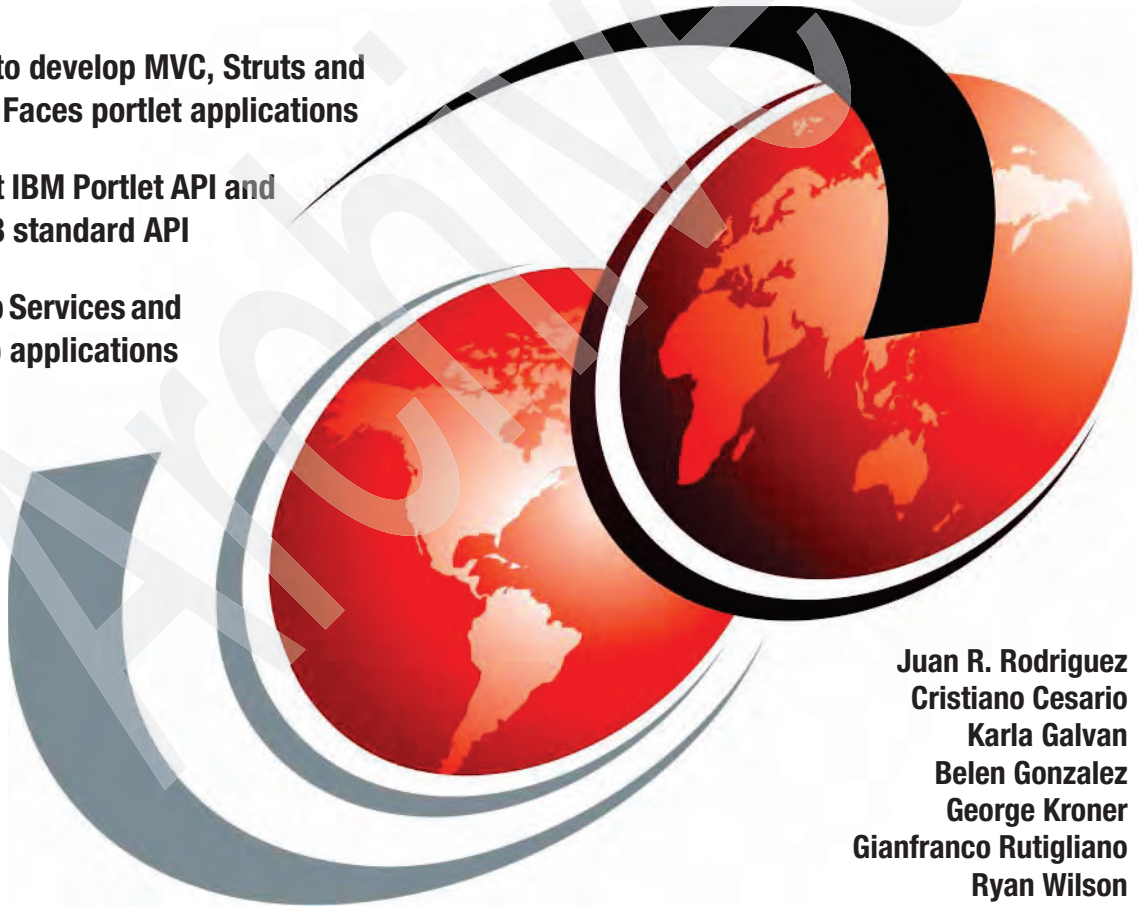
WebSphere. software

IBM

IBM Rational Application Developer V6 Portlet Application Development and Portal Tools

- Learn how to develop MVC, Struts and JavaServer Faces portlet applications
- Learn about IBM Portlet API and the JSR 168 standard API
- Access Web Services and secure Web applications



Juan R. Rodriguez
Cristiano Cesario
Karla Galvan
Belen Gonzalez
George Kroner
Gianfranco Rutigliano
Ryan Wilson

Redbooks

ibm.com/redbooks

Petitioner has added letters to pages (a)-(c) and numbers to pages 1021-1022. Otherwise, it leaves the original page numbering.

SAP AMERICA, INC.
EXHIBIT 1116

(a)



International Technical Support Organization

IBM Rational Application Developer V6 Portlet Application Development and Portal Tools

August 2005

Archived

SG24-6681-00

(b)

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

Archived

First Edition (August 2005)

This edition applies to Version 6 of IBM Rational Application Developer and Version 5.1 of IBM WebSphere Portal.

© Copyright International Business Machines Corporation 2005. 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 xvii
Trademarks xviii

Preface xix
The team that wrote this redbook xix
Become a published author xxii
Comments welcome xxii

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 9
1.2.5 WebSphere Portal tooling 17
1.3 WebSphere Portal 18
1.3.1 Portal concepts 18
1.3.2 Portlets 21
1.3.3 The model-view-controller (MVC) design pattern 23
1.3.4 Standard MVC architecture 23
1.3.5 Portlet MVC architecture 24
1.3.6 Portlet MVC sample 25
1.3.7 WebSphere Portal runtime: the portlet container 26
1.3.8 Page aggregation 26
1.4 Highlights in WebSphere Portal V5.1 31
1.4.1 Portal install 32
1.4.2 General infrastructure 32
1.4.3 Event broker 32
1.4.4 Member subsystem 33
1.4.5 Authentication 33
1.4.6 Authorization 33
1.4.7 URL generation, processing and mappings 35
1.4.8 Search 35
1.4.9 Content management 36
1.4.10 Transcoding 37
1.4.11 Struts Portlet Framework 37

© Copyright IBM Corp. 2005. All rights reserved.

1.4.12 JSF Portlet Runtime 38
1.4.13 User interface 38
1.4.14 Cooperative portlets (Click-To-Action) 39
1.4.15 Portal Toolkit 40
1.5 Portlet solution patterns 41
1.6 Building a war file 44

Chapter 2. Developing Portal applications 47
2.1 Portal overview 48
2.1.1 Portal concepts and definitions 48
2.1.2 IBM WebSphere Portal 51
2.1.3 IBM Rational Application Developer 51
2.2 Developing applications for WebSphere Portal 54
2.2.1 Portal samples and tutorials 54
2.2.2 Development strategy 55
2.2.3 Portal tools for developing portals 58
2.2.4 Portal tools for developing portlets 65
2.2.5 Portal tools for testing and debugging portlets 78
2.2.6 Portal tools for deploying and managing portlets 82
2.2.7 Enterprise Application Integration Portal tools 84
2.2.8 Coexistence and migration of tools and applications 85
2.3 Portal development scenario 86
2.3.1 Preparing for the sample 87
2.3.2 Creating a portal project 88
2.3.3 Adding and modifying a portal page 89
2.3.4 Creating and modifying two portlets 92
2.3.5 Adding portlets to a portal page 95
2.3.6 Running the project in the test environment 98

Chapter 3. Portlet development platform sample installation 101
3.1 Prerequisites 102
3.1.1 Hardware requirements 102
3.1.2 Software requirements 102
3.2 Rational Application Developer and Portal Tools 102
3.3 WebSphere Portal V5.1 Test Environment 106
3.4 Configuration of the Test Environment 110
3.5 WebSphere Test Environment V5.1 (optional) 113

Chapter 4. IBM Portlet API 115
4.1 IBM portlets 116
4.2 IBM portlet application 116
4.3 Servlets versus portlets 117
4.4 Portlet modes 119
4.5 Portlet states 119

4.6 Core objects . . . 120
4.6.1 Hierarchy . . . 120
4.6.2 Portlet . . . 121
4.6.3 PortletAdapter . . . 121
4.6.4 PortletRequest . . . 121
4.6.5 PortletResponse . . . 123
4.6.6 PortletSession object . . . 124
4.6.7 Client . . . 125
4.6.8 PortletConfig object . . . 126
4.6.9 PortletContext object . . . 127
4.6.10 PortletSettings object . . . 128
4.6.11 PortletApplicationSettings object . . . 129
4.6.12 PortletData object . . . 130
4.6.13 PortletLog object . . . 131
4.6.14 PortletException . . . 132
4.6.15 UnavailableException . . . 132
4.6.16 PortletWindow object . . . 132
4.6.17 User object . . . 133
4.6.18 PortletURI . . . 133
4.7 Portlet life cycle . . . 134
4.8 Listeners . . . 137
4.8.1 PortletTitleListener . . . 137
4.8.2 PortletPageListener . . . 138
4.8.3 PortletSessionListener . . . 139
4.8.4 WindowListener . . . 140
4.8.5 PortletSettingsAttributeListener . . . 141
4.8.6 PortletApplicationSettingsAttributesListener . . . 141
4.9 Action event handling . . . 141
4.9.1 ActionListener . . . 142
4.9.2 ActionEvent . . . 142
4.9.3 PortletURI . . . 142
4.9.4 ModeModifier . . . 144
4.10 Attribute storage summary . . . 145
4.11 Portlet JSPs . . . 146
4.11.1 Portlet tag library . . . 147
4.11.2 Portlet events and messaging . . . 152
4.12 Portlet deployment . . . 155
4.12.1 web.xml . . . 157
4.12.2 portlet.xml . . . 160
4.12.3 Parameter summary . . . 168
4.12.4 Descriptors relationship (web.xml and portlet.xml) . . . 170
4.12.5 UID guidelines . . . 171
4.13 Resources . . . 171

Archived

Chapter 5. A first portlet application 173
5.1 Sample scenario 174
5.2 Creating the portlet project 174
5.2.1 Using the Portlet Project wizard 175
5.3 Configuring the test environment 185
5.4 Running the portlet project 188
5.5 Modifying the portlet project and verifying changes 192
5.5.1 Changing the JSP used for the View mode 192
5.5.2 Adding a JavaBean 196

Chapter 6. IBM Portlet API portlet development 203
6.1 About action events 204
6.2 Development scenario 207
6.3 Creating the portlet project 209
6.4 Configuring your project in the test environment 214
6.5 Examining and modifying the source code 216
6.6 Running your project in the test environment 219

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 230
7.6.1 Description 231
7.6.2 Sending a message 232
7.6.3 Creating the target portlet 235
7.6.4 Running the portlet application 240
7.7 Broadcasting messages 242

Chapter 8. JSR 168 API 251
8.1 JSR overview 252
8.1.1 Number of portlet instances 252
8.1.2 Portlet windows 252
8.1.3 Thread safety 253
8.2 JSR 168 comparison to servlets 254
8.3 JSR 168 portlet modes 254
8.4 JSR 168 Portlet window states 256
8.5 Core JSR 168 objects 257
8.5.1 interface javax.portlet.Portlet 257
8.5.2 class javax.portlet.GenericPortlet 259
8.5.3 interface javax.portlet.PortletURL 261
8.5.4 interface javax.portlet.PortletContext 263

Archived

8.5.5 interface javax.portlet.PortletRequest 264
8.5.6 interface javax.portlet.ActionRequest 267
8.5.7 interface javax.portlet.RenderRequest 270
8.5.8 interface javax.portlet.PortletResponse 270
8.5.9 interface javax.portlet.ActionResponse 271
8.5.10 interface javax.portlet.RenderResponse 272
8.5.11 interface javax.portlet.PortalContext 274
8.5.12 interface javax.portlet.PortletPreferences 274
8.5.13 interface javax.portlet.PreferencesValidator 276
8.5.14 interface javax.portlet.PortletConfig 277
8.5.15 interface javax.portlet.PortletSession 278
8.6 JSR 168 Portlet life cycle 280
8.6.1 Instantiation 280
8.6.2 Initialization 280
8.6.3 Request handling 281
8.6.4 End of service 282
8.7 Portlet caching 283
8.7.1 Remote cache 283
8.8 Listeners 284
8.8.1 HttpSessionBindingListener 284
8.8.2 ServletContextListener 285
8.8.3 ServletContextAttributeListener 285
8.8.4 HttpSessionListener 285
8.8.5 HttpSessionAttributeListener 285
8.9 Deployment descriptors 287
8.9.1 Portlet.xml declaration 288
8.9.2 portlet-app - *required, can occur only once* *288*
8.9.3 portlet - *can occur zero or more times* *289*
8.9.4 custom-portlet-mode - *can occur zero or more times* *295*
8.9.5 custom-window-state - *can occur zero or more times* *296*
8.9.6 user-attribute - *can occur zero or more times* *296*
8.9.7 security-constraint - *can occur zero or more times* *298*
8.10 JSR 168 limitations in WebSphere Portal 299

Chapter 9. JSR 168 portlet development 301
9.1 Overview 302
9.2 Creating a JSR 168 portlet project 302
9.2.1 Creating a basic JSR 168 portlet 303
9.2.2 Examining the generated portlet 306
9.3 Updating the generated portlet 306
9.3.1 Modifying the session bean 307
9.3.2 View mode 310
9.3.3 Edit mode 317

9.3.4 Configure mode . . . 326
9.3.5 Updating the portlet descriptor (portlet.xml) . . . 329
9.3.6 Modifying the MySimplePortletPortletPreferenceValidator class . . . 331
9.4 Running the portlet . . . 333
9.4.1 Executing the portlet . . . 333

