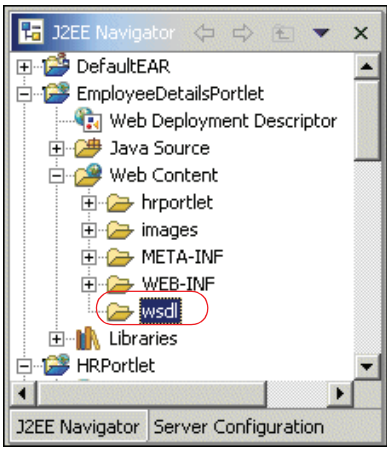


Figure 12-32 New wsdl folder

e. Select the new **EmployeeDetailsPortlet\Web Content\wsdl** folder.

f. Right-click the **wsdl** folder and choose **File -> New -> Other...** from the context menu.

g. In the New window, select **File** from the Simple category. Click **Next**.

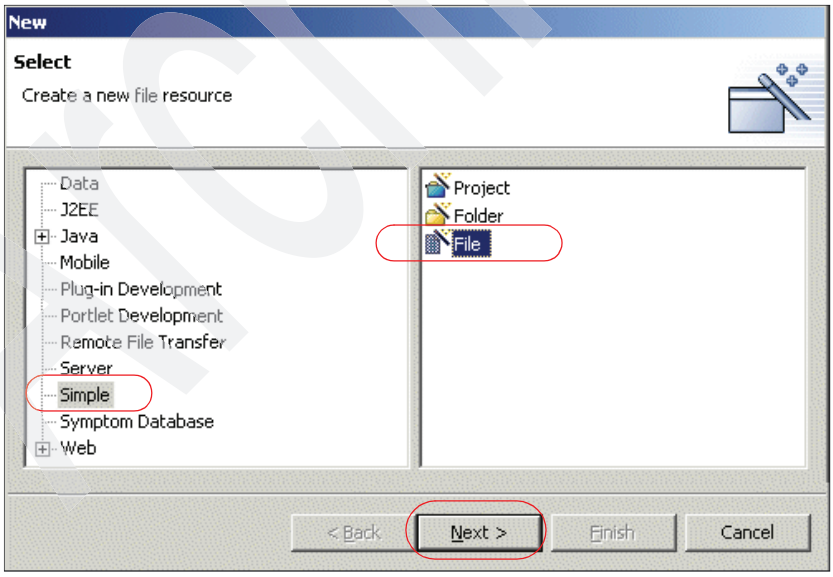


Figure 12-33 Creating a file resource

Archived

h. In the New File dialog, enter a file name of `EmployeeDetailsPortletC2A.wsdl`. Click **Finish**.

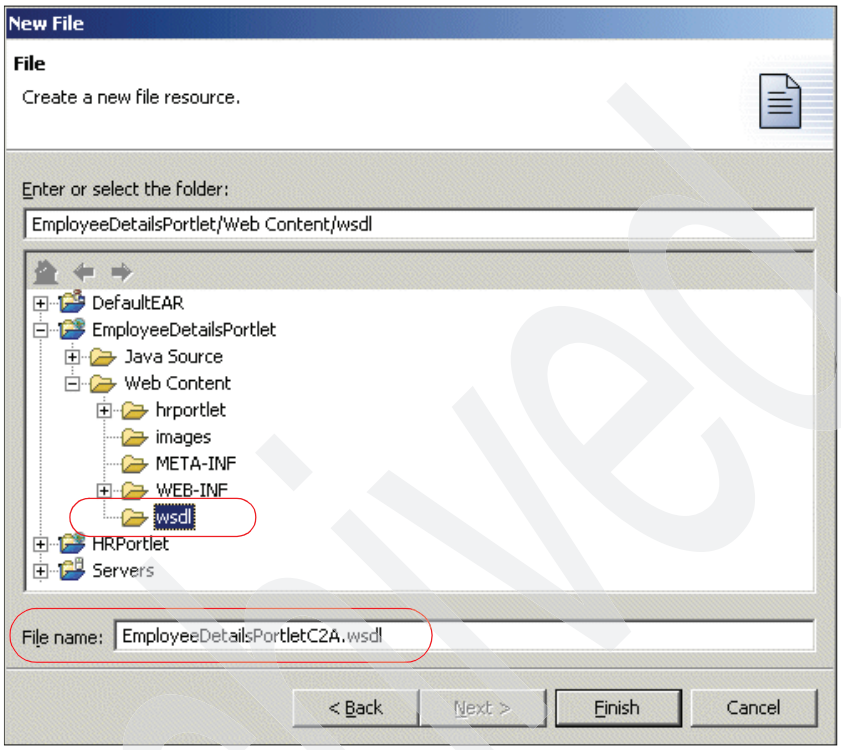


Figure 12-34 Creating file (wsdl) EmployeeDetailsPortletC2A.wsdl

i. In the XML editor, switch to the Source tab and enter the following XML code shown in Example 12-1.

You can also copy and paste this file from c:\LabFiles\C2A\Snippets\EmployeeDetailsPortletC2A.wsdl

Note: You can also download the sample code available as additional materials. See Appendix C, “Additional material” on page 543.

Example 12-1 EmployeeDetailsPortletC2A.wsdl file

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="GetResults_Service"
          targetNamespace="http://www.ibm.com/wps/c2a/examples/hrdetails"
          xmlns="http://schemas.xmlsoap.org/wsdl/"
          xmlns:portlet="http://www.ibm.com/wps/c2a"
          xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
          xmlns:tns="http://www.ibm.com/wps/c2a/examples/hrdetails"
```

Archived

```
          xmlns:xsd="http://www.w3.org/2001/XMLSchema"
          xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://schemas.xmlsoap.org/wsdl/
http://schemas.xmlsoap.org/wsdl/  http://www.w3.org/2001/XMLSchema
http://www.w3.org/2001/XMLSchema.xsd">

   <types>
      <xsd:simpleType name="DEPT_NO">
         <xsd:restriction base="xsd:string"></xsd:restriction>
      </xsd:simpleType>
   </types>

   <message name="GetResultsMessageNameRequest">
      <part name="get_ResultsPartName" type="tns:DEPT_NO" />
   </message>

   <portType name="GetResults_Service">
      <operation name="get_ResultsOperation">
         <input message="tns:GetResultsMessageNameRequest" />
      </operation>
   </portType>

   <binding name="GetResultsBinding" type="tns:GetResults_Service">
      <portlet:binding />
      <operation name="get_ResultsOperation">
         <portlet:action name="hrportlet.HRPortletDetailsAction" type="simple"
caption="Show all employees from this department."
description="Get.Results.for.specified.sql.string" />
         <input>
            <portlet:param name="DEPT_NOParam" partname="get_ResultsPartName"
caption="Show all employees from this department." />
         </input>
      </operation>
   </binding>

</definitions>
```

4. The next step is to update the portlet deployment descriptor to include a reference to the WSDL file:
   a. In the J2EE Navigator view, expand EmployeeDetailsPortlet/Web Content/WEB-INF and double-click **portlet.xml**.
   b. In the portlet deployment descriptor editor, expand the concrete portlet application and select **hrportlet.HRPortlet**.
   c. Change the title to `Employee Details Portlet`.

Archived

d. In the Setting Parameter area, click the **Add...** button to add the following parameter:

   `c2a-action-descriptor`

   Its value should be:

   `/wsdl/EmployeeDetailsPortletC2A.wsdl`

e. Press **Ctrl-S** to save the file and close the deployment descriptor editor.

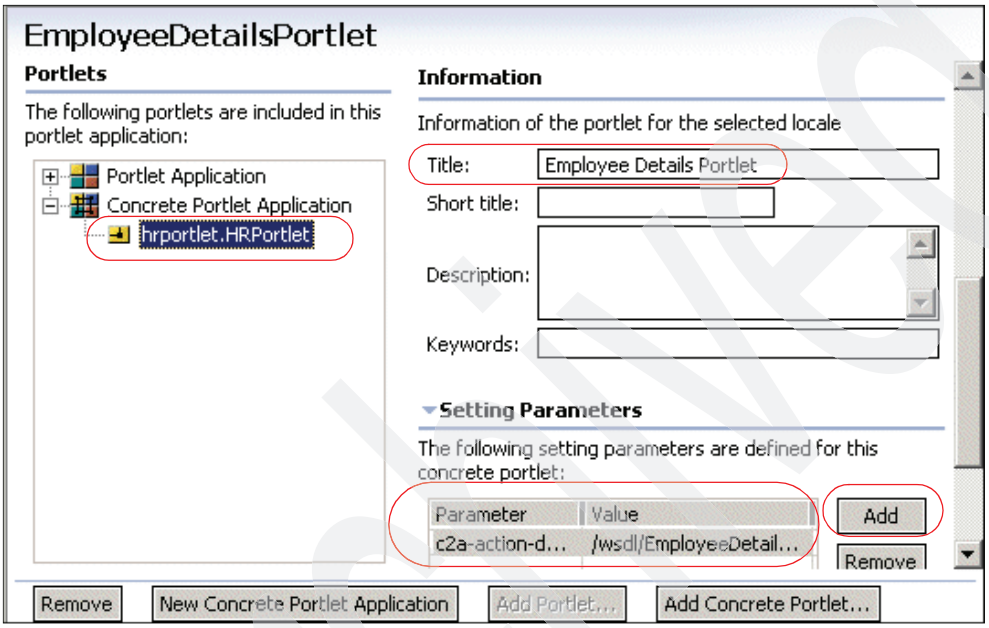


Figure 12-35 Updating portlet descriptor with wsdl file

5. The last step is to update the actionPerformed() method in the portlet class. In this scenario, you will use a special action string.

   a. Open the HRPortlet.java file from the EmployeeDetailsPortlet project.

   b. Insert the lines shown in Example 12-2 on page 405 at the end of the actionPerformed() method.

      You can also copy and paste this code from c:\LabFiles\C2A\Snippets\ap.java

      Note: You can also download the sample code available as additional materials. See Appendix C, “Additional material” on page 543.

Archived

Example 12-2 The hrportlet.HRPortletDetailsAction updates the SQL string

```
if (actionString.equals("hrportlet.HRPortletDetailsAction")) {
          HRPortletSessionBean bean = this.getSessionBean(request);
          bean.setSqlString(
             "select * from employee where workdept='"
                + (String) request.getParameter("DEPT_NOParam")
                + "'");
      }
```

6. Save all your files; you can use **Ctrl-S**.

12.2.5 Running the cooperative portlets

Execute the following steps to run the cooperative portlets scenario:

1. To run a project in the WebSphere Studio Site Developer test environment, it is necessary to add the portlet project to the test environment. You can use a previously created test server or you can create a new server as follows:
   a. Click the **Server Configuration** tab (on the navigator panel).
   b. Expand the Servers tree.
   c. Right-click **WebSphere Portal V5.0 Test Environment** or **Test Environment**.
   d. If needed, click **Add -> DefaultEAR** to add your project to the test environment.

Archived

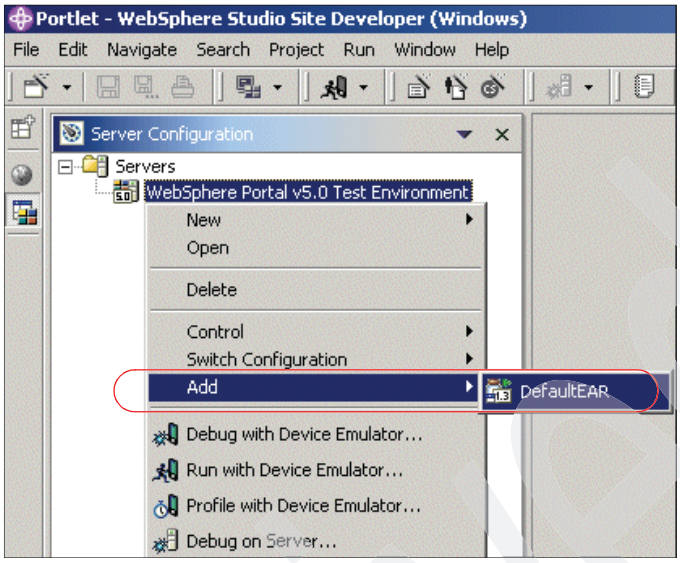


Figure 12-36 Adding a project to the test environment

2. If the Cloudscape sample database has not been populated, run the batch file to populate the test database to be used in this scenario. Click **c:\LabFiles\Cloudscape\CreateCloudTable.bat** to do this.

   Note: You can also download the sample code available as additional materials. See Appendix C, “Additional material” on page 543.

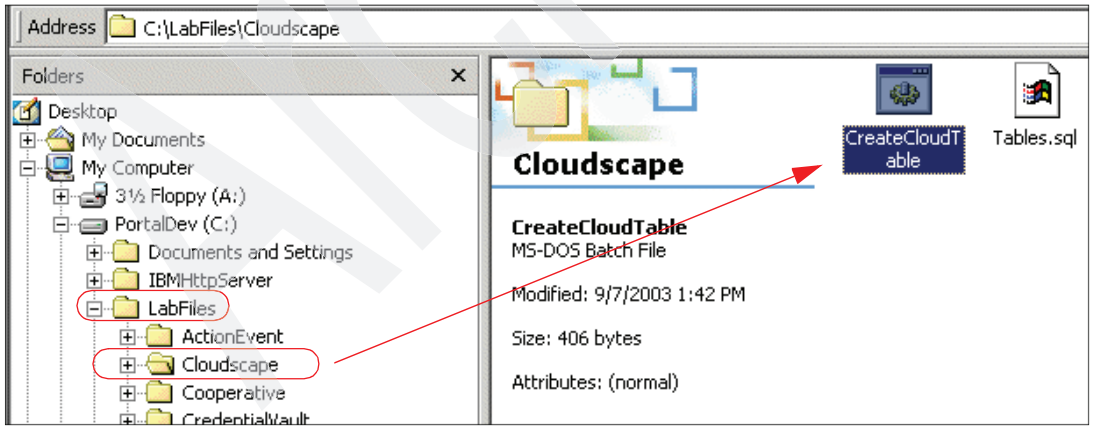


Figure 12-37 Populating test database

3. Next, click the **J2EE Navigator** tab to see your project again.

4. Right-click **HRPortlet**. Then click **Run on Server**. This will load your project into the test environment so that you can view it in the WebSphere Studio Site Developer Web browser. It may take a minute or two for this process to complete.
5. You will now see your newly created portlets project running in the Web browser.
6. Switch to Edit mode in HRPortlet (c2a source portlet).
7. Enter the following information and click **Submit**.
   - Database: `jdbc:db2j:C:\LabFiles\Cloudscape\WSSAMPLE`

     Note: You can also download the sample code available as additional materials. See Appendix C, “Additional material” on page 543.
   - User: `db2admin`
   - Password: `db2admin`
   - SQL: `select * from jobs`

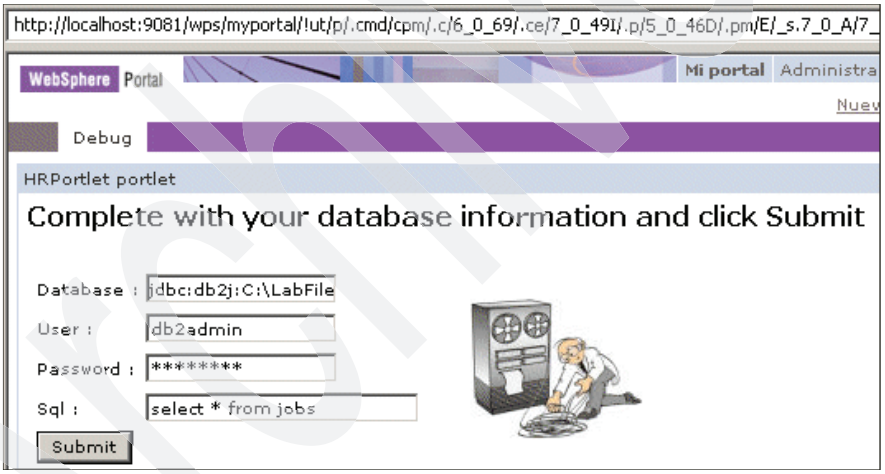


Figure 12-38 HRPortlet portlet in Edit mode

8. The HRPortlet now displays the jobs table, including a cooperative portlet menu in the DEPT_NO column. Before you can use this menu, you have to configure the data source in EmployeeDetailsPortlet.

Archived

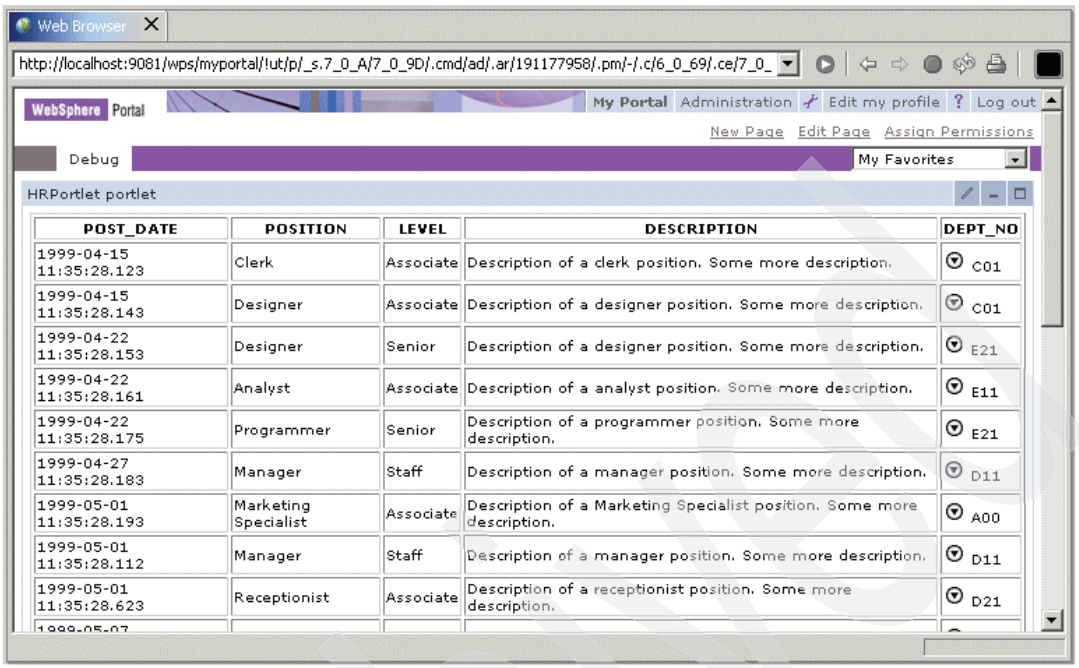


Figure 12-39 HRPortlet displays a cooperative menu in DEPT_NO column

9. Switch to Edit mode in EmployeeDetailsPortlet (c2a target portlet).
10. Enter the following information and click **Submit**. It is not necessary to enter an SQL command here, because it is built during the processing of the cooperative menu.
    - Database: `jdbc:db2j:C:\LabFiles\Cloudscape\WSSAMPLE`

      Note: You can also download the sample code available as additional materials. See Appendix C, “Additional material” on page 543.
    - User: `db2admin`
    - Password: `db2admin`

    Note: There is no need to enter an SQL statement (optional).
11. In the source cooperative portlet View mode, click a DEPT_NO column, for example **C01** or **A00.**
12. Click **Show all employees from this department** (see Figure 12-40 on page 409).

    Note: The EmployeeDetailsPortlet (target portlet) should now display all employees in the selected department.

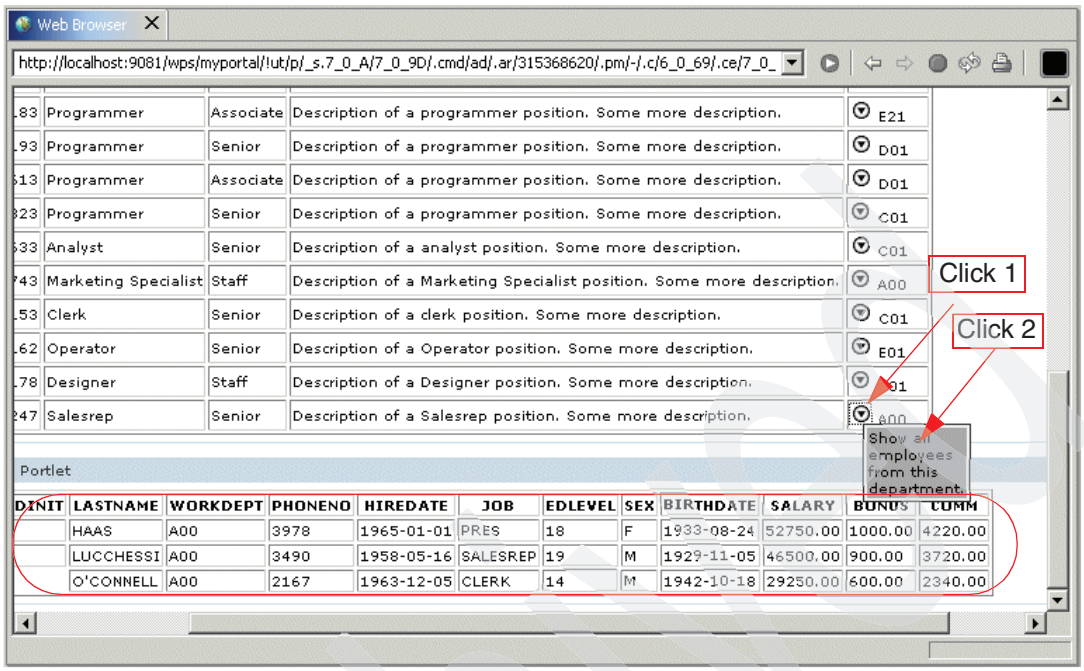


Figure 12-40 EmployeeDetailsPortlet displays all employees from same department

12.3 Hints and tips

In this section, we provide guidelines to troubleshoot some common cooperative portlets problems you may encounter.

Updates in portlet.xml are not reflected after a server restart

When using configuration parameters in portlet.xml, existing parameters are preserved and only new ones are added. To change this behavior, you have to change a special configuration parameter of the Deployment service as follows:

1. Open the file <WSSD_DIR>/runtimes/portal_v50/shared/app/config/services/DeploymentService.
2. Scroll down to the line `update.portlet.preserves.config.settings = true` and change this parameter value to `false`.
3. Restart the server.

Updates in WSDL files are not reflected after a server restart

Because of some limitations, updating the portlet application does not cause the WSDL to be read again.

There are a few ways to achieve an update after changing the WSDL file.

- You can reinstall the portlet application.
- You can program an action forcing the file to be reloaded.
- You can use XMLAccess to update the portlet. This is explained in Appendix B, “Automatically redeploying portlets” on page 535.

To reinstall the portlet application:

1. From the server configuration view, double-click your server.
2. In the server configuration editor, switch to the Portal tab.
3. Check **Enable base portlets for administration and customization**, save the configuration and close the editor.
4. Start the server.
5. Open a browser and log in. Switch to the administration page.



Figure 12-41 Switch to the Administration page to reinstall a portlet

6. From the left menu, select **Portlets -> Manage Applications**.
7. From the Web Modules list, select your cooperative target portlet and click **Uninstall**.

Archive

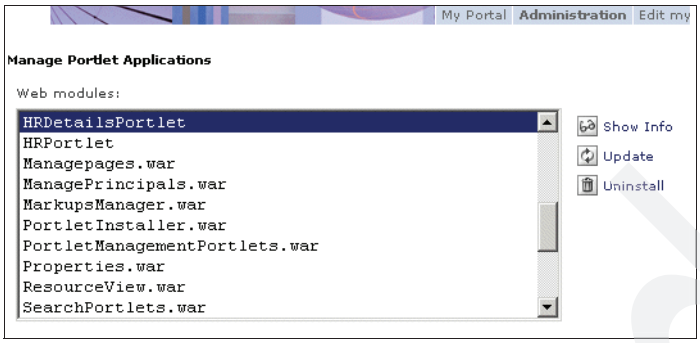


Figure 12-42 To uninstall a portlet, select the portlet and click Uninstall

8. Restart the server.

To force a reload of the WSDL files, you can also program a portlet action as illustrated in Example 12-3. After execution of this action (for example in configuration mode) and a new login, the runtime Portal will parse the changed WSDL file and store the new action set.

Example 12-3 This action removes all registered actions

```
if (actionString.equals("RemoveAllActions")) {
   PropertyBrokerService pbService = null;
   try {
      pbService =
         (PropertyBrokerService) portletConfig
            .getContext()
            .getService(
            PropertyBrokerService.class);
   } catch (PortletServiceUnavailableException e) {
      e.printStackTrace();
   } catch (PortletServiceNotFoundException e) {
      e.printStackTrace();
   }
   PortletSettings settings = request.getPortletSettings();
   try {
      pbService.unregisterActions(request, settings);
   } catch (PropertyBrokerServiceException e) {
      e.printStackTrace();
   }
}
```

Arnion

The server stops immediately after a server start

When you deploy a portlet application with a changed portlet UID which was already deployed before, the server does not start correctly because the old portlet configuration still exists in the configuration database.

This also happens if you create a new portal server within WebSphere Studio. To fix this problem, proceed as follows:

1. Remove all projects from your existing server configuration.
2. Enable the base portlets and start the server.
3. Log in to WebSphere Portal and remove the existing portlet through the Administration page.
4. Stop the server.
5. Add the projects you removed previously.

Archived

13

Advanced cooperative portlets

This chapter discusses advanced cooperative portlet topics.

After reading this chapter, you will be able to:

- Develop source cooperative portlets using a programmatic approach to publish properties.
- Develop target cooperative portlets using a programmatic approach to publish properties.
- Develop source cooperative portlets broadcasting data to two or more portlets.

Note: The sample scenario included in this chapter requires that you have read Chapter 12, "Cooperative portlets" on page 371. You can also download the sample code available as additional materials. See Appendix C, "Additional material" on page 543.

© Copyright IBM Corp. 2004. All rights reserved.

13.1 Publishing properties programmatically

As mentioned in Chapter 12, “Cooperative portlets” on page 371, each action in a cooperative portlet is associated with a single input parameter and zero or more output parameters that provide information to the action about the objects in which the property value should be bound, such as the request object or the session object.

Note: Cooperative portlets using the programmatic approach require that you create a wire; for details about creating wires, see 13.5.4, “Wire portlets” on page 440 and 13.4, “Wiring tool” on page 418. This is because property values are transferred to the target portlets only if wires have been created.

Each parameter is associated with exactly one property. Parameters associated with input properties are called *input parameters*, while those associated with output properties are called *output parameters*. Instead of actions, target portlets can receive property changes directly through the *PropertyListener* interface.

The actual transfer of the property can be initiated by one of the following methods:

1. A user launches a Click-to-Action event from an icon on the source portlet. The icon presents a pop-up menu containing the list of targets for the action. After the user selects a specific target, the property broker delivers the data to the target in the form of the corresponding portlet action.
2. A user holds the **Ctrl** key while clicking an action and chooses to have the selection saved persistently as a connection between two portlets, called a wire.
3. The source portlet can perform a programmatic publish of properties to the broker, when it determines that property values have changed. Such property values are transferred to the target(s) only if wires have been created.

The property broker provides APIs to give developers more control over how portlets handle the input and output properties. In general terms, the programmatic approach might be a better option over the declarative approach when the portlet needs to do the following:

- Activate or deactivate actions for a session.
- Change the portlet state but not requiring the portlet to react immediately.
- Publish output properties using the *changedProperties()* method.
- Register actions programmatically instead of declaring them in a WSDL file. This may be necessary when the action or property is not known at

development time, such as when a portlet is generated by a builder application.

- Generate markup content directly in the portlet rather than using JSPs.

The following packages are provided for portlets to publish properties to the property broker programmatically:

- com.ibm.wps.pb.property
- com.ibm.wps.pb.portlet
- com.ibm.wps.pb.service

13.2 Portlet event handling

The portlet programming model involves an event phase and a render phase in each request-response cycle.

- The event phase is when the property broker delivers notifications to cooperative portlets and when the cooperative portlets can notify the property broker of property value changes. During the event phase, an action may be delivered on one portlet. If the property broker is used, this may result in other actions being triggered on other portlets.
- The event phase is followed by the render phase, in which each portlet is asked to return markup, which is then aggregated in a single page. The markup may embed actions which can be invoked by the user. The page is then returned to the client (such as a browser).

At any point during the event phase, a portlet may explicitly publish the value of an output property to the property broker by invoking the *changedProperties()* method. This is an alternative to the declaration of output parameters for actions and binding the output parameter values to the request or session when the action is invoked. This may happen in the following cases:

- In the callback method associated with the start of the event phase (beginEventPhase method)
- In the invocation of the setProperties() method in a target portlet
- In the portlet action method invocation

The publishing calls are dealt with by the property broker in the same manner as output parameters of actions: wires associated with output properties are examined and the property values propagated using the information in the target end of the wire.

Archived

Note: The process may continue recursively; however, the property broker detects loops and breaks them. Also, during the event phase of the subsequent request, the action is invoked on the corresponding target portlet or portlets.

Figure 13-1 illustrates a simplified version of cooperative portlets implemented using the programmatic approach.

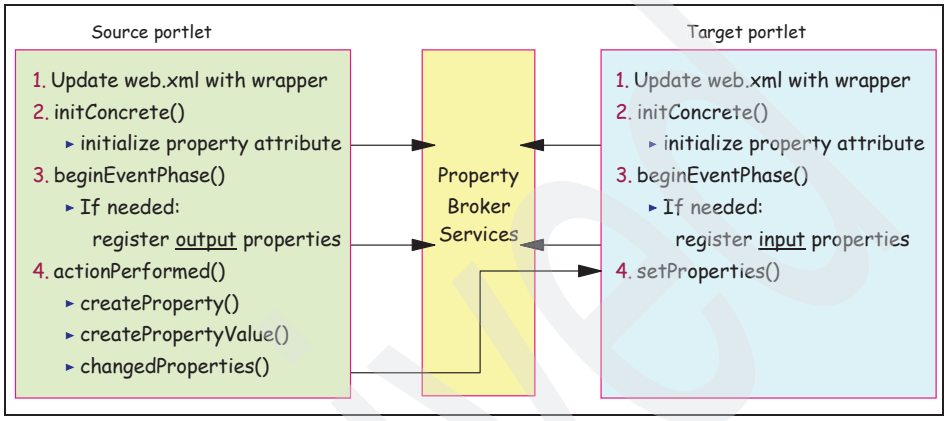


Figure 13-1 Sample summary of a programmatic approach

In the general case using a programmatic approach, the source portlet needs to implement the following:

1. Specify the C2A wrapper in the web.xml descriptor as explained in Chapter 12, “Cooperative portlets” on page 371.
2. The property broker attribute needs to be initialized; this can be done in the *initConcrete()* method.
3. The portlet will need to register its output properties by using the *registerProperties()* method. This can be done by implementing a *beginEventPhase()* method so the portlet is notified when the event phase starts.

   Note: The EventPhaseListener interface requires that you also provide the endEventPhase() method. If needed, some cleanup can be done in this method.
4. The source portlet publishes its output properties, for example when processing an action in the *actionPerformed()* method.

In a similar way, the target portlet needs to be updated as follows:

1. Specify the C2A wrapper in the web.xml descriptor as explained in Chapter 12, “Cooperative portlets” on page 371.

Archive

2. The property broker attribute needs to be initialized; this can be done in the *initConcrete()* method.
3. The portlet will need to register its input properties by using the *registerProperties()* method. This can be done by implementing a *beginEventPhase()* method so the portlet is notified when the event phase starts.
4. The target portlet implements the *setProperties()* method to be notified of property changes reported by other source portlets.

   Note: The target portlet must implement the PropertyListener interface.

13.3 Broadcasting source data

Using the broadcast feature of the cooperative broker, users can send the same data to all portlets on the page with matching actions. The target cooperative portlet of a broadcast can use the declarative or programmatic approach to publish properties.

To include the broadcast menu item to the HRPortlet, proceed as follows:

1. Open the JSP HRPortlet/Web Content/hrportlet/jsp/html/HRPortletView.jsp.
2. In the encodeProperty tag, include the broadcast attribute so it looks as shown in Example 13-1.

Example 13-1 The broadcast attribute enables the broadcast feature.

```
<P>
   <C2A:encodeProperty
   name="<%=results.getColumnName(col).toString()+\"Param\"%>"
   namespace="http://www.ibm.com/wps/c2a/examples/hrdetails"
   type="<%=results.getColumnName(col)%>"
   value="<%=results.getCacheValueAt(row, col).toString()%>"
   broadcast="true"/>
   <%=results.getCacheValueAt(row, col)%>
</P>
```

3. Test the application as described in the last chapter. Do not forget to enter the database attribute in the Edit mode of both the Employee and Department Details Portlets.
4. From the cooperative portlet menu choose **Invoke all actions**. Now both details portlets display the details of the selected department number.

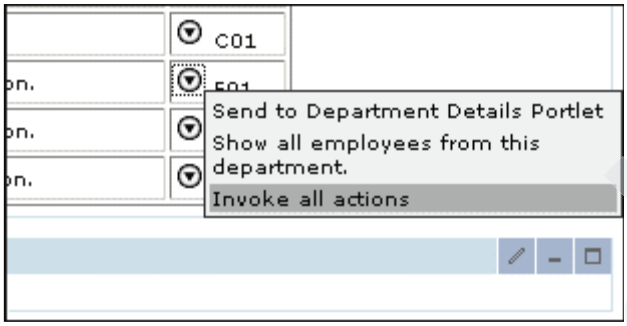


Figure 13-2 Choose Invoke all actions to broadcast the data to all portlets

13.4 Wiring tool

The portlet wiring tool allows you to view the properties that portlets on the page can send or receive. If a match is available between two portlets, you can create a wire between the two portlets. Existing wires may also be deleted using the tool. This is an alternative to the wire creation or deletion while interacting with the portlets using the **Ctrl** key.

The wiring tool allows wires to be created in situations which are not handled by the interactive approach. For example, the tool does not require the existence of Click-to-Action menus to initiate wire creation, and can be used to create multiple wires from a single source property (using the interactive approach, a single source can be wired to a single target or all targets, not an arbitrary subset). Wire creation or deletion is subject to the access control checks.

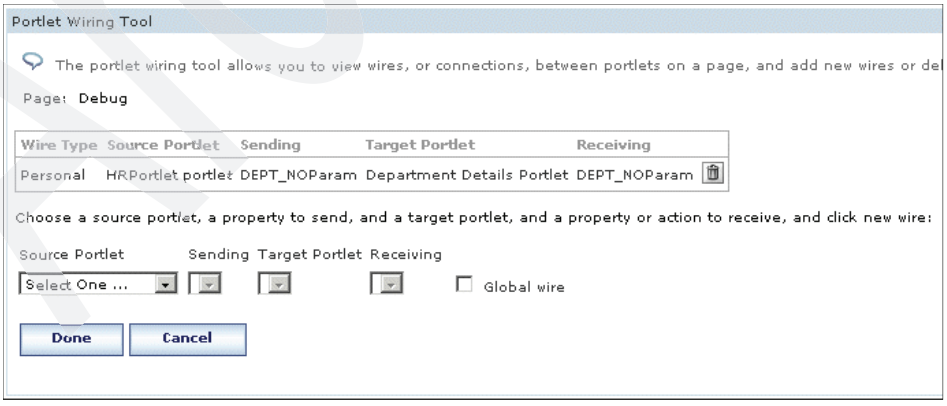


Figure 13-3 Creating wires programmatically using Portlet Wiring Tool

Archived

This portlet is initially provided after the release of WebSphere Portal V5.0. Subsequent versions of WebSphere Portal will include the Portlet Wiring Tool as part of the product and deployed to the Page Customizer.

Note: You can download the tool from the WebSphere portal catalog. The Navigation code is 1WP10004E.

13.5 Sample scenario

In this section, a sample scenario is provided to illustrate how to develop cooperative portlets using the programmatic approach.

13.5.1 Declarative source cooperative portlet

In this scenario, you will implement a combined scenario where the source cooperative portlet uses the declarative approach to interact with a target cooperative portlet using the programmatic approach. The sample scenario is shown in Figure 13-4.

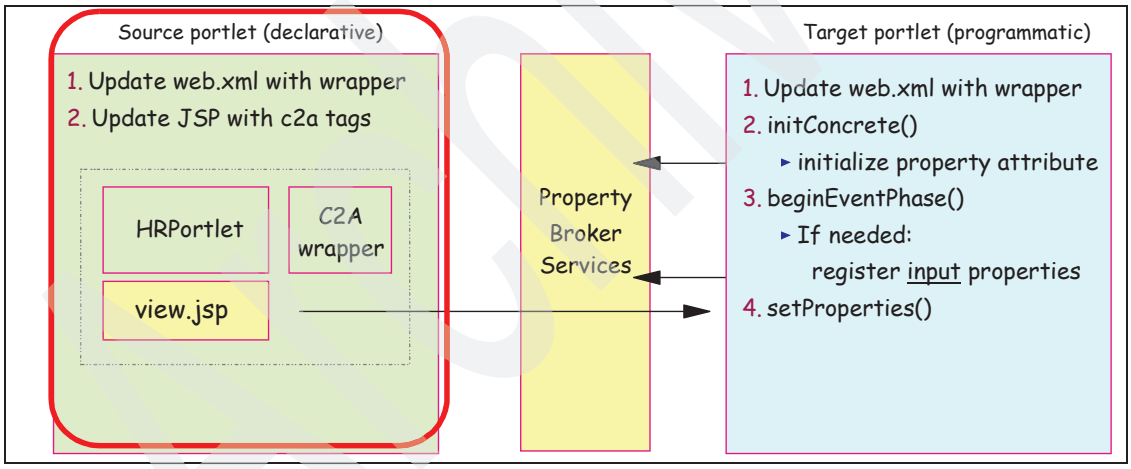


Figure 13-4 A sample scenario using a combined approach

Creating the HRPortlet portlet application project

In this section, the source cooperative portlet from Chapter 12, “Cooperative portlets” on page 371 will be used. Follow these steps if you do not have this portlet project in your workspace:

1. If not already running, start the IBM WebSphere Studio Site Developer, click **Start -> Programs -> IBM WebSphere Studio -> Site Developer 5.0**.

2. Select **File -> New -> Portlet Application Project**.

   Note: If you do not see this option, select **File -> New -> Other** and click **Portlet Development** and **Portlet Application Project**.

3. In the Define Portlet Project window, enter `HRPortlet` for the project name. Click **Next**.

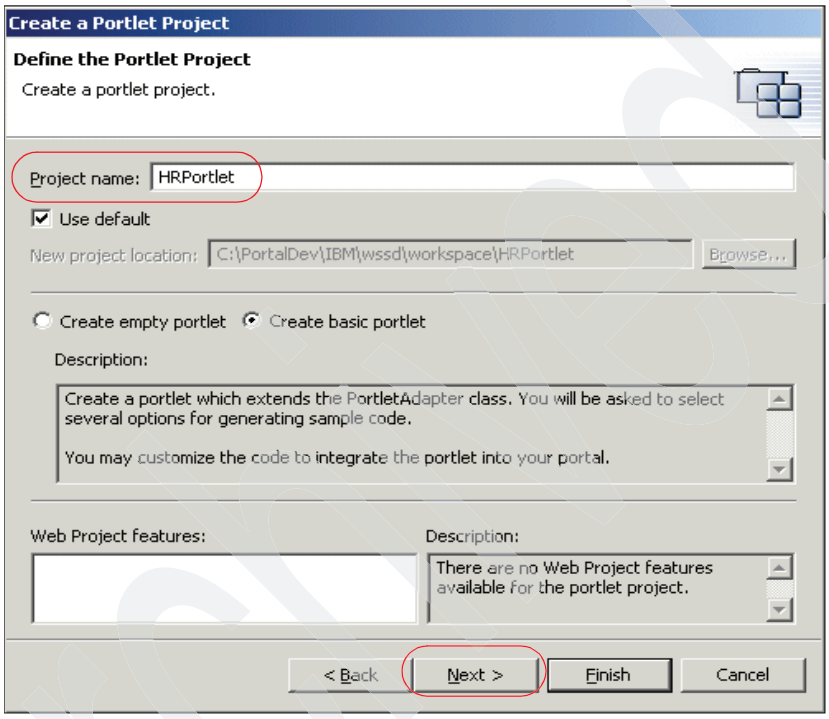


Figure 13-5 Define the Portlet Project

4. In the J2EE Settings Page, click **Next** to take the default values.
5. In the Portlet Settings Page, select **Change code generation options** and enter `HRPortlet` for the Class prefix. Click **Finish** to generate the framework for your project.

Redbooks

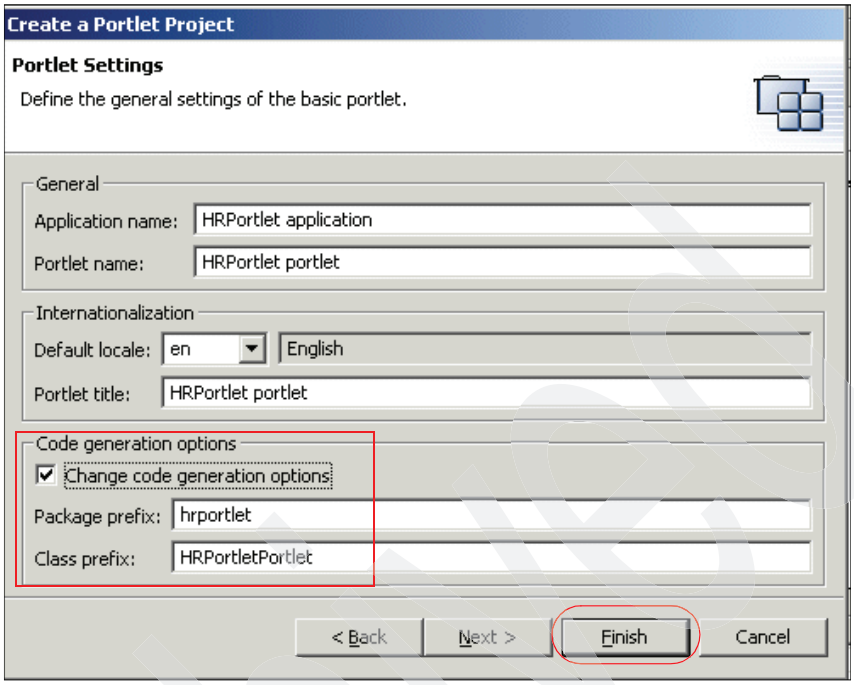


Figure 13-6 Portlet Settings

6. If you have any other portlets in the DefaultEAR project, remove them at this time.

Importing the HRPortlet portlet

The HRPortlet portlet from Chapter 12, “Cooperative portlets” on page 371 will be used as a base for this scenario. This portlet will be enabled to act as a source cooperative portlet in this scenario. You will need to import this portlet if it is not in your workspace.

Follow these steps to import this portlet if it is not in your workspace:

1. Import the WAR file by selecting **File -> Import**.
2. Select **WAR file** and click **Next**.
3. In the *Import Resources from a WAR File* window, enter the following information:
   a. WAR file: browse to `C:\LabFiles\C2A\solutions\HRPortlet.war`.

      Note: The sample scenario included in this chapter requires that you download the sample code available as additional materials. See Appendix C, “Additional material” on page 543.

b. Web project: select **Existing**. In the box that pops up, select **HRPortlet** and click **OK**.

c. Context root: this will change to `/HRPortlet`.

d. In Options, select the **Overwrite existent resources without warning** check box.

e. Click **Finish** to import the WAR file.

4. Make sure the web.xml descriptor has been updated with the c2a wrapper.

5. Make sure the PortletView.jsp has been updated with the c2a tags and c2a tag library. See Example 13-2.

Example 13-2 C2a library and tags