Chapter 10. Migrating to JSR 168 . . . 335
10.1 Modifying the deployment descriptor . . . 336
10.1.1 doctype . . . 336
10.1.2 portlet-app . . . 336
10.1.3 concrete-portlet-app . . . 337
10.1.4 portlet . . . 337
10.1.5 portlet-name . . . 337
10.1.6 web.xml . . . 337
10.1.7 cache . . . 337
10.1.8 supports . . . 338
10.1.9 allows . . . 339
10.1.10 config-param . . . 339
10.1.11 Locale settings . . . 339
10.2 Modifying the Java source . . . 340
10.2.1 Package . . . 340
10.2.2 Superclass . . . 341
10.2.3 doXXX methods . . . 341
10.2.4 actionPerformed . . . 341
10.2.5 ActionEvent . . . 341
10.2.6 Logging . . . 342
10.2.7 JSP includes . . . 342
10.2.8 PortletData and PortletSettings . . . 343
10.2.9 namespace . . . 343
10.2.10 portlet URLs . . . 343
10.3 Modifying the JSP source . . . 344
10.3.1 taglib . . . 344
10.3.2 portletAPI:init . . . 345
10.3.3 namespace . . . 345
10.3.4 Creating URLs . . . 346
10.3.5 portletAPI:text . . . 347
10.3.6 encodeURL . . . 347
10.3.7 CSS . . . 347
10.4 Struts . . . 347
10.5 JSF . . . 348
10.6 Portlet services . . . 348
10.7 Messaging . . . 351

Arrived

Chapter 11. Using JSPs and servlets 353
11.1 Overview 354
11.1.1 Generating output 354
11.2 RequestDispatcher 356
11.2.1 PortletContext.getRequestDispatcher 356
11.2.2 PortletContext.getNamedDispatcher 356
11.2.3 PortletRequestDispatcher.include 356
11.3 JSP tags 360
11.3.1 defineObjects 361
11.3.2 renderURL 361
11.3.3 actionURL 362
11.3.4 namespace 363
11.3.5 param 364
11.3.6 IBM tags 364
11.3.7 JSTL 365
11.4 Cascading style sheets (CSS) 366
11.4.1 WSRP Styles 366
11.4.2 IBM styles 370

Chapter 12. Internationalization 373
12.1 Resource bundles 374
12.1.1 Creating resource bundles in Rational Application Developer . . . 376
12.1.2 Translating resource bundles 379
12.1.3 Accessing resource bundles in portlets 381
12.1.4 Accessing resource bundles in JSPs 382
12.2 Translating whole resources 383
12.3 JSR 168 API considerations 384
12.4 Dynamically changing the language 385
12.5 NLS administration 386
12.5.1 Portlet NLS administration 386
12.5.2 Portal NLS administration 390
12.5.3 Setting NLS titles 390
12.5.4 Supporting a new language 391
12.6 Working with characters 392
12.7 NLS best practices 392
12.8 Sample scenario: NLS bundles 393
12.8.1 NLS bundles 394
12.8.2 Accessing NLS bundles from JSPs 399
12.8.3 Running the NLS scenario 400
12.8.4 Accessing NLS bundles in Java portlets 404
12.9 Sample scenario: translating whole resources 406
12.10 Dynamically changing the language 411

Approved

Chapter 13. Struts portlets 415
13.1 Overview 416
13.2 The Struts portlet framework 419
13.2.1 Struts applications 420
13.2.2 Changes to Struts JSPs 423
13.2.3 Configuration files 426
13.2.4 Creating link tags in Struts 426

Chapter 14. Creating Struts portlets with the IBM Portlet API 429
14.1 Overview 430
14.2 Creating Struts applications with IBM portlet API 430
14.2.1 Creating a Portlet project 430
14.2.2 Inspecting the Struts portlet project 433
14.2.3 Designing the application 434
14.2.4 Realizing the application components 436
14.2.5 Adding logging support 444
14.2.6 Adding support to Edit mode 446
14.2.7 Realizing the new application components 450
14.2.8 Adding internationalization support 460
14.3 Messaging 462
14.4 Migration 463

Chapter 15. Struts portlet development using the JSR 168 API 465
15.1 Overview 466
15.2 Message flow 467
15.3 Creating a Portlet project 469
15.3.1 Inspecting the Struts portlet project 475
15.4 Designing the application (View mode) 477
15.4.1 Realizing the application components 480
15.4.2 Realizing the index.jsp page 482
15.4.3 Realizing the notConfigured.jsp 486
15.4.4 Realizing the Form Bean 486
15.4.5 Realizing configured.jsp 489
15.4.6 Editing the resources file 490
15.4.7 Realizing the welcome action mapping 490
15.4.8 Running the Struts portlet 493
15.5 Designing the application (Edit mode) 494
15.6 Realizing the Edit mode application components 495
15.6.1 Realizing the editBean 495
15.6.2 Realizing index.jsp 498
15.6.3 Realizing the saveConfiguration action 499
15.7 Adding new keys to the resources file 500
15.8 Running the portlet 501

15.9 Adding internationalization support 502
15.9.1 Portlet application View mode internationalization 503
15.9.2 Struts framework internationalization support 504
15.9.3 Editing the Portlet deployment descriptor 505
15.9.4 Testing the new locale 505
15.10 Adding logging support to the application 506

Chapter 16. JavaServer Faces portlets 511
16.1 Overview 512
16.1.1 Life cycle of a JSF page 514
16.2 A simple JSF application 518
16.2.1 Creating the pages 519
16.2.2 Defining navigation rules 521
16.2.3 Developing the beans 523
16.3 User interface component model 525
16.3.1 User interface component classes 525
16.3.2 Component rendering model 527
16.3.3 Conversion model 531
16.3.4 Event and listener model 535
16.3.5 Validation model 536
16.4 Navigation model 539
16.5 Backing bean management 540
16.6 JSF in portlets 542
16.6.1 JSF portlet runtime 542
16.6.2 Mapping between portlet phases and JSF phases 544
16.6.3 Welcome page and navigation in JSF portlets 545
16.6.4 Programming guidelines 546
16.6.5 Limitations in JSF portlets 548
16.7 Migration 548

Chapter 17. JavaServer Faces portlet development 551
17.1 The calculator application 552
17.2 Creating the project 552
17.2.1 Inspecting the JSF portlet project 555
17.3 Creating the page layout 558
17.4 Implementing component attributes and validation 563
17.4.1 Testing the validation 567
17.5 Binding the front end to the calculator 568
17.5.1 Testing the binding 572
17.6 Invoking the business logic of the calculator 572
17.6.1 Implementing an error page 573
17.7 Implementing page navigation 577
17.8 Implementing a validator 580

Archived

17.9 Implementing a value change event 583
17.10 Implementing internationalization 587
17.10.1 Internationalization of standard validator messages 590

Chapter 18. Additional Faces portlet sample scenarios 593
18.1 The call center application 594
18.1.1 Creating the project 594
18.1.2 Creating the page layout 595
18.1.3 Defining a parameter for the list 599
18.1.4 Creating a detail portlet 601
18.1.5 Linking the portlets 602
18.1.6 Testing the application 605
18.2 The Web service client portlet 606
18.2.1 Creating the project 606
18.2.2 Creating a new Web service client 607
18.2.3 Creating the page layout 612
18.2.4 Testing the application 615

Chapter 19. Portlet services 617
19.1 Portlet services 618
19.1.1 ContentAccessService 618
19.1.2 Custom services 620
19.2 Accessing portlet services 627
19.2.1 Accessing a portlet service in an IBM portlet 627
19.2.2 Accessing a portlet service in a JSR 168 portlet 627

Chapter 20. Credential Vault Service 629
20.1 Overview 630
20.1.1 Credentials 630
20.2 Credential Vault organization 631
20.2.1 Vault segments 632
20.2.2 Credential slots 633
20.3 Working with the CredentialVaultService 634
20.3.1 Acquiring a reference to the CredentialVaultService 634
20.3.2 Using the CredentialVaultService 634
20.4 Credential objects 635
20.4.1 Passive credential objects 636
20.4.2 Active credential objects 637
20.4.3 Storing credential objects in the PortletSession 640

Chapter 21. The Credential Vault 641
21.1 Sample scenario 642
21.2 Importing a secure servlet application 644
21.3 Using active credentials 648

Archived

21.3.1 Creating the Credential Vault portlet application 648
21.3.2 Reviewing the generated code 651
21.3.3 Updating the generated portlet 653
21.3.4 Running the portlet 662
21.4 Using passive credentials 667

Chapter 22. Accessing JDBC databases from portlet applications 669
22.1 Creating a portlet project 670
22.1.1 Creating HRPortlet 670
22.2 Creating a sample database 675
22.2.1 Creating the WSSAMPLE database 675
22.2.2 Creating a connection 678
22.2.3 Creating an SQL statement 683
22.2.4 Generating Java classes 685
22.3 Sample scenario 689
22.3.1 Overview 690
22.3.2 Importing the WAR file 692
22.3.3 Reviewing the portlet code 693
22.3.4 Running the HRPortlet application 698

Chapter 23. Accessing JDBC databases using Data Source in standard portlets 703
23.1 Data Source overview 704
23.2 Creating a JSR 168 portlet project 705
23.2.1 Creating HRPortlet 705
23.3 Creating a sample database 710
23.3.1 Creating a connection 710
23.3.2 Creating an SQL statement 716
23.3.3 Generating Java classes 719
23.4 Sample scenario 724
23.4.1 Overview 725
23.4.2 Importing the WAR file 727
23.4.3 Reviewing the portlet code 729
23.4.4 Creating the Data Source 733
23.4.5 Running the HRPortlet168 application 736

Chapter 24. IBM API declarative cooperative portlets 741
24.1 Overview 742
24.1.1 The WebSphere Portal property broker 743
24.1.2 Property broker runtime components 744
24.2 IBM Portlets for cooperation 744
24.2.1 Registering and publishing properties 745
24.2.2 Struts integration 746
24.2.3 Internationalization 746

Archived

24.3 Sample scenario (IBM portlets) . . . 747
24.3.1 Description . . . 747
24.3.2 Source cooperative portlet . . . 750
24.3.3 Target cooperative portlet . . . 761
24.3.4 Running the cooperative portlets . . . 765

Chapter 25. IBM API programmatic cooperative portlets . . . 771
25.1 Publishing properties programmatically . . . 772
25.2 Portlet event handling . . . 773
25.2.1 PropertyListener interface . . . 775
25.2.2 EventPhaseListener interface . . . 776
25.3 Broadcasting source data . . . 777
25.4 Wiring tool . . . 778
25.5 Sample scenario . . . 779
25.5.1 Declarative source cooperative portlet . . . 779
25.5.2 Enabling the portlet for target C2A programmatic . . . 781
25.5.3 Running the cooperative portlets . . . 789
25.5.4 Wire portlets . . . 793
25.5.5 Enabling HRPortlet for programmatic source C2A . . . 794
25.5.6 Running the programmatic source portlet . . . 798

Chapter 26. JSR 168 cooperative portlets . . . 801
26.1 Overview . . . 802
26.2 Source cooperative portlet . . . 803
26.2.1 Importing the HRPortlet168 portlet . . . 804
26.2.2 Internationalization . . . 805
26.2.3 Declaring exchange capabilities using WSDL . . . 806
26.2.4 Updating the portlet deployment descriptor . . . 809
26.2.5 Updating the HRPortlet168 portlet code . . . 811
26.2.6 Updating the JSP to generate a link . . . 815
26.3 Target cooperative portlet . . . 816
26.3.1 Internationalization . . . 818
26.3.2 Declaring exchange capabilities using WSDL . . . 819
26.3.3 Updating the portlet deployment descriptor . . . 821
26.4 Running the cooperative portlets . . . 824
26.4.1 Populating the sample database . . . 824
26.4.2 Creating a data source . . . 825
26.4.3 Running and wiring the cooperative portlets . . . 828

Chapter 27. Struts cooperative portlets . . . 835
27.1 Overview . . . 836
27.2 Source cooperative portlet . . . 837
27.2.1 Importing the Echo portlet . . . 837
27.2.2 Declaring exchange capabilities using WSDL . . . 838

27.2.3 Updating the portlet deployment descriptor 841
27.2.4 Updating the EchoSource portlet code 842
27.2.5 Internationalization 844
27.3 Target cooperative portlet 844
27.3.1 Importing the Echo portlet 845
27.3.2 Declaring exchange capabilities using WSDL 846
27.3.3 Updating the EchoTarget portlet code 849
27.3.4 Internationalization 850
27.4 Running the cooperative portlets 851

Chapter 28. Accessing Web Services from portlet applications 857
28.1 Overview 858
28.2 Sample scenario 858
28.2.1 Creating a Web Service 859
28.2.2 Creating a Web Services client portlet 874

Chapter 29. Web Services for Remote Portlets (WSRP) 885
29.1 Overview 886
29.2 Implementing WSRP in WebSphere Portal 888
29.2.1 Tasks for Producer portals 889
29.2.2 Tasks for Consumer portals 892
29.2.3 Testing the scenario 898
29.3 Security 899

Chapter 30. Portlet debugging 901
30.1 Overview 902
30.2 Sample scenario 902
30.2.1 Fixing compile errors 902
30.2.2 Debugging a portlet application 905

Chapter 31. Remote Server Attach 915
31.1 Overview 916
31.2 Sample scenario 917
31.2.1 Preparing Remote Portal server to debug 917
31.2.2 Creating Remote Portal server users 920
31.2.3 Creating a WebSphere Portal Server Attach 921
31.2.4 Debugging a portlet on WebSphere Portal Server Attach 924
31.3 Defining Web browsers and emulator devices 932

Chapter 32. Updating a portal layout 935
32.1 Overview 936
32.2 Creating a Portal server connection 937
32.3 Importing the WebSphere Portal server configuration 943
32.4 Modifying the Portal Navigation and Layout 945

Archived

32.5 Adding portlets 950
32.6 Additional ways to add portlets 952
32.7 Testing the updated portal configuration 954
32.8 Applying themes 957

Chapter 33. Creating new portal themes 961
33.1 Overview 962
33.2 Creating a new theme 962
33.3 Editing a theme 965
33.4 Editing Styles 976
33.5 Applying MyCorp theme 984
33.6 Applying a Skin 985
33.7 Testing the new Portal Project 986
33.8 Publishing the Portal Project 990
33.9 Deploying the Portal Project 993

Appendix A. Additional material 1003
Locating the Web material 1003
Using the Web material 1003
System requirements for downloading the Web material 1004
How to use the Web material 1004

Related publications 1005
IBM Redbooks 1005
Other publications 1006
Online resources 1006
How to get IBM Redbooks 1007
Help from IBM 1007

Index 1009

Archive

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. 2005. All rights reserved.