```
<%@ page contentType="text/html" import="java.util.*, hrportlet.*"%>
<%@ taglib uri="/WEB-INF/tld/portlet.tld" prefix="portletAPI" %>
<%@ taglib uri="/WEB-INF/tld/c2a.tld" prefix="C2A" %>
.............................................
<TD>
<P><C2A:encodeProperty
   name='<%=results.getColumnName(col).toString()+\"Param\"%>'
   namespace="http://www.ibm.com/wps/c2a/examples/hrdetails"
   type="<%=results.getColumnName(col)%>"
   value="<%=results.getCacheValueAt(row, col).toString()%>" />
<%=results.getCacheValueAt(row, col)%>
      </P>
</TD>
```

6. Save your files (or press **Ctrl-S** to save the files).

13.5.2 Enabling the portlet for target C2A programmatic

In this section, you will import a second copy of the HRPortlet and update it to support Click-to Action as a target cooperative portlet using the programmatic approach. This target portlet will execute a fixed SQL statement with a variable *where* clause.

The code for the target portlet class must meet the following requirements:

- The action must be implemented either as a portlet action or a Struts action. For portlet actions, you should use the simple action Strings rather than the deprecated PortletAction class.
- Portlet actions must accept a single parameter. The parameter may appear as a request parameter, a request attribute, a session attribute, or an action attribute (deprecated), as specified in the action declaration or registration.

The HRPortlet is already prepared for this situation, so only a few changes are needed in the portlet class code. Figure 13-7 illustrates the target cooperative portlet for this sample scenario.

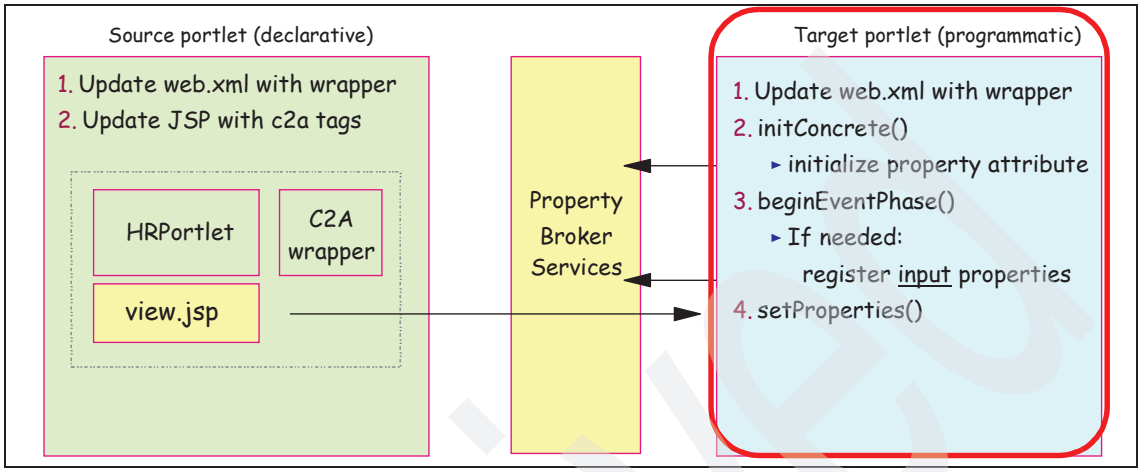


Figure 13-7 Cooperative portlets - programmatic sample scenario (target portlet)

Creating a target portlet application project

To import a second version of the HRPortlet, create a portlet application project as follows:

1. In the Portlet perspective, choose **File -> New -> Portlet Application Project** from the main menu.
2. In the Create a Portlet Project window, enter `DepartmentDetailsPortlet` as the project name and select **Create empty portlet**. Click **Next**.

Archive

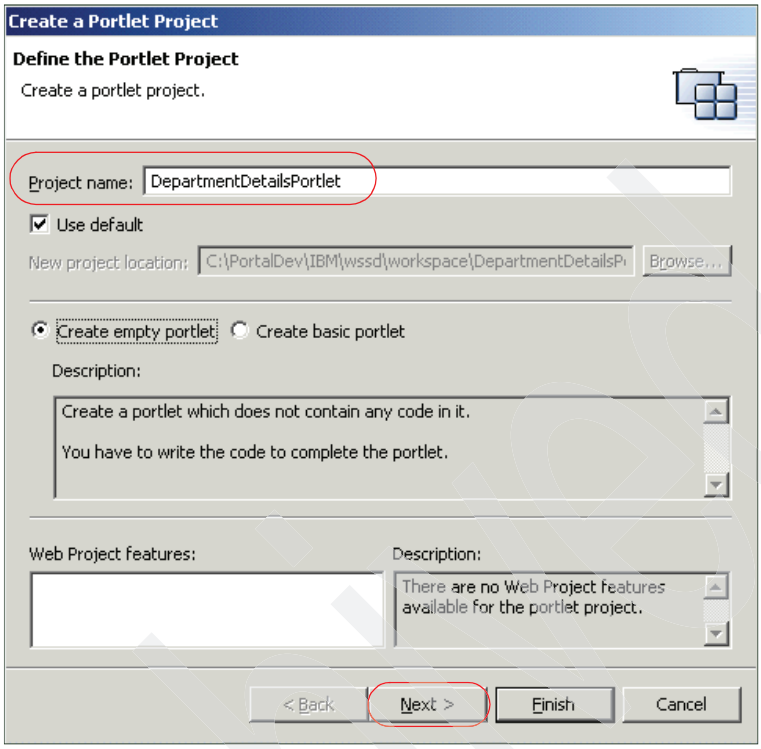


Figure 13-8 Create portlet project DepartmentDetailsPortlet

3. In the J2EE Settings Page, select the **Existing** radio button for the Enterprise application project and enter `DefaultEAR` as its name. Click **Finish** to create the portlet project.

Archive

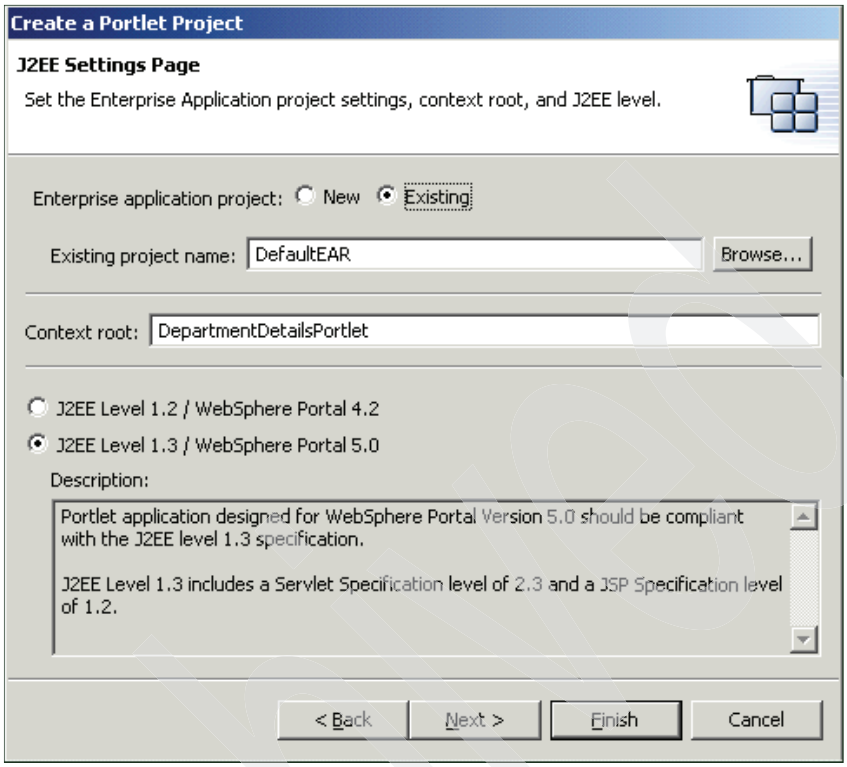


Figure 13-9 J2EE Settings Page

4. Click **OK** if you receive the Repair Service Configuration message indicating that the project will be added to DefaultEAR.

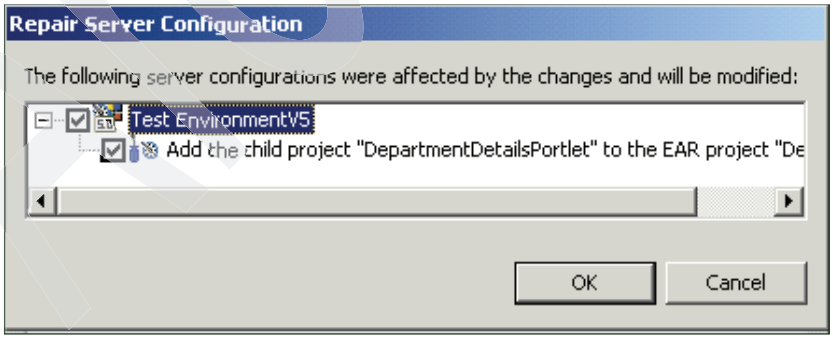


Figure 13-10 Repair Server Configuration message

Importing the original portlet application

1. From the main menu, select **File -> Import** to import the original HRPortlet.
2. Choose **WAR file**, click **Next** and configure as follows:
   a. Browse to the location of the HRPortlet.war file in `c:\LabFiles\AdvC2A\HRPortlet.war`.

      Note: The sample scenario included in this chapter requires that you download the sample code available as additional materials. See Appendix C, “Additional material” on page 543.

   b. As the Web Project, select the existing Web project **DepartmentDetailsPortlet**.
   c. Select **Overwrite existing resources without warning** and click **Finish**.

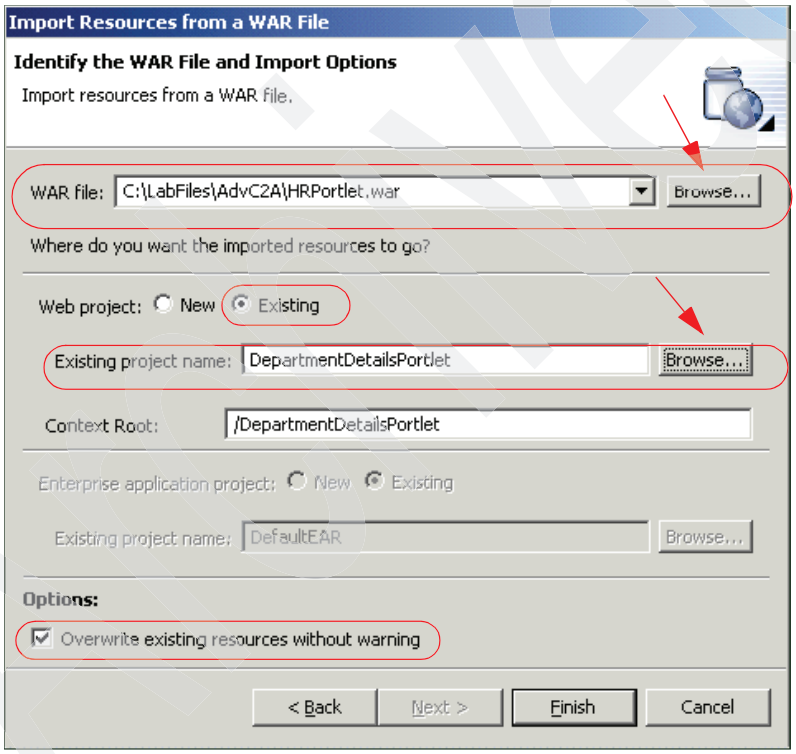


Figure 13-11 Import a second version of HRPortlet (target c2a portlet)

3. Since the HRPortlet and DepartmentDetailsPortlet portlet applications use the same UID, a warning message will appear in the task pane.

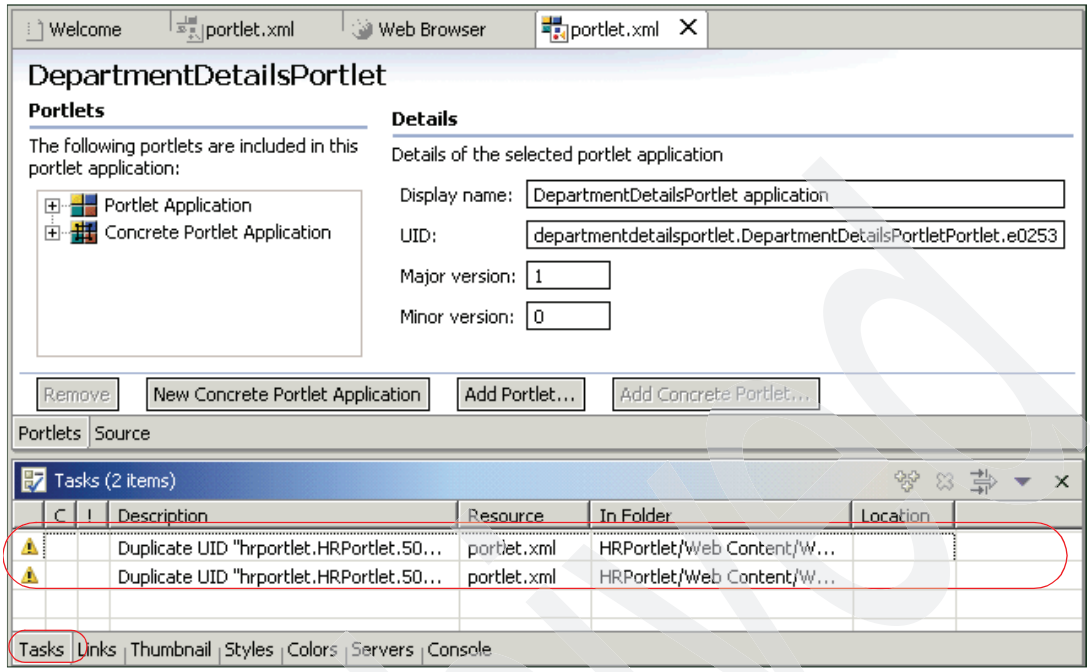


Figure 13-12 Duplicate UID messages

4. To fix this problem, expand DepartmentDetailsPortlet/Web Content/WEB-INF in the J2EE Navigator view. Double-click **portlet.xml**.

Archived

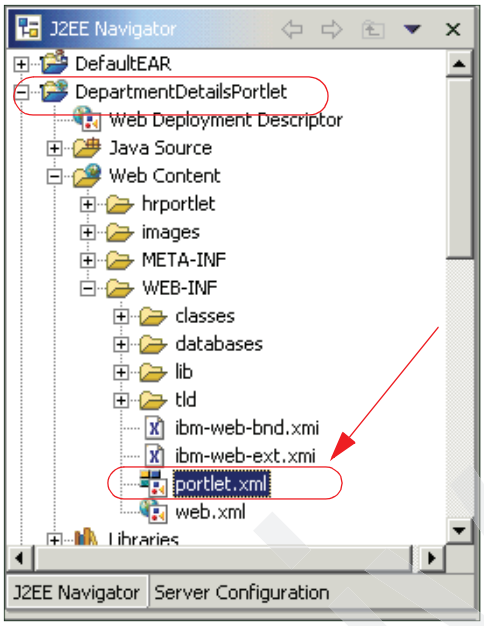


Figure 13-13 Selecting portlet.xml

5. In the portlet deployment descriptor editor, select **Portlet Application** and change the last digit of the UID for this portlet application. For example, in this sample scenario, the last digit was 6 and it was changed to 7.

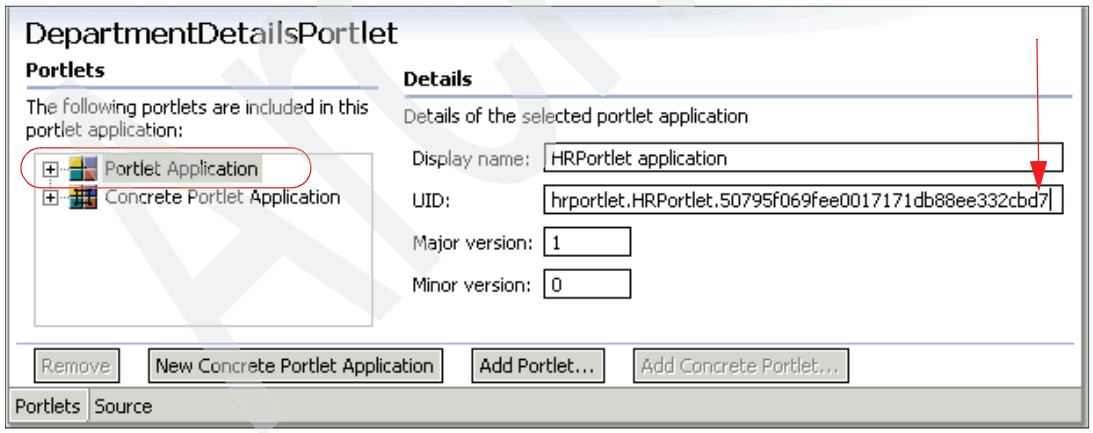


Figure 13-14 Changing a digit in portlet application UID

Archived

6. In a similar way, select **Concrete Portlet Application** and change the last digit before the last dot of the UID to give it the same value as in the previous step.

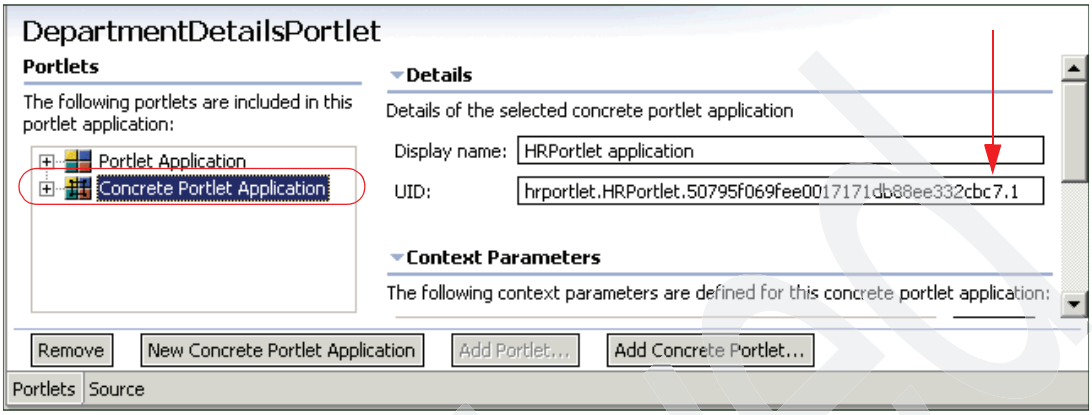


Figure 13-15 Changing digit in concrete portlet application UID

7. Save your changes. The warning messages in the Tasks view should disappear.

Importing pbportlet.jar

Import the property broker (pbportlet.jar) file into the DepartmentDetailsPortlet project.

1. Select **File -> Import -> File system**.
2. For example, for the directory browse to:

   `C:\Program Files\ibm\WebSphere Studio\runtimes\portal_v50\pb\lib`
3. Select **pbportlet.jar**.
4. For the destination, select the folder **DepartmentDetailsPortlet/Web Content/WEB-INF/lib**.

Archived

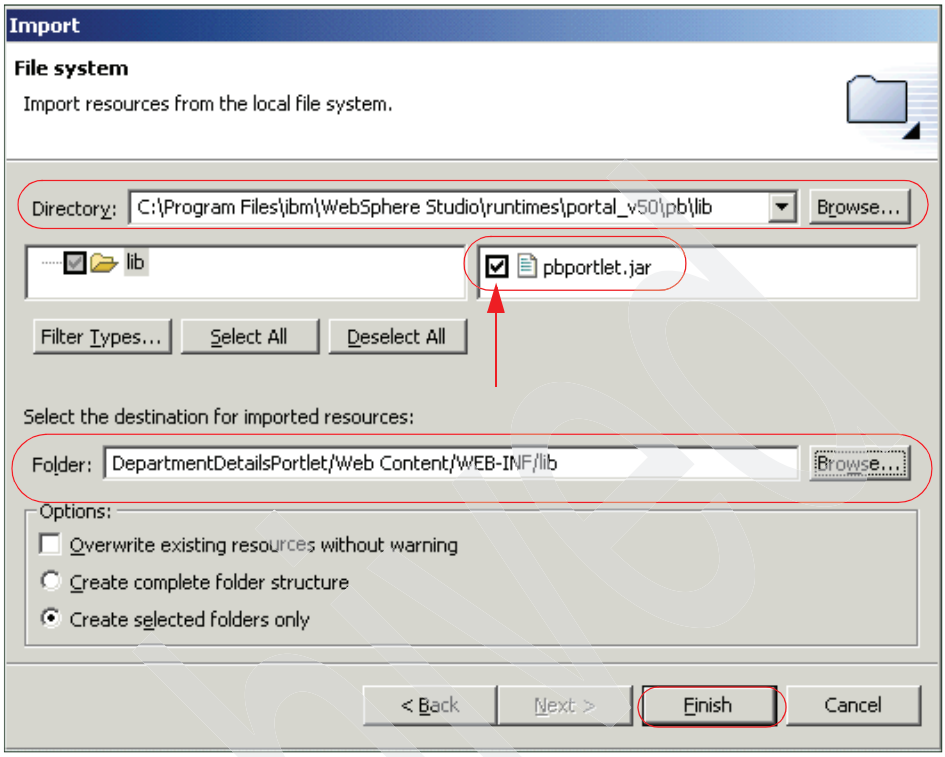


Figure 13-16 Import the property broker jar file (pbportlet.jar)

5. Click **Finish**.

Updating the web.xml descriptor

Update the Web deployment descriptor by changing the servlet class to PortletWrapper and including the `c2a-application-portlet-class` parameter.

1. In the J2EE Navigator view, expand DepartmentDetailsPortlet and double-click **Web Deployment Descriptor**.
2. Switch to the Servlets tab and select the **hrportlet.HRPortlet** servlet (Figure 13-18 on page 432).
3. In the Details area, click the **Browse...** button to change the servlet class (Figure 13-18 on page 432).
4. In the Servlet selection dialog, select the **PortletWrapper** class from the com.ibm.wps.pb.wrapper package and click **OK**.



Figure 13-17 Adding PortletWrapper

5. In the Initialization area, click the **Add...** button to add a new parameter.
6. Enter a parameter name of `c2a-application-portlet-class` and a parameter value of `hrportlet.HRPortlet`. The final editor should look as shown in Figure 13-18 on page 432.

Archived

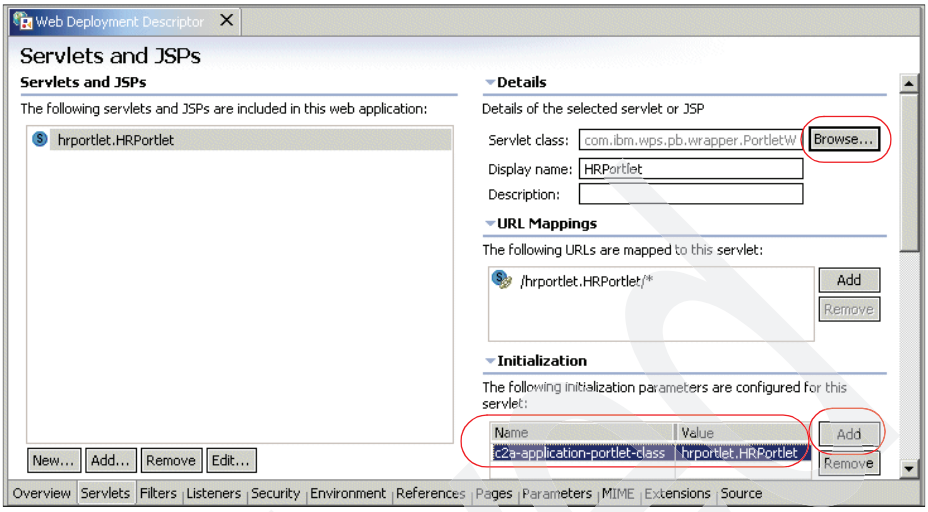


Figure 13-18 Web Deployment Descriptor for the target cooperative portlet

7. Save your files (or press **Ctrl-S** to save the file) and close the deployment descriptor editor.

Updating the DepartmentDetailsPortlet

In this section, you will update the portlet to act as a target C2A portlet using the programmatic approach. The DepartmentDetailsPortlet is a copy of the HRPortlet and it will register the DEPT_NO property by using the property broker API instead of a WSDL file used in the declarative approach.

Follow these steps to update the portlet class to publish the property and create call back methods to recognize the property changes:

1. Open the file DepartmentDetailsPortlet/Java Source/hrportlet/HRPortlet.java.
2. Update the class definition so it implements the PropertyListener and EventPhaseListener interfaces.

```
public class HRPortlet extends PortletAdapter implements PropertyListener,
EventPhaseListener, ActionListener
```

   Note: The setProperties method of the PropertyListener interface receives updates of property values. To register properties, you will use the *beginEventPhase* method of the EventPhaseListener interface, since only during the event phase is it possible to register and unregister properties.

3. Insert the following two new class attributes so the code looks like this:

```
public class HRPortlet extends PortletAdapter implements PropertyListener,
EventPhaseListener, ActionListener {
```

```
PropertyBrokerService pbService;
PortletConfig portletConfig;
```

Note: `pbService` is an interface to the property broker and `portletConfig` stores the portlet config.

4. Update the *init* method so it looks as shown in Example 13-3.

Example 13-3 Update init method to obtain portlet configuration

```
   public void init(PortletConfig portletConfig) throws
UnavailableException {
      super.init(portletConfig);
      this.portletConfig=portletConfig;
   }
```

5. Insert the *initConcrete* method shown in Example 13-4 to initialize the property broker attribute.

Example 13-4 Initialize property broker

```
   public void initConcrete(PortletSettings settings)
      throws UnavailableException {
      try {
         pbService =
            (PropertyBrokerService) getPortletConfig()
               .getContext()
               .getService(
               PropertyBrokerService.class);
      } catch (PortletServiceUnavailableException e) {
         throw new UnavailableException("Could not locate
PropertyBrokerService.");
      } catch (PortletServiceNotFoundException e) {
         throw new UnavailableException("Could not locate
PropertyBrokerService.");
      }
   }
```

6. For simplicity, create the *registerPropertiesIfNecessary* method to register the property we are interested in.

   Note: This method performs the same as the WSDL file when using the declarative approach. Notice also that you need to specify a direction of *Property.IN*. In addition, actions are not registered (which is possible using the registerActions method) in this scenario. In other words, this portlet will be invoked by the property broker using the setProperties method instead of actionPerformed.

Archived

Example 13-5 registerPropertiesIfNecessary method

```
   private void registerPropertiesIfNecessary(PortletRequest request)
      throws PropertyBrokerServiceException {
      PortletSettings settings = request.getPortletSettings();
      Property[] properties = pbService.getProperties(request, settings);
      if (properties == null || properties.length == 0) {
         PortletContext context = getPortletConfig().getContext();
         //not registered, register now
         properties = new Property[1];
         properties[0] = PropertyFactory.createProperty(settings);
         properties[0].setName("DEPT_NOParam");
         properties[0].setDirection(Property.IN);
         properties[0].setType("DEPT_NO");

properties[0].setNamespace("http://www.ibm.com/wps/c2a/examples/hrdetails");
         properties[0].setTitleKey("HRDetails.Department");
         properties[0].setDescriptionKey(
            "Display department details");
         pbService.registerProperties(request, settings, properties);
      }
   }
```

7. Add the *beginEventPhase* and *endEventPhase* methods which are call back methods called during the event phase.

Example 13-6 The beginEventPhase methods registers the property if necessary

```
public void beginEventPhase(PortletRequest request) {
      try {
         registerPropertiesIfNecessary(request);
      }
      catch (Throwable e) {
         e.printStackTrace();
      }
   }
public void endEventPhase(PortletRequest request) {
   }
```

8. Add the setProperties method. Using this method, the class is notified about property changes.

Example 13-7 setProperties updates the sql statement

```
public void setProperties(
   PortletRequest request,
   PropertyValue[] properties) {
   PortletSession session = request.getPortletSession();
   HRPortletSessionBean sessionBean = getSessionBean(request);
   for (int i = 0; i < properties.length; i++) {
```

```
            System.out.println(properties[i].toString());
            if (properties[i].getProperty().getName().equals("DEPT_NOParam")) {
               String value = (String)properties[i].getValue();
               if (value.equals("*"))
                  sessionBean.setSqlString("select * from department");
               else
                  sessionBean.setSqlString(
                     "select * from department where deptno='"
                        + value
                        + "'");
            }
         }
      }
```

9. Right-click anywhere in the Java editor and select **Source -> Organize Imports** to include the missing import statements. In the Organize Imports dialog, choose to import the **com.ibm.wps.pb.property.Property** class.

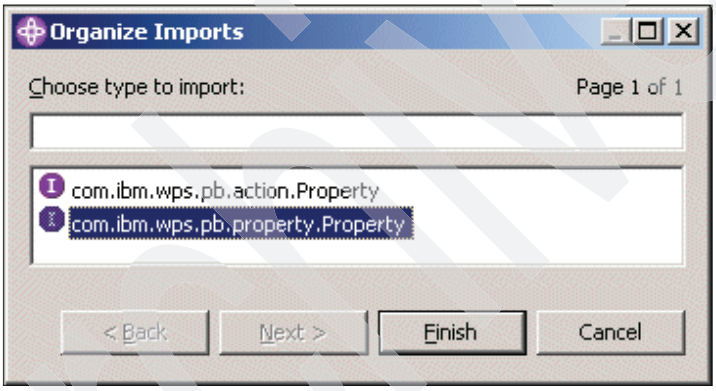


Figure 13-19 The Organize Imports dialog.

10. Save and close the HRPortlet.java file.

13.5.3 Running the cooperative portlets

Execute the following steps to run the cooperative portlets scenario:

1. To run a project in the WebSphere Studio Site Developer Test Environment, it is necessary to add the portlet project to the test environment. You can use a previously created test server or you can create a new server. Follow these steps:
   a. Click the **Server Configuration** tab (on the navigator panel).
   b. Expand the Servers tree.

c. Right-click **WebSphere Portal V5.0 Test Environment** or **Test Environment**.

d. If needed, click **Add -> DefaultEAR** to add your project to the Test Environment.

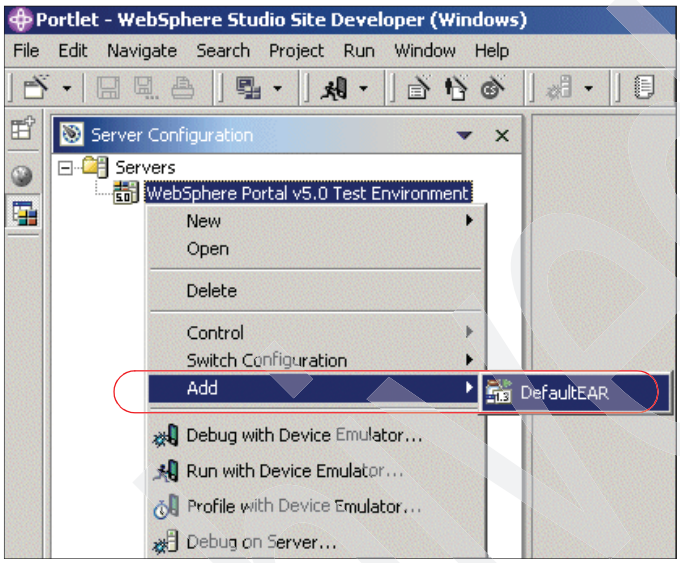


Figure 13-20 Adding a project to the Test Environment

2. If the Cloudscape sample database has not been populated, run the batch file to populate the test database to be used in this scenario. Click **c:\LabFiles\Cloudscape\CreateCloudTable.bat** to do this.

   Note: The sample scenario included in this chapter requires that you download the sample code available as additional materials. See Appendix C, “Additional material” on page 543.

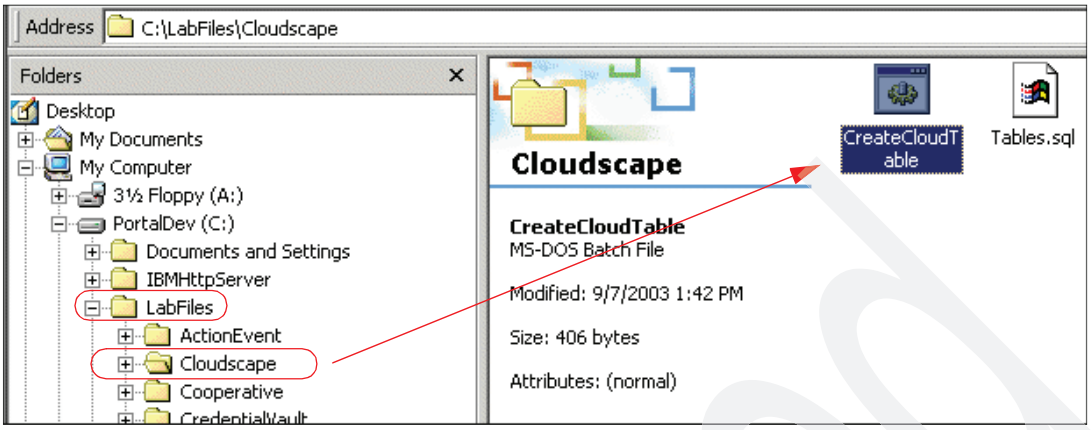


Figure 13-21 Populating test database

3. Next, click the **J2EE Navigator** tab to see your project again.
4. Right-click **HRPortlet**. Then click **Run on Server**. This will load your project into the Test Environment so that you can view it in the WebSphere Studio Site Developer internal Web browser. It may take a minute or two for this process to complete.
5. You will now see your newly created portlets project running in the Web browser.
6. Switch to Edit mode in HRPortlet (c2a source portlet).
7. Enter the following information and click **Submit**.
   - Database: `jdbc:db2j:C:\LabFiles\Cloudscape\WSSAMPLE`
   - User: `db2admin`
   - Password: `db2admin`
   - SQL: `select * from jobs`

Archived

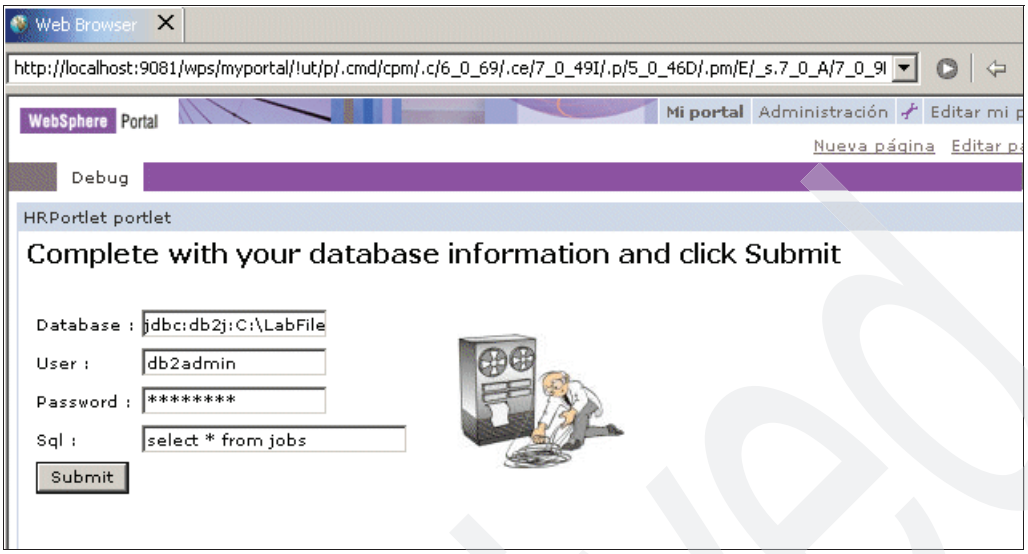


Figure 13-22 HRPortlet portlet in Edit mode

8. The HRPortlet now displays the jobs table, including a cooperative portlet menu in the DEPT_NO column. Before you can use this menu, you have to configure the data source in DepartmentDetailsPortlet.

Archived

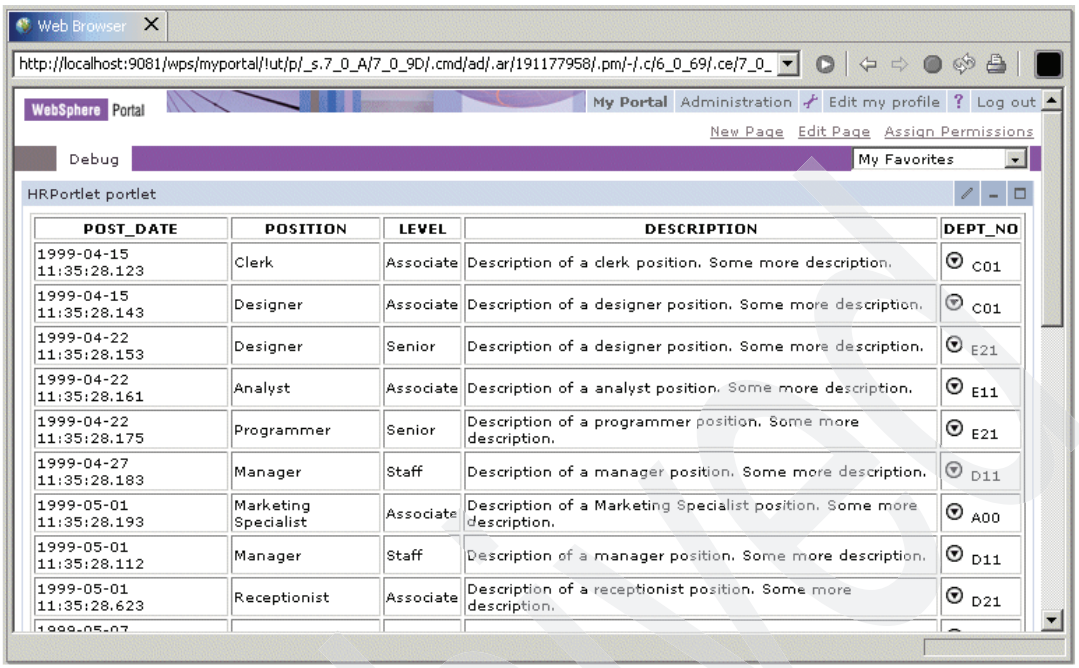


Figure 13-23 HRPortlet displays a cooperative menu in the DEPT_NO column

9. Switch to Edit mode in DepartmentDetailsPortlet (c2a target portlet).
10. Enter the following information and click **Submit**. It is not necessary to enter an SQL command here, because it is built during the processing of the cooperative menu.
    - Database: `jdbc:db2j:C:\LabFiles\Cloudscape\WSSAMPLE`
    - User: `db2admin`
    - Password: `db2admin`

    Note: There is no need to enter an SQL statement (optional).
11. In the source portlet View mode, click the **c2a** icon in the DEPT_NO column, for example before C01 or A00.
12. Click the **c2a** menu again to send the changed property to C2A.



Figure 13-24 Cooperative portlets

13.5.4 Wire portlets

In this section, you will wire the source and target portlets for C2A. Follow these steps:

1. Select one of the C2A menu items but this time press the **Ctrl** key during the selection.
2. A new dialog opens asking whether to `Automate the action in future`. Select **Yes**.

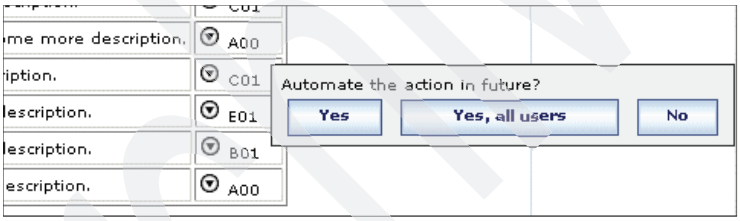


Figure 13-25 Pressing Ctrl key during menu selection to create a wire

3. Click **Yes** to activate the wire.
4. Try other departments again.
5. Press the **Ctrl** key again when using C2A to disable the wire.

Note: Wiring can also be accomplished by using the C2A wiring tool.

13.5.5 Enabling HRPortlet for programmatic source C2A

In this section, you will be required to enhance the source C2A portlet application to implement the programmatic approach. Figure 13-26 on page 441 illustrates the source cooperative portlet for this sample scenario.

Archived

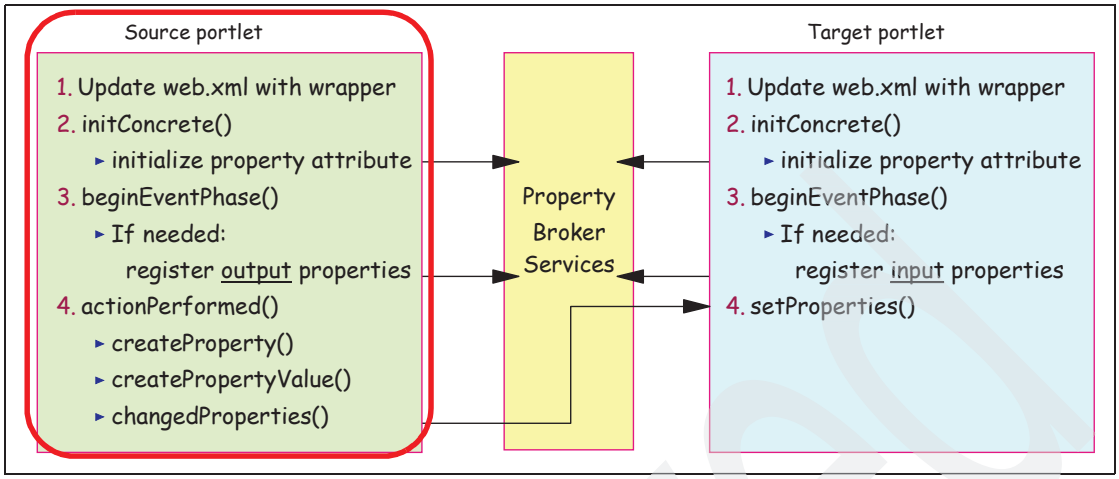


Figure 13-26 Cooperative portlets - programmatic sample scenario (source portlet)

You will update HRPortlet to change property data using the programmatic approach. You will also insert a button in the HRPortlet page to offer the display of department details in the DepartmentDetailsPortlet (C2A target portlet).

Notice that in this portlet, you will need to register the DEPT_NO property using the Property.OUT direction. In addition, the update of the property will be done in the actionPerformed method.

Note: For this scenario, you may want to copy and paste the code provided in the folder c:\LabFiles\AdvC2A\snippets\HRPortlet(source)\. The sample scenario included in this chapter requires that you download the sample code available as additional materials. See Appendix C, “Additional material” on page 543.

Proceed as follows to update the HRPortlet:

1. Open the file HRPortlet/Java Source/hrportlet/HRPortlet.java.
2. Update the class definition so it implements the *EventPhaseListener* interface:

   ```
   public class HRPortlet extends PortletAdapter implements
   EventPhaseListener, ActionListener {
   ```

   Note: The *beginEventPhase* method of the *EventPhaseListener* interface is used to register the output properties; this is because it is only possible to register and unregister properties during the event phase.
3. At the beginning of this class, insert two new class attributes so the code looks like this:

   ```
   public class HRPortlet extends PortletAdapter implements
   EventPhaseListener, ActionListener {
   ```

```
PropertyBrokerService pbService;
PortletConfig portletConfig;
```

Note: `pbService` is an interface to the property broker and `portletConfig` stores the portlet configuration.

4. Change the *init* method so it looks as follows:

```
   public void init(PortletConfig portletConfig) throws
UnavailableException {
      super.init(portletConfig);
      this.portletConfig=portletConfig;
   }
```

5. Add the following initConcrete method to initialize the property broker attribute.

Example 13-8 The initConcrete method initializes the property broker attribute

```
   public void initConcrete(PortletSettings settings)
      throws UnavailableException {
      try {
         pbService =
            (PropertyBrokerService) getPortletConfig()
               .getContext()
               .getService(
               PropertyBrokerService.class);
      } catch (PortletServiceUnavailableException e) {
         throw new UnavailableException("Could not locate
PropertyBrokerService.");
      } catch (PortletServiceNotFoundException e) {
         throw new UnavailableException("Could not locate
PropertyBrokerService.");
      }
   }
```

6. For simplicity, you will now add a new method called registerPropertiesIfNecessary to register the property we are interested in.

Example 13-9 Register a new output property

```
   private void registerPropertiesIfNecessary(PortletRequest request)
      throws PropertyBrokerServiceException {
      PortletSettings settings = request.getPortletSettings();
      Property[] properties = pbService.getProperties(request, settings);
      if (properties == null || properties.length == 0) {
         PortletContext context = getPortletConfig().getContext();
         //not registered, register now
         properties = new Property[1];
         properties[0] = PropertyFactory.createProperty(settings);
         properties[0].setName("DEPT_NOParam");
```


Archived

```
            properties[0].setDirection(Property.OUT);
            properties[0].setType("DEPT_NO");

properties[0].setNamespace("http://www.ibm.com/wps/c2a/examples/hrdetails");
            properties[0].setTitleKey("HRDetails.Department");
            properties[0].setDescriptionKey(
               "Display department details");
            pbService.registerProperties(request, settings, properties);
         }
      }
```

7. Implement the eventPhaseListener interface by inserting the *beginEventPhase* and *endEventPhase* methods, which are callback methods invoked during the event phase. The beginEventPhase invokes the method to register the output property DEPT_NOParam.

   Note: The endEventPhase method does nothing in this scenario but needs to be included in the interface.

Example 13-10 beginEventPhase and endEventPhase methods

```
public void beginEventPhase(PortletRequest request) {
      try {
         registerPropertiesIfNecessary(request);
      }
      catch (Throwable e) {
         e.printStackTrace();
      }
   }
public void endEventPhase(PortletRequest request) {
   }
```

8. For simplicity, insert a new method with name changeProperty to notify the broker about a changed property value. This method uses the *changedProperties* method to notify property changes.

Example 13-11 Notify the broker of a property value change

```
private void changeProperty(PortletRequest request, String value) {
      System.out.println("send data");
      PortletSettings settings = request.getPortletSettings();
      if (pbService != null) {
         try {
            Property p = PropertyFactory.createProperty(settings);
            p.setName("DEPT_NOParam");
            p.setDirection(Property.OUT);
            p.setType("DEPT_NO");
            p.setNamespace("http://www.ibm.com/wps/c2a/examples/hrdetails");
            PropertyValue[] pva = new PropertyValue[1];
```

Archive

```
            pva[0] = PropertyFactory.createPropertyValue(p, value);
            pbService.changedProperties(request, getPortletConfig(), pva);

         } catch (Exception e) {
            e.printStackTrace();
         }
      }
   }
```

9. At the end of the actionPerformed method, include the following code to invoke the internal changeProperty method and notify the output property change.

Example 13-12 Invoke method to notify the property value change

```
      if (actionString.equals("DisplayAllDepartmentDetails")) {
         this.changeProperty(request, "*");
      }
```

10. Right-click anywhere in the Java editor and select **Source -> Organize Imports** to include the missing import statements. In the Organize Imports dialog, choose to import the **com.ibm.wps.pb.property.Property** class.

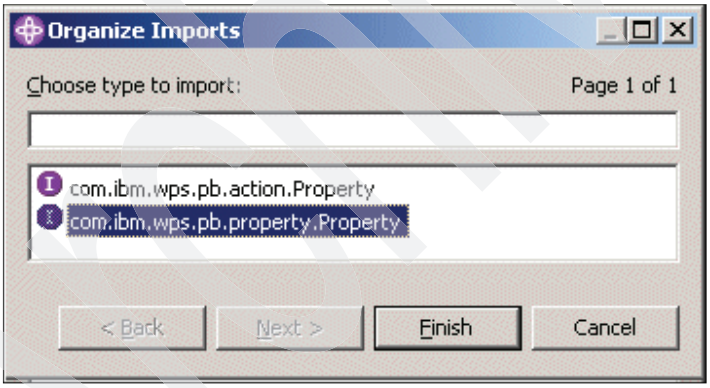


Figure 13-27 The Organize Imports dialog

11. Save and close the updated source cooperative portlet HRPortlet.java file.
12. Update the HRPortletView.jsp to insert a new action button offering to display department details from all departments. Open the HRPortletView.jsp file and insert the following code at the end of the file and before the </BODY> tag.

    Note: The action name is DisplayAllDepartmentDetails and it will be processed in the actionPerformed method. See Example 13-13 on page 445.

Example 13-13 New button to offer display of all department details

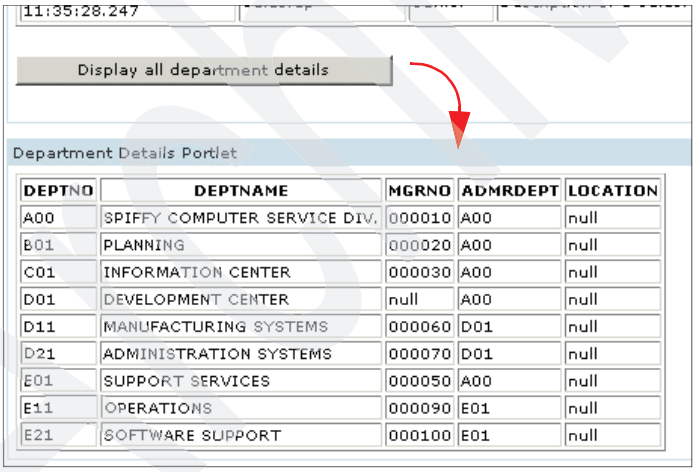
```
<p align="center">
<FORM method="post"
action="<portletAPI:createReturnURI><portletAPI:URIAction
name='DisplayAllDepartmentDetails'/></portletAPI:createReturnURI>"">
<INPUT type="submit" name="submit" value="Display all department details">
</FORM>
</p>
```

13. Optionally, preview the JSP and see the new button.
14. Save and close the file.

13.5.6 Running the programmatic source portlet

Follow these steps to run the updated scenario:

1. Right-click **HRPortlet** and select **Run on Server**.
2. Click the **Display all department details** button. The DepartmentDetailsPortlet displays all department details, as shown in Figure 13-28.



DEPTNO	DEPTNAME	MGRNO	ADMRDEPT	LOCATION
A00	SPIFFY COMPUTER SERVICE DIV.	000010	A00	null
B01	PLANNING	000020	A00	null
C01	INFORMATION CENTER	000030	A00	null
D01	DEVELOPMENT CENTER	null	A00	null
D11	MANUFACTURING SYSTEMS	000060	D01	null
D21	ADMINISTRATION SYSTEMS	000070	D01	null
E01	SUPPORT SERVICES	000050	A00	null
E11	OPERATIONS	000090	E01	null
E21	SOFTWARE SUPPORT	000100	E01	null

Figure 13-28 Department Details Portlet displays details from all departments

3. A portlet wire is required. Try the **Display all department details** button before and after adding the portlet wire.

Important: The method changedProperties() will only trigger events on wired targets. If no wires exist for the published properties, it has no effect.

Archived

14

Struts portlets

Struts is a very popular framework for Web applications using a Model-View-Controller (MVC) design pattern. The Struts framework is an open source subproject of the Apache Software Foundation's Jakarta project and can be used to effectively design Web applications.

This chapter discusses the Struts Portlet Framework, which adds support for writing Struts application that can be deployed in WebSphere Portal.

After reading this chapter, you will be able to:

- Understand the architecture of the Struts framework
- Know how to develop Struts application in WebSphere Studio
- Know how to transform a Struts Web application into a Struts portlet

© Copyright IBM Corp. 2004. All rights reserved.

14.1 Overview

The Struts framework divides your application into three functional areas.The *model* is the business logic, which in most cases involves access of data stores such as relational databases. The development team that handles the model may be expert at writing DB2 COBOL programs, or EJB entity beans, or some other technology appropriate for storing and manipulating enterprise data.

The *view* is the code that presents images and data on Web pages. The code comprises JSPs and the JavaBeans that store data for use by the JSPs. The *controller* is the code that determines the overall flow of events.

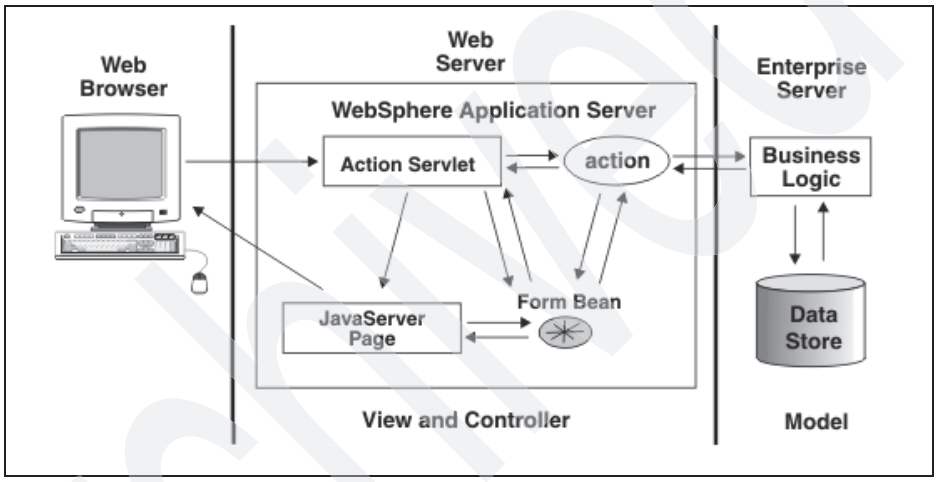


Figure 14-1 The three tiers of the Struts framework

Struts supports this model by providing the following components:

- The Struts action servlet handles run-time events in accordance with a set of rules that are provided at deployment time. Those rules are contained in a Struts configuration file (struts-config.xml) and specify how the servlet responds to every outcome received from the business logic. Changes to the flow of control require changes only to the configuration file.

  Struts also provides the Java class org.apache.struts.action.Action, which a Java developer subclasses to create an "action class". At runtime, the ActionServlet is said to "perform actions", which means that the servlet invokes the perform method of each of the instantiated action classes.

  Note: The object returned from the perform method directs ActionServlet as to what action or JSP to access next.

Archived

- Struts provides the Java class org.apache.struts.action.ActionForm, which a Java developer subclasses to create a form bean. At runtime, the bean is used in two ways: when a JSP prepares the related HTML form for display, the JSP accesses the bean, which holds values to be placed into the form. Those values are provided from business logic or from previous user input. When user input is returned from a Web browser, the bean validates and holds that input either for use by business logic or (if validation failed) for subsequent redisplay.
- Struts provides numerous, custom JSP tags which are simple to use but are powerful in the sense that they hide information. The page designer does not need to know much about form beans beyond, for example, the bean names and the names of each field in a given bean.

Note: For more detailed information, you can find numerous sources available about Struts. The Struts Web site offers resources for developers, including a User Guide, informative JavaDoc, and the source code. For more information about Struts, see the Struts Application Framework at:

`http://jakarta.apache.org/struts/`

14.1.1 The Struts portlet framework

Struts application can run within WebSphere Portal using the Struts portlet framework. Developers that have worked with Struts in the servlet environment should adapt easily to the Struts Portlet Framework. The packaging of a Struts portlet application is very similar to a Struts application in the servlet environment.
However, WebSphere Portal also introduces additional concepts, such as portlet modes, multiple device support, and portlet communication, which might need to be addressed by the Struts application. Because of this, developers have to consider the following main issues when developing Struts portlets:

- **Action processing and rendering**
  In a normal servlet application all servlet processing occurs during the service() method. The Struts rendering of the page is usually immediately preceded by action processing; they are essentially part of one step. Portlet processing, however, is implemented in two phases, an action phase and a rendering phase.

  When a Struts application is migrated to the portlet environment, some of the information that was available during the action phase, namely the request parameters, is no longer available during the rendering phase. Additionally, since rendering methods, such as doView(), can be called when the portlet page is refreshed without a new event occurring for that portlet, all information

required to render the page must be available every time that method is called.

A command pattern can be used to encapsulate the rendering of the view, and the information required during this rendering. The pattern is implemented using the IViewCommand interface.

- **URI construction**
  URIs are constructed differently for portlets than for servlets. The portlet creates the URI using the PortletResponse.createURI() method. The Struts Portlet Framework has modified the tags in Struts so that they create portal links.
- **Changes to the configuration files**
  To run a Struts Web application within WebSphere Portal, developers have to add new init parameters to the Web deployment descriptor (web.xml). The Struts configuration file (struts-config.xml) must also be changed to specify the WpsRequestProcessor as the controller

For more detailed information about the Struts portlet framework, see the Struts Portlet Framework chapter of the Portal Infocenter:

`http://publib.boulder.ibm.com/pvc/wp/500/ent/en/InfoCenter/index.html`

14.2 Developing Struts Web applications

WebSphere Studio provides numerous tools for developing Struts Web applications. For example, a visual assembly tool, the Web diagram editor, helps developers to design a Struts-based Web application by creating Struts artifacts and connecting these artifacts visually.
In this section you will develop a Struts Web application. This application adds a new book to a fictitious bookshop. Please note this is only a sample application which does not execute any real business logic.

To develop a Struts application proceed as follows:

1. Create a Struts Web application:
   a. Switch to the Web perspective.
   b. Select **File -> New -> Web -> Web Project**.
   c. Enter the project name of `WebBookStruts`.
   d. Select the Web Project features **Add Struts support**. This adds the necessary support for Struts into your Web project.
   e. Click **Next >**.
   f. In the J2EE Settings Page, click **Next >** again.

g. In the Struts settings page, select **Override default settings** and select the Struts version **1.1**.

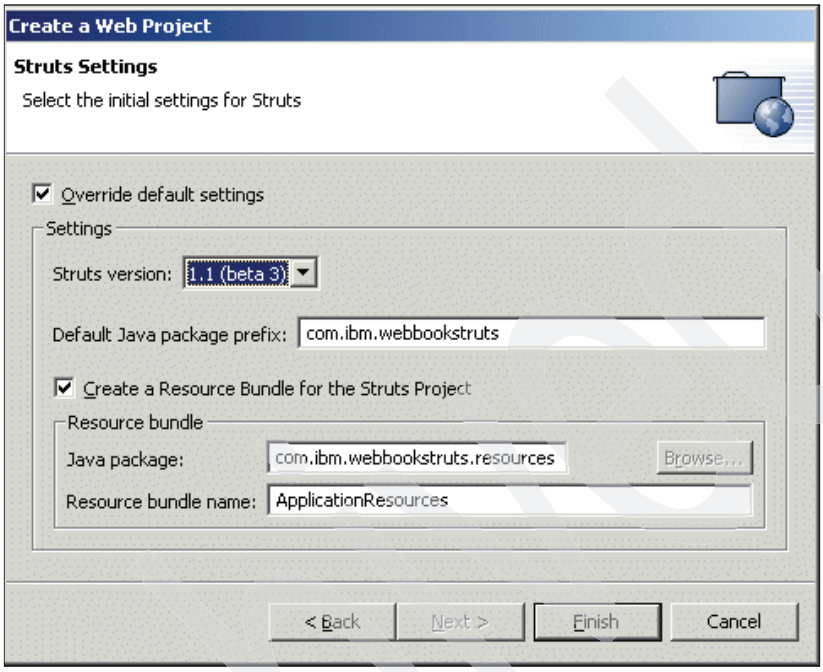


Figure 14-2 For Struts portlets you need Struts version 1.1

h. Click **Finish**.

2. Next, graphically lay out the Web application using the Web Diagram Editor:

   a. Select **File -> New -> Other -> Web -> Struts -> Web Diagram** to create a new Web diagram.

   b. Select the **WebBookStruts** Web project and enter the name `BookWeb.gph`. Click **Finish**. The Web diagram editor opens.

   c. Right-click anywhere in the editor and select **New -> Web Page Node**. Name the Web page node `addABook.jsp`.

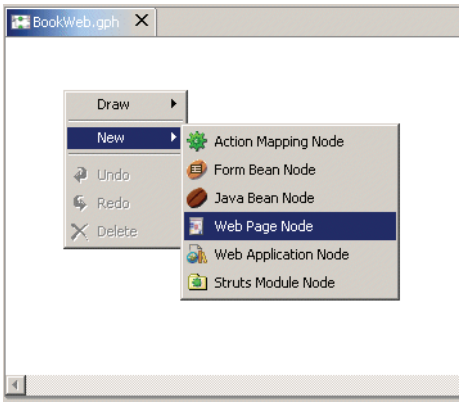


Figure 14-3 In the Web diagram editor, create a new Web page

d. Add two Web page nodes. Change their names to `addABookResponse.jsp` and `error.jsp`.

e. Select **New -> Action Mapping Node** to add a new action mapping node. Change its name to `addABook`.

f. Select **New -> Form Bean Node.** Name it `newBook` and set the scope to `request`.

g. Select `addABook.jsp` and choose Connection from the context menu. Draw a connection to `addABook`.

h. Connect `addABook` to `addABookResponse.jsp`. Label the connection `success`.

i. Connect `addABook` to `error.jsp`. Label the connection `failure`.

j. Connect `addABookResponse.jsp` to `addABook.jsp`.

k. Connect `error.jsp` to `addABook.jsp`.

l. Connect `newBook` to `addABook`.

m. Connect `addABook.jsp` to `newBook`. The final Web diagram should look as in Figure 14-4 on page 453.

Archived

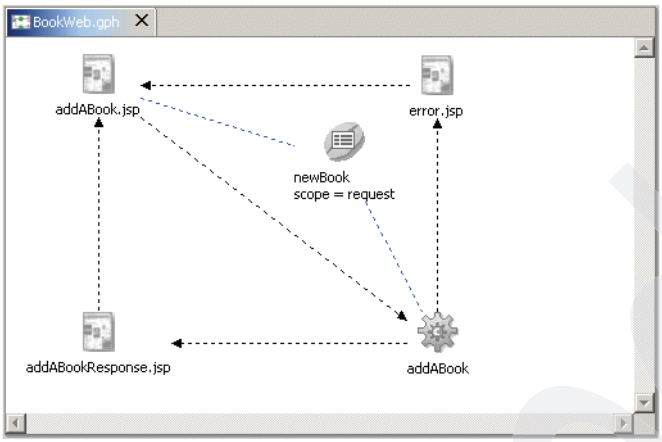


Figure 14-4 The final Web diagram of the book application.

n. Press **Ctrl-S** to save the Web diagram.

3. From the Web diagram, we create first the newBook action form class as follows:
   a. Double-click **newBook** to launch the New Action Form Class file wizard.
   b. On the first page, click **Next >**.
   c. On the Choose new accessors page, click **Next >** again.
   d. On the Create new accessors page, add four new accessors of type String: `title`, `isbn`, `author` and `category`.
   e. Add one accessor `price` of type java.mathBigDecimal.
   f. Click **Next >**, then **Finish**. Close the Java editor.
4. From the Web diagram, we can now create the JSP and complete its content as follows:
   a. Double-click `addABook.jsp` to launch the New JSP file wizard.
   b. Select **XHTML** as the Markup Language. Click **Next >** and on the second page **Next >** again.
   c. On the Page Directive Information Page, disable **Generate a Page Directive Information**.
   d. Click **Next >** three times.
   e. On the Form Field Selection page, select **newBookForm** as the form bean.
   f. By pressing the **shift** key, select all four fields of the bean.

Archived

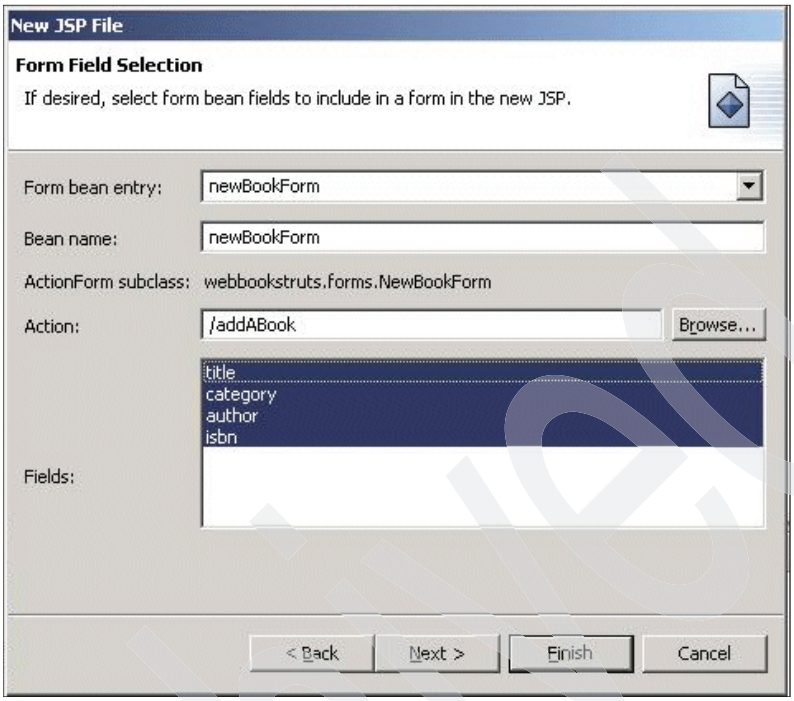


Figure 14-5 JSP File wizard to create an html form

g. Click **Next >** and **Finish**.Close the Web designer.

5. From the Web diagram, double-click the **error.jsp**. Select **XHTML** as the Markup language and disable the page directive generation. Click **Finish**.
6. In the Web page designer, enter an error message and close the Web designer.
7. From the Web diagram, double-click the **addABookResponse.jsp**. Select **XHTML** as the Markup language and disable the page directive generation. Click **Finish**.
8. In the Web page designer, enter a success message and close the Web designer.
9. Next, create the Struts Action class. The action class receives control from the input page using the input data stored in the form bean.
   a. Double-click the **addABook** action mapping to launch the New Action Mapping dialog.
   b. In the Form Bean Name field, select the **newBookForm** bean (see Figure 14-6 on page 455).

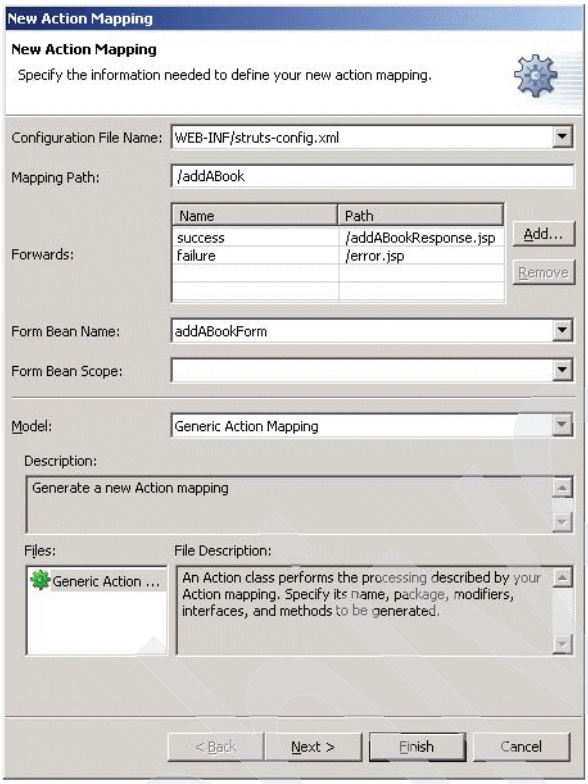


Figure 14-6 Create the action mapping

c. Click **Finish**. This creates the com.ibm.webbookstruts.actions.AddABookAction class.

d. Save the file and close the Java editor. Close the Web diagram editor.

10. To test the Struts Web application, we will create a server project and run the application on it.

a. In the J2EE Navigator view, select the **addABook.jsp** file and select **Run On Server** from the context menu. In the server selection dialog, select the **WebSphere Express 5.0 Test Environment**. Click **Finish**. The following page is displayed. Enter some values to add a new book.

b. From the Servers control panel, stop the server: select the server and click the **Stop** button.

Archived

14.3 Migrating Struts Web applications

The following list presents the steps you have to follow to migrate an existing Struts Web application into a Struts portlet application. Most of the changes only affect the configurations files.

1. Make sure the existing Struts application has been built as a Struts 1.1 application.
2. Update the Struts action so that it does not write to the response object. This is not supported in the Struts Portlet Framework because of WebSphere Portal's two phase processing.
3. The Web deployment descriptor must be updated to use the WpsStrutsPortlet as the servlet class instead of the ActionServlet.
4. The servlet mapping for Struts actions must be specified as the struts-servlet-mapping init parameter.Create a portlet.xml. The PortalStrutsExample.war can be used as an example.
5. Modify the struts-config.xml to specify the WpsRequestProcessor as the controller.
6. Modify tags that use pageContext.forward() to use the PortletApiUtils forward.
7. Modify JSPs that use a forward to use the logic forward tag.
8. Modify JSPs as necessary to use the Struts tags for creating URLs, like the Struts Link and Form tags.
9. The JAR files from the WEB_INF/lib directory of the PortalStrutsBlank.war must be used. These files are the Struts JARs and the required Struts Portlet Framework JARs.
10. The TLD files must be updated from the WEB_INF/lib directory of PortalStrutsBlank.war. Verify the taglib attributes and that the JSP correctly reference the TLD files. This has been a common source of problems when migrating existing applications.
11. The JSPs should be modified so they do not use html, head and body elements. All HTML output to the portal is written in the context of an HTML table cell.

To migrate the WebBookStruts Web application into a portlet application, proceed as follows.

1. In the Portlet perspective choose **File -> New -> Portlet Application Project** from the main menu.
2. In the Create a Portlet Project window, enter a project name of `StrutsBookPortlet` and check **Create empty portlet**. Click **Next >**.

3. In the J2EE Settings page, select a J2EE Level of **1.2**, check **Existing** enterprise application project and select **DefaultEAR**. Click **Finish**.
4. From the main menu, select **File -> Import...** Choose **WAR file** and click **Next >**. Browse to WSAD_INSTALL_DIR\runtimes\portal_v50\dev\struts\StrutsPortlet. Check **PortalStrutsBlank.war**. As the Web Project, select the existing Web project **StrutsBookPortlet**. Check **Overwrite existing resources without warning** and click **Finish**.

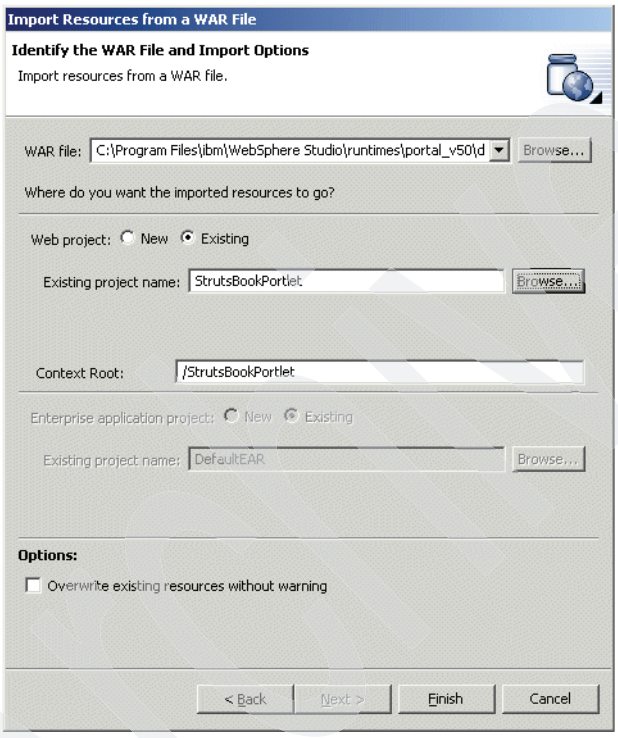


Figure 14-7 Import the PortletStrutsBlank.war file

5. Next, we copy all resources from the Web application to the portlet application.
   a. In the J2EE Navigator, select all packages from the WebBookStruts project and choose **Copy** from the context menu.

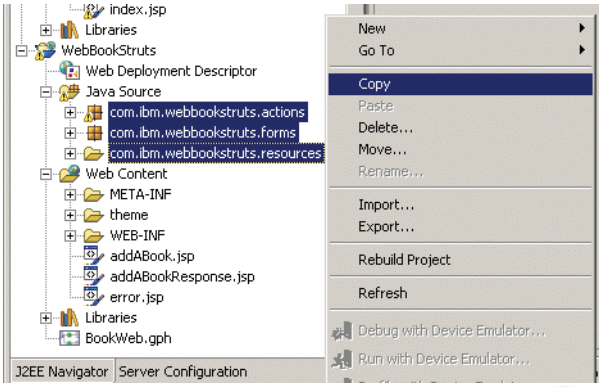


Figure 14-8 Copy the java files from the Web to the portlet project

b. Select the **Java Source** folder from the StrutsBookPortlet project and choose **Paste** from the context menu.

c. Copy all JSP from the WebBookStruts/Web Content folder to the StrutsBookPortlet/Web Content folder.

d. Delete index.jsp from the StrutsBookPortlet/Web Content folder.

e. In the StrutsBookPortlet/Web Content folder, rename addABook.jsp to `index.jsp`.

f. Open index.jsp. Change the lines:

```
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
```

to:

```
<%@ taglib uri="/WEB-INF/tld/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/tld/struts-bean.tld" prefix="bean" %>
```

g. Remove all tags between `<head>` and `</head>`, then remove all head and body tags.

h. Repeat steps f and g for the remaining two JSPs in StrutsBookPortlet.

i. Delete struts-config.xml from the StrutsBookPortlet/Web Content/WEB-INF folder.

j. Copy struts-config.xml from WebBookStruts/Web Content/WEB-INF folder to the StrutsBookPortlet/Web Content/WEB-INF folder.

k. Open StrutsBookPortlet/Web Content/WEB-INF/struts-config.xml. The Struts config editor opens. Switch to the XML Source tab.

Archived

l. Include the following code before the `message-resources` tag:

```
<!-- ============================ Controller Configuration -->
<controller
processorClass="com.ibm.wps.portlets.struts.WpsRequestProcessor">
</controller>
```

Note: Because WebSphere Portal does not support the back button of the Web browser, it can be difficult to navigate back through a number of screens to a location earlier in the browser history. To browse back from the addABook.Result.jsp or error.jsp, we have to add a homeFromStatic forward to ort action mapping which references the static content.

m. Switch to the Global Forwards tab of the Struts configuration editor. Click the **New** button. Enter a forward name of `homeFromStatic` and a path of `/index.jsp`.

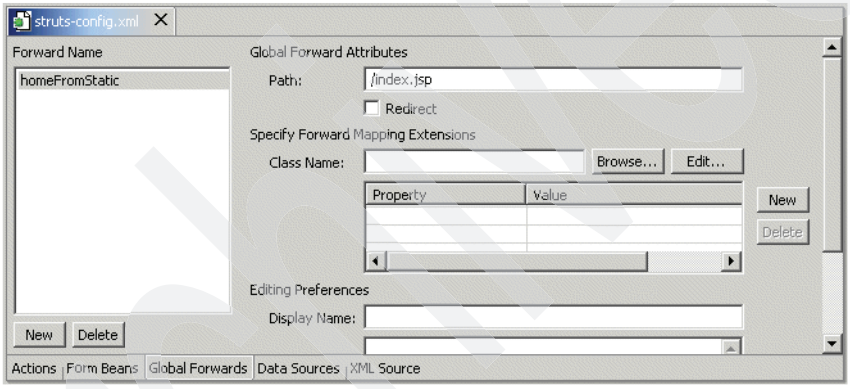


Figure 14-9 The Struts configuration editor

n. Open the addABookResponse.jsp file. Insert the following code snippet before the </html> tag.