Trademarks

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

@server®	DPI®	Redbooks™
@server®	Everyplace®	Roma®
Redbooks (logo) ™	Illustra™	SecureWay®
ibm.com®	Informix®	Tivoli®
z/OS®	IBM®	TME®
ClearCase®	IMS™	WebSphere®
Cloudscape™	Lotus Notes®	Workplace™
CrossWorlds®	Lotus®	Workplace Web Content
CICS®	Notes®	Management™
DB2®	Rational®	

The following terms are trademarks of other companies:

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. 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.

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

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

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

Archived

Preface

This IBM Redbook provides an overview and hands-on scenarios to help you design, develop and implement portlet applications using the Rational® Application Developer V6.0 and the provided Portal Tools. The sample scenarios included in this redbook target Business-to-Employee (B2E) enterprise applications, but most of the scenarios presented will also apply to Business-to-Consumer (B2C) applications. 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 Tools to develop portlets. You will also learn how to extend your portlet capabilities to use other advanced functions such as cooperative portlets, internationalization, action events, using the Credential Vault to enable Single Sign-On, Web Services, remote portlets, portal design and portlet debugging capabilities.

Elements of the Portlet API and the standard JSR168 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 that follow the MVC design pattern, the Struts framework and JavaServer Faces technology.

A basic knowledge of Java™ technologies such as servlets, JavaBeans, EJBs, JavaServer Pages (JSPs), as well as of 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 IBM ITSO, Raleigh Center. He received his M.S. degree in Computer Science from Iowa State University. He writes extensively and teaches IBM classes worldwide on such topics as networking, pervasive computing, Web technologies, and information security. Before joining the IBM ITSO, he worked at IBM Research Triangle Park, North Carolina as a designer and developer of networking products.

© Copyright IBM Corp. 2005. All rights reserved.

Cristiano Cesario is an Application IT Architect at IBM Global Services in Brazil. He has experience in WebSphere products and Portal solutions. He holds a Bachelor's degree in Computer Systems Enginnering from the Universidade do Estado do Rio de Janeiro (UERJ) in Brazil and a Post-Graduate Degree in Information Systems Project and Management from Universidade Federal do Rio de Janeiro (UFRJ) in Brazil.



Karla Galvan is a WebSphere IT Specialist at IBM Software Group in Monterrey, Mexico. She is responsible for Application Integration and Middleware (AIM) Technical Sales in the northeast region of Mexico. She holds a Bachelor's degree in Computer Science from Universidad Autónoma de Nuevo León, Mexico. Her areas of expertise include WebSphere Portal, Workplace™ Web Content Manager and Business Intregration Solutions.



Belen Gonzalez is an IT Specialist in IBM Global Services in IBM Spain. She has seven years of experience in the e-business field and three years of experience in WebSphere Portal products. She holds a degree in Computer Science Engineering from Universidad Autonoma de Madrid. Her areas of expertise include J2EE application development with WebSphere Studio and Rational Application Developer and e-commerce projects.



George Kroner is a recent graduate of Penn State University, having pursued a Bachelor of Science degree in Information Sciences and Technology. He has several years of experience in portal development and design. His interests include Web applications, wireless mobile applications and integration, portlet applications, and innovative business strategy.



Gianfranco Rutigliano holds a degree in Systems Engineering from the University of Lima (Peru) and has been a member of IBM Global Services. He has experience designing and implementing pervasive and Java-based Web solutions. He is now a Software Architect for Avatar e-Business Solutions, an IBM Business Partner, and has participated in several ITSO residencies for pervasive computing and WebSphere technologies.



Ryan Wilson is a WebSphere Portal support IT Specialist in RTP, North Carolina. He was the technical team lead for WebSphere Studio site developer support before joining Portal. His areas of expertise include J2EE application development with IBM WebSphere Studio and Rational Application Developer. He has participated in many projects, including internal tools development.



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

Masato Noguchi
IBM Software Group, Japan

Marshall Lamb, Ryan E. Smith, Shannon Pixley
IBM Research Triangle Park, North Carolina, USA

Stefan Hepper, Oliver Koeth
IBM Software Group, Germany

Cecilia Bardy
International Technical Support Organization, Raleigh Center

Archived

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

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 email 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 a best of breed solution.

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. 2005. 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, we see that the advent of using windows 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.

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 with a complete set of comprehensive tools to enhance their productivity.

These 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 based on business rules. The key to their further evolution is their open framework for common services.

Archived

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.

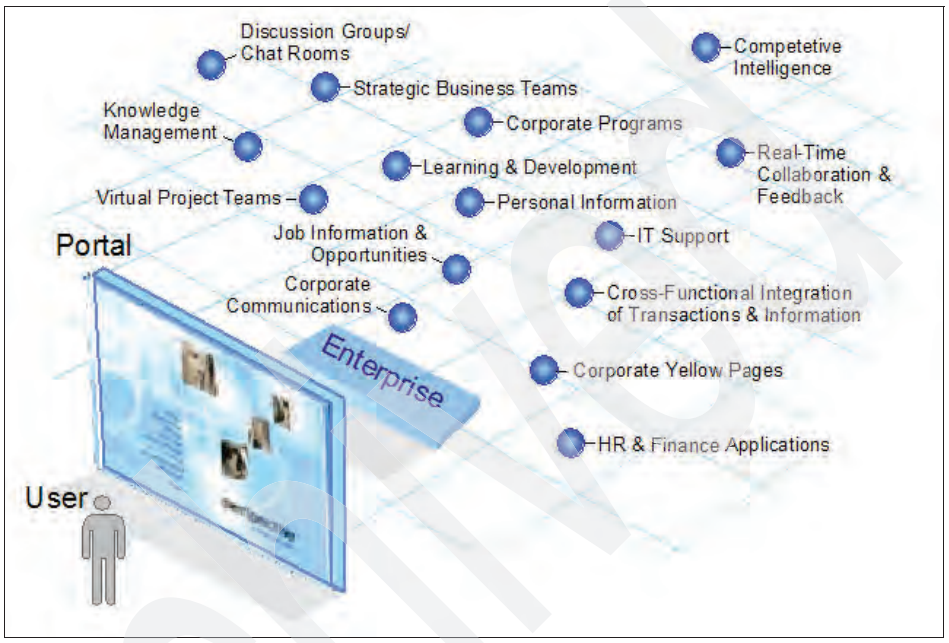


Figure 1-1 e-business needs

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 generate revenue for the clients.

Most companies have developed their Business to Consumer (B2C), Business to Business (B2B) and Business to Employee (B2E) strategies. Many 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 integration with legacy systems. Companies are often faced with the decision of whether to build or 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 uses, 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.

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.

Archive

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.

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.

Archived

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) 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.

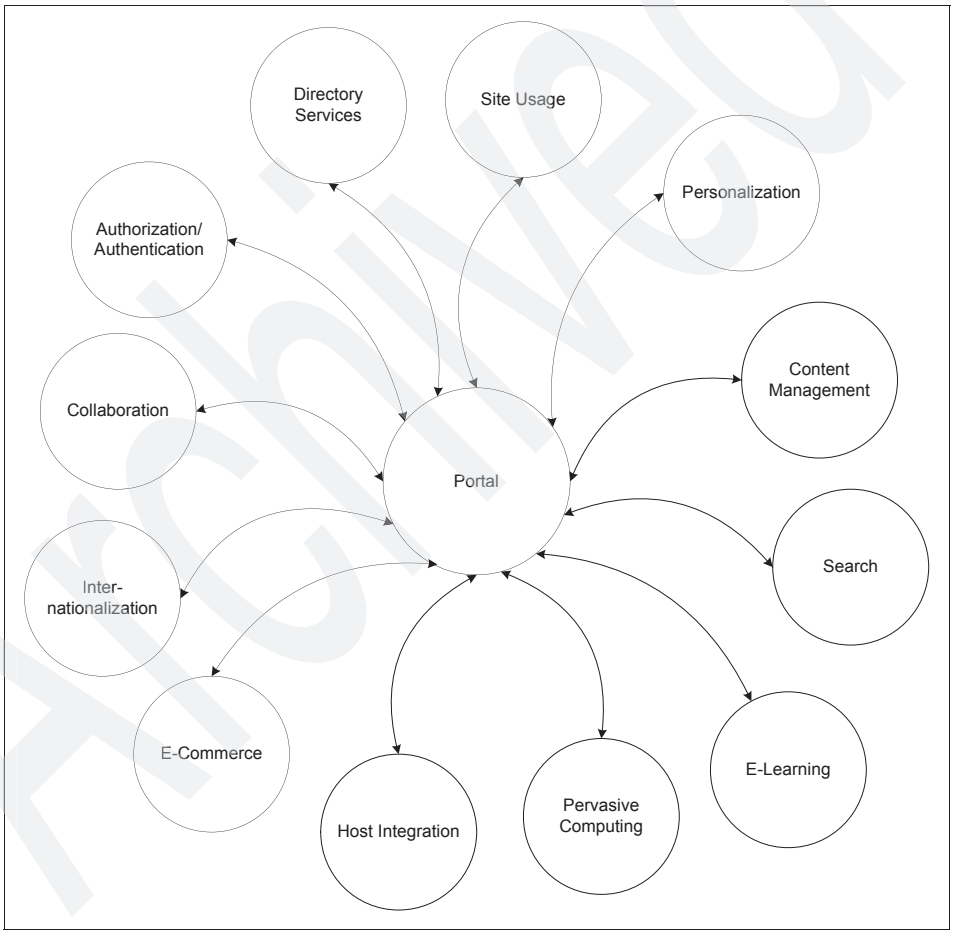


Figure 1-2 Portal context diagram

IBM WebSphere Portal provides a single, secure, interactive point of access to dynamic applications, information, people and processes to help build successful

Archived

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 5.1 platform, using J2EE standards to optimize performance
- Provides integrated Web services so you can quickly deploy portlets
- 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*. WebSphere Portal supports two different portlet containers, one for running IBM portlets and the other for running JSR 168 portlets.

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.

Note: The IBM portlet API and JSR 168 portlets will interact with portlet services in different ways, including the Credential Vault and the Cooperative portlet features. Please see the respective sections for more information.

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 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 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 14. 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.

Archived

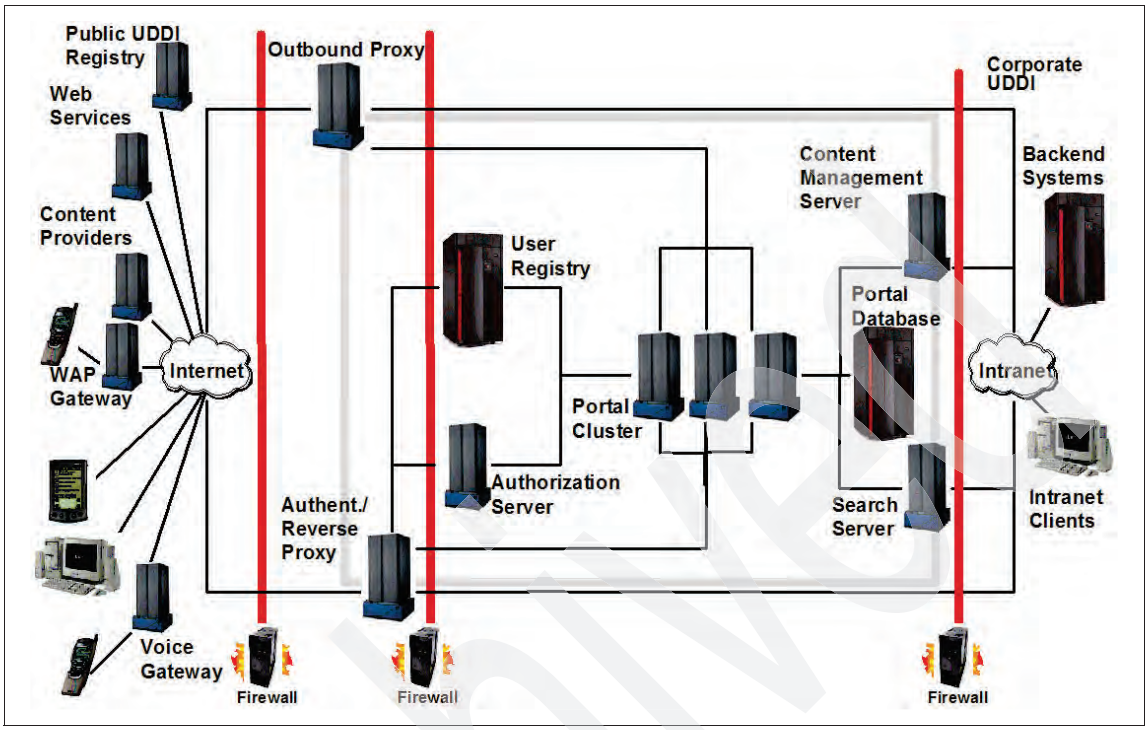


Figure 1-3 Distributed Portal system

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

Figure 1-4 on page 13 depicts 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 failover 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 services

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 in 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.

IBM along with Sun and several other companies have created the JSR 168 Portlet specification. This new portlet API is designed to provide a standard for portlet development.

OASIS created the Web Services for Remote Portals (WSRP) Technical Committee. Chaired by IBM, the WSRP committee has created an XML and Web services standard that allows 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.

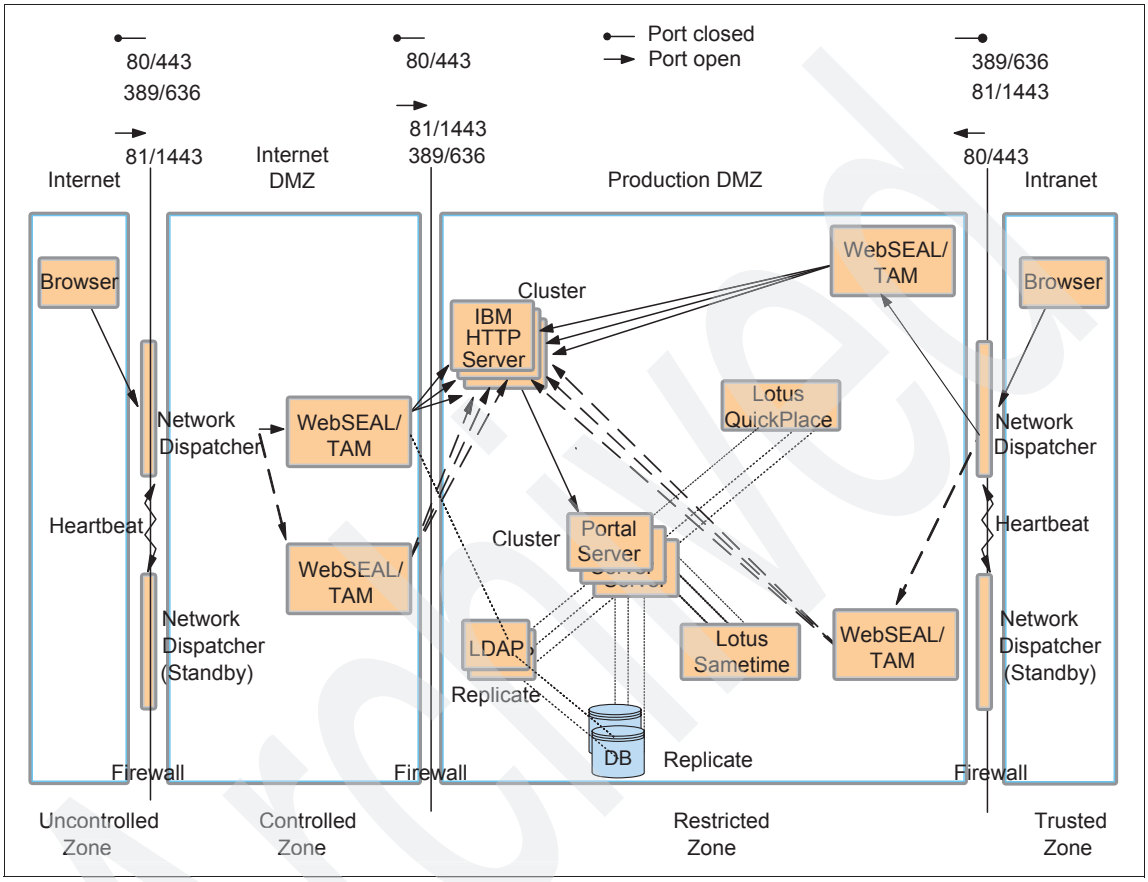


Figure 1-4 High availability portal solution

Building on the Jetspeed implementation, WebSphere Portal provides an architecture for building and running portal applications. WebSphere Portal V5.1 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 on page 14. The core of WebSphere Portal architecture is composed of the presentation services, the portal infrastructure, and the portal services.

Reviewed

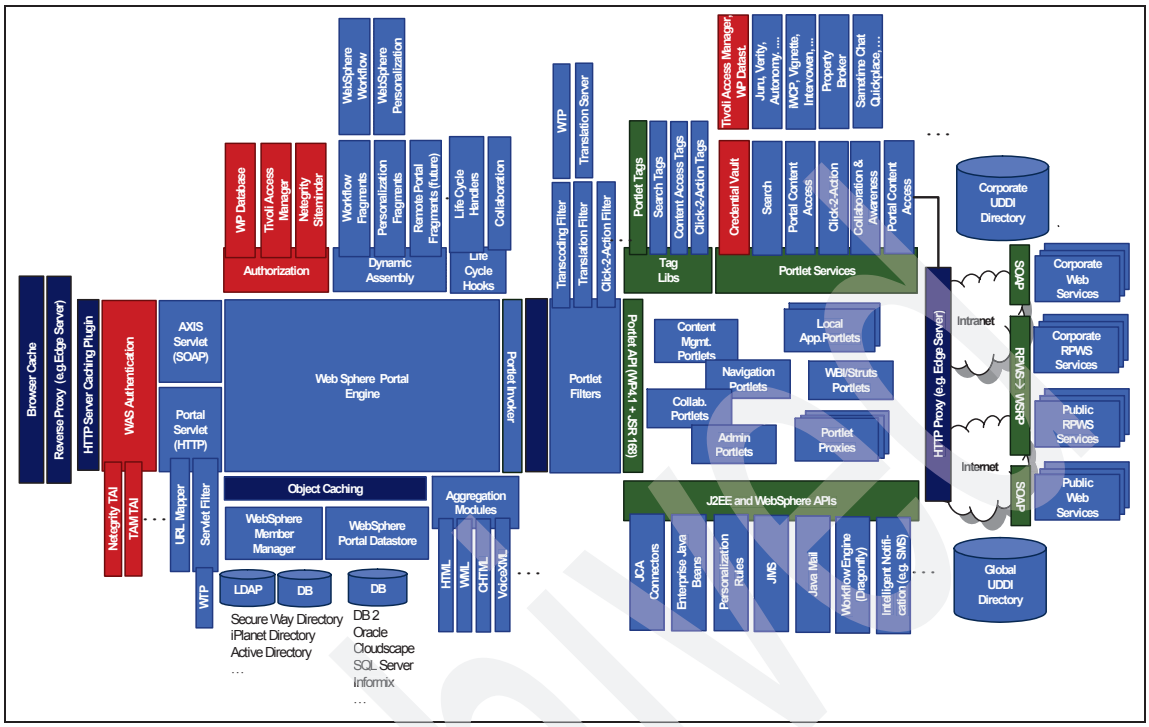


Figure 1-5 WebSphere Portal architecture

Presentation services

WebSphere Portal presentation services provide customized and personalized pages for users through 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 an 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 on page 15 shows the WebSphere Portal engine components.

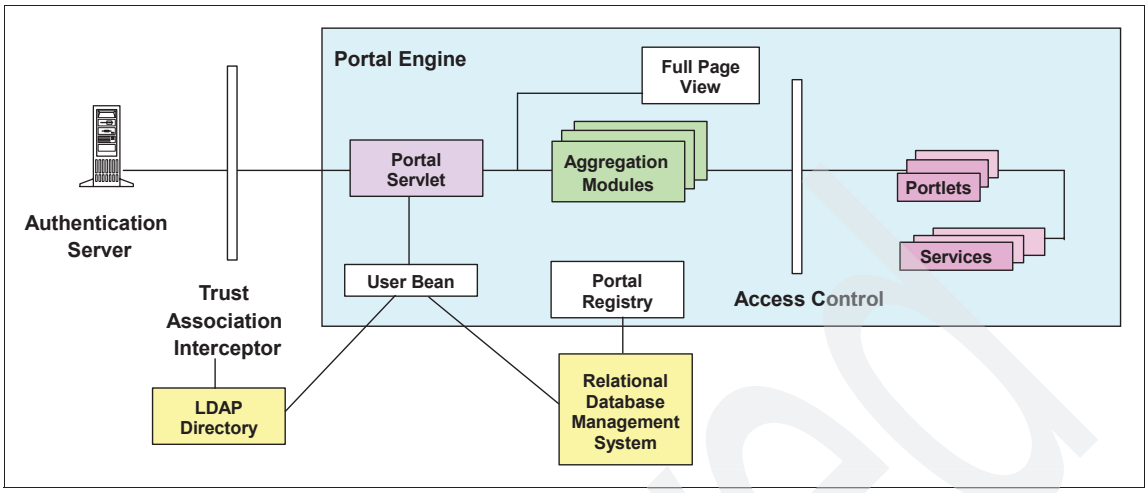


Figure 1-6 WebSphere Portal engine

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 action phase is used for processing actions. This is also the phase where messaging is processed in the IBM portlet API. The other is the redner phase. In the render 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 or JNDI in the case of JSR 168 portlets.

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 IBM portlets. JSR 168 portlets can take advantage of J2EE security methods or they can use the user attributes stated in the specification.

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 14. 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 (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. WebSphere Portal V5.1 provides a set of portlets for providing and consuming remote portlets.

1.2.5 WebSphere Portal tooling

WebSphere Portal and Rational Application Developer, 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.

Archived

Rational Application Developer

WebSphere Portal tools provided with Rational Application Developer provides an environment for developing portlets for both APIs, IBM and JSR. With Rational Application Developer, you can now also develop portal themes and skins.

Rational Application Developer 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.

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.

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

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.

Archive

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 V5.1 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:

- **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 with 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.

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

Archived

V4.x Permissions	V5.0 Roles
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.

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 page. The portal administrator or the designer determines the theme for pages 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.

Archived

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 modes along with event. 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.

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 GoF 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.

Archived

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.

Portlets

Portlets are invoked by the portlet container. A portlet is a Web application 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 deployment descriptor. IBM portlets use the Web deployment descriptor while JSR portlets are defined in the portlet deployment descriptor.

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.

State

This is the current state of the portlet window. Valid states in both portlet APIs are Normal, Minimized and Maximized. IBM portlet API will also support SOLO while JSR 168 does not.

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. IBM portlet API supports View, Edit, Config and Help modes, while JSR 168 portlets supported View, Edit and Help modes, as well as custom modes. The Config mode for JSR 168 portlets is supported as a custom mode.

Note: All portlets must support the default mode, View.

1.3.3 The model-view-controller (MVC) design pattern

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 1-7.

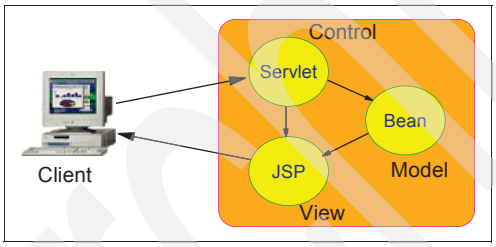


Figure 1-7 Simple MVC architecture

1.3.4 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.

Archived

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.

1.3.5 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.

1.3.6 Portlet MVC sample

In the simplest of portlet applications, the MVC architecture would be applied as in Figure 1-8 on page 26. Note that Figure 1-8 on page 26 does not reflect the architecture of the Portal Server, simply the portlets executed by the server in any given single request.

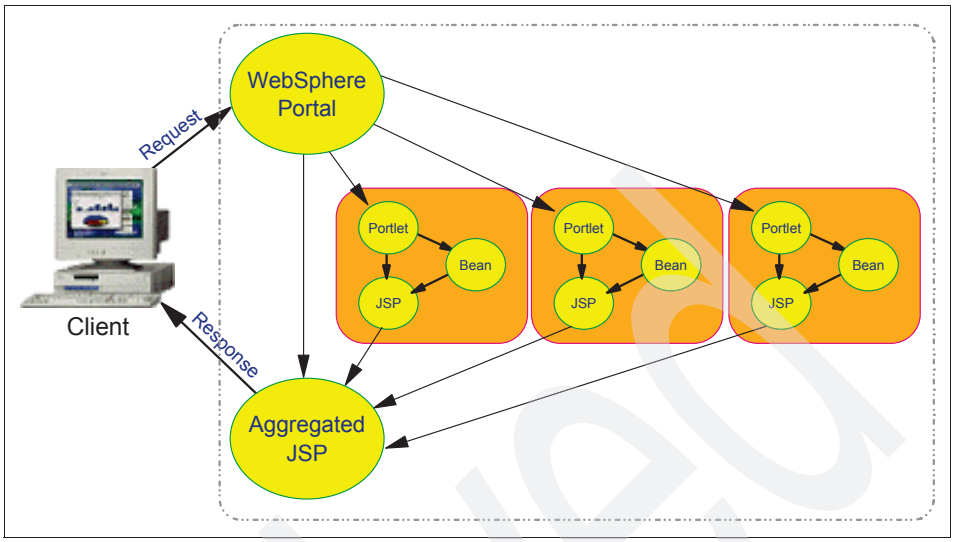


Figure 1-8 Portlet MVC architecture

1.3.7 WebSphere Portal runtime: the portlet container

WebSphere Portal is a J2EE application based on the servlet technology. In fact, IBM Portlets inherit from HTTP Servlet in the Java hierarchy, providing the servlet functionality. JSR 168 portlets do not inherit from servlets but they are model after them. 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.

Both portlet APIs provides the standard interfaces for accessing the services provided by the portlet container. As previously mentioned, the Portlet Container is implemented on top of the servlet container and thus both portal APIs are similar to the servlet API.

1.3.8 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 1-9 on page 28.

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.

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.

Archive

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 page for the current user. Aggregation then continues with the rendering of the page.

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 1-9 shows the layout of a portal page.