```
<html:link forward="homeFromStatic">Back to index</html:link>
```

o. Repeat the last step for the error.jsp.

p. Save and close the struts-config.xml file.

Running the Struts portlet

Execute the following steps to run the Struts portlet:

1. To run a project in the WebSphere Studio Site Developer Test Environment, it is necessary to add the portlet project to the test environment. You can use a test server you created before or create a new server.

   a. Click the **Server Configuration tab** (on the navigator panel).

b. Expand the Servers tree.

c. Right-click **WebSphere Portal V5.0 Test Environment or your custom name...**.

d. Click **Add -> DefaultEAR** to add your project to the test environment.

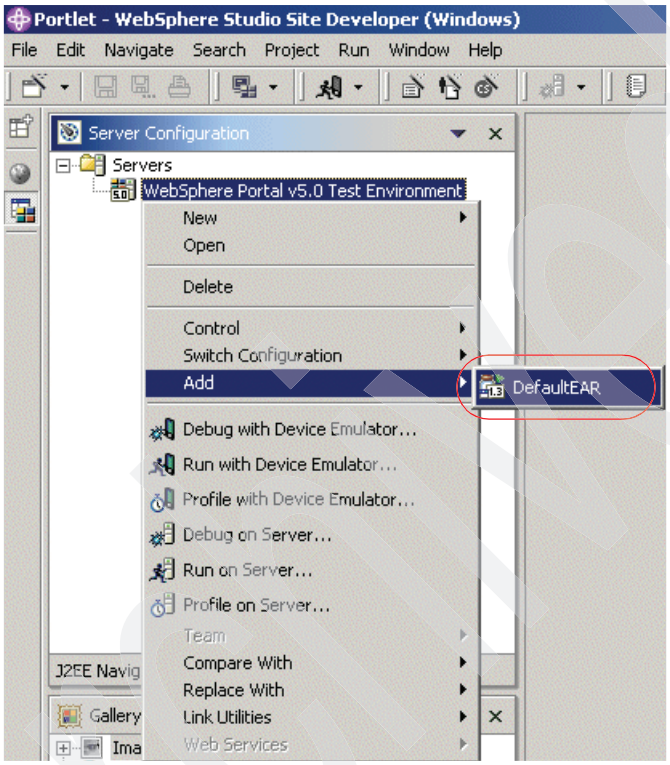


Figure 14-10 Adding the Struts portlet project to the test environment

e. Now click the **J2EE Navigator** tab to see your project again. Right-click **StrutsBookPortlet**. Then click **Run on Server**. If the Server Selection window opens, select your portal test environment and click **Finish**.

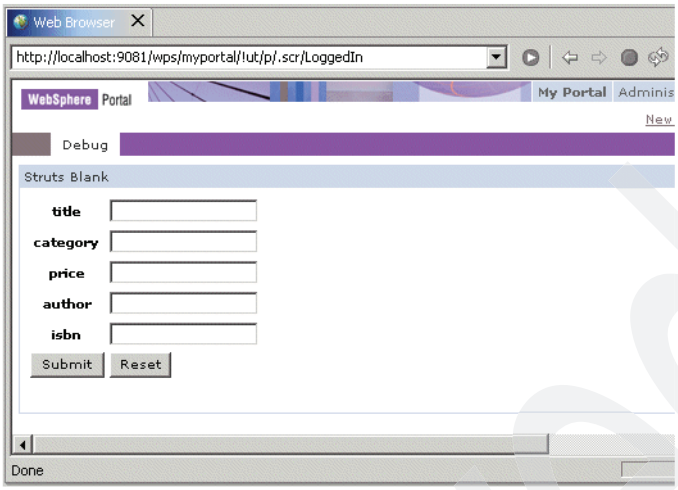


Figure 14-11 WebBook Struts application running within WebSphere Portal

When you press the **Submit** button, the Struts framework within WebSphere Portal executes the AddABook action and opens the addABookResponse.jsp or error.jsp.

Archived

Archived

15

Portlet preview

This chapter provides exercises to preview a portlet project using a remote WebSphere Portal server. This function does not require you to manually export and import files. The provided overview and sample scenario will allow you to understand the techniques and the configuration required to preview portlets.

Archived

© Copyright IBM Corp. 2004. All rights reserved.

15.1 Overview

During portlet development using the Portal V5 Toolkit, you can preview a portlet application running on WebSphere Portal on a local or remote machine. When you want to preview a portlet application that you have already developed or modified, select the provided **Portlet Preview** function.

In this environment, you do not have to send the rebuilt WAR file to the target runtime portal, or launch the browser. The Portlet Preview option automatically executes these tasks for you when you invoke this function. The Portlet Preview option also uninstalls and re-installs the portlet application for you by using the portal's XML configuration interface.

The development workstation and the remote Portal are shown in Figure 15-1.

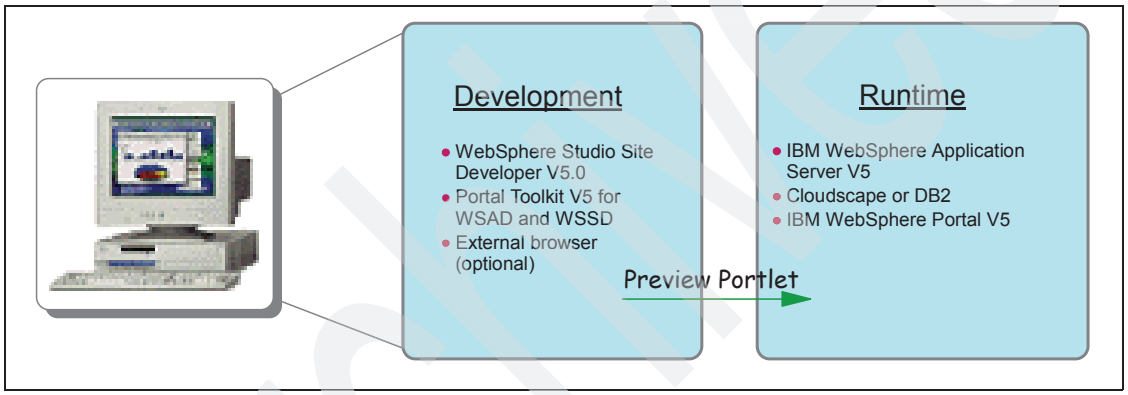


Figure 15-1 Development workstation

Note: The Portlet Preview is a different function from the preview page function which shows how a current page is likely to look when viewed with Microsoft Internet Explorer.

To preview a portlet, you need WebSphere Portal 5.0 or 4.2 externally running on a remote machine, your machine is already set with this scenario.The Portlet Preview does not support WebSphere Portal installed as the test environment. Instead, you can use Run on Server in the test environment to run the portlet easily; this is similar to the Portlet Preview method.

Multiple users can preview portlets since normal HTTP communication channel to a portal is used, and a single WebSphere Portal can support multiple users of the Portlet Preview. A preview page is generated for every user. Every user must use his/her own user ID.

The Portlet Preview action automatically executes the following tasks for you:

- Builds a selected portlet project.
- Exports it as a WAR file.
- Creates a preview place unless it already exists. The name is defined in the Portlet Preview preference.
- Creates a preview page for the user. The name is the same as the user ID.
- Installs or updates the portlet application into WebSphere Portal.
- Adds the portlet application to the preview page and activates the page.
- Sets the edit permission for the user.
- Launches a browser.
- Logs in to the portal server as the preview user and locates the preview page.

Temporary resources and configurations are generated when previewing a portlet. After previewing, you can delete them by executing the Reset Portlet Preview action, which automatically executes the following tasks for you.

- Deletes all portlet applications deployed by the user invoking the reset Portlet Preview function.
- Deletes the preview page.

Note: You need to delete the preview place (Portal V4.x) manually using Portal Administration.

15.1.1 Portlet Preview buttons available in the toolbar

Optionally, you may want to have the Preview Portlet and Reset Preview Portlet buttons available in the toolbar (see Figure 15-4 on page 467). If you want this option and it is not available, follow these steps to add them to the toolbar:

1. Select **Window -> Customize Perspective**.

Archived

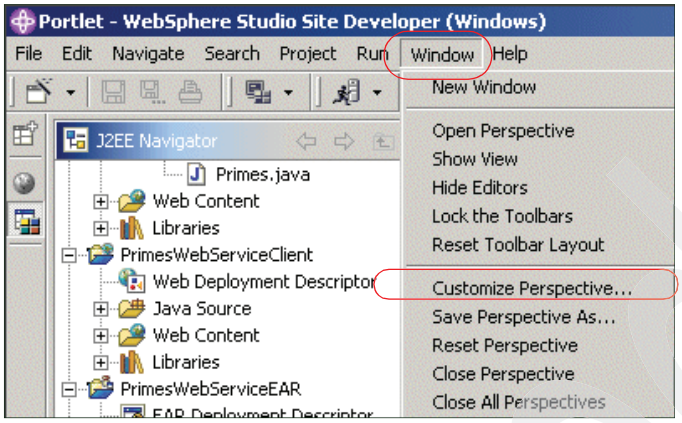


Figure 15-2 Customize portlet perspective

2. Select **Other** and check the **Preview Portlet** box to display items in the portlet perspective. Click **OK**.

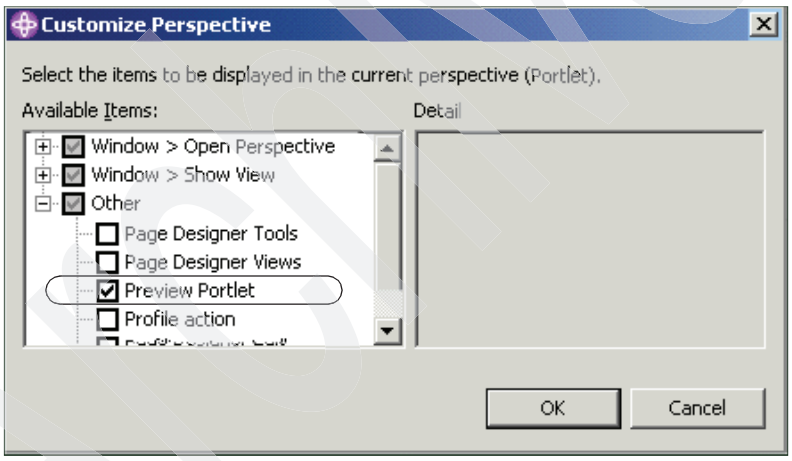


Figure 15-3 Customize Perspective - Other - Preview Portlet

3. Select the **Preview Portlet** drop-down list.

Archived

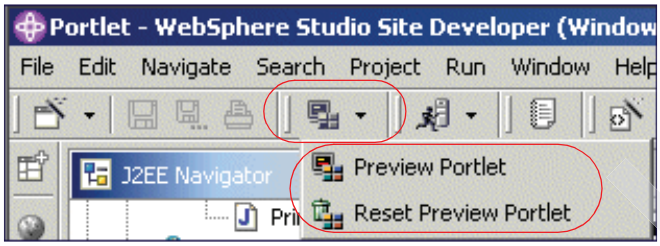


Figure 15-4 Preview Portlet available options

15.2 Sample scenario

This sample scenario illustrates how you configure and implement the Portlet Preview facility using WebSphere Studio Site Developer V5 and the Portal Toolkit V5.

15.2.1 Defining the Portlet Preview preference

The first task is to configure your Portlet Preview preferences. Start the remote Portal server and configure preferences in the Portlet Preview function. For example, execute these steps:

1. If required, start the IBM WebSphere Studio Application Developer; click **Start -> Programs -> IBM WebSphere Studio -> Site Developer 5.0**.
2. Stop any server running on the WebSphere Test Environment inside WebSphere Studio Site Developer. Right-click the started server and select **Stop**.

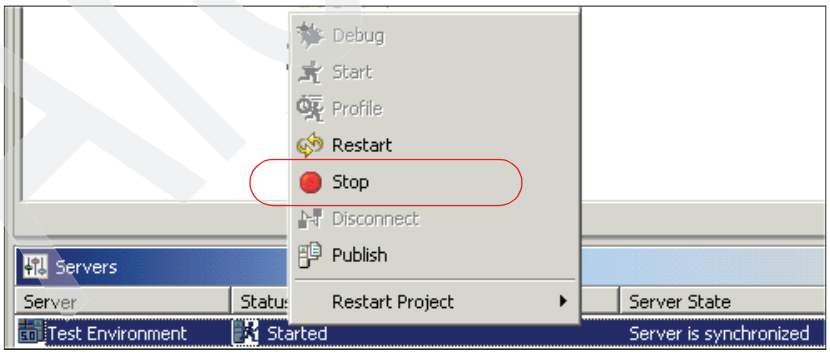


Figure 15-5 Stop servers

Archived

3. Start the external WebSphere Portal; click **Start -> Programs -> IBM WebSphere -> Portal Server V5.0 -> Start the Server**.
4. Wait a few minutes for the external Portal server to start.
5. Using the Preferences page, define the attributes used for previewing portlets. To define these preferences, from the Window menu select **Preferences -> Portlet Preview**.
6. Specify the following (see Figure 15-6 on page 470):
   a. **Host Address**: `http://portaldev.itso.ral.ibm.com:9081`

      Where `portaldev` is the fully-qualified host name of the machine where WebSphere Portal is running. For this sample scenario, the remote server has been configured to use port 9081.

      Note: You cannot use the WebSphere Portal installed for the test environment.

   b. **Base URI**: `/wps`

      The base URI for the portal, which is defined during WebSphere Portal installation. The default URI is /wps. Be sure you enter the one your server is using (default or customized value).

   c. **Default page**: `/portal`

      The home page. The default home page is `/portal`. Be sure you enter the one your server is using (default or customized value).

   d. **Version**: `5.0`

      Select the version of the Portal Server.

   e. **WebSphere Security**: uncheck

      Check whether WebSphere Security is enabled. This is not required in this sample scenario.

   f. **WebSphere Portal Administrator User ID**: provide the proper Portal server administrator user ID

      Note: The default ID is wpsadmin. This ID is used for administration tasks to prepare for previewing portlets.

   g. **WebSphere Portal Administrator Password**: provide the proper Portal server administrator password

   h. **WebSphere Portal Log In User ID**: provide the proper Portal server user ID to log in to Portal server and access the portlet

   i. **WebSphere Portal Log In Password**: provide the proper password

Archived

Note: The user must be created manually in the Portal Administration before previewing. The permissions will be automatically assigned to the user at the edit level. The user ID will also be used for the name of the preview page. To use a single WebSphere Portal for multiple users, you must use an owner specific user ID.

j. **Place for Preview (optional)**: leave blank

It is recommended to keep these fields blank.

i. **Name**:

The name of the place to be used for previewing. The default name is Preview. The place will be created automatically by the Portlet Preview if the specified place does not exist.

The portlet preview will set the user permission level to access the place for preview even if the place was already created and the current permission level is not adequate. The original permission level will be restored with the Reset Portlet Preview action.

ii. **Ordinal**:

The ordinal attribute which specifies the sorting order of portal resources. The default value is 20.You can specify where to insert the preview place among places.

Note: For information on the ordinal attributes, refer to "Special configuration data entries" in the InfoCenter of WebSphere Portal.

Archived

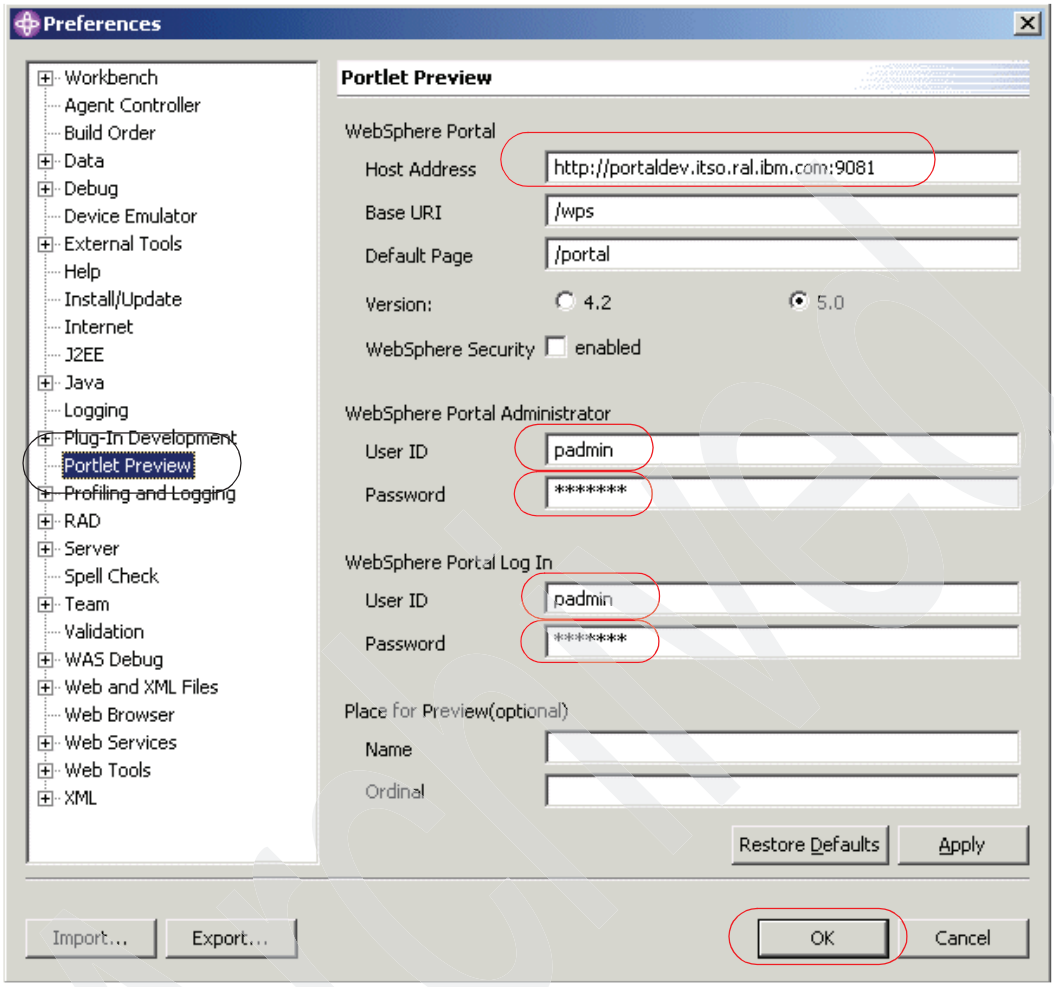


Figure 15-6 Portlet Preview configuration panel

7. Click **OK**.

15.2.2 Previewing the portlet

In this section, we show how to preview the portlet. You can preview a portlet application in the internal browser and also using an external browser. To preview a portlet application, follow these steps:

1. In the J2EE Navigator view, right-click the portlet project you want to preview, for example **HelloWorld**.
2. Select **Preview Portlet...** in the pop-up menu.

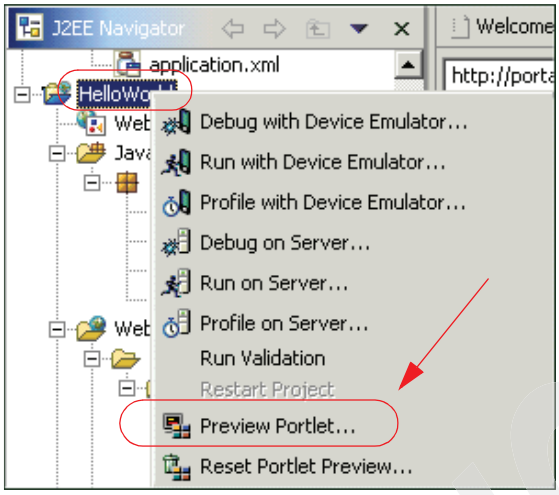


Figure 15-7 Invoking Preview Portlet option

Note: If any unsaved changes exist in the project, the Export Problems dialog appears. Click **Yes** to save the changes and export the project. Click **No** to export the project without saving them.

3. The configured browser will be launched (the default uses the internal browser) and the portlet will be previewed.

   Note: If the portlet does not immediately appear, click the page name (**padmin** page in this scenario).

Archived

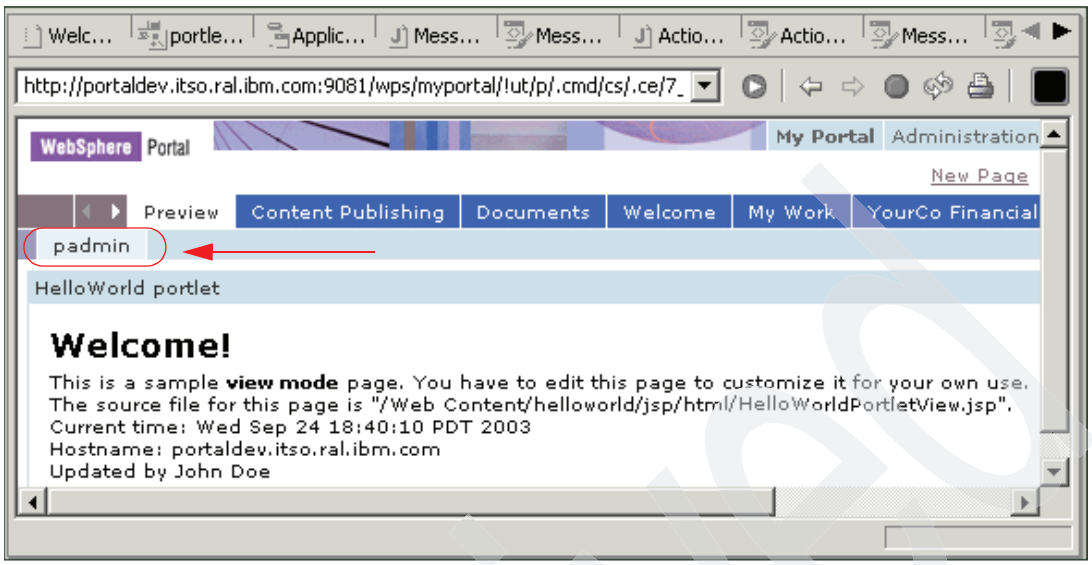


Figure 15-8 Portlet preview in internal browser

Note: Alternatively, you can preview the portlet by selecting the portlet project you want to preview, then clicking the **Preview Portlet** button in the toolbar.

Note also that if there are any problems in the portlet, it will not be previewed correctly and you will need to verify the Tasks view for errors.

4. Reset Portlet Preview (see Figure 15-7 on page 471) or use the proper option in toolbar. Temporary resources and the configuration of the portal server are generated to preview the portlet. You can delete them by performing the reset portlet preview action. For example, follow these steps:

   a. In the J2EE Navigator view, right-click the portlet project and select the **Reset Preview Portlet** in the pop-up menu.

   b. Alternatively, you can reset the portlet preview by selecting any portlet project, then clicking the toolbar **Preview Portlet** button and selecting **Reset Portlet Preview**.

      Note: The following tasks will be carried out by this action:

      i. In the portal server, all portlet applications previewed by the current user will be uninstalled.

      ii. In the portal server, the preview page of the current user will be deleted.

      iii. In your development machine, WAR files exported and previewed by the current user will be deleted.

Archived

5. Configure an external browser to be launched and preview the portlet. Select **Window -> Preferences**.

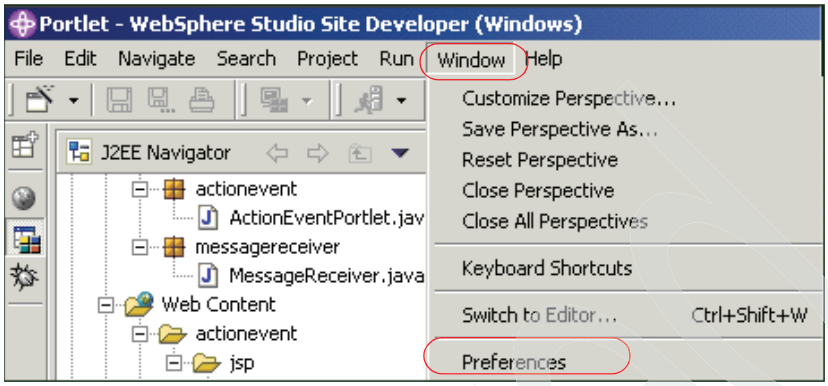


Figure 15-9 Preferences

6. Select **Web Browser** and check the box **Use external Web Browser**.
7. Configure the external browser location by browsing to the Internet Explorer location, for example: c:\Program Files\Internet Explorer\IEXPLORE.EXE.

Archive

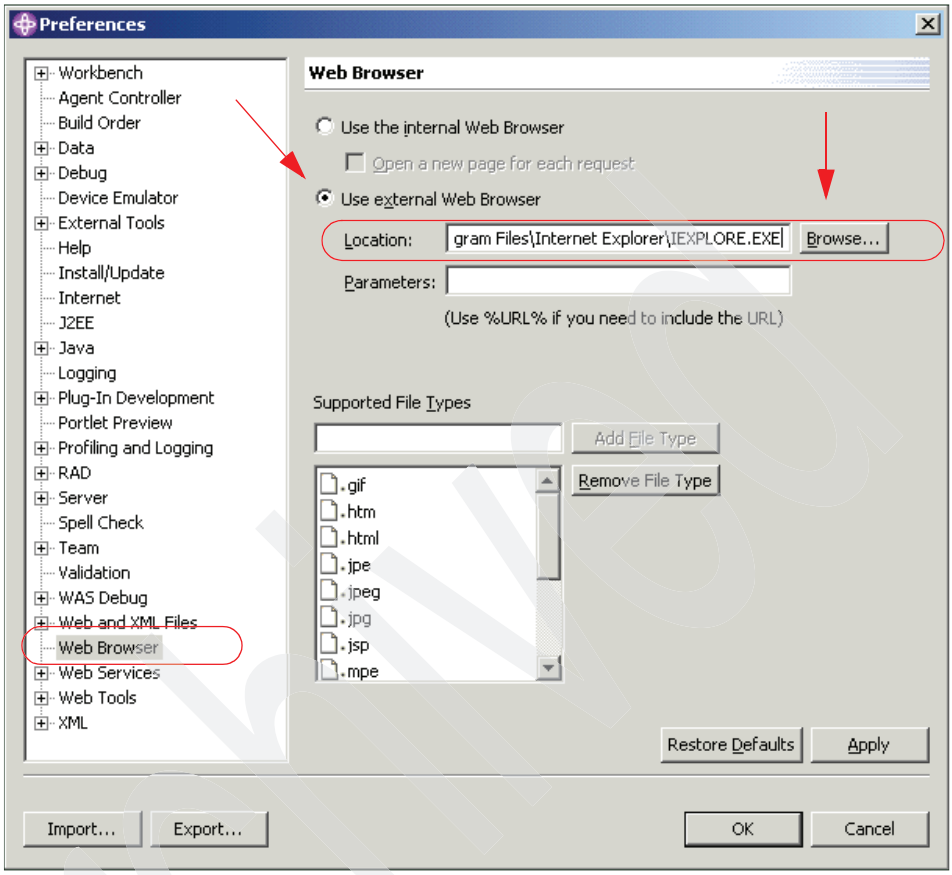


Figure 15-10 External Web browser configuration

8. Select the portlet application and click **Preview Portlet** (see Figure 15-7 on page 471) or use the option in the toolbar. This will bring up Internet Explorer to log in to the remote portal, deploy and run your selected portlet application. Click the page name if you do not see the portlet.

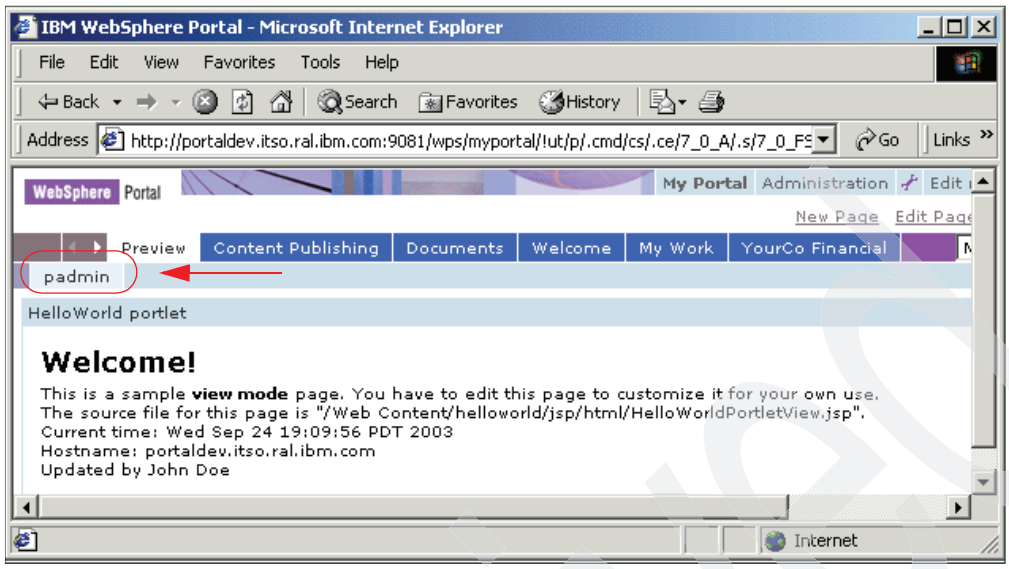


Figure 15-11 Portlet preview in external browser (IE)

Archived

Archived

16

Remote Server Attach

This chapter provides an overview and a sample scenario to test and debug portlet projects running on a remote WebSphere Portal server. The topics presented in this chapter will allow you to understand the techniques and the configuration required to test and debug portlets remotely.

The sample development workstation and the remote Portal server are illustrated in Figure 16-1.

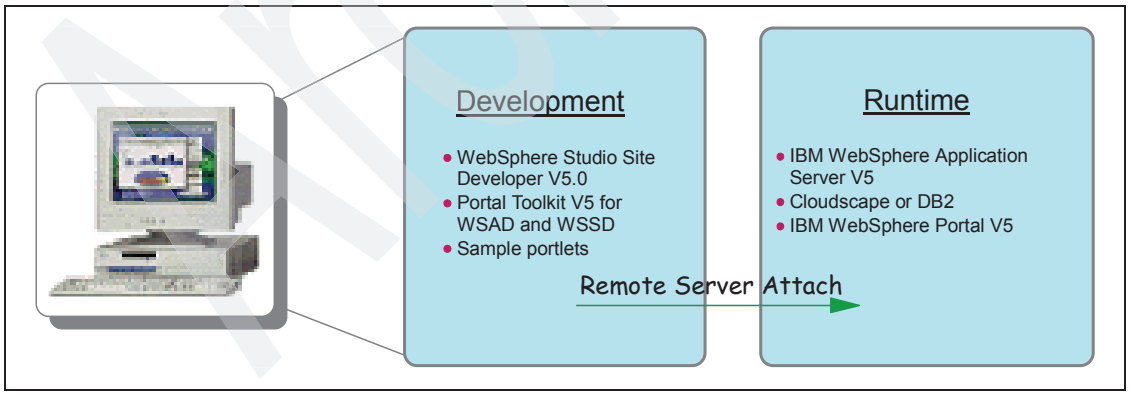


Figure 16-1 Development workstation

© Copyright IBM Corp. 2004. All rights reserved.

16.1 Overview

You can run and optionally debug a portlet application running on a remote WebSphere Portal system accessible through a network connection. The recommended steps to implement this function are as follows:

1. Prepare Portal server for Remote Server Attach.
   a. Install WebSphere Portal on a remote machine. In this scenario, it has already been installed for you.
   b. Configure JVM in the WebSphere Application Server Administrative Console.
2. Configure WebSphere Studio for Remote Server Attach.
   a. Select **WebSphere Portal 5.0 Remote Server Attach**.
   b. Specify the Host (Portal server), JVM debug port, HTTP port.
3. If needed, install a portlet application in Portal for Remote Server Attach.
   a. Build and export a portlet application project as a WAR file.
   b. Manually install and configure the portlet application (WAR file) using Portal Administration functions.
   c. Add the portlet to a Portal page using Portal Administration functions.
4. Run and debug the portlet application using Remote Server Attach.
   a. Select the portlet application project.
   b. Select **Debug on Server** from the context menu.
   c. Debug the portlet running in the remote Portal server.

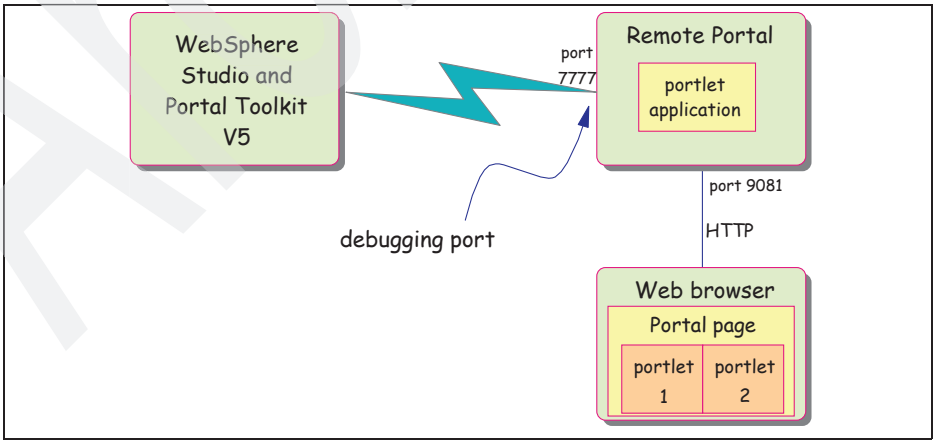


Figure 16-2 Remote Server Attach sample scenario

Archived

16.2 Preparing Portal for Remote Server Attach

In order to test and debug portlets remotely, some configuration is required.

1. First, stop any servers running on WebSphere Test Environment inside Studio.
2. Start the external WebSphere Application Server; click **Start -> Programs -> IBM WebSphere -> Application Server v5.0 -> Start the Server**.
3. Once WebSphere Application Server has started, open the Administrative Console. Click **Start -> Programs -> IBM WebSphere -> Application Server v5.0 -> Administrative Console**.
4. Log on to the Administrative Console with user ID wsadmin. Click **OK**.

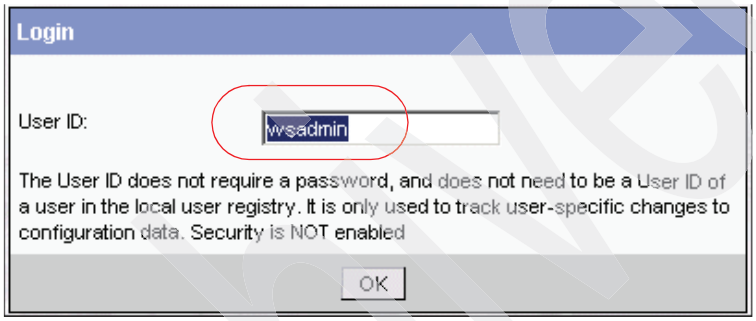


Figure 16-3 WebSphere Application Server Administrative Console Login

5. Click **No** if you get the auto complete option message. It is not needed for this scenario.
6. Expand Servers and select **Application Servers -> WebSphere_Portal**.

Archived

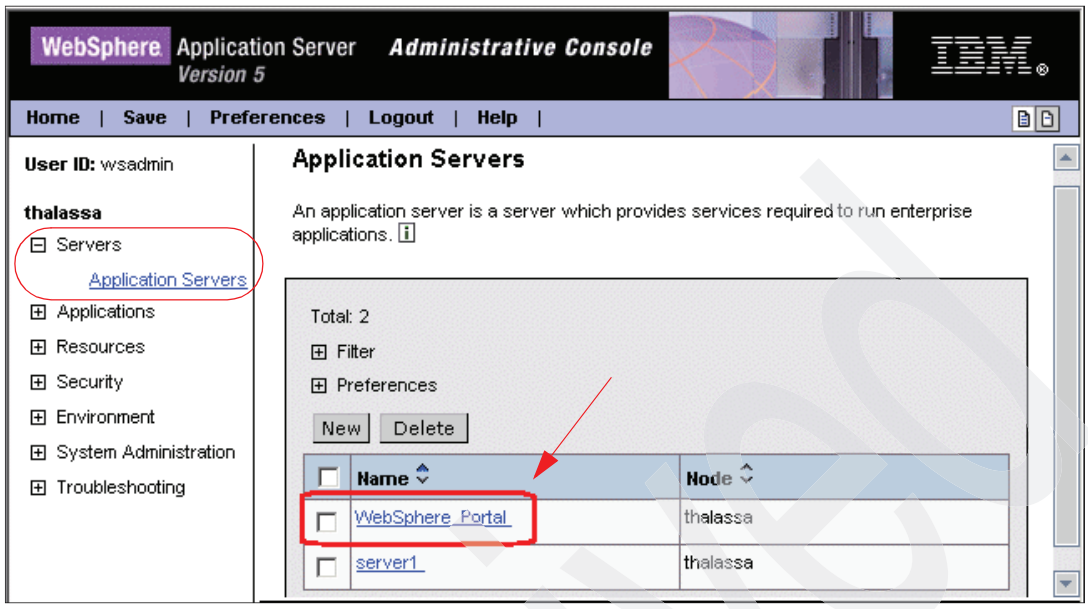