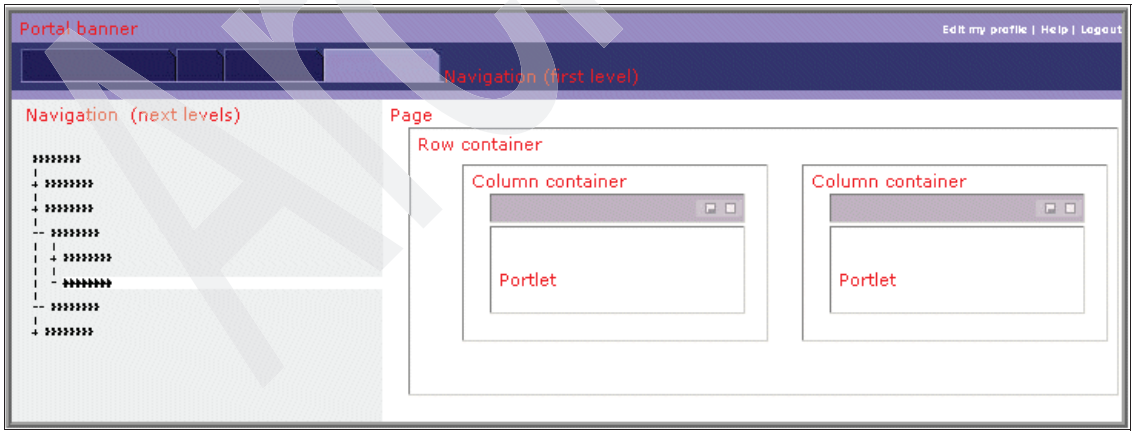


Figure 1-9 Portal page layout

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 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.

Archive

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:

- 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 1-10 on page 31.

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.

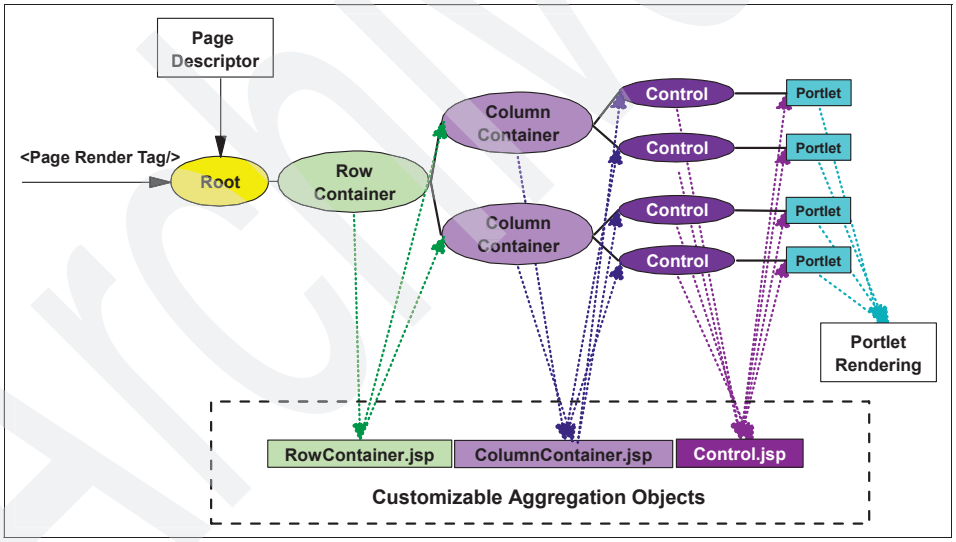


Figure 1-10 Page aggregation

1.4 Highlights in WebSphere Portal V5.1

In this section, we present general information about Portal V5.1.

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 5.1 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

The general infrastructure of Portal V5.1 consists of the following elements.

Support for WebSphere Application Server V5.1 Enterprise

WebSphere Portal 5.1 runs on WebSphere Application Server 5.1 Enterprise to take advantage of better performance and scalability.

Enabling for Communities

WebSphere Portal 5.1 through its WebSphere Member Manager Component provides support for Communities as special groups with additional meta-information.

1.4.3 Event broker

WebSphere Portal 5.1 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.

Archive

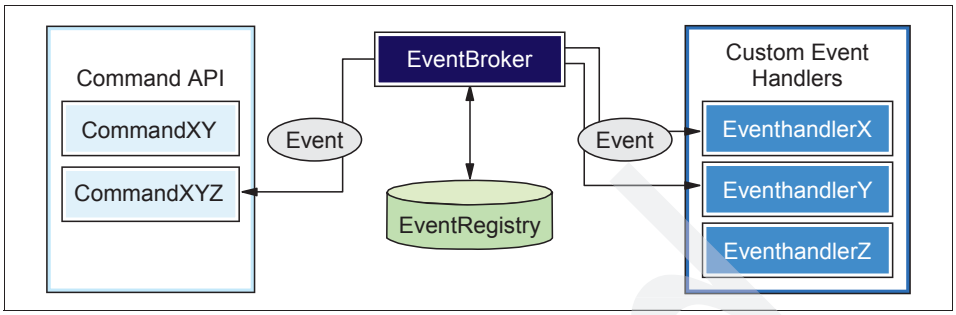


Figure 1-11 Event broker

1.4.4 Member subsystem

WebSphere Portal Server 5.1 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

Authentication in this version consists of the following elements.

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 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

Authorization in this version consists of the following elements.

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

1.4.7 URL generation, processing and mappings

WebSphere Portal 5.1 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.

1.4.8 Search

WebSphere Portal 5.1 introduces major improvements in its search functionality, adding additional content sources and a new Search Center Portlet.

New in WebSphere Portal 5.1:

- You can make intranet sites available for searches.
- WebSphere Portal site are now also available for searches. Only portlets for which you have access to will be indexed. Furthermore only the main panels of portlets are indexed.

Archived

- Lotus® Workplace Web Content Management™ is now available for indexing as well.
- You can configure the use of external search engine such as Google.
- The new Search Center portlet provides many new useful features.

See the WebSphere Portal InfoCenter for more details:

http://publib.boulder.ibm.com/infocenter/wp51help/topic/com.ibm.wp.ent.doc/wps/admsrch.html

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.

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®.

IBM Lotus Workplace Web Content Manager

IBM Lotus Workplace Web Content Manager (ILWWCM) 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, page designs, workflow, and versioning, that make the content creation process simple, yet controlled. ILWWCM 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.

1.4.10 Transcoding

Transcoding technology was incorporated into WebSphere Portal 4.1. As transcoding technology serves different market through various IBM offerings, including WebSphere Portal, a number of markup language transformations 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 reduce 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.

1.4.11 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.

WebSphere portal version 5.1 provide a Struts portlet framework to be used with JSR 168 portlets.

1.4.12 JSF Portlet Runtime

JavaServer Faces (JSF) is a technology defined by JSR 127 standard, that helps you build user interfaces for dynamic Web applications that run on the server. The JavaServer Faces framework manages UI states across server requests and offers a simple model for the development of server-side events that are activated by the client. JSF is consistent and easy to use.

WebSphere Portal 5.1 includes support for JSF portlet applications by providing a JSF portlet runtime that makes possible to run JSF applications as portlets in WebSphere Portal.

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.

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 about 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.

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

Archived

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.

1.4.15 Portal Toolkit

The portlet tools are integrated into Rational Application 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 API, optional frameworks, 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 Rational Application 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 Rational Application 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.

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.

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.

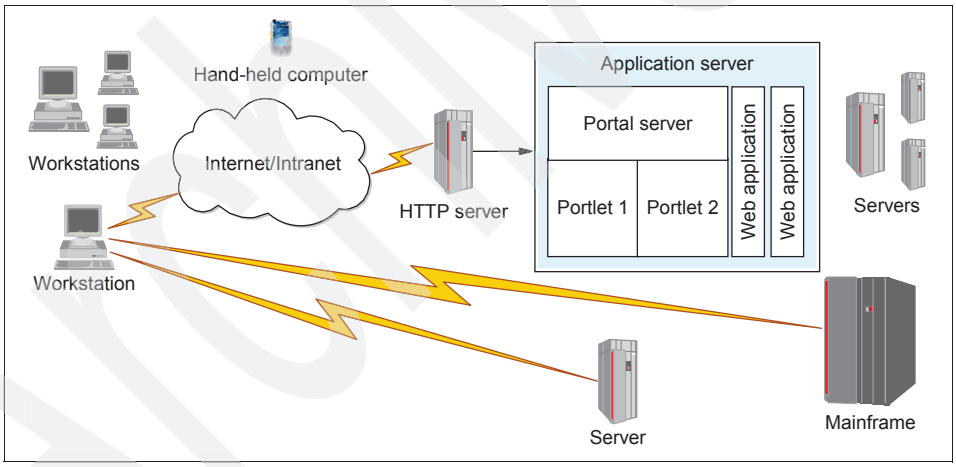


Figure 1-12 Portal Solutions - client to remote application

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).

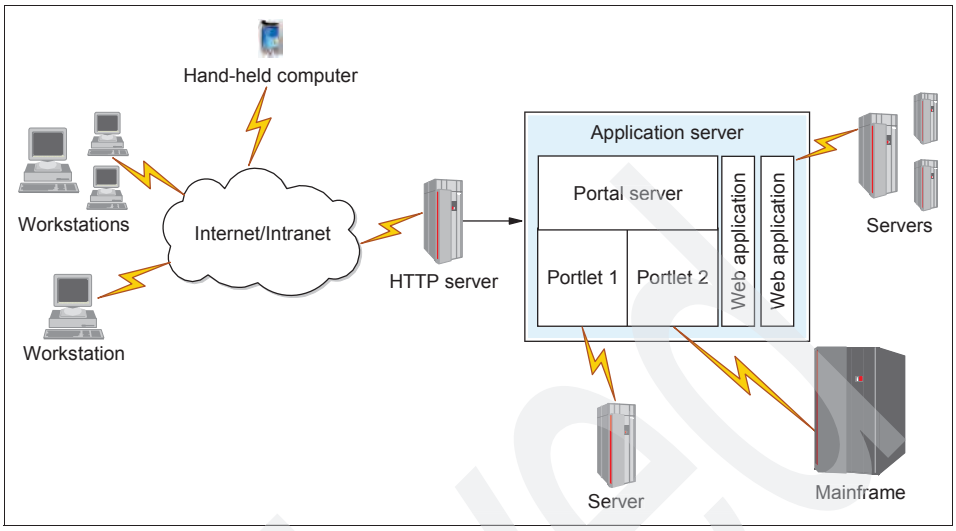


Figure 1-13 Portal Solutions - portlet to remote application

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.

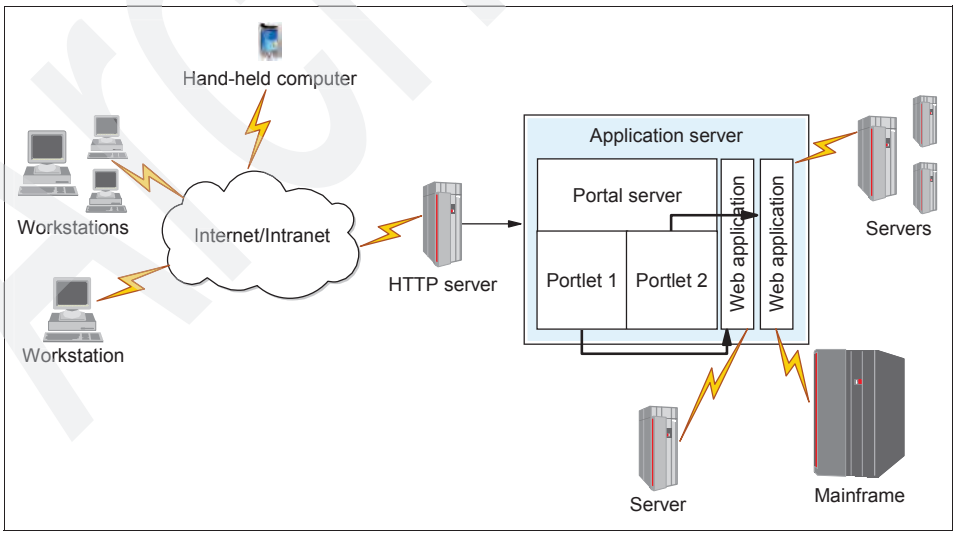


Figure 1-14 Portal Solutions - Portlet to Web application

Archive

1.6 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 Rational Application Developer. Each war file should contain elements listed in Figure 1-15.

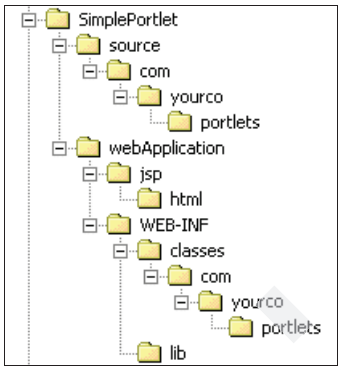


Figure 1-15 Basic WAR contents

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 National Language support for HTML requests, the JSP folder would be organized as in Figure 1-16.

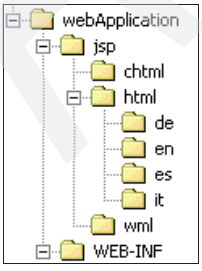


Figure 1-16 WAR structure for a portlet with NLS and multi-device support

Archived

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 1-17.

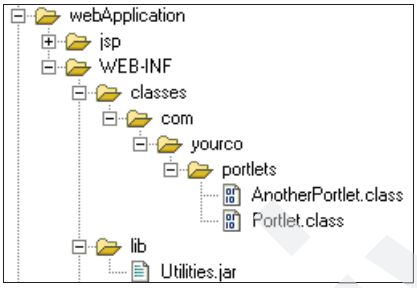


Figure 1-17 Storing classes in the WEB-INF folder

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

Archived

2

Developing Portal applications

This chapter describes the portal development tools and test environment for IBM WebSphere Portal that are now included with IBM Rational Application Developer V6.0. We will also highlight how the portal tools in Rational Application Developer can be used to develop a portal and associated portlet applications for WebSphere Portal. Lastly, we have included a development scenario to demonstrate how to use the new integrated portal tooling and test environment to develop a portal, customize the portal, and develop two portlets.

The chapter is organized into the following topics:

- Portal overview
- Development applications for WebSphere Portal
- Portal development scenario

© Copyright IBM Corp. 2005. All rights reserved.

2.1 Portal overview

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.

Because of the increasing popularity of portal technologies, the tooling and frameworks used to support the building of new portals has evolved. 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, many portals offer personalization and customization features. 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 preferences.

Portals are the next-generation desktop, delivering e-business applications over the Web to many types of client devices from PCs to PDAs. 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 relevant to the user.

The goal of implementing an Enterprise portal is to enable a working environment that integrates people, their work, personal activities and supporting processes and technology.

2.1.1 Portal concepts and definitions

Before beginning development for portals, you should become familiar with some common definitions and descriptions of portal-related terminology.

Portal page

A portal page is a single Web page that can be used to display content aggregated from multiple sources. The content that appears on a portal page is displayed by an arrangement of one or more portlets. For example, a World Stock Market portal page might contain two portlets that display stock tickers for popular stock exchanges and a third portlet that displays the current exchange rates for world currencies.

Portlet

A portlet is an individual application that displays content on a portal page. To a user, a portlet is a single window or panel on the portal page that provides information or Web application functionality. To a developer, portlets are Java-based pluggable modules that can access content from a source such as another Web site, an XML feed, or a database, and display this content to the user as part of the portal page.

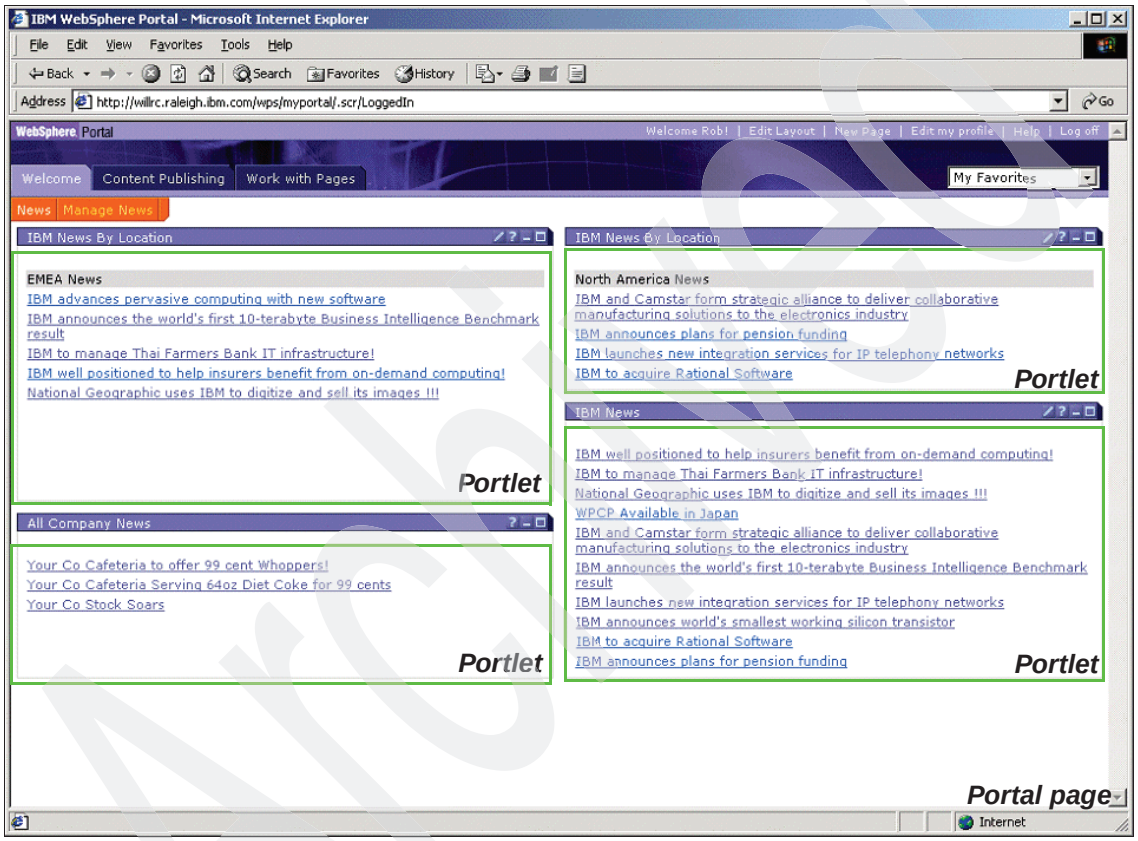


Figure 2-1 Portal page and portlets

Portlet application

Portlet applications are collections of related portlets and resources that are packaged together. Portlets within the same portlet application can exchange and share data and act as a unit. All portlets packaged together share the same context which contains all resources such as images, properties files and classes.

Archived

Portlet states

Portlet states determine how individual portlets look when a user accesses them on the portal page. These states are very similar to minimize, restore, and maximize window states of applications run on any popular operating system just in a Web-based environment.

The state of the portlet is stored in the *PortletWindow.State* object and can be queried for changing the way a portlet looks or behaves based on its current state. The IBM portlet API defines three possible states for a portlet:

- **Normal**: The portlet is displayed in its initial state as defined when it was installed.
- **Minimized**: Only the portlet title bar is visible on the portal page.
- **Maximized**: The portlet fills the entire body of the portal page hiding all other portlets.

Portlet modes

Portlet modes allow the portlet to display a different *face* depending on how it is being used. This allows different content to be displayed within the same portlet depending on its mode. Modes are most commonly used to allow users and administrators to configure portlets or to offer help to the users. There are four modes in the IBM Portlet API:

- **View**: Initial face of the portlet when created. The portlet normally functions in this mode.
- **Edit**: This mode allows the user to configure the portlet for their personal use (for example, specifying a city for a localized weather forecast).
- **Help**: If the portlet supports the Help mode, this mode displays a help page to the user.
- **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.

Portlet events

Some portlets only display static content in independent windows. To allow users to interact with portlets and to allow portlets to interact with each other, portlet events are used. Portlet events contain information 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 a given event, the portlet must also have the appropriate *event listener* implemented within the portlet class. There are three commonly used types of portlet events:

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

Archived

- **Message events:** Generated when one portlet within a portlet application sends a message to another portlet.
- **Window events:** Generated when the user changes the state of the portlet window.

2.1.2 IBM WebSphere Portal

IBM WebSphere Portal provides an extensible framework that 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 such as Single Sign-On (SSO), security, credential vault, directory services, document management, Web 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.

WebSphere Portal allows you to plug in new features or extensions using 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 when providing a portal that functions as the user's interface 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.

Portal solutions such as IBM WebSphere Portal are proven and shorten 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.

2.1.3 IBM Rational Application Developer

IBM Rational Application Developer provides development tools for applications destined to WebSphere Portal. Bundled with IBM Rational Application Developer V6.0 are a number of Portal Tools that allow you to create, test, debug, and deploy portal and portlet applications. These tools are described in more detail in 2.2, "Developing applications for WebSphere Portal" on page 54.

Portal Tools

Unlike WebSphere Studio Application Developer where the tools where installed as a separate toolkit, Portal Tools can now be installed as a feature when installing IBM Rational Application Developer V6.0. For this reason, there is no longer a separate Portal Toolkit or separate installation procedure.

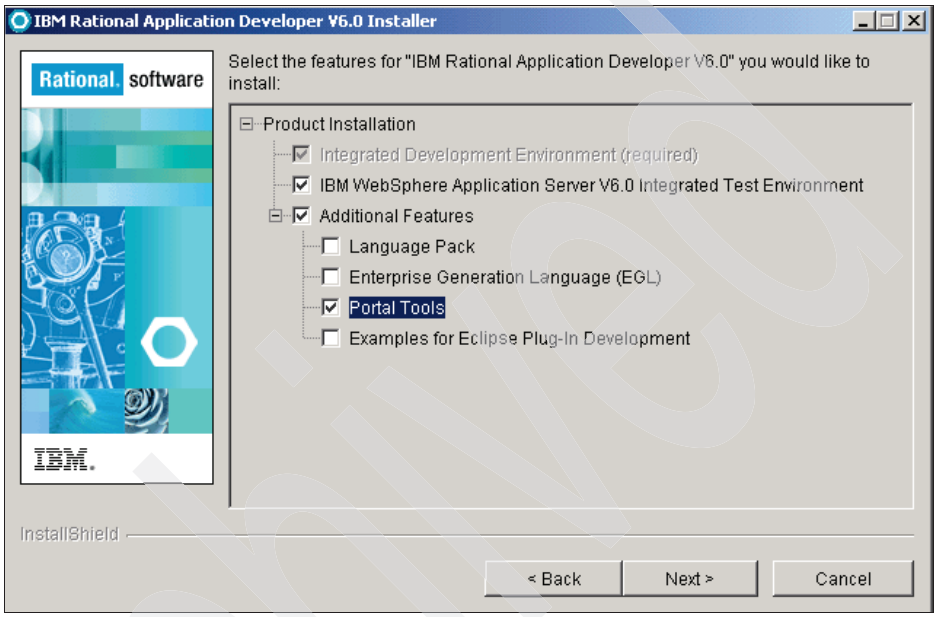


Figure 2-2 Portal tools installation

Portal Test Environments

As part of this tool set, Rational Application Developer provides an integrated test environment to run and test your portal and portlet projects from within the Rational Application Developer workbench.

At the time IBM Rational Application Developer V6.0 was released, the WebSphere Portal V5.0.2.2 Test Environment was included as an installed component of the Rational Application Developer installer known as the Launchpad (see Figure 2-3 on page 53).

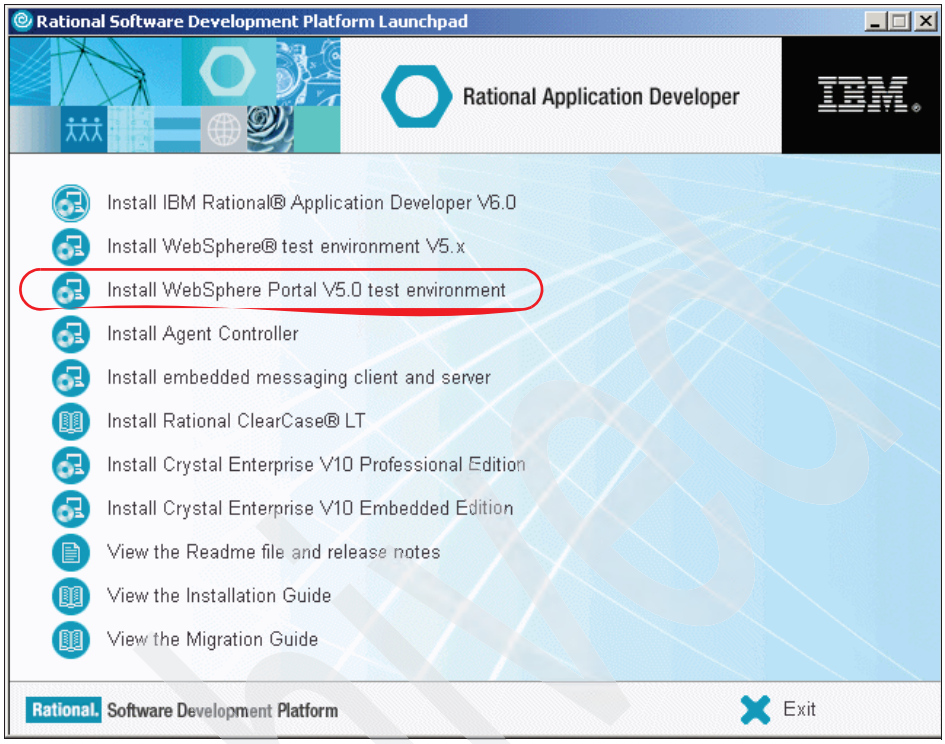


Figure 2-3 Product installation launchpad

CDs to install the WebSphere Portal V5.1 Test Environment are also included with the IBM Rational Application Developer V6.0 distribution. To install the WebSphere Portal V5.1 Test Environment, you must run the WebSphere Portal V5.1 setup program and select Test Environment as the setup type as seen in Figure 2-4 on page 54. Follow the instructions in the InfoCenter to configure this test environment so that it works from within Rational Application Developer. The WebSphere Portal V5.0.2.2 and V5.1 Test Environments can co-exist within the same product installation.