Figure 16-4 Select WebSphere_Portal

7. In Additional Properties (Configuration page), scroll down and select **Process Definition**.

Archived

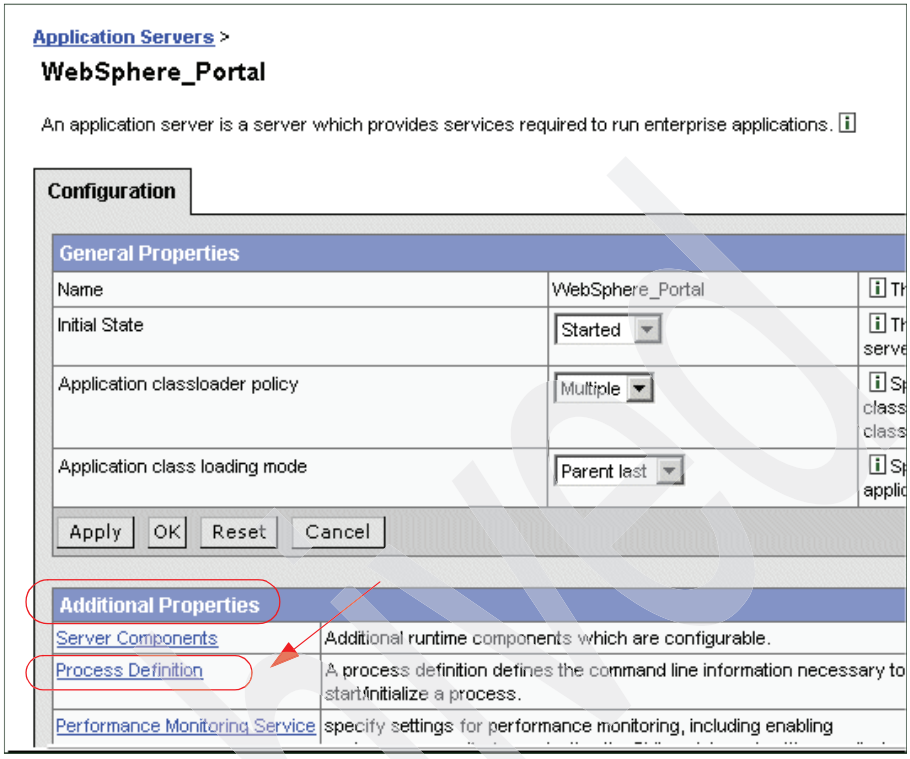


Figure 16-5 Select Process Definition

8. Select **Java Virtual Machine** (Additional Properties).

Archive

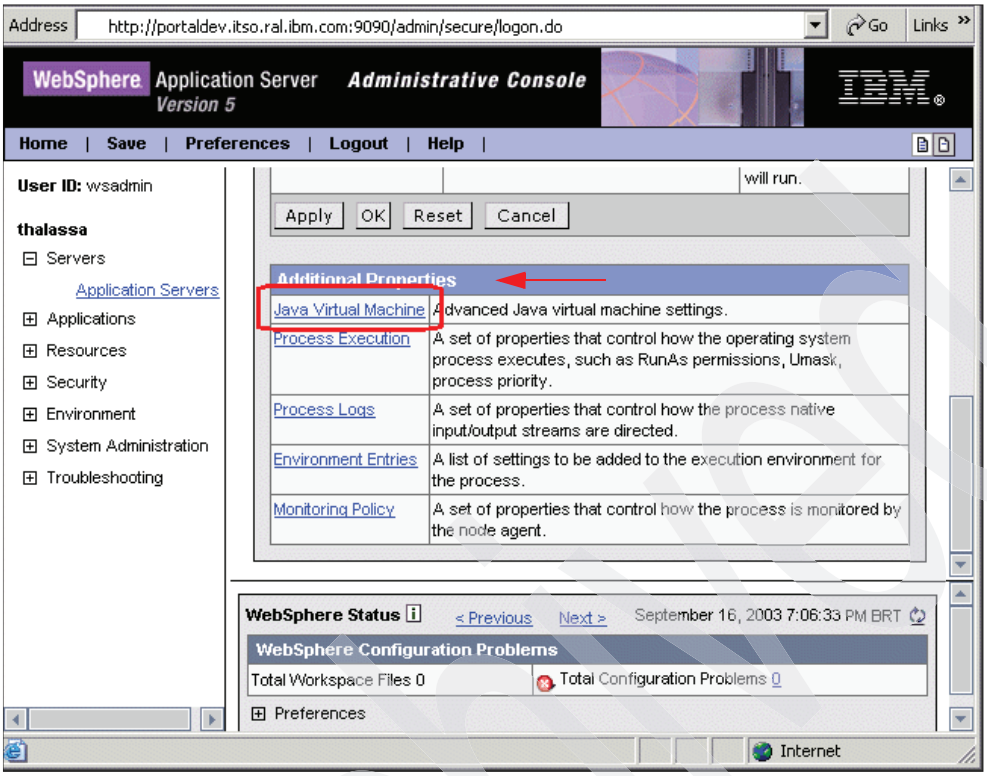


Figure 16-6 Java Virtual Machine

9. In General Properties:
   a. Select the **Debug Mode** checkbox.
   b. Verify that the following string is defined as the Debug arguments:

      ```
      -Djava.compiler=NONE -Xdebug -Xnoagent
      -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=7777
      ```

      Note: The address specifies the JVM debug port number; the default value is 7777.
   c. Click **Apply**.

Archive

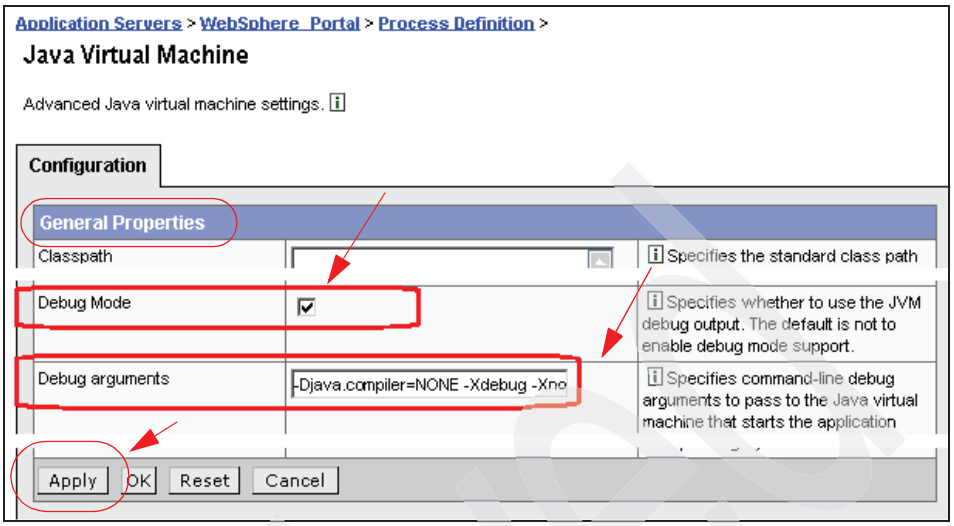


Figure 16-7 JVM parameter enabling debug

10. In the menu bar, click **Save.** Also, click **Save** in Save Master Configuration.

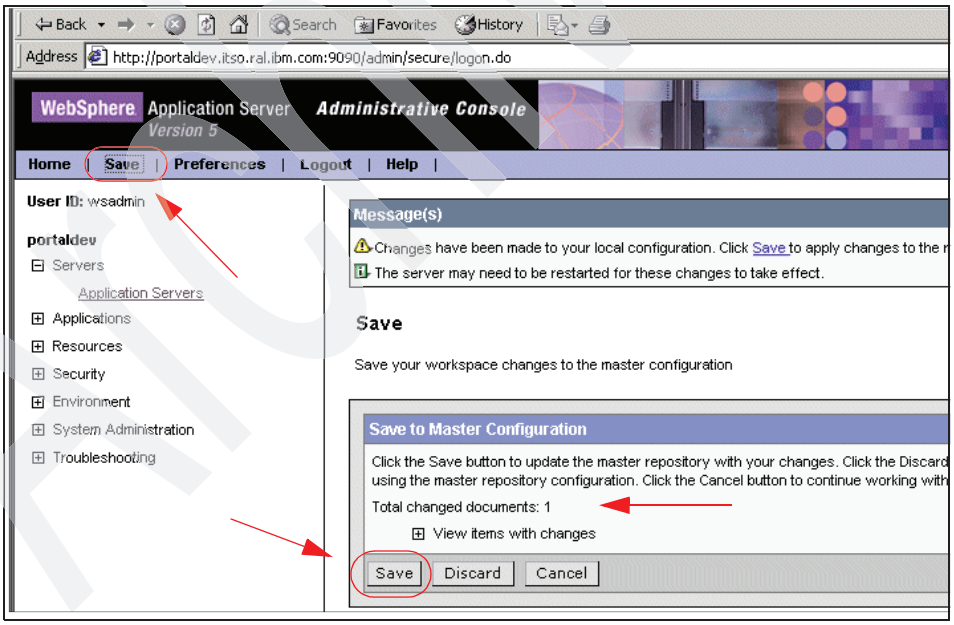


Figure 16-8 Save the JVM configuration

11. Stop the Application Server by clicking **Programs -> IBM WebSphere -> Application Server -> Stop the Server.**

12. Close the desktop browser.

16.3 Remote Server Attach configuration

In this section of the lab, you will start the Portal server and configure WebSphere Studio to connect to the remote Portal server.

1. Start WebSphere Portal by clicking **Programs -> IBM WebSphere -> Portal Server -> Start the Server**. Wait a few minutes for Portal to be started (open for e-business message).
2. If needed, start the WebSphere Studio Site Developer by selecting **Start -> Programs -> IBM WebSphere Studio -> Site Developer 5.0**.
3. Create a new server and server configuration by selecting the **Server Configuration** tab on the left pane and **Server and Server Configuration** on the right side.

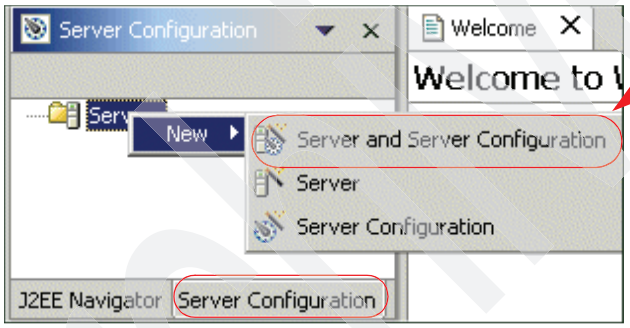


Figure 16-9 Server and Server Configuration

4. A new window will be displayed to allow you to define the new server and server configuration.
   a. Enter for example `Test Remote Server Attach` as the server name.
   b. Select **WebSphere Portal version 5.0 -> Remote Server Attach** as the server type.
   c. Click **Next** to continue.

Abstract

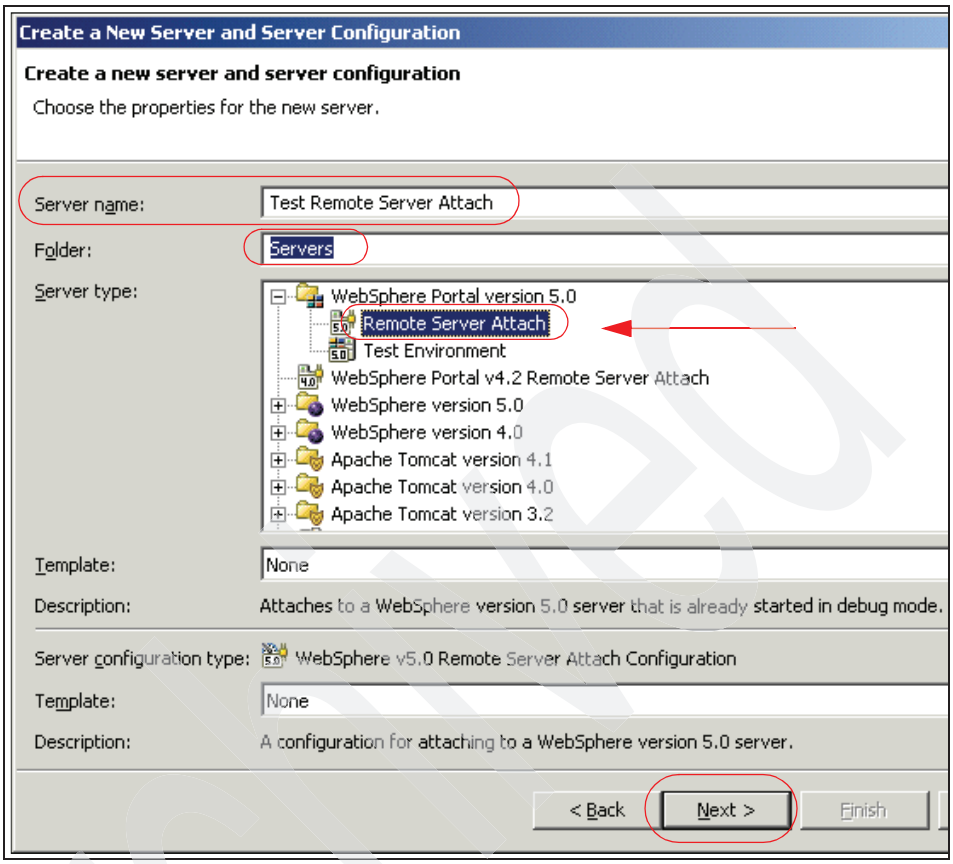


Figure 16-10 Server configuration settings window

5. In the Remote Attach Server Instance Page; enter as host: `portaldev.itso.ral.ibm.com` and click **Next.**

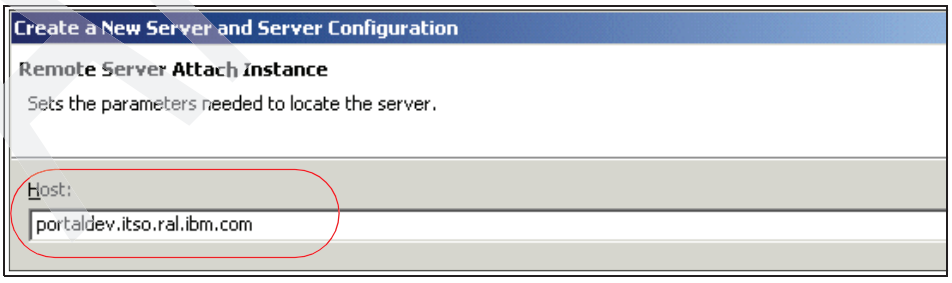


Figure 16-11 Host Name of the remote machine running Portal

6. In the *WebSphere v5.0 Remote Server Attach Configuration* configure the following:

   a. JVM debug port: 7777

      The debug port number provided by the server JVM.

   b. HTTP port: use port 9081 (default value is 9080)

      The transport port to access the remote Portal server.

   c. Enable JavaScript debugging: do not check

      Select this checkbox if you want to enable JavaScript debugging. It is not needed for this lab.

   d. BSF debug port: disabled

      This is the port used to attach to the Bean Scripting Framework Manager to enable JavaScript debugging.

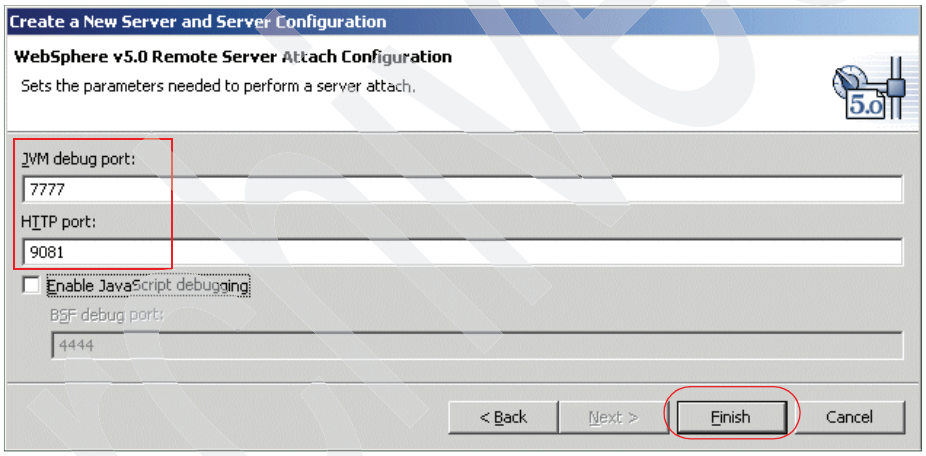


Figure 16-12 Configurations for debugging

7. Click **Finish**.

16.4 Installing a portlet in Remote Portal

When the portlet is not available in the remote Portal server, you will need to install it using the administration tools. In this section of the lab, you will export a portlet application (HelloWorld) from WebSphere Studio and manually install it in the remote Portal server.

Exporting the portlet application (WAR file)

The HelloWorld portlet will be used for this scenario. Follow these steps to export the portlet project and create a WAR file:

1. In the J2EE Navigator panel, right-click the **HelloWorld** portlet project and select **Export**.
2. Select **WAR file**.
3. For example, provide the project name `HelloWorld` and the target location as `c:\LabFiles\RemoteAttach`.

   Note: You can also download the sample code available as additional materials. See Appendix C, “Additional material” on page 543.

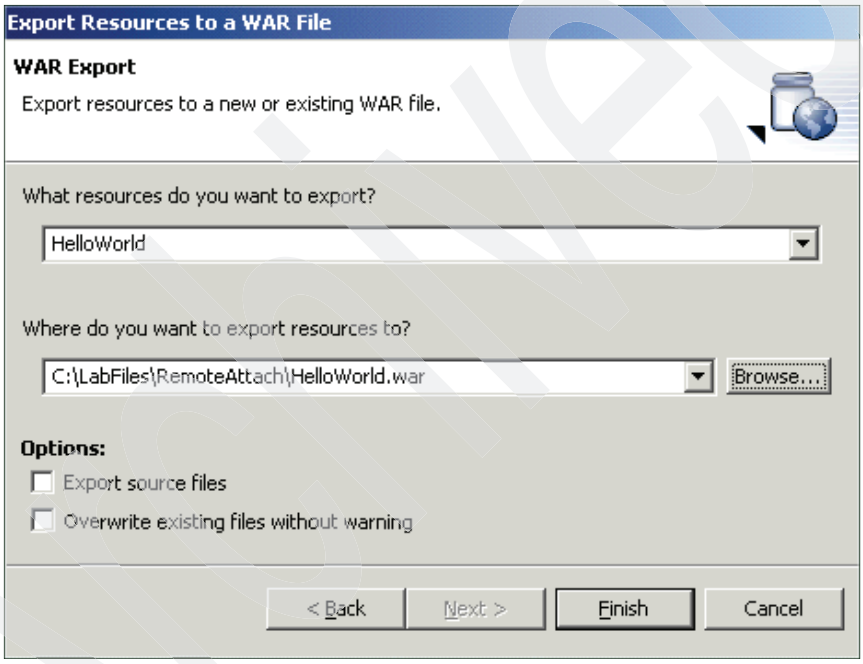


Figure 16-13 Exporting HelloWorld

Installing the portlet in Portal using Portal Administration

In this section, we provide a sample administration procedure to create a new page, install a portlet and add this portlet to the created page.

1. Start the external Web browser (IE) and connect to the portal:

   `http://portaldev.itso.ral.ibm.com:9081/wps/portal`

2. Log in to the portal as the administrator (user ID = `padmin` and password = `itsOlab`).

Note: Password uses the number 0.

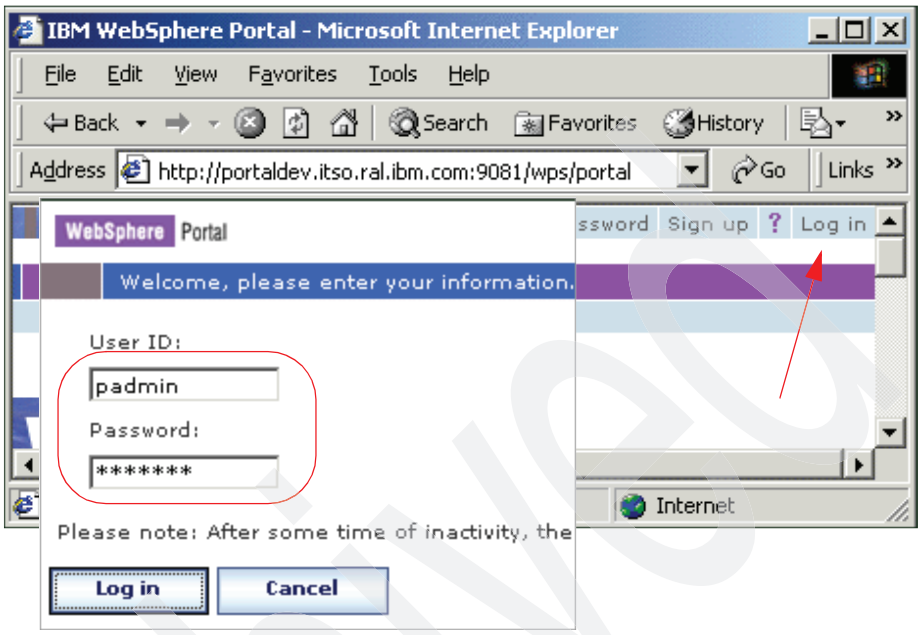


Figure 16-14 Login to WebSphere Portal

3. Click the **Administration** link.



Figure 16-15 Administration link

4. Choose **Portal User Interface -> Manage Pages**.

Archive

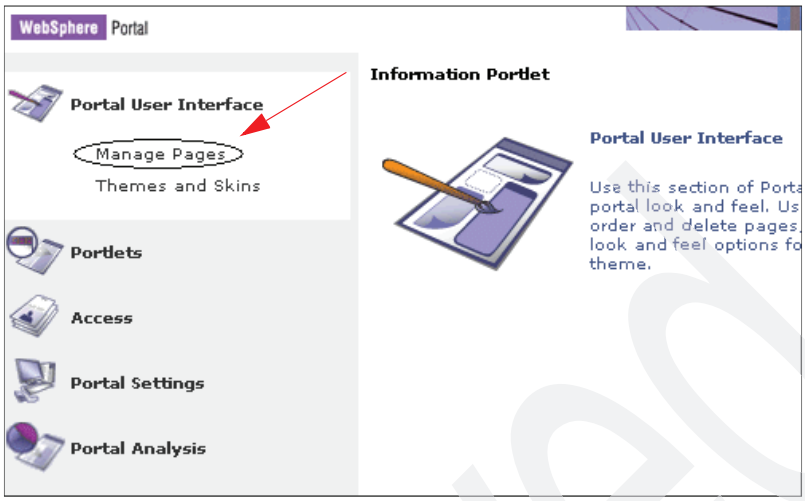


Figure 16-16 Manage Pages option

5. Click the **My Portal** label.

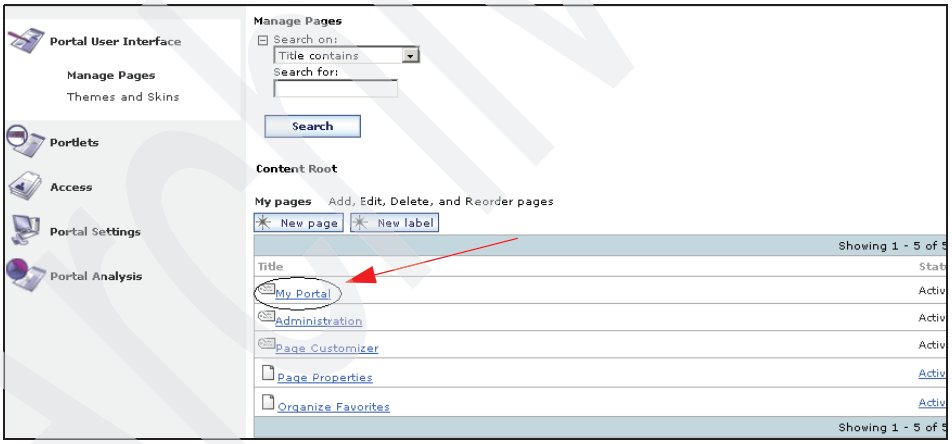


Figure 16-17 My Portal Label

6. Click **New label** to create a new label.

Archived

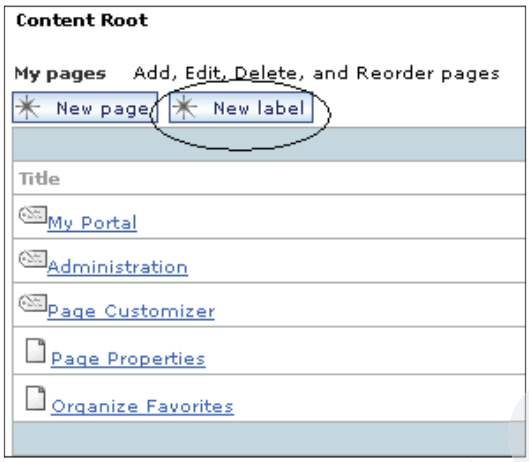


Figure 16-18 Select New Label

7. On the Create Label page, enter this information:
   - Place Name: `My Label`
   - Theme: `Science` or any other theme of your preference.
   - Supported markup: `HTML`. Click the **+** sign to see this option in Advanced Options.

     Click **OK** to create the new label (My Label).

Archived

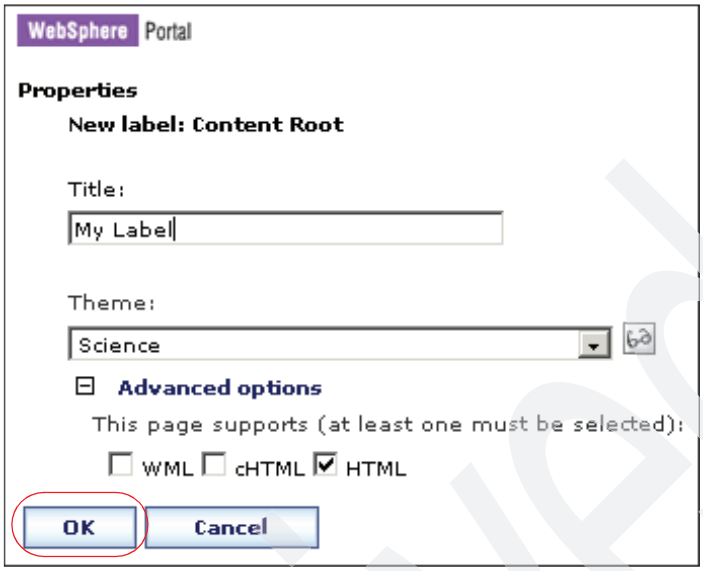


Figure 16-19 Create label

8. You will see a message indicating that the label was successfully created. Click **OK**.



Figure 16-20 My Label successfully created

9. Click the **My Label** label.

Archived

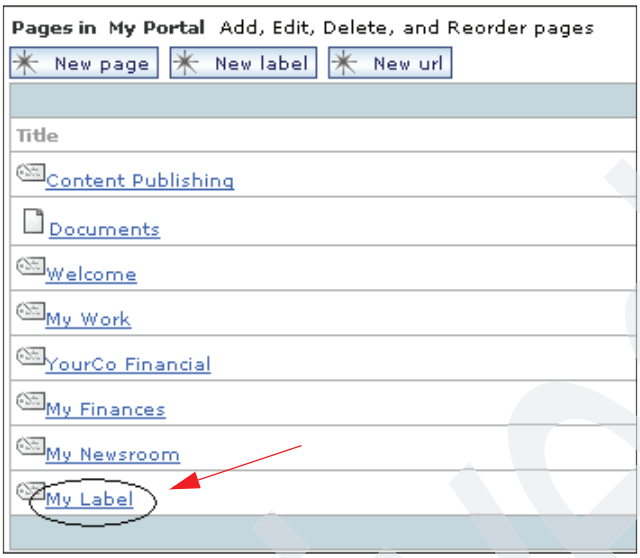


Figure 16-21 Select My Label

10. Click **New Page** to create a new page. The new page will be created inside My Label.



Figure 16-22 New Page creation

11. On the Create Label page, enter this information:
    - Title: `New Page`
    - Layout: two column portal page. Click the **+** plus sign to see the Advanced Options.
    - Click **OK** to create the page.

Archived

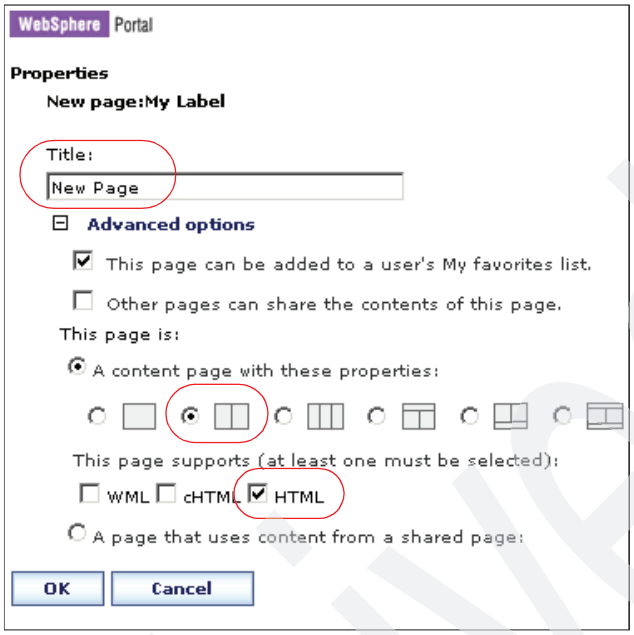


Figure 16-23 Create new page

12. Verify that the page has been successfully created. Click **OK**.

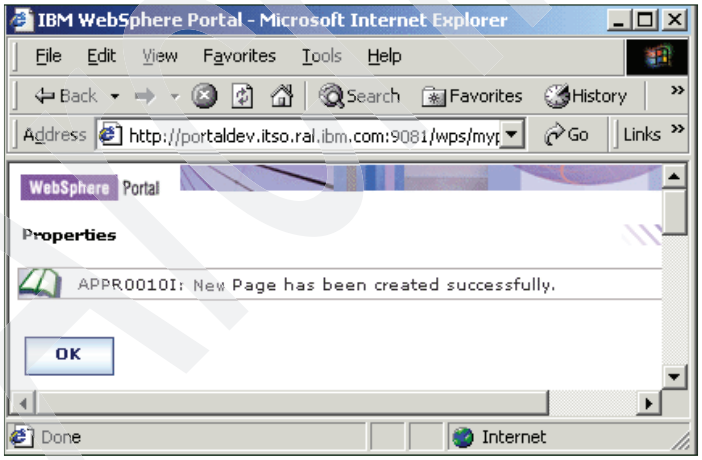


Figure 16-24 New page has been created

Archived

13. You will now install and add the portlet to the page. Navigate to the link **Administration -> Portlets -> Install**. Do the following:

    a. Click **Browse**.

    b. Select the directory **c:\LabFiles\RemoteAttach\HelloWorld.war** and click **Open** to select the file.

    Note: You can also download the sample code available as additional materials. See Appendix C, “Additional material” on page 543.

    c. Click **Next** to continue.

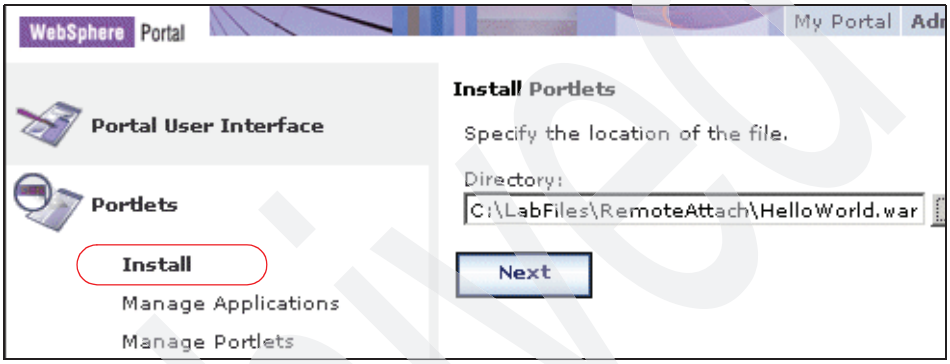


Figure 16-25 Select directory

14. Click **Try Again** if you get the `no Internet access` message. Internet access is not needed.

15. Verify that you are installing the HelloWorld portlet and click **Install** to continue the portlet installation. This step may take some time.

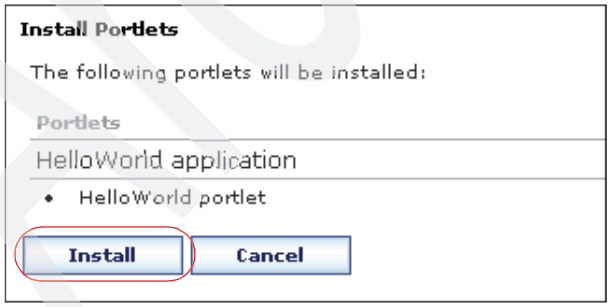


Figure 16-26 Portlet install

16. Once the installation is complete, the message `Portlets were successfully installed` appears at the bottom of the page.

Archived



Figure 16-27 Successful install message

17. To add the Hello World portlet to **My Label -> New Page**, do the following:
    a. Navigate to **Administration -> Portal User Interface -> Manage pages**.
    b. Select the labels **My Portal -> My Label**.
    c. You should see the page New Page (inside **My Portal -> My Label**).
    d. Click the Edit Page Layout icon (looks like a pencil) on the right of New Page.

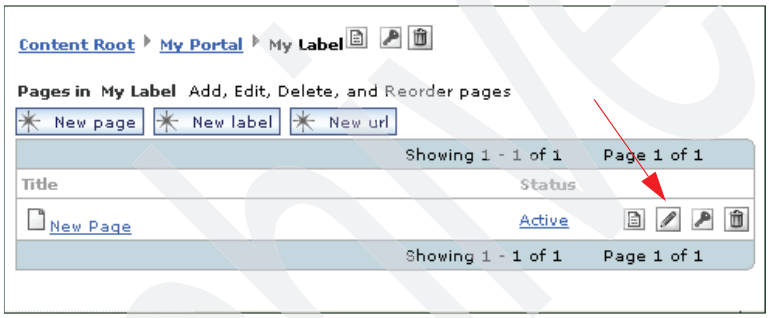


Figure 16-28 Edit Page Layout

    e. Click **Add portlets** on the left column in the portal page.

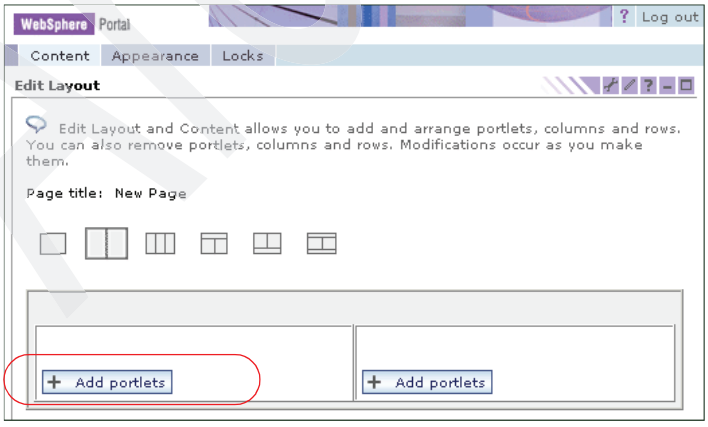


Figure 16-29 Add portlets

Archived

f. Enter `HelloWorld` in the *Search for* field and click **Search**. The Hello World portlet should appear in the result list.

g. Click the checkbox on the left of the HelloWorld portlet to select the portlet.

h. Click **OK**.

i. Click **Done**.

18. Navigate to **My Portal -> My Label -> New Page** to verify that the portlet appears on the page.

.



Figure 16-30 New Page with installed HelloWorld portlet

16.5 Running the portlet

Once the portlet has been installed in WebSphere Portal, you will need to connect to the server in Debug mode. If required, you can also trace the execution of the portlet program by using debugging facilities such as breakpoints and so on.

1. To run a project using the Remote Server Attach, you will need to add the portlet project to the remote server.
   a. Click the **Server Configuration** tab (on the navigator panel).
   b. Expand the Servers tree.
   c. Right-click **Test Remote Server Attach.**
   d. Click **Add -> DefaultEAR** (or any other project that you want to run) to add your project to the remote server.
2. In the Servers view, right-click the server and select **Debug** to connect to the remote server.

Archived

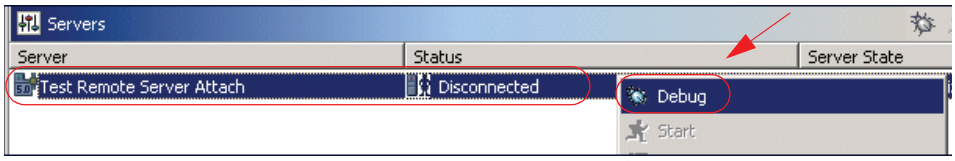


Figure 16-31 Debugging remotely

3. The server status should change to `Connected in debug mode` and the server will be ready for portlet debugging using the Debug perspective.

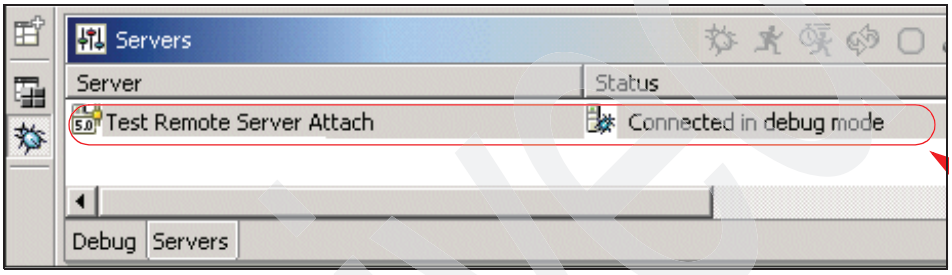


Figure 16-32 Connected in Debug mode

4. As an example, set a breakpoint in the doConfigure() method in the HelloWorld portlet (see Figure 16-33). That is, right-click the selected statement and click **Add Breakpoint**.

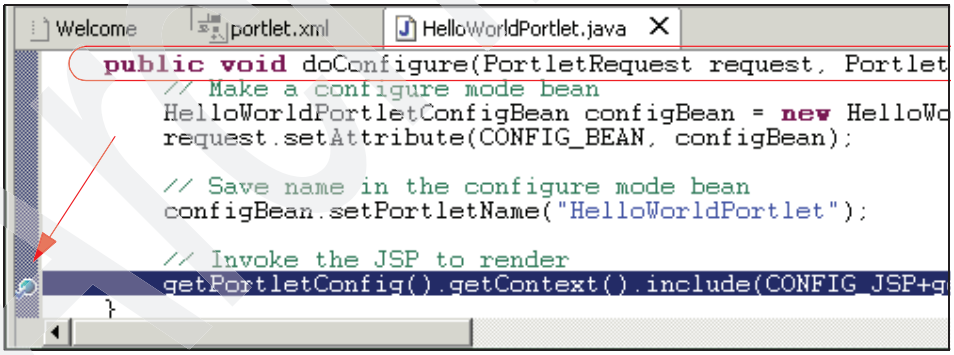


Figure 16-33 Setting a breakpoint in doConfigure() method

5. Start debugging the portlet application by launching a new external browser (IE) and log in to WebSphere Portal. For example, for this scenario use the following values:

   a. `http://portaldev.itso.ral.ibm.com:9081/wps/portal`

   b. User ID is `padmin` and password is `itso1ab`.

6. Navigate to execute the portlet: **My Portal -> My Label -> New Page**.
7. Initially, you will receive the Step-by-Step Debug window. At this time, you may want to skip the doGet method with the request as well as disable the step-by-step mode.

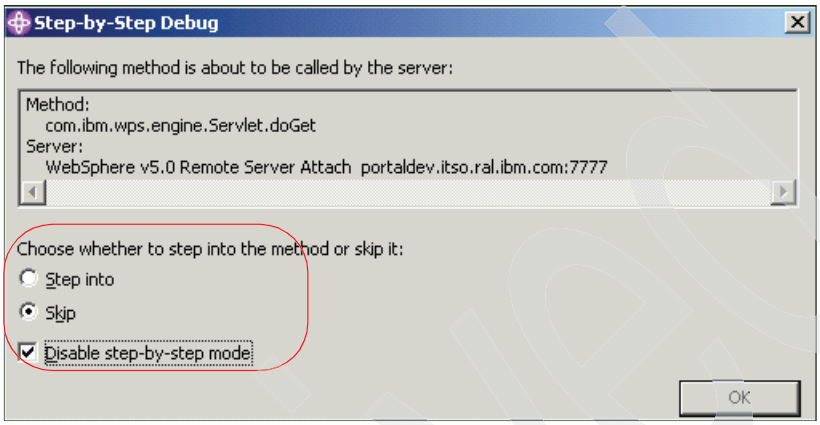


Figure 16-34 Step-by-Step Debug window

8. When a breakpoint is reached, you will see a window similar to Figure 16-35 on page 499. It shows the breakpoint and you can also inspect your variables.

Archive

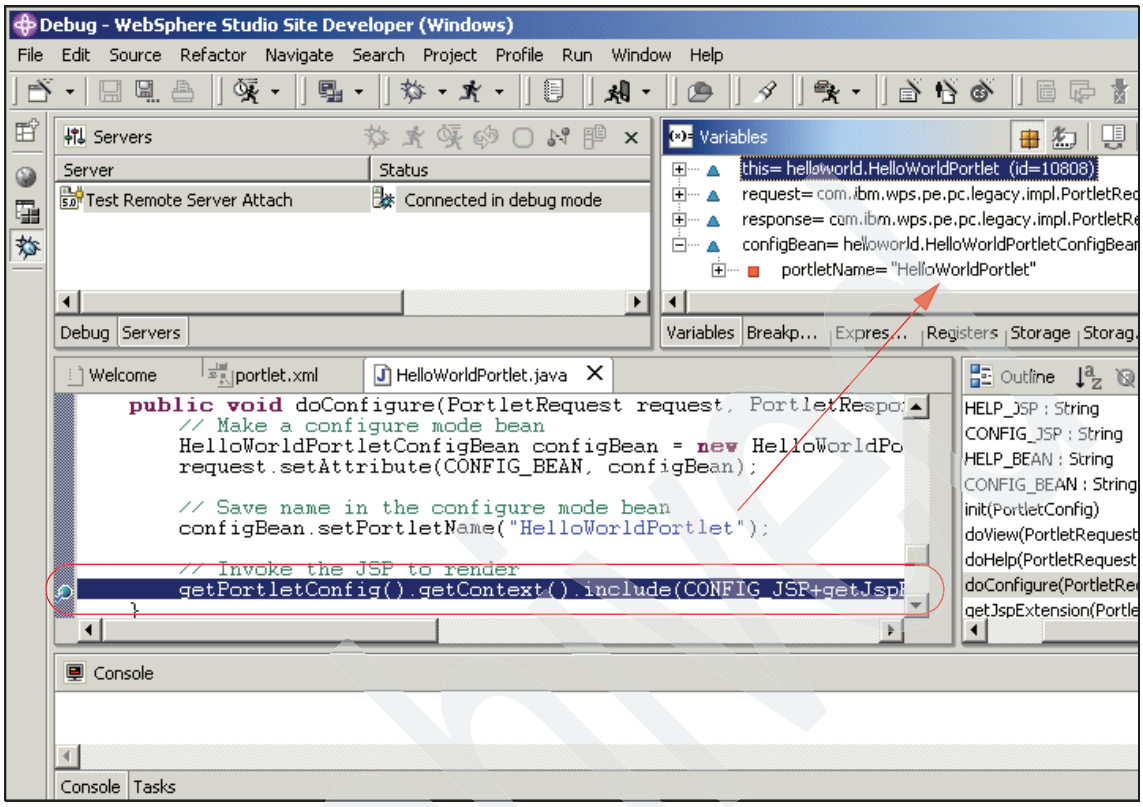


Figure 16-35 Debugging the portlet

Note: The process of debugging portlet applications is the same as for Web applications. You control and trace the execution of the portlet since you can set breakpoints in Java source code and JSPs.

9. Click the triangle in the Debug menu bar to resume execution of the portlet.

Archive

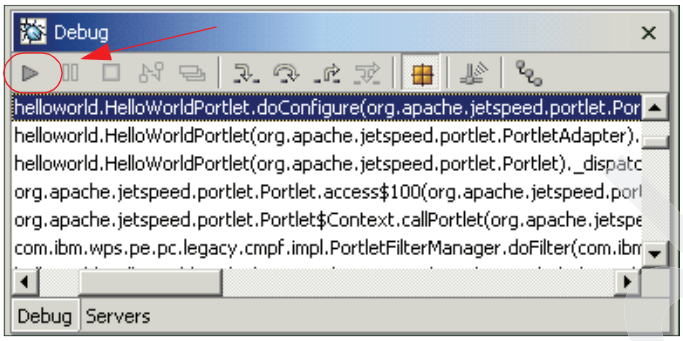


Figure 16-36 Debug window

10. You can disconnect the remote WebSphere Portal by right-clicking the server name in the Servers view and clicking **Disconnect**.

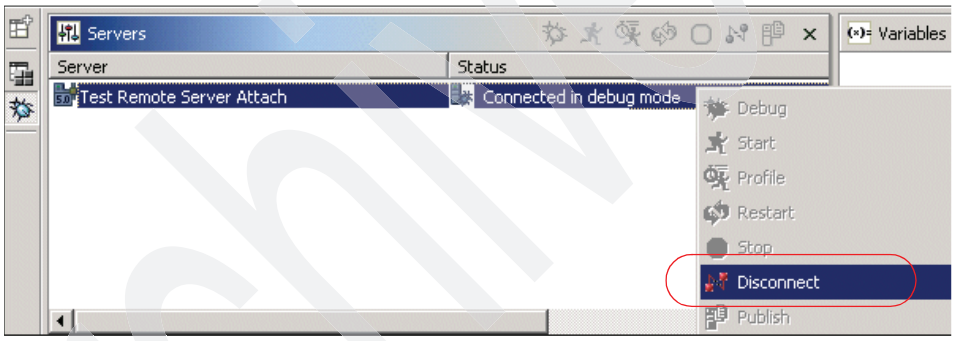


Figure 16-37 Disconnecting from the Remote Server Attach

Archive

A

Portlet development platform sample installation

This appendix shows a sample portlet development installation by describing the steps necessary to prepare a single computer running Windows 2000 Professional or Server for WebSphere application development and testing.

1. Installation of the WebSphere Studio Site Developer (WSSD) V5.0, the WebSphere Application Server V5.0 Test Environment, and related fix packs
2. Installation of the Portal Toolkit and WebSphere Portal V5.0 for WebSphere Studio Site Developer
3. Configuration and preparation of the workstation for lab exercises
4. Installation of the WebSphere Portal V5.0 (runtime environment)

© Copyright IBM Corp. 2004. All rights reserved.

Prerequisites

Minimum machine requirements

- 1GHz Pentium® III or higher
- 768MB RAM
- 20 GB hard drive, LAN Attached or loopback support
- TCP/IP stack must be installed

Software requirements

- Windows 2000 Professional or Server and Service Pack 3 (MS Hotfix KB823980 is recommended)
- Cloudscape V5.1 or DB2 V8.1 FP 1(optional)
- WebSphere Studio Site Developer (WSSD) V5.0 and fix packs
- WebSphere Portal Toolkit V5.0 for WebSphere Studio Site Developer
- WebSphere Portal V5.0
- WebSphere Application Server V5.0 and fix packs

Figure A-1 illustrates the sample portlet development platform for the scenarios in this redbook.

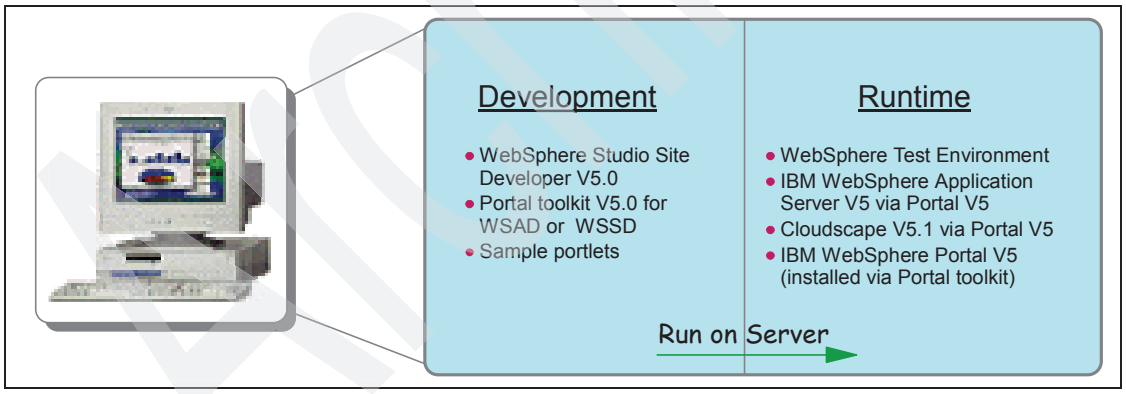


Figure A-1 Portlet development workstation

Installing a loopback adapter

Each target machine should have a static IP address assigned to it. For the purposes of this installation, the *hosts* file will be used to provide address resolution.

1. Prior to software installation, disable your Ethernet or Token Ring adapter by clicking **Start -> Settings -> Network and Dial-up Connections.** If Ethernet or Token Ring adapters appear, right-click and disable them. You should see something like Figure A-2.

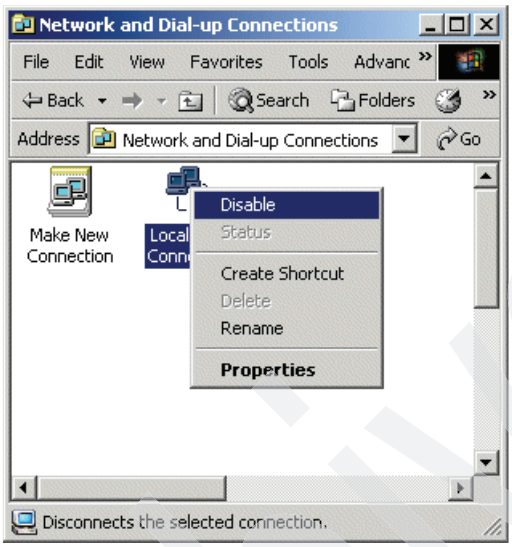


Figure A-2 Network and dial-up connections

2. Install the Loopback Adapter.
   a. Select **Start -> Settings -> Control Panel**.
   b. Double-click **Add/Remove Hardware**. This will start the Add/Remove Hardware Wizard. Click **Next**.
   c. Select **Add/Troubleshoot a device** and click **Next**.
   d. Select **Add a new device** and click **Next**.
   e. Select **No, I want to select the hardware from a list** and click **Next**.
   f. Select **Network Adapters**.
   g. From the Manufacturers column, select **Microsoft** and from the Network Adapter column, select **Microsoft Loopback Adapter**, then click **Next**.
   h. The Loopback adapter will install.
3. Configure the Loopback Adapter.
   a. Select **Start -> Settings -> Network Adapters -> MS Loopback adapter** (or whatever you named it) **-> Internet Protocol (TCP/IP) -> Properties**. You should see a window like Figure A-3 on page 504.

Archived

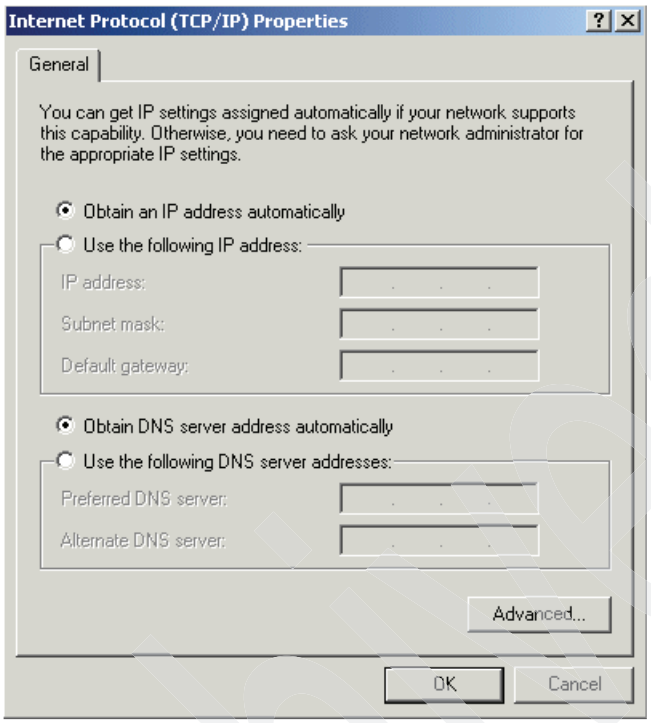


Figure A-3 TCP/IP properties

b. Change to the option **Use the following IP address** then we suggest that you specify `10.1.1.1` for the IP address and `255.255.255.0.` for the Subnet mask, then click the **OK** button.

4. Configure the hostname of your computer.

   a. Go to My Computer and click **Properties**.

   b. To change your hostname, click **Network Identification**, and then click **Properties**; we suggest that you specify `portaldev` in the Computer name field.

   c. Click **More...**, and change the Primary DNS suffix; we suggest `itso.ral.ibm.com`.

   d. Click **OK** in the DNS Suffix window. Click **OK** in the Identification Changes window; you should see a window like Figure A-4 on page 505.

Arcinco

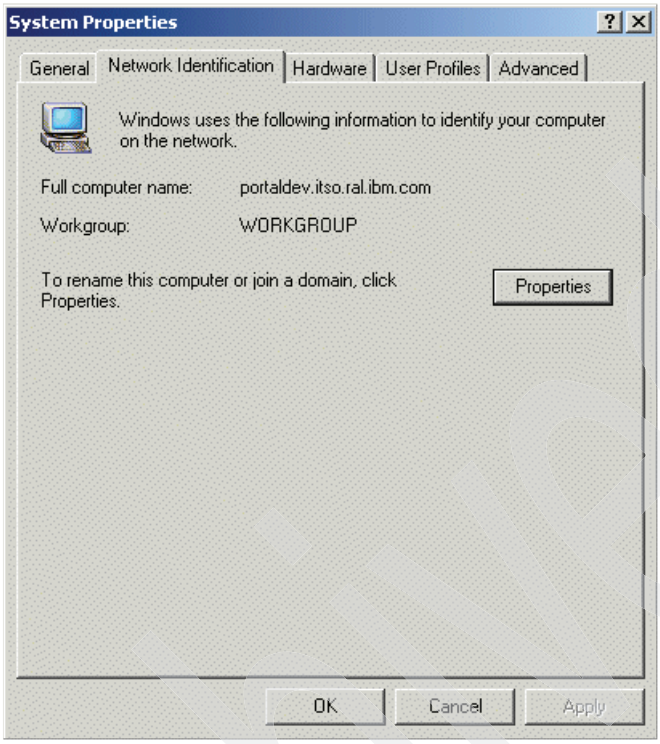


Figure A-4 Network Identification window with a full computer name

5. Obtain the fully qualified host name by executing `ipconfig /all` in a DOS window. The connection-specific DNS suffix appended to the hostname will be the fully qualified name of your computer. In the illustration shown in Figure A-5 on page 506, the fully qualified name of the computer is *portaldev.itso.ral.ibm.com*.