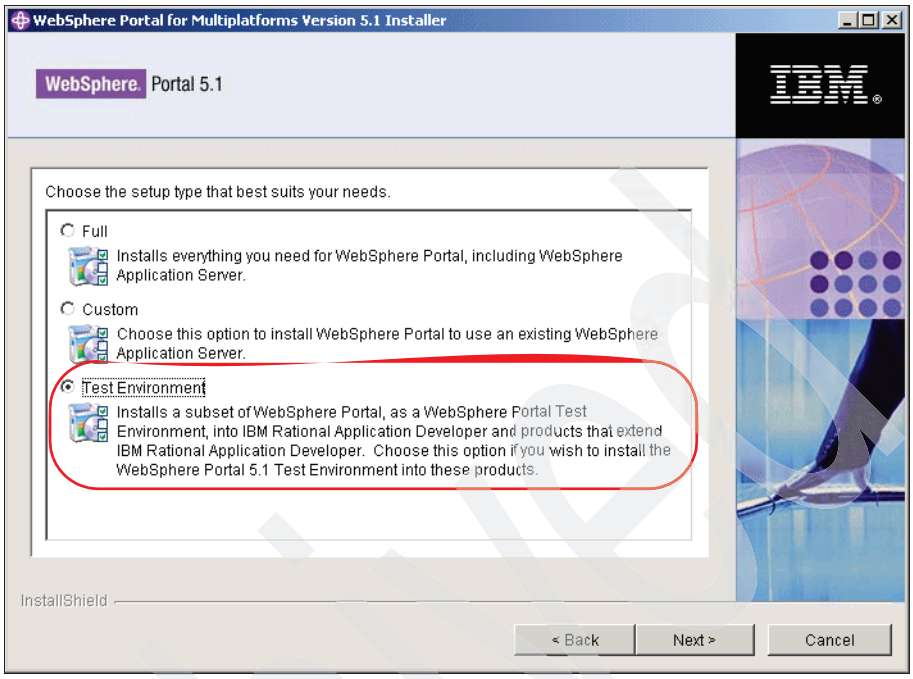


Figure 2-4 Installing the WebSphere Portal V5.1 Test Environment

2.2 Developing applications for WebSphere Portal

Rational Application Developer includes many tools to help you quickly develop portals and individual portlet applications. In this section, we will cover some basic portlet development strategies and provide an overview of the tools included with IBM Rational Application Developer V6.0 to aid in development for WebSphere Portal.

2.2.1 Portal samples and tutorials

Rational Application Developer also comes with several samples and tutorials to aid you in development for WebSphere Portal. The Samples Gallery provides sample portlet applications to illustrate portlet development.

To access portlet samples, click **Help** → **Samples Gallery**. Then expand **Technology samples** and **Portlet**. Here you can select a Basic Portlet, Faces Portlet, or Struts Portlet Framework to view sample portlet application projects that you can then modify and build upon for your own purposes.

The Tutorials Gallery provides detailed tutorials to illustrate portlet development. These are accessible by selecting **Help** → **Tutorials Gallery**. Then expand **Do and Learn**. You can select **Create a portal application** or **Examine the differences between portlet APIs** to view the content of these tutorials.

2.2.2 Development strategy

A portlet application consists of Java classes, JSP files, and other resources such as deployment descriptors and image files. Before beginning development, several decisions must be made regarding the development strategy and technologies that will be used to develop a portlet application.

Choosing an API - IBM or JSR 168

WebSphere Portal supports portlet development using the IBM portlet API and the JSR 168 portlet API standard. The Portal Tools included with IBM Rational Application Developer V6.0 support both APIs.

The IBM portlet API was initially supported in WebSphere Portal V4.x and will continue to be supported by WebSphere Portal. It is important to note that the IBM portlet API extends the servlet API. More information about the IBM portlet API can be found at:

`http://www.ibm.com/developerworks/websphere/zones/portal/portlet/5.0api/WPS`

JSR 168 is a Java specification from the Java Community Process Program that addresses the requirements of content aggregation, personalization, presentation, and security for portlets running in a portal environment. It was finalized in October of 2003. Portlets conforming to JSR 168 are more portable and reusable because they can be deployed to any JSR 168 compliant portal. Rational Application Developer supports Faces portlet development based on the JSR 168 specification.

Unlike the IBM portlet API, the JSR 168 API does not extend the servlet API. It does, however, share many of the same characteristics such as servlet specification, session management, and request dispatching. The JSR 168 API as implemented in WebSphere Portal V5.0.2.2 does not support Click-to-Action cooperative portlets or portlet messaging. However, in WebSphere Portal V5.1, the JSR 168 container has been enhanced with Property Broker support which can act as a messaging broker for either portlet messaging or wired (automatic cooperating) portlets. At the time of writing, the support for Click-to-Action (user-initiated cooperating portlets) was still under development.

For new portlets, consider using JSR 168 when the functionality it provides is sufficient for the portlet's needs or when the portlet is expected to be published as a Web Service for Remote Portlets (WSRP) service.

IBM will further support JSR 168 in follow-on versions to make the JSR 168 portlet API as robust as the current IBM counterpart and offer tooling to support JSR 168 development. IBM is committed to wider adoption of open standards in WebSphere Portal.

More information can be found on JSR 168 at:

`http://www.jcp.org/en/jsr/detail?id=168`

Choosing markup languages

WebSphere Portal supports mobile devices by generating pages in any markup language. Three markup languages are officially supported in Rational Application Developer:

- **HTML (Hyper Text Markup Language)** is used for Web browsers on desktop computers. All portlet applications support HTML at a minimum.
- **WML (Wireless Markup Language)** is used for WAP devices which are typically Web-enabled mobile telephones.
- **cHTML (compact Hyper Text Markup Language)** is used for mobile devices in the NTT DoCoMo i-mode network. For more information about the i-mode network, visit the following Web site:

  `http://www.nttdocomo.co.jp/english/`

Adding emulator support for other markup languages

To run a portlet application which supports WML or cHTML, you must use an emulator provided by the device vendor. To add device emulator support to Rational Application Developer, do the following:

1. Select **Window** → **Preferences.**
2. Expand **Internet** and select **Web browser.**
3. Click the **Add** button to locate the device emulator appropriate for the device that you wish to test and debug.

Enabling transcoding for development in other markup languages

Transcoding is the process by which WebSphere Portal makes portal content displayable on mobile devices. By default, it is *not enabled* in the WebSphere Portal Test Environment. Therefore, you need to make some configuration changes before you can test and debug applications on mobile device emulators. You will need to remove the comments from lines beginning with `#Disable Transcoding` from three files.

The PortletFilterService.properties and PortalFilterService.properties files are all located by default in the following directory:

<rad_home>\runtimes\portal_v50\shared\app\config\services

The services.properties file is located by default in the following directory:

<rad_home>\runtimes\portal_v50\shared\app\config

Choosing other technologies

Struts technology and JavaServer Faces technology can also be incorporated into a portlet development strategy.

Struts

Struts-based application development can be applied to portlets, similar to the way that Struts development is implemented in Web applications. The Struts Portal Framework (SPF) was developed to merge these two technologies. SPF support in Rational Application Developer simplifies the process of writing Struts portlet applications and eliminates the need to manage many of the underlying requirements of portlet applications. In addition, multiple wizards are present to help you create Struts portlet-related artifacts. These are the same wizards used in Struts development. These wizards include: Action Class, Action Mapping, ActionForm, Form-Bean Mapping, Struts Configuration, Struts Module, Struts Exception, and Web Diagram. Refer to the Rational Application Developer Struts documentation for usage details.

In WebSphere Portal V5.0.2.2, Struts is only supported using the IBM portlet API. Struts is fully supported in both the IBM and JSR 168 APIs in WebSphere Portal V5.1; however, there is no tooling support in Rational Application Developer for this configuration.

More information about Struts can be found at:

`http://struts.apache.org/`

JavaServer Faces (JSF)

JavaServer Faces is a GUI framework for developing J2EE Web applications (JSR 127). It includes reusable user interface components, input validation, state management, server-side event handling, page life cycle management, accessibility, and internationalization. Faces-based application development can be applied to portlets, similar to the way that Faces development is implemented in Web applications. Similar to Struts, there are many wizards to help you with Faces development. Both WebSphere Portal V5.0.2.2 and V5.1support JavaServer Faces.

There are certain limitations to Faces portlet development in the current release. Service Data Objects (SDO), formerly referred to as WebSphere Data Objects (WDO), are limited to prototyping purposes only. Applications that rely on SDOs

Archived

should be limited in a production environment. File upload and binary download are not supported for Faces components. Finally, document root-relative URLs are not supported for images.

Refer to the Rational Application Developer Faces documentation in the InfoCenter for usage details. Alternatively you can refer to the following Web site:

http://www.jcp.org/en/jsr/detail?id=127

Beginning development

After making these decisions, you can now begin development using the Portal Tools included with Rational Application Developer.

2.2.3 Portal tools for developing portals

A portal is essentially a J2EE Web application. It provides a framework where developers can associate many portlets and portlet applications via one or more portal pages.

Rational Application Developer includes several new portal site creation tools that enable you to visually customize portal page layout, themes, skins, and navigation.

Portal Import Wizard

One way to create a new portal project is to import an existing portal site from a WebSphere Portal V5.0 server into Rational Application Developer. Importing is also useful for updating the configuration of a project that already exists in IBM Rational Application Developer.

The portal site configuration on WebSphere Portal server contains the following resources: the global settings, the resource definitions, the portal content tree, and the page layout. Importing these resources from WebSphere Portal server to Rational Application Developer overwrites duplicate resources within the existing portal project. Non-duplicate resources from the server configuration are copied into the existing portal project. Likewise, resources that are unique to the portal project are not affected by the import.

Rational Application Developer uses the XML configuration interface to import a server configuration, and optionally retrieves files under the websphere_installation_directory/installedApps/node/wps.ear directory. These files include the JSP, CSS, and image files for themes and skins. When creating a new portal project, retrieving files is mandatory. To retrieve files, Rational Application Developer must have access to this directory as specified when you define a new server for this project.

You can access the Portal Import Wizard by selecting **File** → **Import**, then selecting **Portal**. You will need to specify the server and options for importing the project into Rational Application Developer.

Follow the instructions in the Help Topics on Developing Portal Applications to ensure that the configuration in the development environment accurately reflects that of the staging or runtime environment. If you do not do this, you may experience compilation errors after the product is imported or unexpected portal behaviors.

Portal Project Wizard

The New Portal Project wizard will guide you through the process of creating a portal project within Rational Application Developer.

During this process, you are able to:

- Specify a project name
- Specify the default server
- Choose a default theme (optional)
- Choose a default skin for the theme (optional)

Important: You should not name your project *wps* or anything that resembles this string to avoid internal naming conflicts.

The project that you create with this wizard will not have any portlet definitions, labels or pages. The themes and skins that are available in this wizard are the same as if you had imported a portal site from a WebSphere Portal server.

You can access this wizard by clicking **File** → **New** → **Project** and then selecting **Portal Project** from the list. Figure 2-5 on page 60 displays the options to specify when creating a Portal Project after clicking the **Show Advanced** button.

Archived

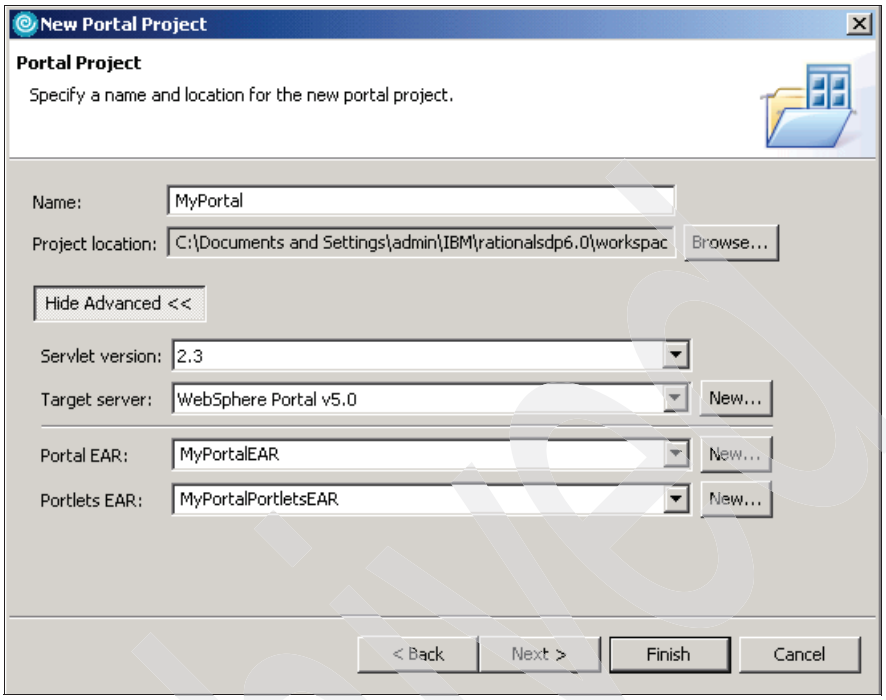


Figure 2-5 Portal Project Wizard

Portal Designer

Rational Application Developer allows for editing both the graphic design and portlet layout within your portal. Portal Designer is the workbench interface that you see upon opening a portal project.

When using Portal Designer, the portal page layout can be altered. The layout refers to the number of content areas within a page and the number of portlets within those content areas. Page content includes rows, columns, URLs and portlets.

Once the project is published to the Portal server, Portal administrators can use the administration portlets to give site users permission to edit the page layout.

In terms of portal layout and appearance, you can think of Portal Designer as a What-You-See-Is-What-You-Get (WYSIWYG) editor. It will render graphic interface items such as themes, skins, and page layouts.

Portal Designer will also render the initial pages of JSF and Struts portlets within your portal pages, but not anything else with regard to portlet content.

Portal Designer provides the capability of altering the layout of the content within a portal page with respect to navigation (the hierarchy of your labels, pages, and URLs) and content (the arrangement of portlets via rows and columns on the portal pages).

Portal Configuration is the name of the layout source file that resides in the root of the portal project folder (see Figure 2-6). To open Portal Designer, double-click the file in the Project Explorer.