```
C:\>ipconfig /all

Windows 2000 IP Configuration
        Host Name . . . . . . . . . . . . : portaldev
        Primary DNS Suffix  . . . . . . . : itso.ral.ibm.com
        Node Type . . . . . . . . . . . . : Hybrid
        IP Routing Enabled. . . . . . . . : No
        WINS Proxy Enabled. . . . . . . . : No

Ethernet adapter MS LoopBack:
        Connection-specific DNS Suffix  . : isto.ral.ibm.com
        Description . . . . . . . . . . . : Microsoft Loopback Adapter
        Physical Address. . . . . . . . . : 02-00-4C-4F-4F-50
        DHCP Enabled. . . . . . . . . . . : No
        IP Address. . . . . . . . . . . . : 10.1.1.1
        Subnet Mask . . . . . . . . . . . : 255.255.255.0
        Default Gateway . . . . . . . . . :
        DNS Servers . . . . . . . . . . . :
```

Figure A-5 Output from ipconfig command

6. Open C:\WINNT\system32\drivers\etc\hosts with a text editor and add a line with the static IP address and the fully qualified name of the computer. Be sure to end the line by pressing the **Enter** key. The file should look like Figure A-6.Save the file and close the editor.

.



Figure A-6 Lines from the hosts file

7. Reboot Windows for these changes to become effective.

WebSphere Studio Site Developer (WSSD) V5.0

Prepare the following WebSphere Portal V5.0 CDs:

- WebSphere Studio Site Developer for Windows, V5.0.1 - CD # 4-1
- WebSphere Studio Site Developer for Windows, V5.0.1 - CD # 4-2
- WebSphere Studio Site Developer PTFs for Windows, V5.0.1- CD # 4-3
- WebSphere Application Server Fix Pack 1 for Windows, V5.0 - CD # 1-6

Archive

This section will explain the procedure to install the WebSphere Studio Site Developer.

1. Insert WebSphere Studio Site Developer installation CD (CD # 4-1). Navigate to the \wssd directory and double-click **setup.exe**. You will see the welcome window as illustrated in Figure A-7. Click **Install IBM WebSphere Studio Site Developer** to continue.

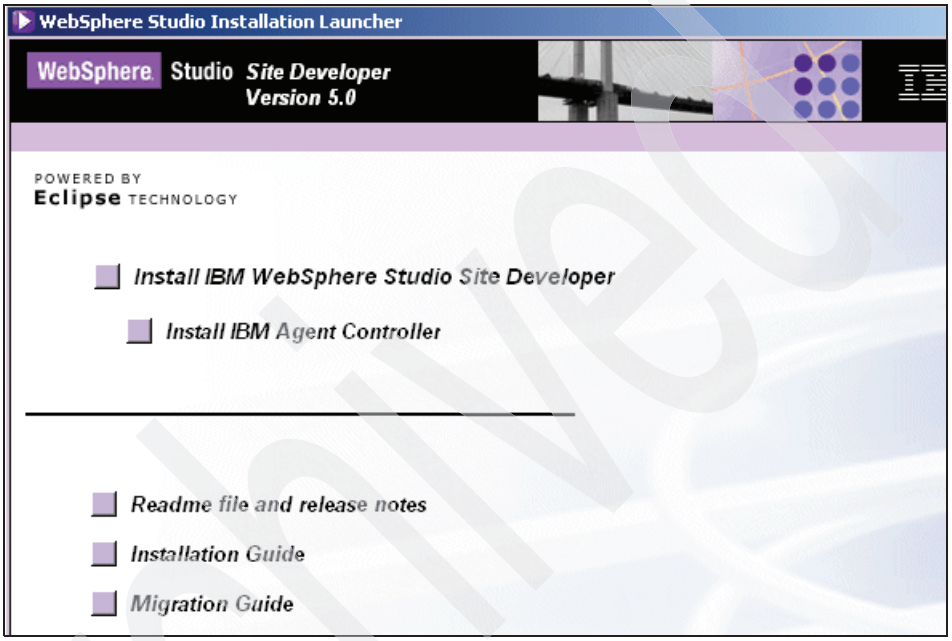


Figure A-7 WebSphere Site Developer installation welcome screen

2. Click **Next** to continue past the copyright notice screen.

Archive



Figure A-8 Copyright notice screen

3. Select **I accept the terms of this license agreement**. Then click **Next** to continue.
4. Accept the default settings for directory locations as shown in Figure A-9. Click **Next** to continue.

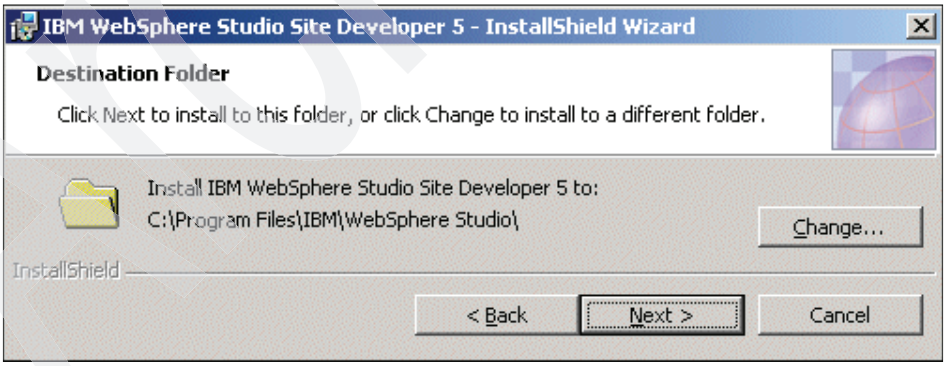


Figure A-9 Installation destination folder

5. Select **WebSphere Application Server - Express** and **WebSphere Application Server v5.0**. Click **Next** to continue.

Archived

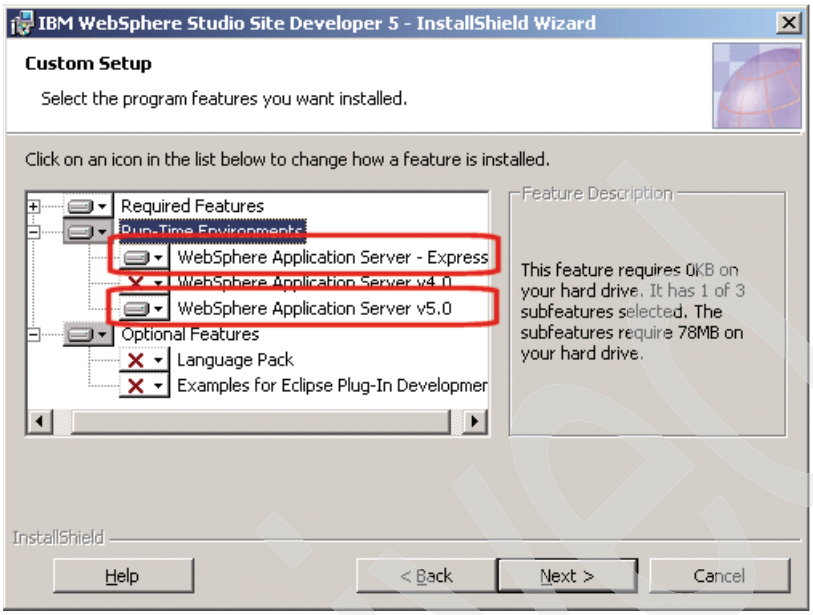


Figure A-10 WebSphere Studio Site Developer features to install

6. Click **Install** to begin the installation.

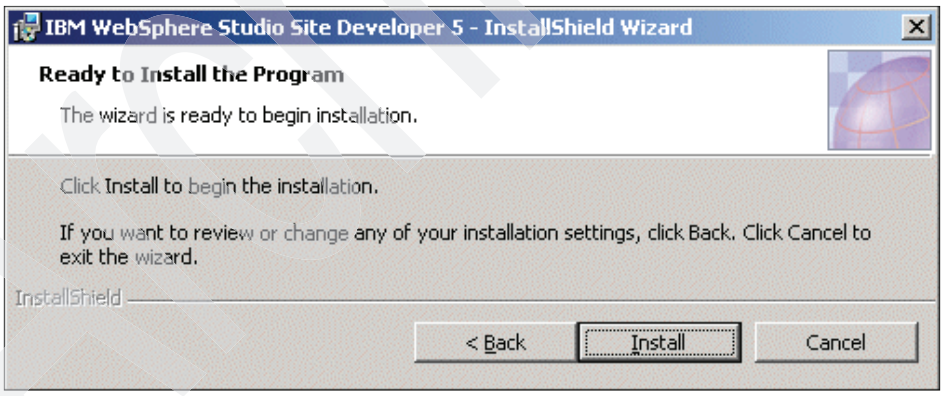


Figure A-11 Last screen before installation begins

7. Midway through the installation, you will be prompted to insert the disk IBM WebSphere Studio Site Developer - DISK2 (Portal CD # 4-2). Insert this disk and click **OK** to continue the installation.

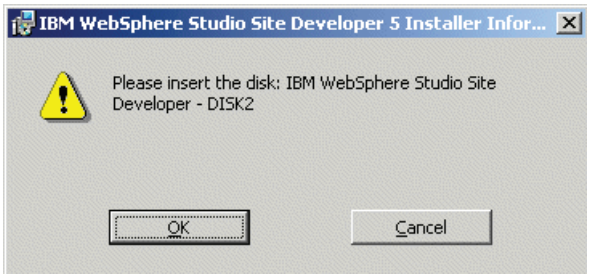


Figure A-12 Insert the disk and click OK to continue the installation

8. At the completion of the installation process, click **Finish** to exit.
9. If you are prompted to restart the machine, click **Yes** to do so.

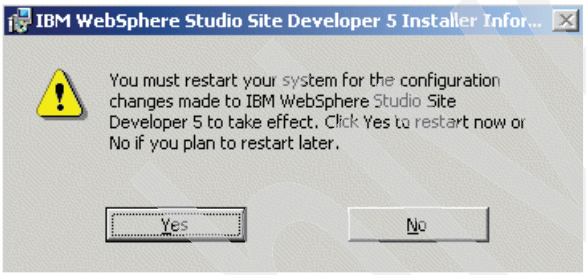


Figure A-13 Click Yes to restart the machine

WebSphere Studio Site Developer - WSSD Fix Pack 1

After installing WebSphere Studio Site Developer, you have to update this installation with Fix Pack 1; to do this, you have two options:

- Software Update function from the help menu (requires Internet connection)
- Using WebSphere Portal CD # 4-3

Software Update function from the help menu

When you run WebSphere Studio Site Developer for the first time, you will get a dialog box to confirm the location of your development workspace. Check the box to not show this window again (you can change this later) and click **OK**.

Archived

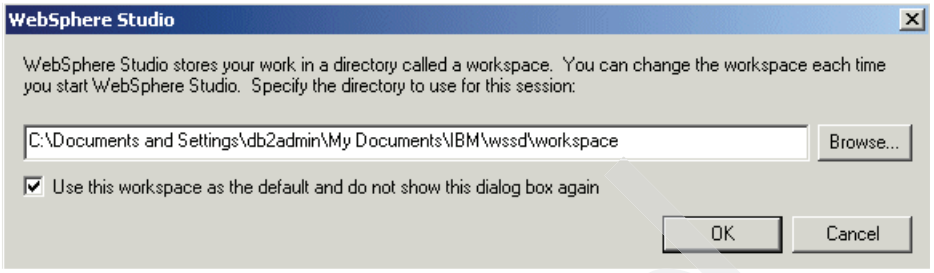


Figure A-14 Setting your workspace

1. Click **Help -> Software Updates -> New Updates**.

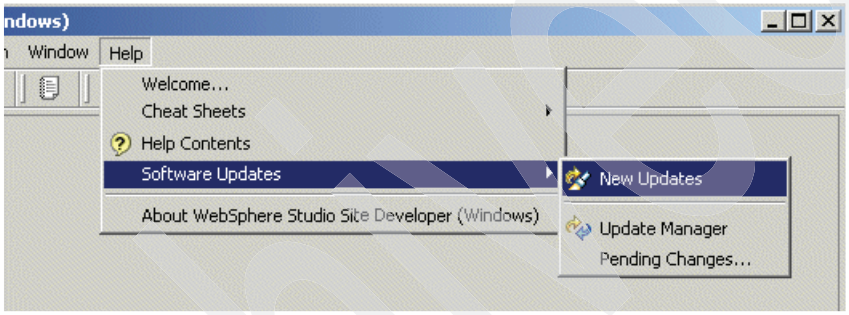


Figure A-15 Help menu to install new updates

2. A dialog window will appear, indicating that WebSphere Studio Site Developer is searching for new updates. This will take a few minutes.

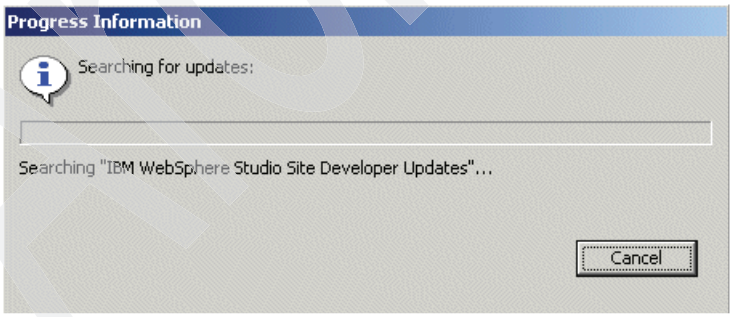


Figure A-16 Updates dialog

3. Choose the two updates that it finds. Click **Next** to continue.

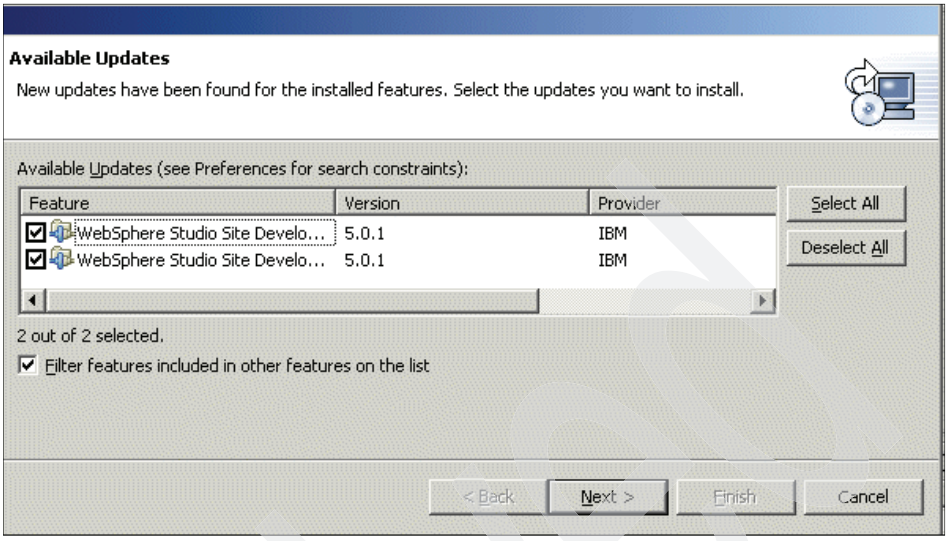


Figure A-17 Selecting updates to install

4. Accept the license agreement and click **Finish** to install.
5. You will receive a verification window for each update that is installed. Click **Install** for each of these.

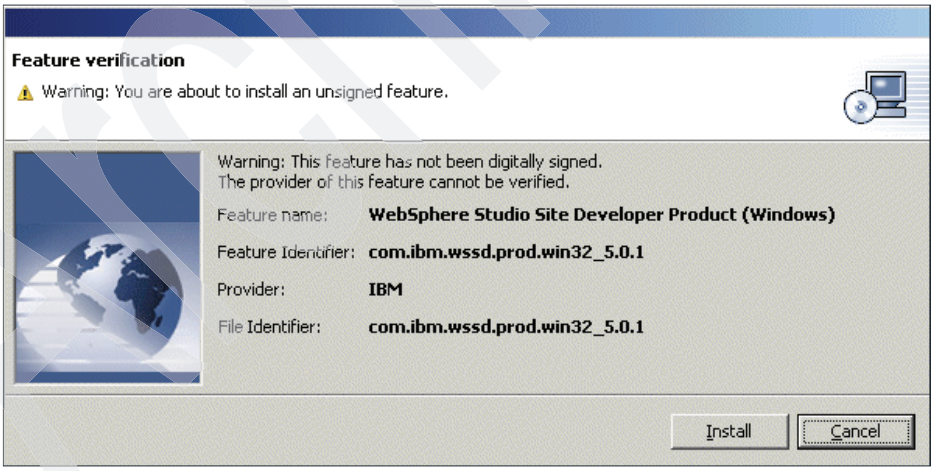


Figure A-18 Installation warnings

6. When the operation is complete, you will be asked to restart WebSphere Studio Site Developer. Click **Yes** to do so.

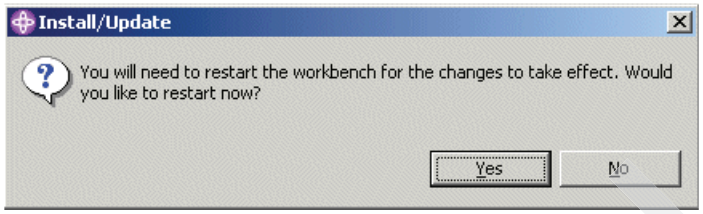


Figure A-19 Restart the workbench dialog

Using Portal CDs

When you run WebSphere Studio Site Developer for the first time, you will get a dialog box to confirm the location of your development workspace. Check the box to not show this window again (you can change this later) and click **OK**.

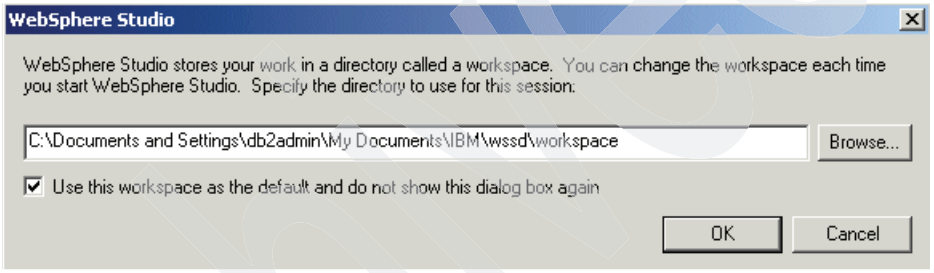


Figure A-20 Setting your workspace

The WSSD Fix Pack 1 is available in the CD # 4-3 of the WebSphere Portal V5.0. Insert the CD # 4-3 into the drive; you can also download this fix pack from the Internet as a zip file from the following site:

`www3.software.ibm.com/ibmdl/pub/software/websphere/studiotools/html/501/wssd/download.html`

1. Start the Update manager by selecting **Help -> Software Updates -> Update Manager**.

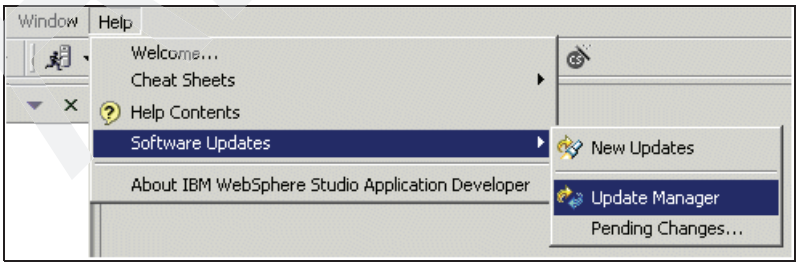


Figure A-21 Update Manager

2. In the Feature Updates view, select **My Computer** and locate the CD drive or the temporary directory you have extracted the zip file to.
3. Expand to the following.
   - If you have downloaded the fix pack: **wssd501 -> update -> IBM WebSphere Studio Site Developer Updates**.
   - If you are using WebSphere Portal CD # 4-3: **wssdptf -> IBM WebSphere Studio Site Developer Updates**.

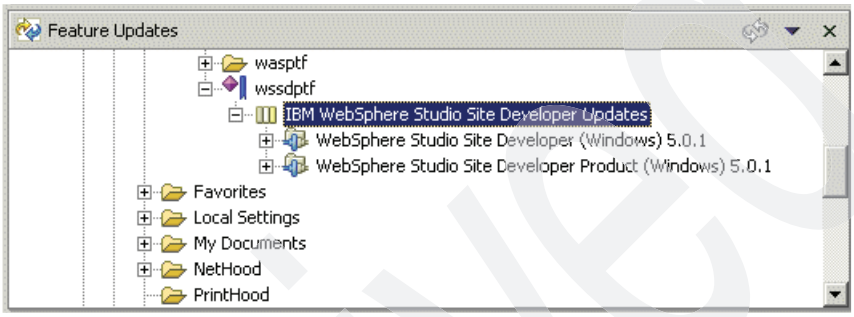


Figure A-22 Navigating in the Feature Updates view

4. Select **WebSphere Studio Site Developer (Windows) 5.0.1** and then click **Update** in the right pane.

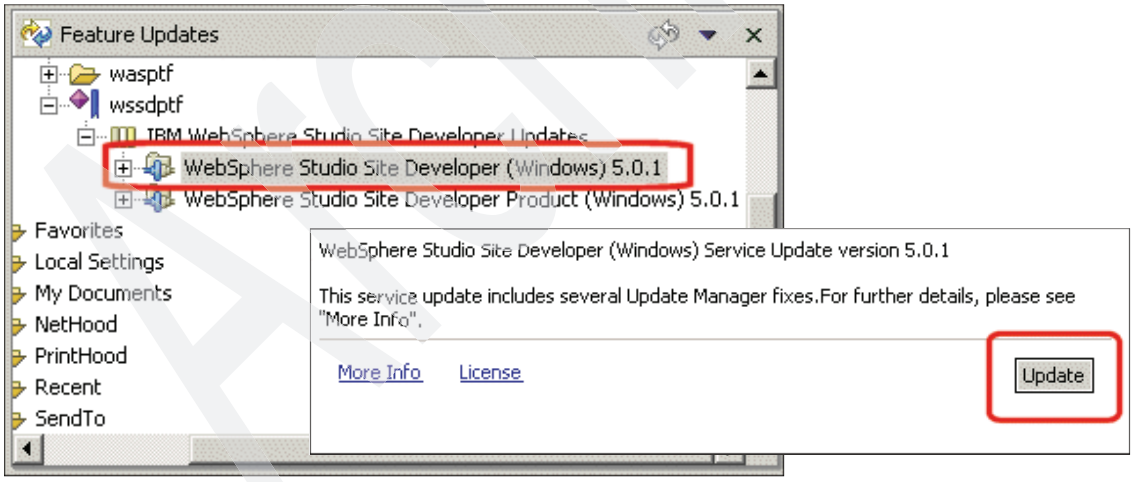


Figure A-23 Update feature of WebSphere Studio Site Developer

5. When the Feature Install screen appears, click **Nex**t.

Archived

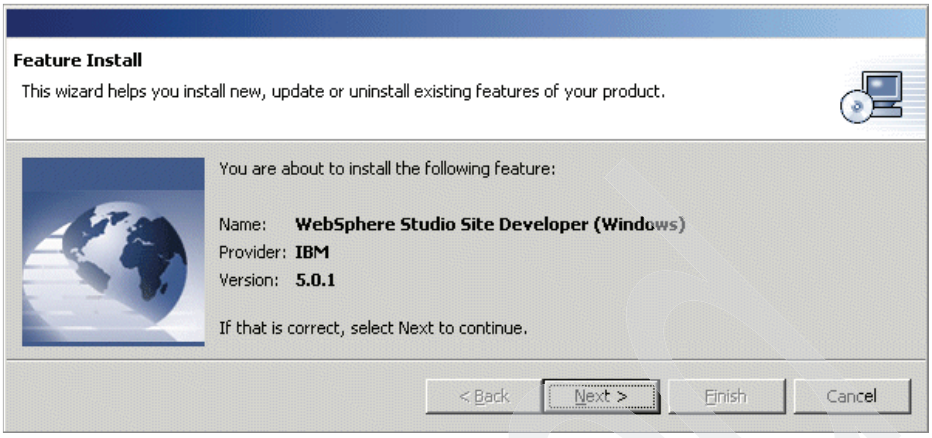


Figure A-24 Feature Install window

6. Once you have read and accepted the license, click **Next**.
7. You will see the Optional Features panel. Do not change any settings. Click **Next**.

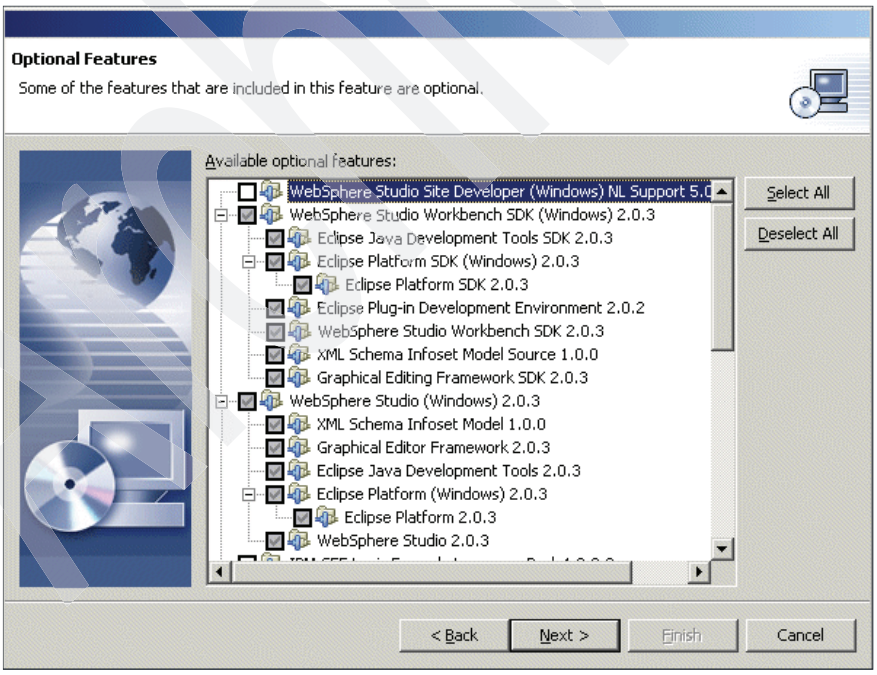


Figure A-25 Optional features screen

8. Click **Finish** on the Install Location panel. You will be warned that you are about to install an unsigned feature. There is no need to worry about this warning, so click **Install** to continue.
9. Upon completion of the installation, you will be asked whether you want to restart the workbench. Click **No** to continue.
10. Select **WebSphere Studio Site Developer Product (Windows) 5.0.1** and then click **Update** in the right pane.

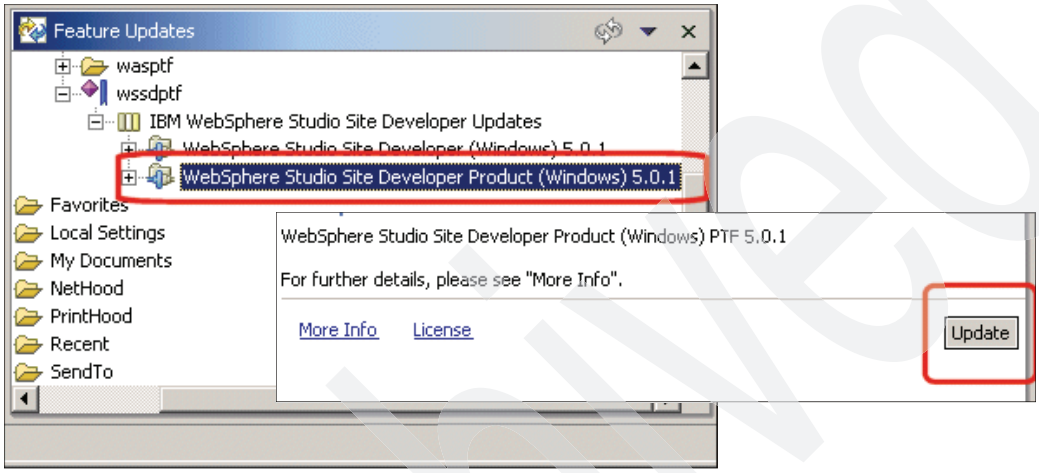


Figure A-26 Update feature of WebSphere Studio Site Developer

11. Go through the Feature Install, Feature License, Optional Features by successively clicking **Next**. Do *not* change any settings in the Optional Features panel. Click **Finish** in the Install Location panel.
12. You will be warned that you are about to install an unsigned feature. There is no need to worry about this warning so click **Install** to continue.
13. Upon completion of the installation, you will be asked whether you want to restart the workbench. Click **Yes** to complete the installation.

WebSphere Studio Site Developer - WebSphere Application Server Fix Pack 1

1. For local debugging, you will need to update the WebSphere Application Server Test Environment; use the WebSphere Portal CD # 4-3 or download the fix pack:

Archived

a. Go to:

   http://www3.software.ibm.com/ibmdl/pub/software/websphere/studiotools/html/501/wssd/install_was.html

   b. Click **WebSphere Application Server v5.0, Windows download**.

2. Ensure that WebSphere Studio is not running.
3. Navigate to the CD drive or to downloaded file and extract the fix pack file X:\wasptf\was50_fp1_win.zip to a temporary folder (for example c:\temp) then copy the extracted ptf folder to C:\Program Files\IBM\WebSphere Studio\runtimes\base_v5\ptf.

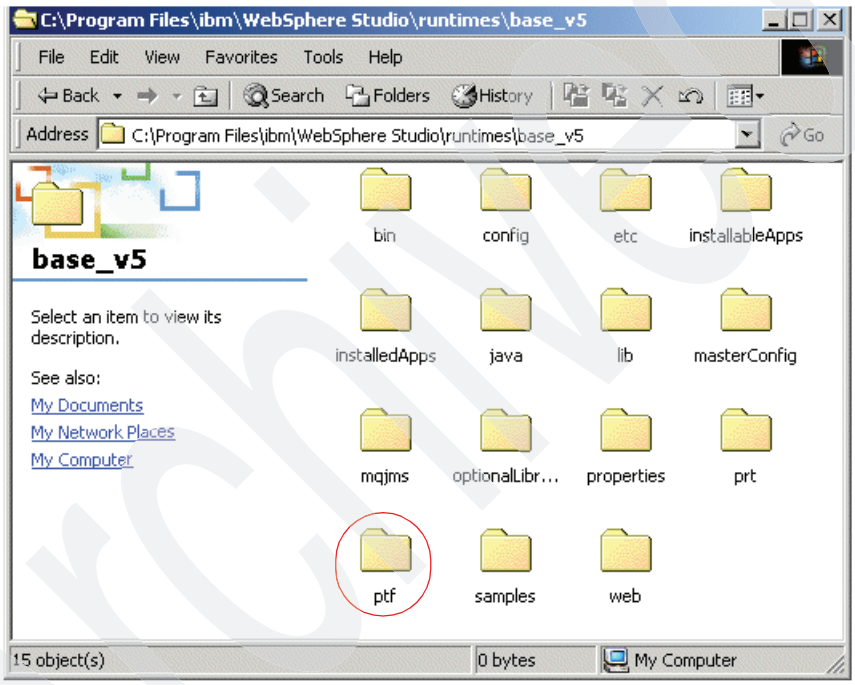


Figure A-27 PTF directory

4. Run wteInstall.bat from the directory C:\Program Files\ibm\WebSphere Studio\runtimes\base_v5\ptf directory to update the software.

Archived

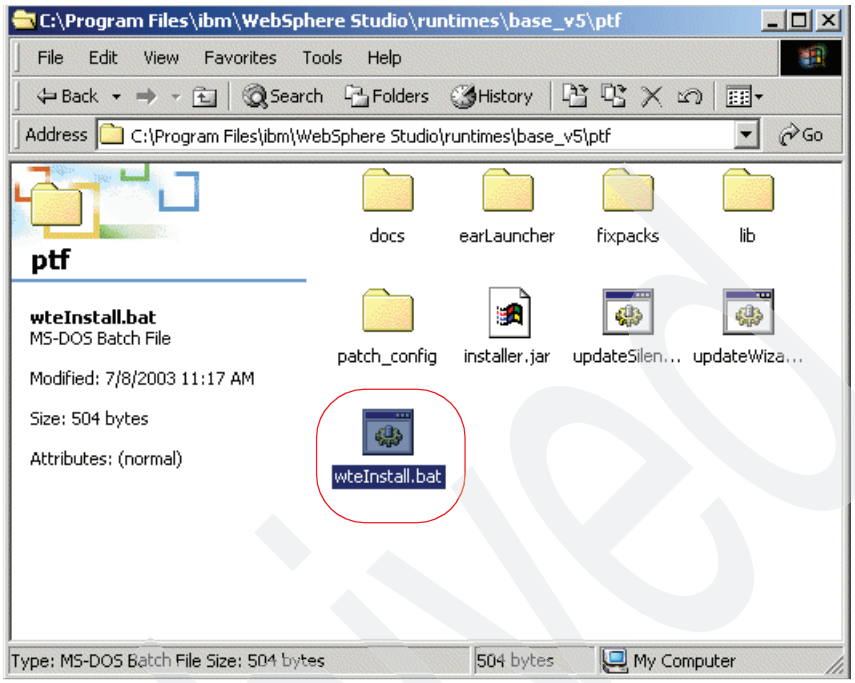


Figure A-28 Running wteInstall.bat

5. Installation will finish silent.

WebSphere Studio Site Developer - WebSphere Application Server Interim Fixes

1. Copy the entire fixes directory (including all its subdirectories and contents) from the WebSphere Portal CD # 1-6 into C:\Program Files\ibm\WebSphere Studio\runtimes\base_v5.
2. Run updateWizard.bat from the directory C:\Program Files\ibm\WebSphere Studio\runtimes\base_v5\fixes to update the software.

Archived

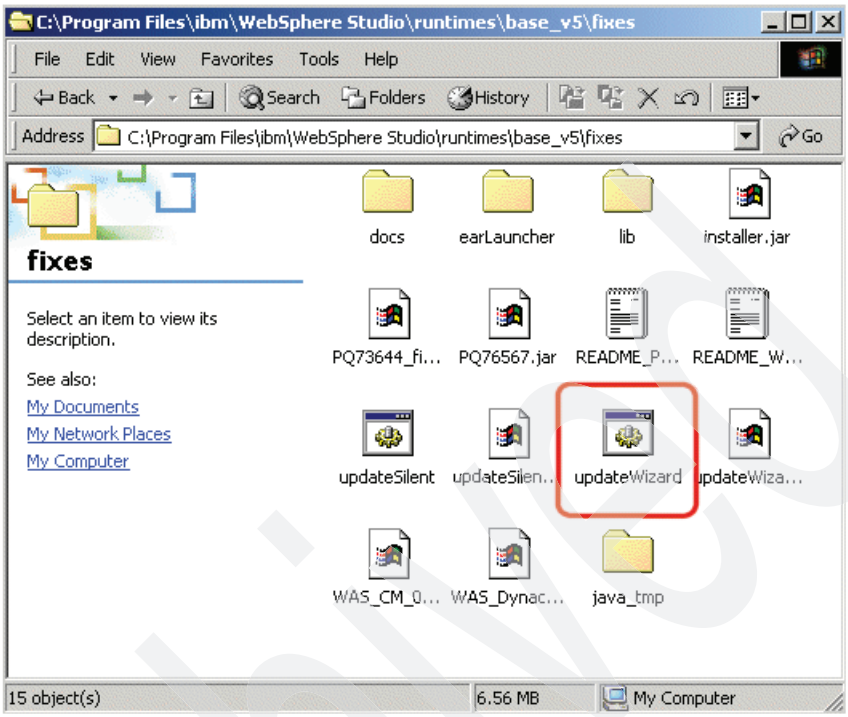


Figure A-29 WebSphere Application Server fixes directory

3. If you have problems launching this file, edit the updateWizard.bat file, verify if the JAVA_HOME variable is correct; if not, correct it to match your environment and run the updateWizard.bat again.

Archived

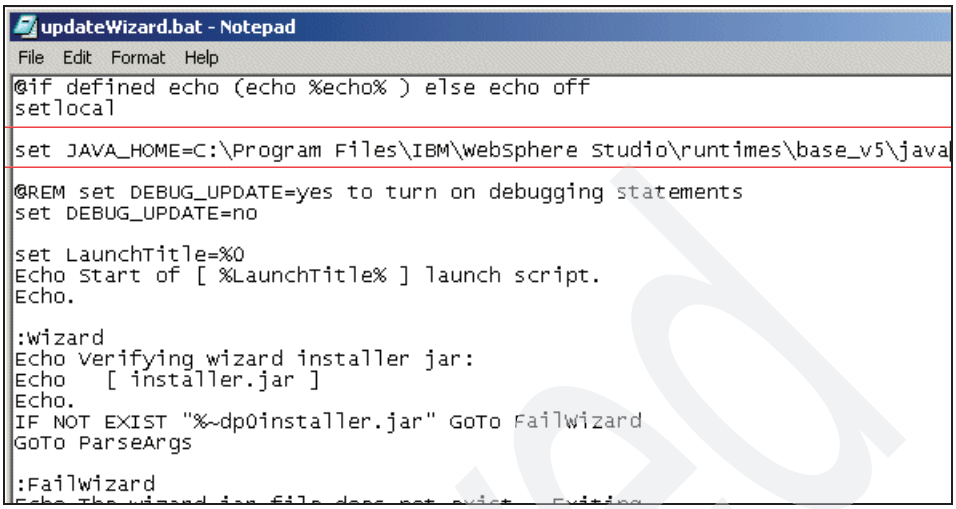


Figure A-30 Modifying the JAVA_HOME setting

4. Click the **Next** button to begin the Update Installation Wizard.

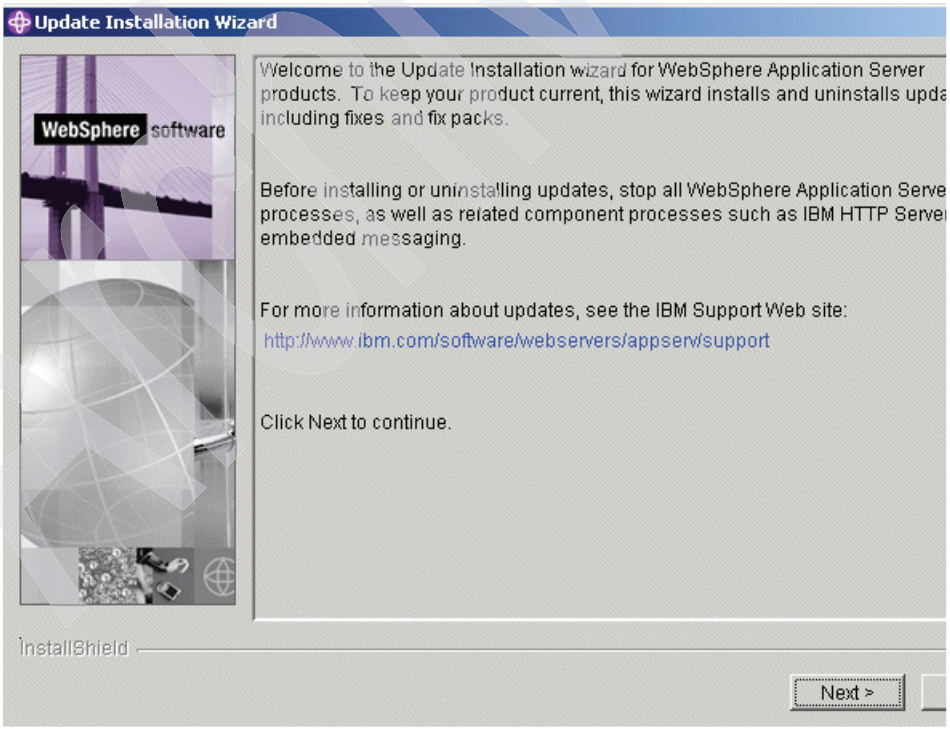


Figure A-31 The Update Installation Wizard

5. On the Specify Product Information window, enter C:\Program Files\IBM\WebSphere Studio\runtimes\base_v5\fixes as the Installation Directory. Click **Next** to continue.

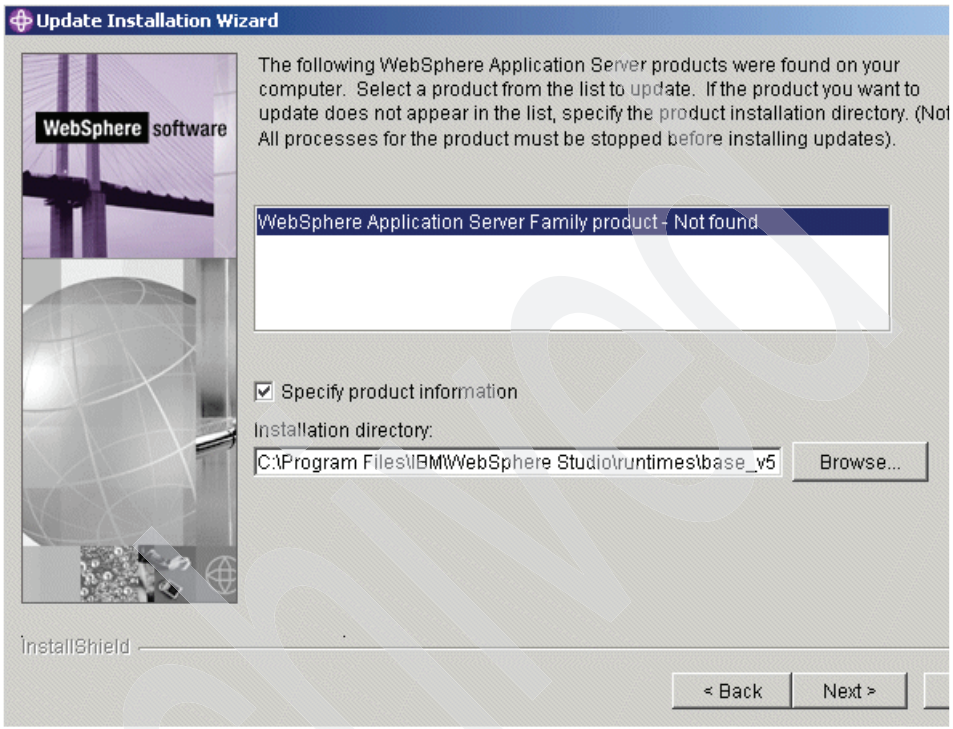


Figure A-32 Selecting the installation directory

6. Click the option **Install fixes**. Then click **Next** to continue.

Archived

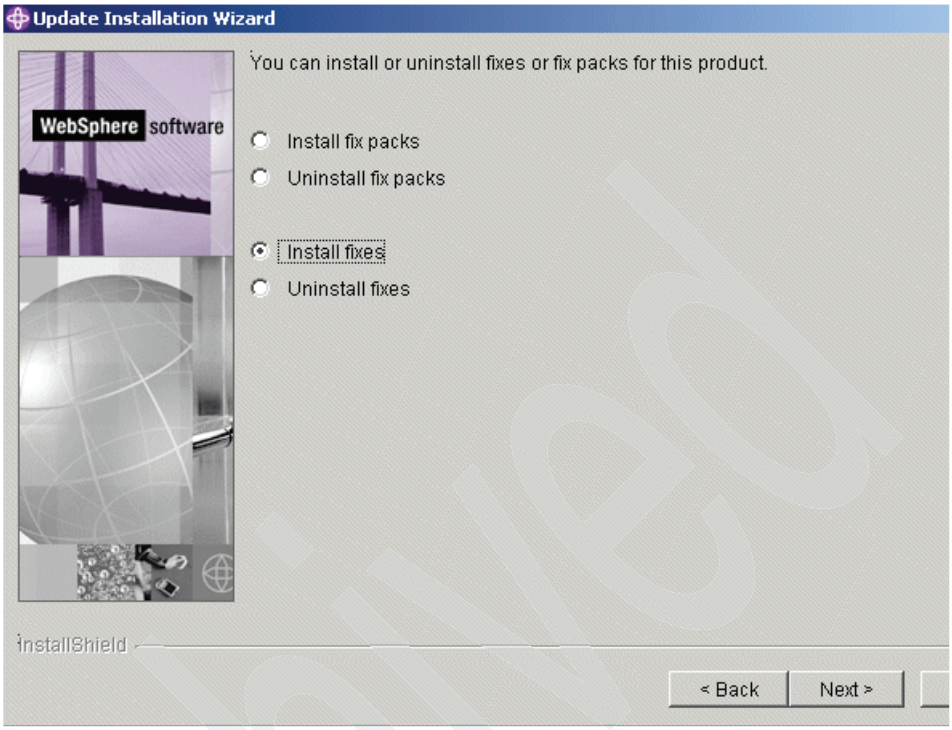


Figure A-33 Choosing install fixes

7. Type C:\Program Files\IBM\WebSphere Studio\runtimes\base_v5 as the directory where fixes are located. Click **Next**.

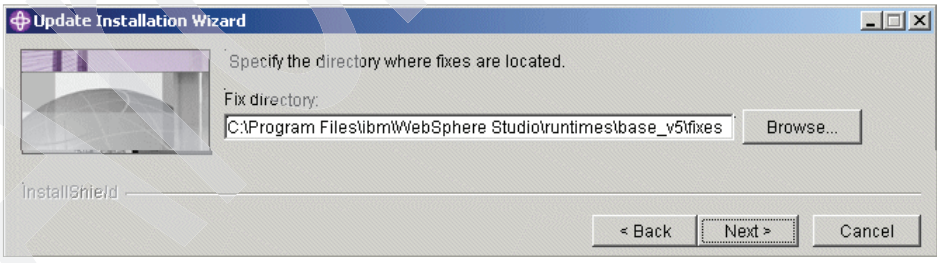


Figure A-34 Choosing the location of the fixes to install

8. Select the four fixes to install. Click **Next**.

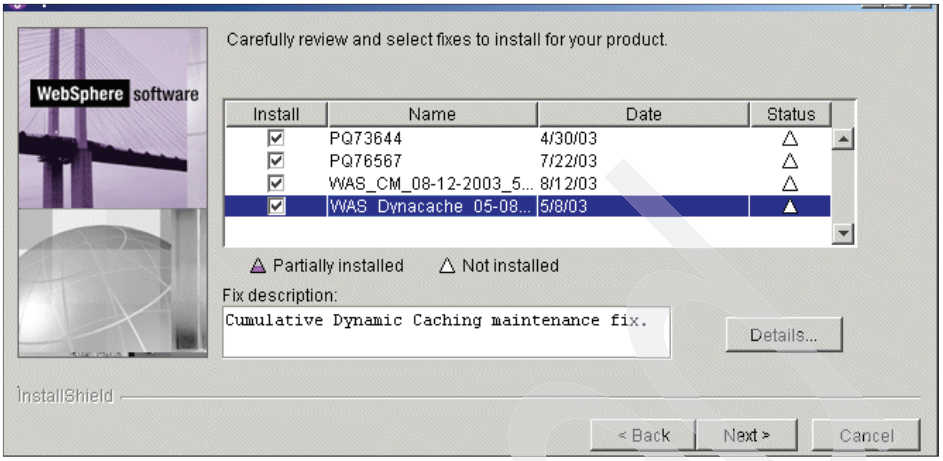


Figure A-35 Selecting the fixes to install

9. Click **Next** to begin the installation of the fixes.

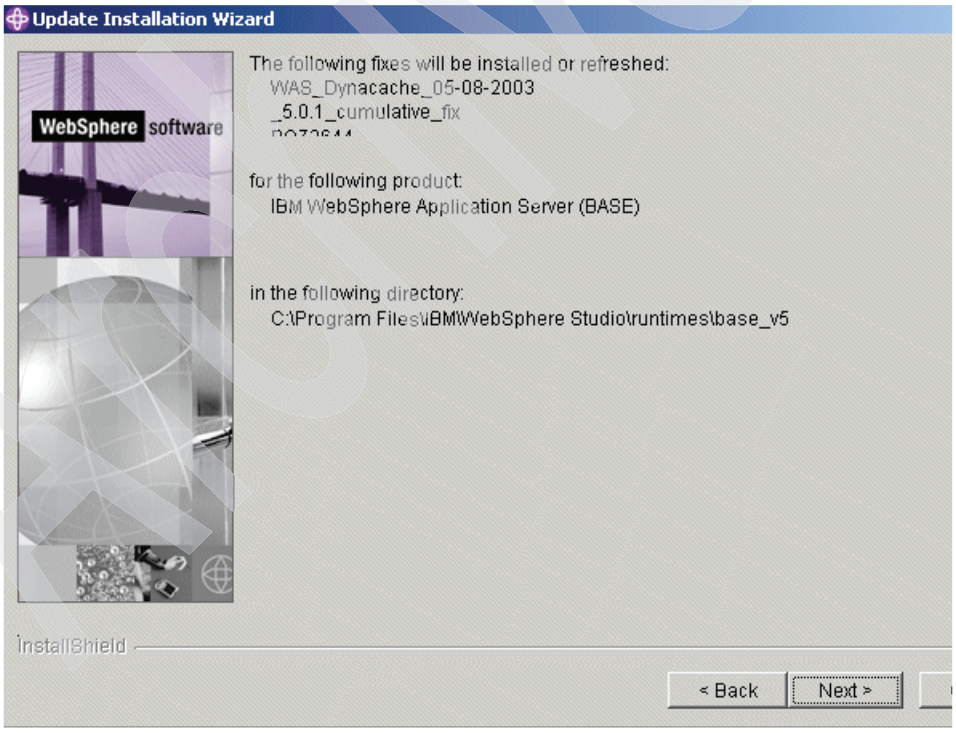


Figure A-36 Beginning the installation

10. When installation is complete, click **Finish** to exit.



Figure A-37 Completing the installation

WebSphere Portal Toolkit V5.0

Following are some considerations regarding WebSphere Portal Toolkit V5.0.

Toolkit installation

This section provides information on the installation of the WebSphere Portal Toolkit.

Prepare the following WebSphere Portal V5.0 CDs:

- Portal Install, Portal InfoCenter, Portal Toolkit (Setup) Windows, V5.0 - Setup CD
- Portal Server, WebSphere Portal content publishing for Windows, V5.0 - CD # 2

1. Hold down the **Shift** key to bypass the automatic starting.
2. Insert WebSphere Portal Setup CD. Navigate to the \PortalToolkit directory and double-click **Install.bat**.

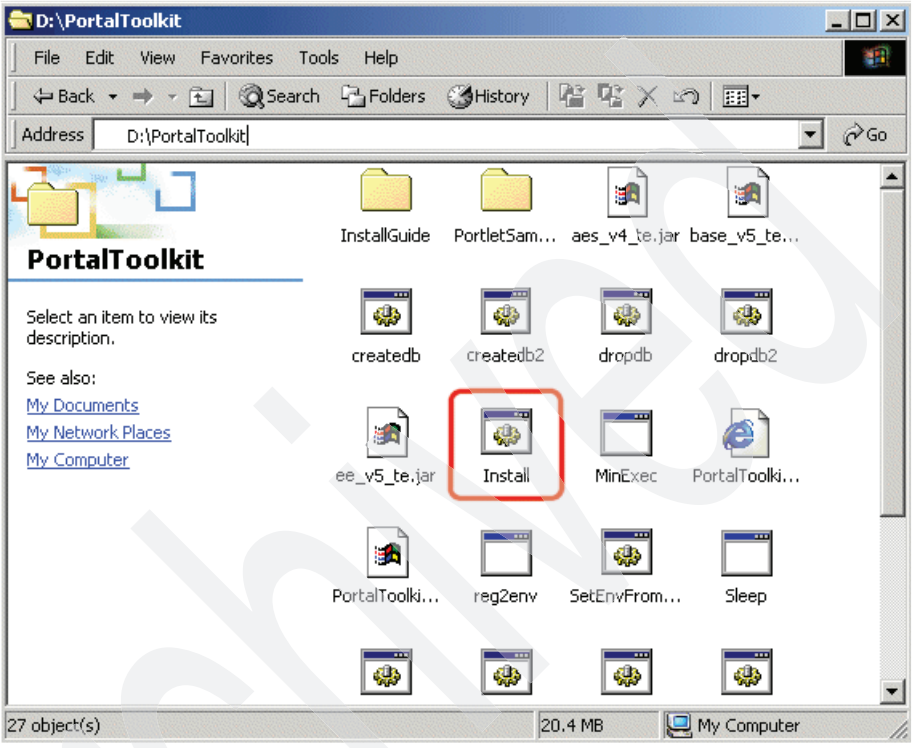


Figure A-38 Toolkit installation directory

3. Select the language to be used then click **OK and** wait for the installation screen to load. Then click **Next** to continue.

Archive

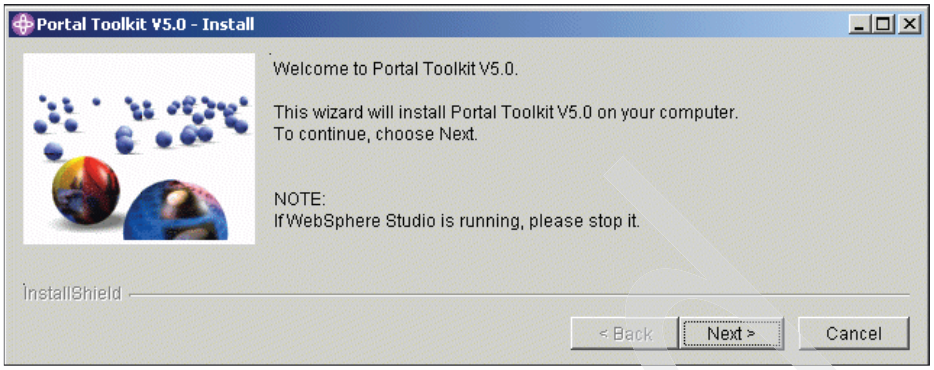


Figure A-39 Welcome screen for Toolkit installation

4. Read and accept the license agreement, then click **Next**.
5. Make sure the directory displayed is the installation directory of WebSphere Studio and click **Next**.

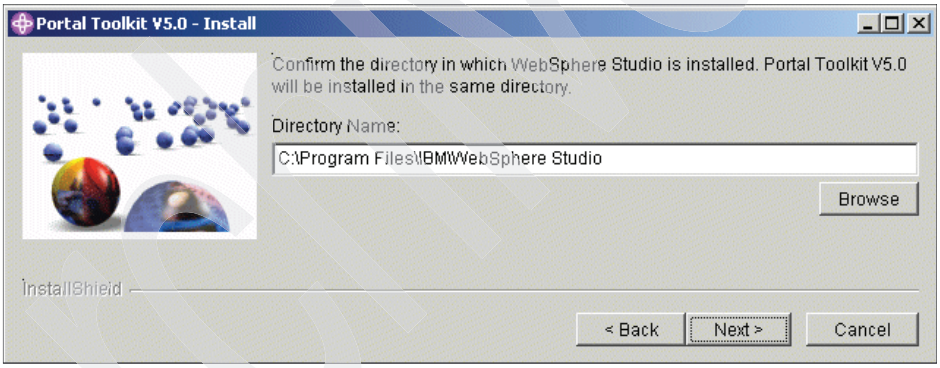


Figure A-40 WebSphere Studio Site Developer installation directory

6. Select **Portal Toolkit V5.0** and **WebSphere Portal V5.0 for Test Environment** then click **Next**.

Archive

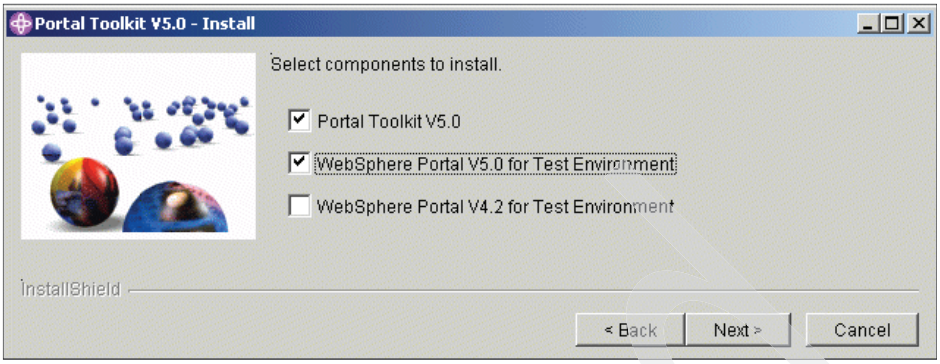


Figure A-41 Select components to install

7. Verify the install location and click **Next**.
8. Hold down the **Shift** key to bypass the automatic starting.
9. Insert WebSphere Portal CD # 2, select the **install.bat** file in the \wps directory and click **Next**.

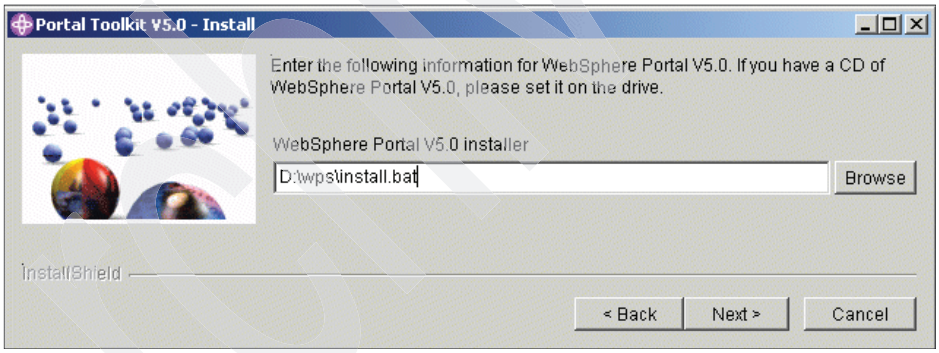


Figure A-42 Portal Toolkit V5 install

10. Wait until the installation complete, then click **Finish**.

Configuring Studio Site Developer and the Portal Toolkit

To use the WebSphere Studio Site Developer and WebSphere Portal Toolkit, you must first configure a Test Environment. It is not necessary for a separate instance of WebSphere Application Server or WebSphere Portal to be running to run the Portlet code that you developed in the WebSphere Studio Site Developer IDE.

Start the WebSphere Studio Site Developer. Select **Start->Programs->IBM WebSphere Studio-> Site Developer 5.0**.

1. The first time you start WebSphere Studio Site Developer, it will ask you to specify a directory to be used as your development workspace. Accept the default directory, check the box next to **Use this workspace as the default and do not show this dialog box again**, and click **OK** to open Site Developer.

   Note: You have already checked **OK** to this dialog, so you probably will not receive it a second time.

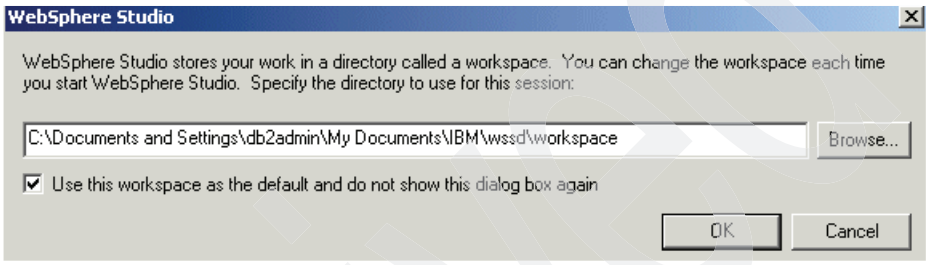


Figure A-43 Workspace directory

2. It may take a while for Site Developer to load for the first time. Please be patient.



Figure A-44 Loading WebSphere Studio Site Developer

Archived

3. Once WebSphere Studio Site Developer is loaded, you should see the following message as shown in Figure A-45; click **Yes**.

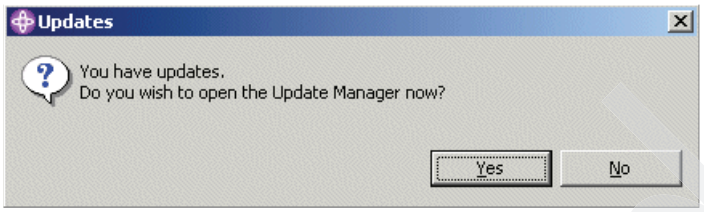


Figure A-45 Updates pop-up window

4. Select the pending configuration change and click **Finish**.

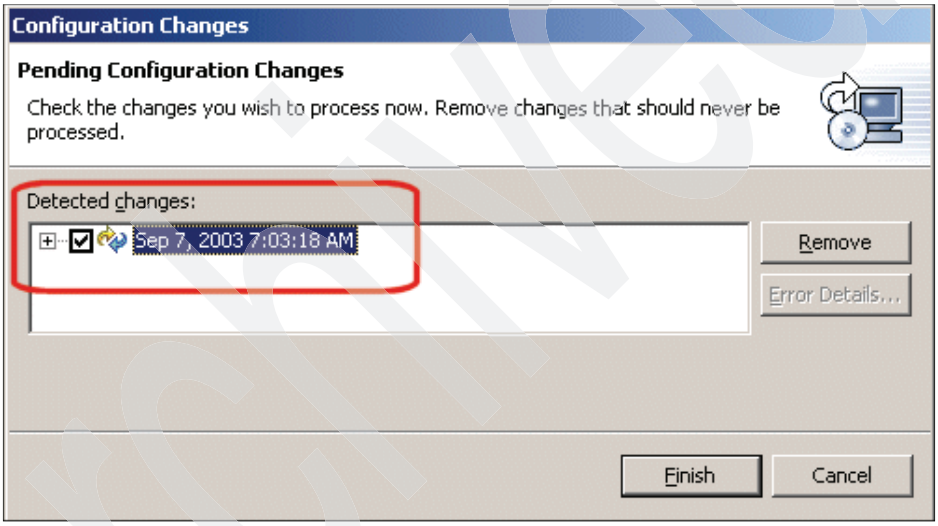


Figure A-46 Configuration changes

5. Click **Yes** when asked to restart the workbench.
6. Once WebSphere Studio Site Developer is restarted, click the **Server Configuration** tab as shown in Figure A-47 on page 530.

Archived

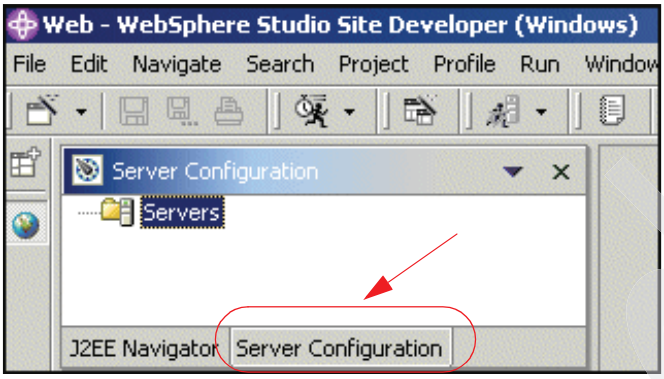


Figure A-47 Server Configuration tab

7. Right-click **Servers**, then click **New**, then **Server and Server Configuration** as shown in Figure A-48.

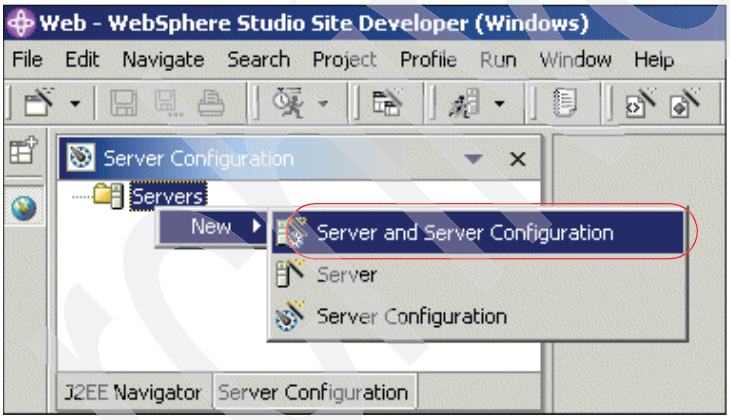


Figure A-48 Creating a new server and configuration

8. A new window will be displayed to allow you to configure the new server and server configuration. Enter `Test Environment` as the server name. Select **Test Environment** under *WebSphere Portal version 5.0* as the server type. Click **Next** to continue.

   Note: You must enter a server name to continue. Also, be sure to select **Test Environment**.

Archived

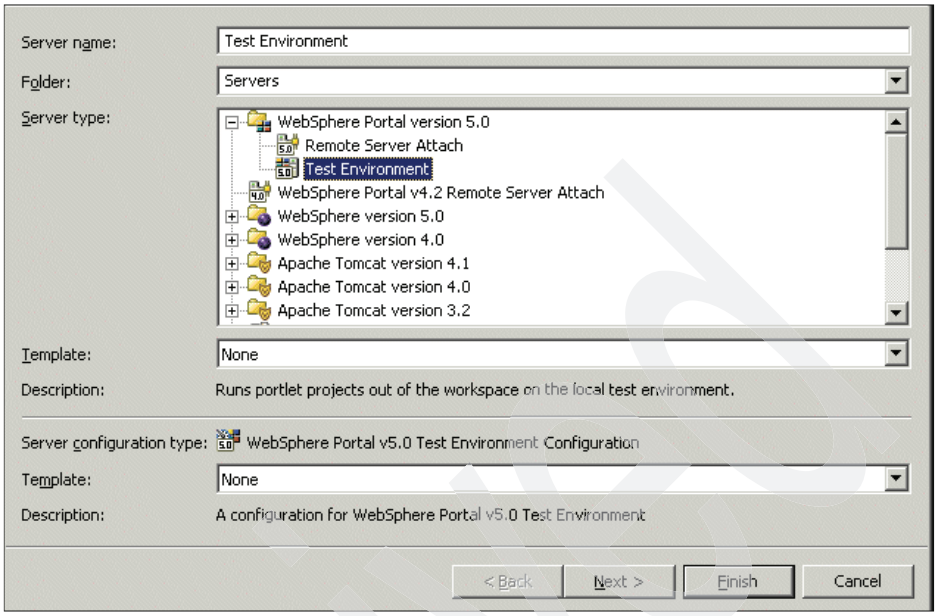


Figure A-49 Server configuration settings window

9. This will bring up a window to select the HTTP port number to be used by Site Developer. The default is 9081. Click **Finish** to add the Test Environment.

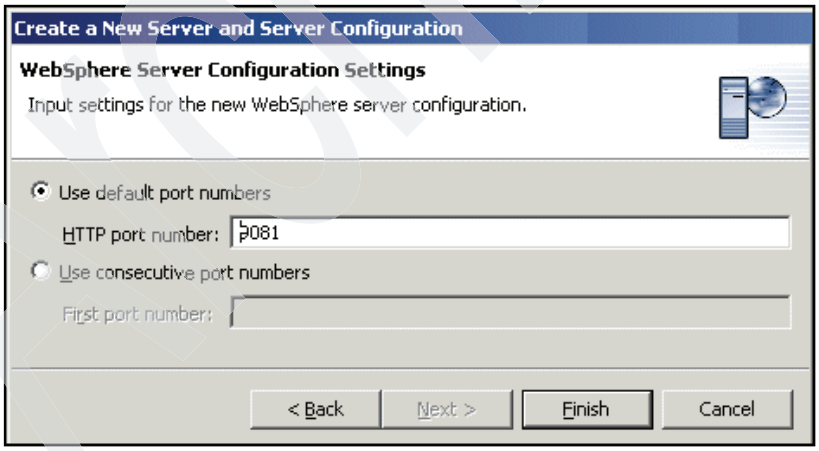


Figure A-50 HTTP port selection

10. The server has been successfully added and can now be seen in the Server Configuration tab.

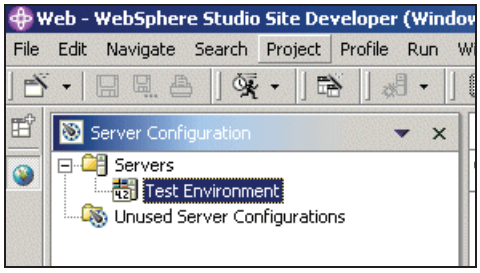


Figure A-51 Test Environment has been added successfully

11. WebSphere Site Developer and the Portal Toolkit are configured so that whenever you test a portlet by right-clicking it and selecting **Run on Server**, it will execute in this Test Environment.

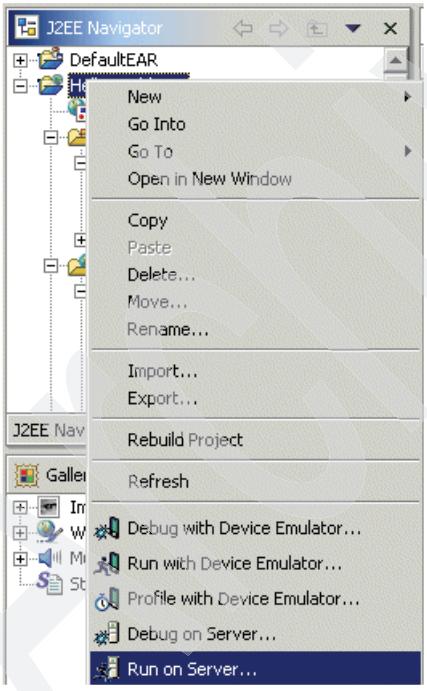


Figure A-52 Running a portlet in the Test Environment

Archived

Configuration and preparation of the workstation

This section details how to create a Cloudscape sample database for use with the scenarios in this redbook when accessing a back-end database using the JDBC interface.

Installing the Cloudscape sample database

1. If needed, download the additional materials associated with this redbook (lab files) and find the provided CreateCloudTable.bat file.
2. From the Windows Explorer, navigate to C:\LabFiles\Cloudscape and execute CreateCloudTable.bat as show in Figure A-53. This command will create and populate the WSSAMPLE database with sample data.

   Note: The CreateCloudTable.bat file can be downloaded from the sample code available as additional materials. See Appendix C, “Additional material” on page 543.

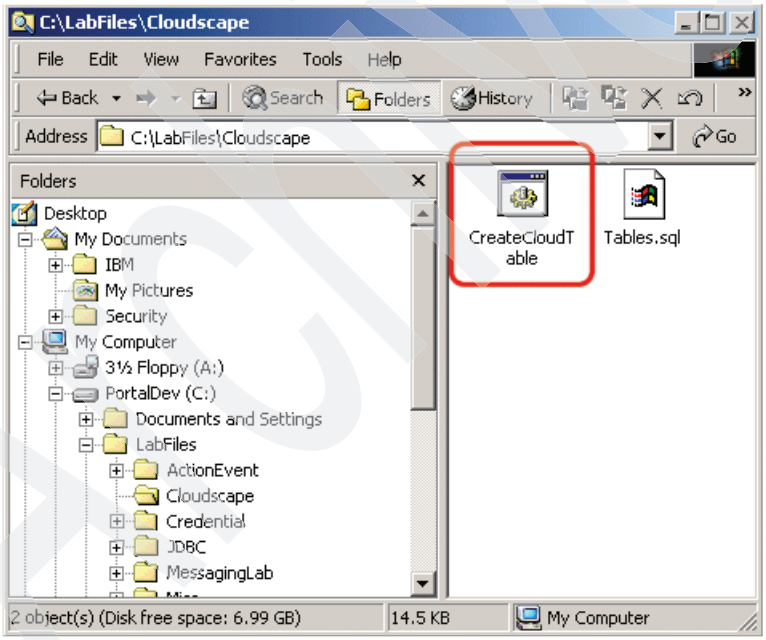


Figure A-53 Cloudscape script to create and populate a sample database

3. Make sure that the database was populated; to do this, you can use a tool that comes with Cloudscape: Cloudview.
   a. Make sure that JAVA_HOME directory is in your PATH, so you can run “java” in your command prompt.

b. In a command prompt window, navigate to C:\Program Files\IBM\WebSphere Studio\runtimes\portal_v50\shared\app\cloudscape\bin and execute the setCP.bat; this will set all the environment variables in order to run the CloudView.

c. Execute the cview.bat, then when it starts click **File -> Open** and navigate to C:\LabFiles\Cloudscape\WSSAMPLE, then click **Open**. Navigate to the CloudView interface to becme familiar with it and check the sample data available in the WSSAMPLE database.

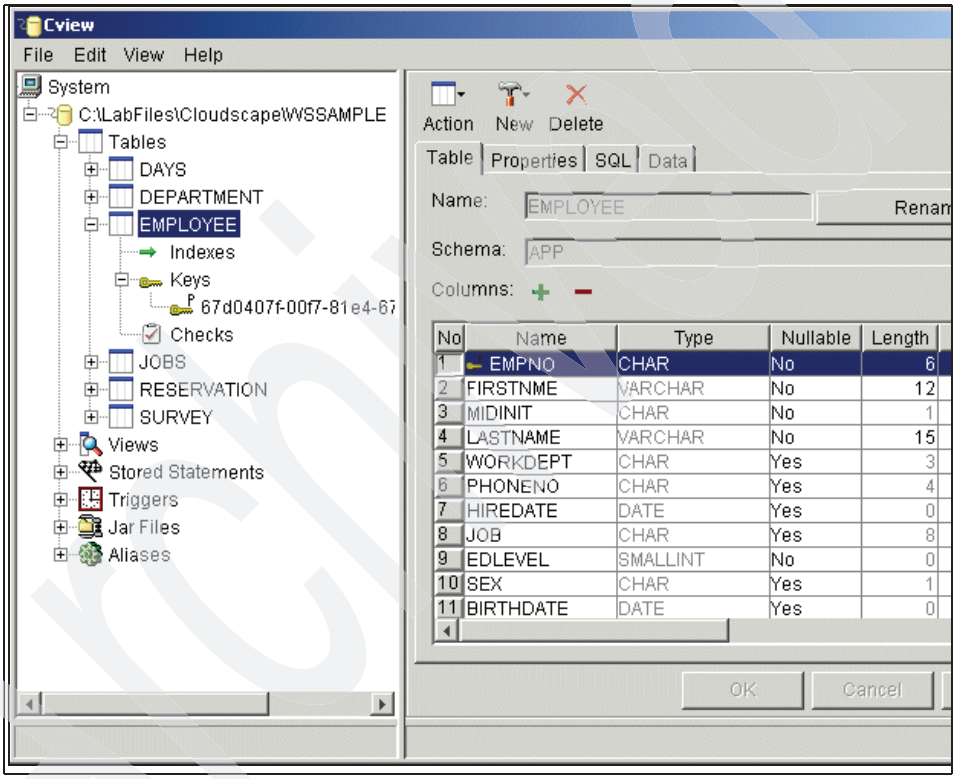


Figure A-54 Checking the database creation with CloudView

Archive

B

Automatically redeploying portlets

Often when you develop portlets, for example Struts or Cooperative Portlets, you have to redeploy the portlet after changing configuration files. You can do this manually using the administration portlets. This appendix describes how to configure WebSphere Studio to redeploy portlets automatically without restarting your test portal server.

Archived

© Copyright IBM Corp. 2004. All rights reserved.

Description

To configure WebSphere Studio, proceed as follows:

1. Log in to WebSphere Portal and open the Administration page.
2. Select **Portlets -> Manage Applications**.

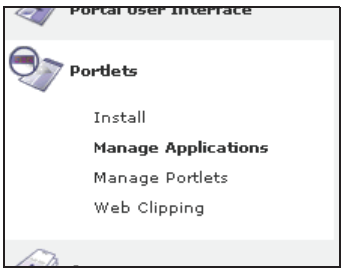


Figure B-1 Select Manage Applications to get the portlet name

3. Select the Web module you want to redeploy and click the **Show Info** button.
4. In the portlet info page, mark the Portlet Application Name and copy it to the clipboard.



Figure B-2 The portlet info portlet

5. In WebSphere Studio, select **File -> New -> Other...**.
6. In the New Projects wizard, select **Project** from the Simple category. Click **Next** and enter a project name of `PortletReConfigurationFiles`.
7. Switch to the XML perspective. In the Navigator view, right-click the **PortletReConfigurationFiles** project and select **New -> XML**.
8. Select **Create XML file from Scratch** and click **Next**. Enter a xml file name of `UpdatePortlet.xml` and click **Finish**.
9. In the Source tab of the XML editor, enter the XML code shown in Example 16-1 on page 537.

Archived

Example 16-1 XML file for updating the portlet using XMLAccess

```
<?xml version="1.0" encoding="UTF-8"?>
<request xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="PortalConfig_1.2.xsd" type="update"
create-oids="true">
   <portal action="locate">
      <web-app action="update" active='true'
uid="com.ibm.myportlet.MyPortlet.7058b2390e0e001817ab8f2fcd901ef9">
         <url>file://localhost/D:/wps5/installableApps/MyPortlet.war</url>
      </web-app>
   </portal>
</request>
```

10. Replace the uid value with the Portlet Application Name you copied in step 4. Replace the url value with the absolute portlet path you want to use. We will use this path to export the portlet application.
11. Save and close the XML file.
12. Switch to the Java perspective. Select **File -> New -> Project...**. Select **Java Project** from the Java category. Click **Next** and enter the project name `XMLAccess`. Click **Finish**.
13. Right-click the **XMLAccess** project and select **Properties**.
14. In the Properties window, switch to the Java Build Path category.
15. Switch to the Libraries tab. Click the **Add External JARs...** button. Browse to the installation location of WebSphere Portal or WebSphere Portal Test Environment. Select **tools.jar** from the WPS_INSTALL_DIR/bin directory.

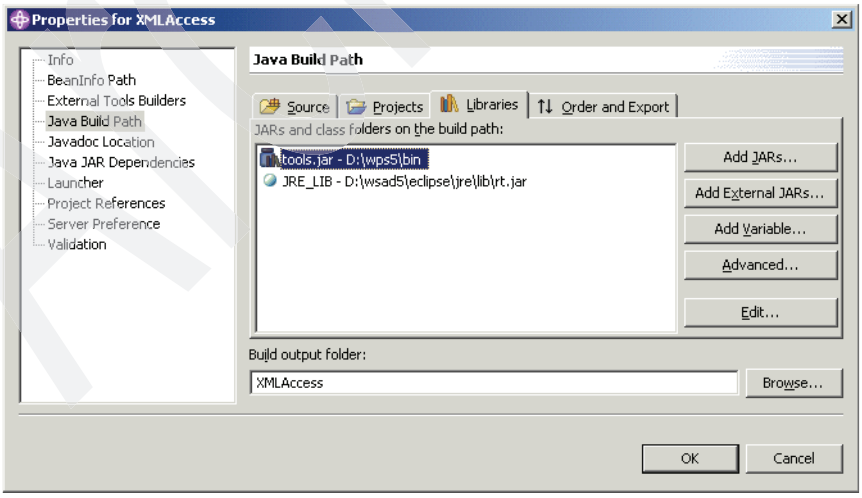


Figure B-3 Add the tools.jar file to the classpath

Archived

16. Click **OK** to close the Properties window.
17. Select **Run...** from the Run tool menu.

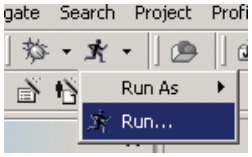


Figure B-4 Select Run... to create a new Launch configuration

18. Select **Java Application** from the Launch Configuration list and click the **New...** button.
19. Enter the name `UpdatePortlet` and check **Include external jars...**.

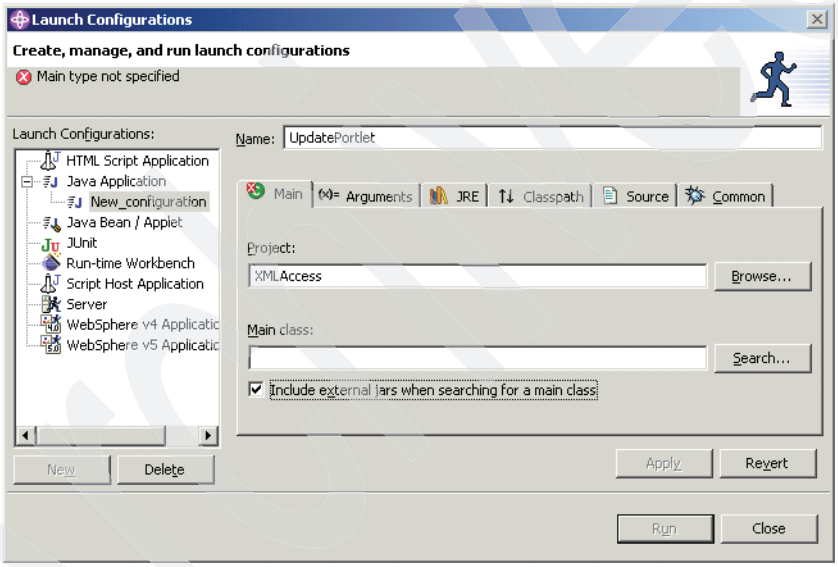


Figure B-5 Launch configurations

20. Click the **Search...** button near the Main class field.
21. In the Choose Main Type dialog, select the **XmlAccess** class and click **OK**.
22. Switch to the Arguments tab. In the Program arguments field, enter the following text:

```
-in UpdatePortlet.xml -user wpsadmin -pwd wpsadmin -url
http://localhost:9081/wps/config
```

23. Replace the user, pwd and url arguments with the values of your server.

Archived

24. Uncheck **Use default working directory**. Select **Workspace** and browse to the PortletReConfigurationProject.

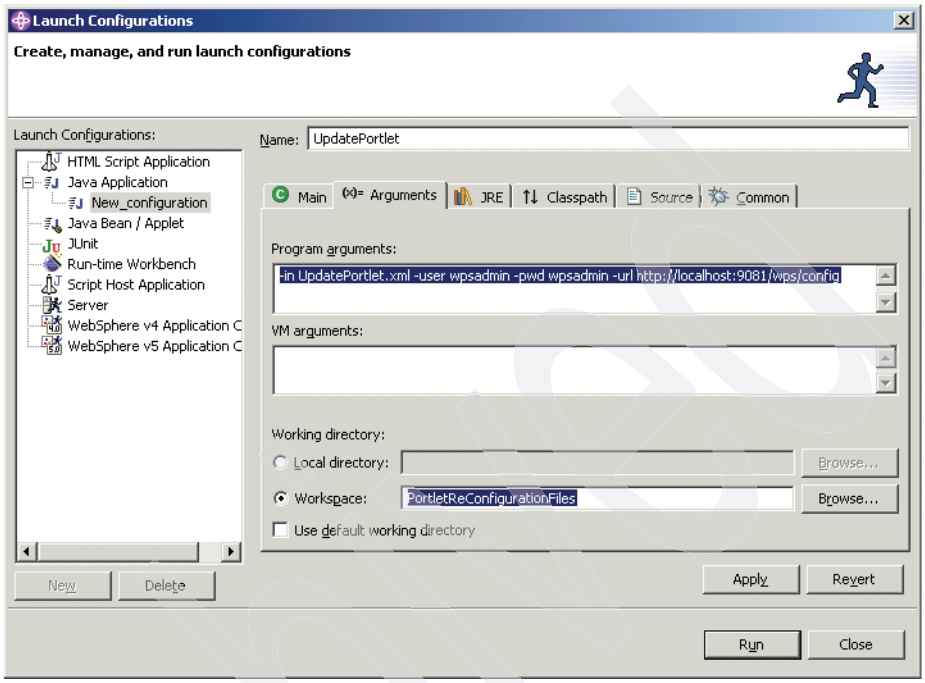


Figure B-6 Arguments tab in Launch Configurations dialog

25. Click **Run**.

26. If you did not export the Portlet war file previously, you will see an error message in the Console view. That is fine for now. Otherwise, you should see an output as in Figure B-7 on page 540.

Archive

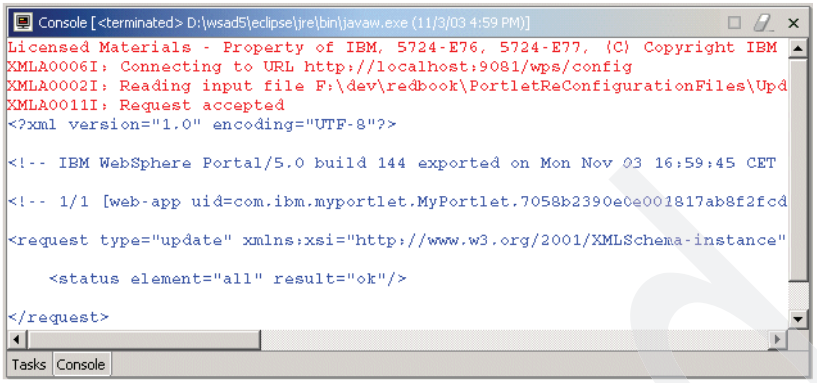


Figure B-7 Console output after redeploying the portlet

27. Switch to the portlet perspective.
28. In the J2EE Navigator view, select your portlet and choose **Export...** from the context menu.
29. In the Export wizard, select **WAR file** and click **Next >**. Select the portlet you want to export and enter the absolute path you configured in Example 16-1 on page 537.

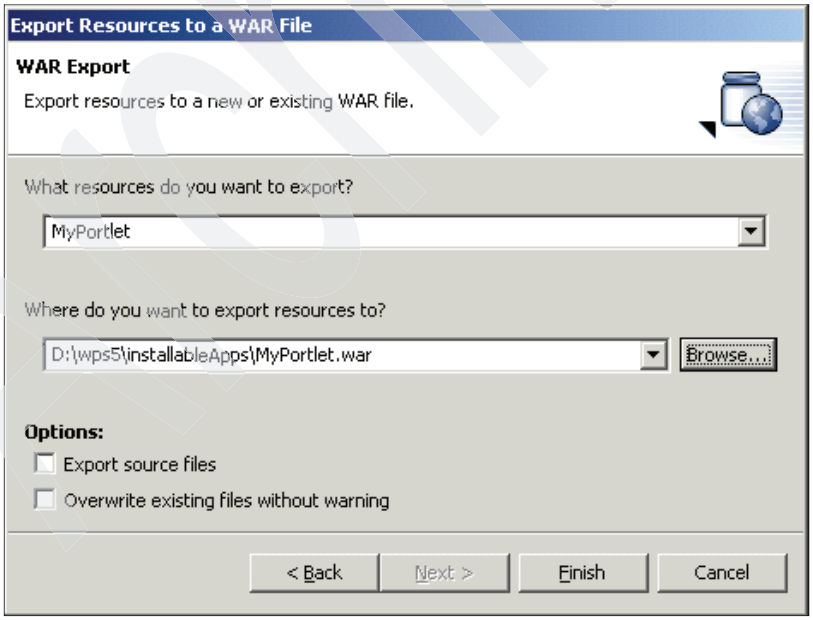


Figure B-8 Export your Portlet

Archived

30. Click **Finish**.

31. Select **Window ->Customize Perspective...**.

32. In the Customize Perspective window, check **Launch from the Other sub tree**.

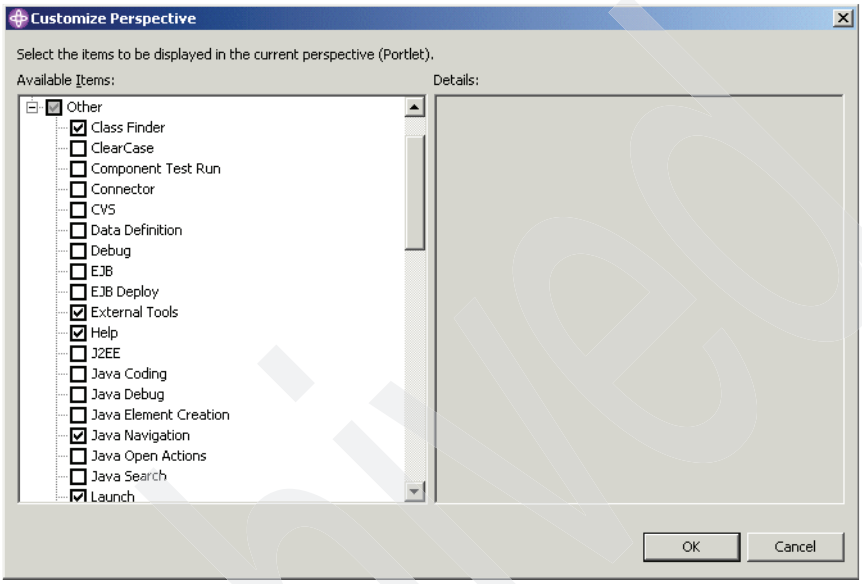


Figure B-9 Check Launch to get new items in the toolbar

To redeploy your portlet, proceed as follows:

1. Export your portlet war file as described in Step 28 on page 540.
2. Select **UpdatePortlet** from the Run tool menu.

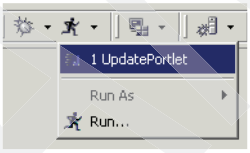


Figure B-10 Select UpdatePortlet to redeploy the portlet

Archived

C

Additional material

This redbook refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this 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/SG246076

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the redbook form number, SG24-6076.

© Copyright IBM Corp. 2004. All rights reserved.

Using the Web material

The additional Web material that accompanies this redbook includes the following files:

File name	*Description*
SG246076.zip	Zipped code samples

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space:	3 MB minimum
Operating System:	Windows
Processor:	1GH or higher
Memory:	512 MB or higher

How to use the Web material

Create a subdirectory (folder) on your workstation and unzip the contents of the Web material zip file into this folder.

Archived

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 546. Note that some of the documents referenced here may be available in softcopy only.

- *IBM WebSphere Portal V4 Developer’s Handbook,* SG24-6897
- *IBM WebSphere Portal V4.1 Handbook Volume 2,* SG24-6920
- *IBM WebSphere Portal V4.1 Handbook Volume 3,* SG24-6921
- *IBM WebSphere V4.0 Advanced Edition Handbook,* SG24-6176
- *IBM WebSphere V4.0 Advanced Edition Security,* SG24-6520
- *WebSphere Portal V4.1 AIX 5L Installation,* REDP3594
- *WebSphere Portal V4.1 Windows 2000 Installation,* REDP3593
- *IBM WebSphere Portal V4.1.2 in a Linux Environment,* REDP0319
- *WebSphere Portal Collaborative Components,* REDP0319

Other publications

These publications are also relevant as further information sources:

- *WebSphere Portal Primer,* ISBN: 1-931182-13-2

Online resources

These Web sites and URLs are also relevant as further information sources:

- IBM Portal Information Kit:

  `http://www-3.ibm.com/software/info1/websphere/index.jsp?tab=landings/portalkit&S_TACT=1O2BBW01&S_CMP=campaign`

© Copyright IBM Corp. 2004. All rights reserved.

- IBM WebSphere Software Platform:

  http://www-3.ibm.com/software/info1/websphere/index.jsp?tab=highlights/

- IBM WebSphere Application Server Support:

  http://www-3.ibm.com/software/webservers/appserv/support.html

- IBM WebSphere Portal - InfoCenter:

  http://www.ibm.com/software/webservers/portal/library/enable/InfoCenter/

- IBM WebSphere Application Server AE - InfoCenter:

  http://www-3.ibm.com/software/webservers/appserv/doc/v40/ae/infocenter/index.htmlu

- IBM WebSphere Portal for Multiplatform:

  http://www.ibm.com/software/info1/websphere/solutions/offerings/portallibrary.jsp

- Using WebSphere Portal log files:

  http://www7b.software.ibm.com/wsdd/zones/portal/V41InfoCenter/InfoCenter/wpf-ena/en/InfoCenter/wps/trouble.html#portal_log

- WebSphere Developer Domain - Search:

  http://www7b.software.ibm.com/webapp/dd/transform.wss?URL=/wsdd/library/index.xml&xslURL=/wsdd/xsl/document.xsl&format=two-column&site=wsdd

- WebSphere Portal Primer book:

  http://mc-store.com/bookfromibmp.html

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Archived

Index

A

Abstract and concrete portlet applications 62
accessibility 7
Accessing resource bundles in JSPs 257
Accessing resource bundles in portlets 256
Accessing Web Services 291
Action Event Handling 98
ActionEvent 99
ActionListener 98–99
Active Credential 116
active credential 323
Add a JavaBean 174
Add actionlistener 160
Adjusting Portal resource bundles 264
Administrative slot 115
Apache Jetspeed 10
architecture 10
Attribute storage summary 107
Authentication 5
Authorization 5

B

bidi 123
breakpoint 213
Building a war file 78
Business to Business 5, 8
Business to Consumer 5, 8
Business to Employee 5, 8

C

c2a.tld 118
Canonical Portal URLs 41
categorization 42
Click-to-Action 371
Cloudscape 533
Cloudview 533
Collaboration 6, 18
Content Management 6, 17, 43
Content management 6
content.tld 118
ContentAccessService 109
Controller 154
cooperative broker 417
Cooperative Portlets 371
  beginEventPhase() 416
  broadcast 417
  broadcasting 413
  C2A wrapper 416
  callback method 415
  changedProperties() 414
  Click-to-Action architecture 379
  Click-to-Action event 414
  Click-to-Action menus 418
  combined scenario 419
  Ctrl key 414
  encodeProperty 374
  encodeProperty tag 417
  event phase 415
  Hints and tips 409
  Import Resources from a WAR File 421
  input parameters 414
  J2EE Settings page 420
  output parameters 414
  Overview 372
  Portlet Messaging 372
  portlet.xml 427
  PortletWrapper 398
  programmatic approach 413, 416
  Programming model 373
  property broker 414
  PropertyListener 375
  Register and publish properties 375
  registerProperties() 416
  Run the cooperative portlets 405
  sample scenario 419
  setProperties() 415
  Source cooperative portlet 380
  Steps to program a source cooperative portlet 374
  Target cooperative portlet 390
  Web Deployment Descriptor 398
  Web Service Description Language 374
  WebSphere Portal Property Broker 373
Cooperative portlets 371
Core event objects 99
Core Portlet Objects 83

© Copyright IBM Corp. 2004. All rights reserved.

Create a portlet project 154
Create the Service Factory 111
createReturnURI 119
createURI 119
Credential 115
credential slots 321
Credential Vault 319–320, 330
  Active credential objects 323
  Add credential vault handling 333
  administrative slot 322
  Administrator-managed segments 321
  Components of the Credential Vault 321
  Credential slots 322
  Credentials objects 323
  HttpBasicAuth 323
  HttpFormBasedAuth 323
  Import a protected servlet application 325
  J2EE Settings Page 332
  JaasSubjectPassive 323
  JavaMail 323
  LtpaToken 323
  Overview 320
  Passive credential objects 323
  portlet private slot 322
  Portlet Settings 332
  private keys 320
  shared slot 322
  SimplePassive 323
  SiteMinderToken 323
  SSL client certificates 320
  system slot 322
  user credentials 320
  User-managed segments 321
  UserPasswordPassive 323
  Using active credentials 330
  Using passive credentials 341
  Vault segments 321
  WebSealToken 323
Credential Vault organization 321
Credential Vault Portlet Service 321
credentials 320
CredentialsVaultService 109
CredentialVaultService Methods 117
Custom developed portlets 49
Custom Services 109
Customizable portlets from a vendor 49

D
dataAttribute 120
database connection 344
dataLoop 120
Debug a portlet application 216
declarative approach 373
DefaultPortletMessage 103, 227
Demilitarized Zone 11
deployment concerns 53
Directory services 5
Document Categories 42
Document Filter Technology 43
document filters 42

E
e-commerce 6
e-learning 6
encodeNameSpace 121
encodeURI 122
engine.tld 118
Eureka! categorizer 42
Event Handling 60
extend.tld 118
extranet 11

F
first generation portals 3
first portlet application 153
Flyweight pattern 22
Fourth generation portals 3

G
generations of portal technology 3
getURL 109

H
high availability 11
Highlights 37
Highlights in WebSphere Portal V5 37
Host integration 7

I
infrastructure 17
init 118
Installation
  Configuration and preparation of the workstation

Archived

533
Configuring WebSphere Studio Site Developer and the Portal Toolkit 527
hosts 506
Install the Cloudscape sample database 533
Installing a loopback adapter 502
Minimum machine requirements 502
Network Identification 504
Portal Toolkit 501
Prerequisites 502
Primary DNS suffix 504
Server Configuration 529
Software requirements 502
Software Update function from the help menu 510
Toolkit installation 524
Using Portal CDs 513
WebSphere Application Server - Express 508
WebSphere Application Server Interim Fixes 518
WebSphere Studio Site Developer 501, 506
WebSphere Studio Site Developer PTFs 506
installation 501
Internationalization 6–7
Internet 11
Inter-Portlet Communication 60

J

Java Community Process 13
JDBC 343
create a database connection 344
Create a new connection 344
Create a SQL statement 347
Data Definition view 346
DB Servers view 344
Generate Java classes 347
Importing the WAR file 359
Overview 353
Run the SQL statement 351
Jetspeed implementation 14

L

layout 20
Listeners 95
log 123
Loopback Adapter 503

M

menu.tld 118
MessageEvent 103, 227
MessageListener 102, 226
Mode 54
Model 154
Model-View-Controller 22, 154
Model-View-Controller (MVC) 447
ModeModifier 101
multiple devices 10
MVC
Model, View and Control 56
MVC architecture 55

N

National Language Support 249
Accessing resource bundles in JSPs 257
Accessing resource bundles in portlets 256
Adjusting Portal resource bundles 264
Creating Resource Bundles in WebSphere Studio 252
NLS administration 260
NLS best practices 265
Portal NLS administration 263
Portlet NLS administration 260
Resource bundles 250
Sample scenario 266
Setting NLS titles 263
Translating Resource Bundles 254
Translating whole resources 258
Working with characters 265
next-generation desktop 5
NLS administration 249, 260
NLS best practices 249, 265

O

Organization for the Advancement of Structured Information Standards 13
Overview 1, 464
Aggregation Module 16
Authentication Server 16
Authorization 40
Click-To-Action 46
Client to remote application 49
Comparison of V4.x Permission vs. V5.x Roles 20
Content Publishing 44
credentials 9

Archived

Customer Relationship Management 48
database structures 17
Document Categories and Summaries 42
e-Business needs 4
Enabling for Communities 38
Enterprise Resource Planning 48
Eureka! Categorizer 42
Event Broker 39
events 9
evolution 2
evolution process 3
First generation portals 3
Fourth generation portals 3
General Infrastructure 38
generations of portal technology 3
high availability 11
IBM Portlet Wiring Tool V5.0 47
J2EE platform 4
J2EE Security 39
Jetspeed implementation 10
LDAP 17
Member Subsystem 39
model-view-controller 26
Open Source Portal 10
Overview 4
Page content 15
page structure 15
page structure. 15
Permissions 20
Portal concepts 19
Portal Document Manager 43
Portal engine 16
Portal Install 37
Portal Servlet 16
portal technology 4
Portlet container 16
portlet container 27
Portlet events and messaging 29
portlets 9, 16
Presentation services 15
profile information 9
Property Broker 38
remote content 9
Reverse Proxy Security Server 12
Roles 20
Search 42
Second generation portals 3
Security services 17
services 17
Site Analysis 18
Skins 22
SSO Functionality 39
Struts Portlet Framework 45
Summarizer 43
Themes 21
Third generation portals 3
Toolkit 19
Transcoding 45
Transcoding Technology 17
User and Group Management 17
WebSphere Portal 16–17

P

Page Aggregation 59
Parameter summary 76
Passive Credential 115
passive credentials 323
person.tld 118
Personalization 6, 17
Pervasive computing 6
PortletRequest request 109
Portal access control 40
portal administrator 22
Portal Document Manager 43
Portal Framework 7, 9
Portal NLS administration 263
Portal technology 1–2
Portal Toolkit 47, 125
Adding portlets to applications 149
Additional markups 136
Additional modes 136
Application name 133
Basic portlet 138
Code generation options 133
Configure a server and server instance 146
Context Root 132
Default locale 133
Deploying portlets 146
Developing portlet applications 136
Enterprise Application project 131
Examples 150
Generated classes 138
IBM WebSphere Studio Workbench 128
images 138
J2EE level 132
J2EE Settings Page 131
Java Source 137

Archived

META-INF 137
Package 137
Portal Toolkit installation 128
Portlet application contents 137
Portlet Application wizard 129
Portlet name 133
Portlet title 133
Portlet.xml interface 140
Run the application 148
Web Content 137
WEB-INF 138
portals 1
Portlet 19, 83
Portlet API 53, 82
Abstract and concrete portlet applications 62
Action Event Handling 98
ActionEvent 99
ActionListener 99
Active Credentials 116
Administrator Managed 115
Attribute storage summary 107
Client 87
Configure 55
Control 57
Core Credential Vault Objects 114
Core event objects 99
Create the Service Factory 111
Credential Vault 113
Custom Services 109
DefaultPortletMessage 103
Define the Service 110
Edit 55
Event Handling 60
EventPhaseListener 106
Help 55
Hierarchy 82
Implement the Service 110
Inter-Portlet Communication 60
Listeners 95
MessageEvent 103
Mode 54
Model 56
Model View Control architecture 56
ModeModifier 101
MVC architecture 55
Page Aggregation 59
Parameter summary 76
Portlet deployment 61
Portlet JSPs 118
Portlet life cycle 53
Portlet messaging 102
Portlet Services 108
Portlet Tag Library 118
portlet terms 54
Portlet window 54
portlet.xml 67
PortletAdapter 83
PortletApplicationSettings object 90
PortletApplicationSettingsAttributesListener 98
PortletConfig object 88
PortletContext object 88
PortletData object 91
PortletException 93
PortletLog object 92
PortletMessage 104
PortletPageListener 95
PortletRequest 84
PortletResponse 85
PortletSession object 86
PortletSessionListener 97
PortletSettings object 89
PortletSettingsAttributesListener 98
PortletTitleListener 95
PortletURI 94, 100
PortletWindow object 93
PropertyListener 105
Real Estate 59
Register the Service 112
Security 60
Segment 114
Slot 115
State 54
Test the service 113
Third party authentication 113
Tivoli Access Manager 113
UnavailableException 93
User Managed 115
User object 94
Vault 114
View 55, 57
web.xml 64
Web.xml and Portlet.xml relationship 76
What is a portlet 54
What is a Portlet Application 61
WindowListener 97
Portlet application 19, 61
Portlet Containers 109
Portlet debugging 213

Archived

Add Breakpoint 217
Fix compile errors 214
Java validator 214
Step-by-Step Debug 220
Portlet deployment 61
Portlet development platform sample installation 501
Portlet life cycle 27, 53, 80
Portlet life cyle 80
Portlet messaging 102, 226, 372
Portlet Messaging vs. Cooperative Portlets 372
Portlet Modes 25
Portlet MVC architecture 57
Portlet MVC sample 58
Portlet NLS Administration 260
Portlet NLS administration 260
Portlet Preview 464
Define the Portlet Preview preference 467
Overview 464
Portlet Preview buttons 465
Preview Portlet 466
Preview preference 467
Sample scenario 467
Portlet preview 463
Portlet Private slot 115
Portlet solution patterns 48
Portlet states 26
portlet terms 54
Portlet to remote application 50
Portlet to Web application 51
Portlet window 54
portlet wiring tool 418
portlet.tld 118
portlet.xml 67
PortletApplicationSettings object 90
PortletApplicationSettingsAttributesListener 98
PortletConfig object 88
PortletContext object 88
PortletData object 91
PortletException 93
PortletLog object 92
PortletMessage 104, 228
PortletPageListener 95
PortletRequest 84
PortletResponse 85
PortletResponse response 109
Portlets 22
Core Portlet Objects 83
DefaultPortletMessage 103, 227
Portlet 83
Portlet API 82
Portlet deployment 61
Portlet JSPs 118
Portlet life cycle 80
Portlet MVC architecture 57
Portlet MVC Sample 58
Portlet Services 108
portlet terms 54
Portlet window 54
portlet.xml 67
PortletAdapter 83
PortletApplicationSettings object 90
PortletApplicationSettingsAttributesListener 98
PortletConfig object 88
PortletContext object 88
PortletData object 91
PortletException 93
PortletLog object 92
PortletPageListener 95
PortletRequest 84
PortletResponse 85
PortletSession object 86
PortletSessionListener 97
PortletSettings object 89
PortletSettingsAttributesListener 98
PortletTitleListener 95
PortletURI 94, 100
PortletWindow object 93
Servlets versus Portlets 59
Web.xml and Portlet.xml relationship 76
What is a Portlet Application? 61
WindowListener 97
PortletSession object 86
PortletSessionListener 97
PortletSettings 24
PortletSettingsAttributeListener 98
PortletSettingsAttributesListener 98
PortletTitleListener 95
PortletURI 94, 100
Preview Portlet 474
programmatic approach 374
property broker 382

R

Redbooks Web site 546
Contact us xvi
Register the Service 112

Archived

relational database 353
remote content 9
Remote Server Attach 477
  Add Breakpoint 497
  BSF debug port 486
  Debug Mode 482
  Disconnect 500
  Enable JavaScript debugging 486
  Exporting the portlet application 487
  HTTP port 486
  Install a portlet in Remote Portal 486
  Installing the portlet in Portal using Portal Administration 487
  JVM debug port 486
  Preparing Portal for Remote Server Attach 479
  Process Definition 480
  Remote Server Attach configuration 484
  Run the portlet 496
  Server Configuration 496
  Test Remote Server Attach 496
Remove Breakpoint 222
Reset Preview Portlet 472
Resource bundles 249–250
Resources 124

S

Search 6, 18
search and taxonomy 7
search functionality 42
Second generation portals 3
Segment 114
Service Provider Interface 40
ServletResponse 85
Servlets vs. Portlets 59
Setting NLS titles 263
settingsAttribute 120
settingsLoop 121
Shared slot 115
simple action Strings 390
Single Sign-On 161, 320
site analytics 7
Site usage 7
Slot 115
Standard MVC architecture 56
State 54
String url 109
Struts actions 456
Struts applications
  action processing 449
  components 448
  controller 448
  custom JSP tags 449
  Java developer subclasses 448
  model 448
  Overview 448
  portlet environment 449
  rendering 449
  resources 457
  Struts Portlet Framework 45, 447
  struts-config.xml 448
  view 448
  Web application 456–457
  Web page designer 454
Struts portlet 459
Summarizer 43
System slot 115

T

target portlet application 423
Test Environment 330
Test the service 113
text 123
The Portlet Wiring Tool 47
Third generation portals 3
Tivoli Access Manager lock box 322
Tooling 18
Toolkit installation 125
Toolkit V5 125
Transcoding technology 45
Translating Resource Bundles 254
Translating whole resources 249, 258

U

UID Guidelines 77
Update the portlet project 171
URIAction 119
URIParameter 120
URL 41
URL Generation, Processing and Mappings 41
URL Mappings 41
User object 94
User-friendly Portal URLs 41
Using resource bundles 249

Archived

V

validator 214
vault segment 321
View 154

W

Web crawler 43
Web Service client 291
Web Service Description Language 374
Web Services 18, 316
Web Services Client
  Create a Web Service 299
  Create a Web Services client portlet 308
  Generate a proxy 300
  sample Web Service 298
  simple Web Service project 293
  Test the generate proxy 300
  Web Service Bean Identity 302
  Web Service Bind Proxy Generation 304
  Web Service Client Portlet Project 309
  Web Service Deployment Settings 301
  Web Service Java Bean Selection 302
  Web Service Portlet Parameters 311
  Web Services 300
Web Services for Remote Portals 13
web.xml 64
Web-based content 9
WebSphere Portal 19
WebSphere Portal content publishing 44
WebSphere Portal Property Broker 373
WebSphere Portal Toolkit V5.0 524
WebSphere Studio Application Developer 53
WebSphere Test Environment Express 293
What is a portlet 54
What is a Portlet Application 61
WindowListener 97
wiring tool
  download 419
Working with characters 249, 265

Archived

Redbooks

IBM WebSphere Portal V5 A Guide for Portlet Application Development

(1.0" spine)
0.875"<->1.498"
460 <-> 788 pages

Archived



IBM WebSphere Portal V5 A Guide for Portlet Application Development



Learn about Portal Toolkit and portlet application development

Actions, messaging, Credential Vault, cooperative portlets

Access Web Services from portlet applications

This IBM Redbook helps you design, develop and implement portlet applications using the IBM WebSphere Studio Site Developer and the Portal Toolkit V5. The information provided in this redbook targets Business-to-Employee (B2E) enterprise applications, but most of the scenarios presented apply to Business-to-Consumer (B2C) applications as well. In this redbook, you will find step-by-step examples and scenarios showing ways to integrate your enterprise applications into an IBM WebSphere Portal environment using the WebSphere Portal APIs provided by the Portal Toolkit to develop portlets as well as extend your portlet capabilities to use other advanced functions such as cooperative portlets, national language support, action events, portlet messaging, Credential Vault, Web Services and portlet debugging capabilities.

Elements of the portlet API are described and sample code is provided. The scenarios included in this redbook can be used to learn about portlet programming and as a basis to develop your own portlet applications. You will also find numerous scenarios describing recommended ways to develop portlets and portlet applications using the APIs provided by the IBM WebSphere Portal Toolkit. The sample scenarios in this redbook have been developed using the WebSphere Studio Site Developer but they can also be developed using the WebShere Studio Application Developer. A basic knowledge of Java technologies such as servlets, JavaBeans, EJBs, JavaServer Pages (JSPs), as well as XML applications and the terminology used in Web publishing, is assumed.

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

SG24-6076-00 ISBN 0738498513