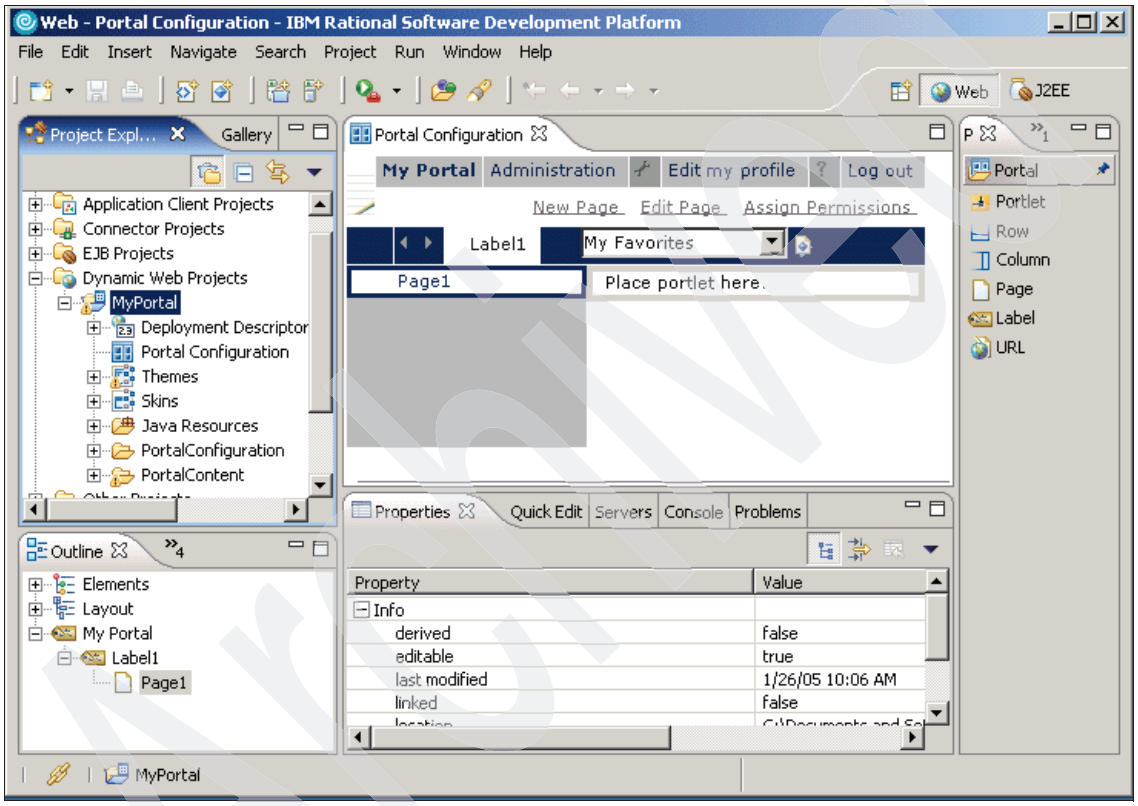


Figure 2-6 Portal Designer

Skin and theme design and editing

A skin is the border around each portlet within a portal page. Unlike themes, which apply to the overall look and feel of the portal, skins are limited to the look and feel of each portlet that you insert into your portal application.

The IBM Rational Application Developer installation includes pre-built themes and skins to use with portal projects. There are also wizards to create new themes and skins. Changing themes and skins was previously done through

portal administration. In addition to these wizards for creating new skins and themes, there are tools that can be used to change or edit these.

Once created, skins and themes will be displayed in the Project Explorer view. Double-click a skin or theme to manually edit it.

New Skin wizard

In addition to using the pre-made skins that came with the installation, you can use the New Skin wizard to create customized skins for your project. Right-click the portal project in the Project Explorer view and select **New** → **Skin**.

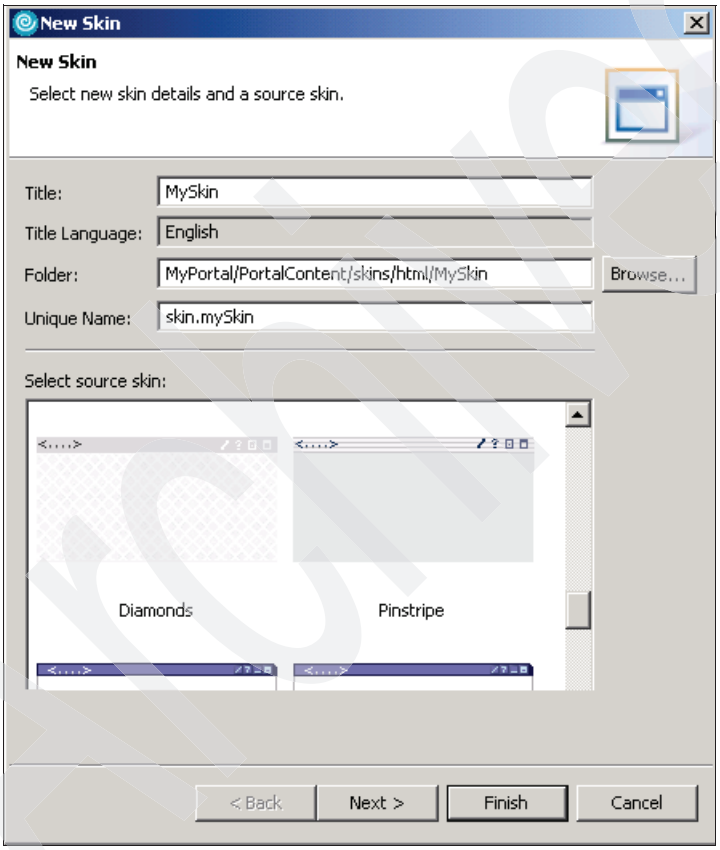


Figure 2-7 New Skin wizard

New Theme wizard

Themes provide the overall look and feel of your portal application. In addition to using the pre-existing themes, you can use the New Theme wizard to create

customized themes for your project. Right-click the portal project in the Project Explorer view and select **New** → **Theme** (see Figure 2-8).

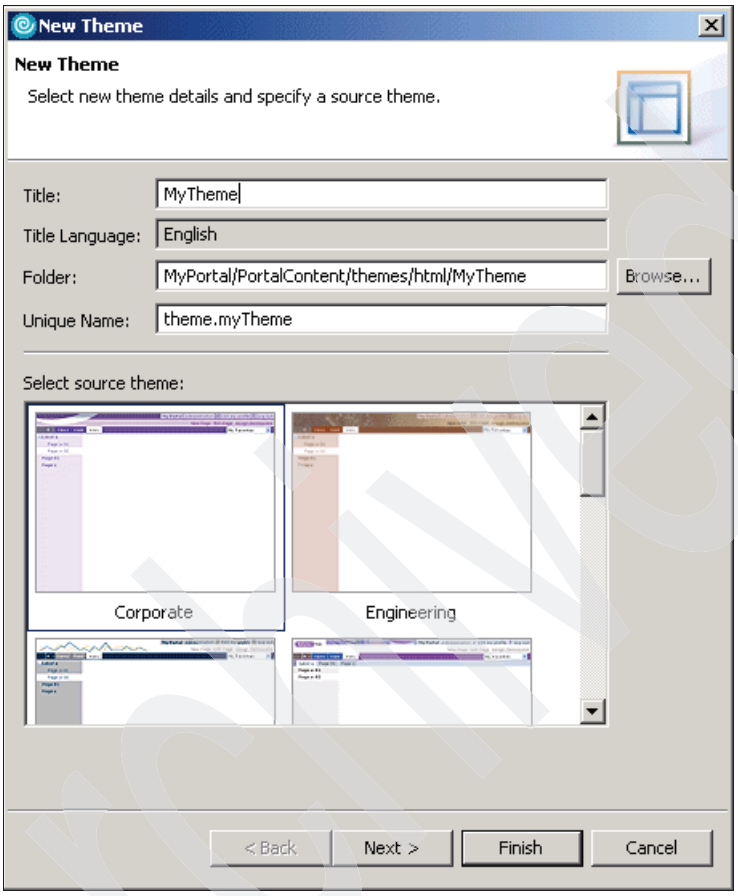


Figure 2-8 New Theme wizard

Deploying portal projects

From Rational Application Developer, you can choose to publish a portal project to a WebSphere Portal server either manually (export) or automatically (deploy).

There are two models of publishing your portal project to WebSphere Portal:

- **Export**: This method is recommended for publishing to a staging or production server. You need to manually copy the packaged portal project to the portal server. Since exporting does not require FTP or copy access to the portal server, there is very little chance of interruption during publishing.

Archived

To export, select **File** → **Export**, then select **Portal Project Deploy Set**. You will need to specify the Portal EAR file and specify a target Portal server. The wizard examines the Portal server so that it can generate specific deploy information. It then generates a set of files for manually deploying the portal project to the Portal server. These files include:

- WPS.ear
- XmlAccess for deployment portal configuration (contained in the EAR)
- Readme file with instructions for deploying to a server
- WAR files for each portlet project used in the portal project
- XMLAccess script for deploying portlets

The wizard also created a file named `DeployInstructions.txt`, which is a set of instructions that will guide you through the process of manually deploying your exported project to the server.

Note: Do not attempt to manually deploy the exported files to a portal server other than the one you specify. The export operation contains information from this portal server, and it will not work with other servers.

- **Deploy**: This method automatically publishes a configuration from a portal project to a Portal Server. The deploy method is recommended for publishing to a test, integration or staging server.

  If you are also transferring the theme and skin files during the deployment, you must also have FTP or copy access to the portal server.

  To deploy a portal, right-click the portal project and select **Deploy Portal**. From here, a wizard will guide you through the deployment process. This will include specifying the portal server to where you are deploying.

  Once you start the deploy process, do not interrupt it. Errors in a project or an unfinished deploy may cause a portal server to become inoperable. As such, you should not deploy directly to a production server. Before running deploy, it is recommended that you back up or image your server.

  Note: If a transfer interruption (for example, network failure) occurs during deployment, there is a slight chance that the portal server will become inoperable.

  Since the portal project does not have any access control information, use administration portlets in the published Portal site to set appropriate access control.

2.2.4 Portal tools for developing portlets

Whether beginning or continuing development of individual portlets and portlet projects, Rational Application Developer has tools that can make this process easier.

Project Interchange files

If you are not using a software configuration management (SCM) system, such as ClearCase or CVS, and you want to share portlet projects with team members or develop a portlet application among multiple computers, you can use the Project Interchange feature.

There are other ways that you can share projects and files including manually copying the project's workspace and importing via WAR files. The recommended method of accomplishing project portability is using the Project Interchange feature. When you export using Project Interchange, the entire project structure is maintained, including metadata files. You can also export several unrelated projects or include required projects for an entire application. Projects exported using this feature can be easily imported into another workspace with a single action.

The Project Interchange mechanism exports projects as they exist in the workbench, including the project property that specifies the target server runtime for the project. If a user imports the exported project and does not have the same target server runtime installed, the project will not compile. This can be corrected by modifying the target server for the project.

Important: It is important that the IBM Rational Application Developer V6 install path is common for all team members sharing code to avoid absolute library path problems found in projects when importing.

Exporting a Project Interchange file

To export to a Project Interchange file, follow these steps:

1. Right-click the project that you want to export, and select **Export**.
2. Select **Project Interchange**, and click **Next**.
3. Select the projects that you want to export. You have the following options for selection:
   - Click **Select All** to select all projects in the window.
   - Click **Deselect All** to clear all the check boxes.
   - Click **Select Referenced** to automatically select projects that are referenced by any of the currently selected projects.

Archived

4. In the To zip file field, enter the full path, including the ZIP file name, where you want to export the selected projects.
5. Click **Finish**.

Importing a Project Interchange file

To import a Project Interchange file, do the following:

1. Click **File** → **Import**.
2. Select **Project Interchange** and click **Next**.
3. In the From ZIP file field, click **Browse** to navigate to the ZIP file that contains the shared projects. The Import wizard lists the projects that are in the ZIP file.
4. Select the projects that you want to import. You have the following options for selection:
   - Click **Select All** to select all projects in the window.
   - Click **Deselect All** to clear all the check boxes.
   - Click **Select Referenced** to automatically select projects that are referenced by any of the currently selected projects.
5. Click **Finish**.

Importing WAR files

An alternate method of transferring a portlet project to another computer is via a WAR file. WAR files package all pieces of a portlet project into a single file. They are most commonly used to manually deploy portlet projects to Portal Servers.

For development purposes, WAR files can be used to move portlet projects from one computer to another. WAR files are not optimized for this purpose, and moving projects in this way may result in lost meta data or lost time due to any reconfigurations that may be required upon import.

For portlet projects completed with a version of the Portal Toolkit prior to V5.0.2.2, importing by WAR file is the only supported migration path.

To import a project by WAR file, follow these steps:

1. Select **File** → **Import** and select **WAR File.** Then click **Next**.
2. Locate the WAR file to import by using the **Browse** button.
3. The wizard assumes you want to create a new Web project with the same name as the WAR file. Accepting these defaults will create a new project with the same servlet version as specified by the WAR file and in the same location. To override these settings, click **New** and specify the new settings in the Dynamic Web Project wizard.

Archived

4. To import a WAR file into an existing Web project, select the project from the Web project drop-down list. If this method is used, the overwrite existing resources without warning option can be selected.
5. Click **Finish** to populate the Web project.

Note: When a portlet project is exported to a WAR file, the source files must be included. This procedure is detailed in “Exporting WAR files” on page 82.

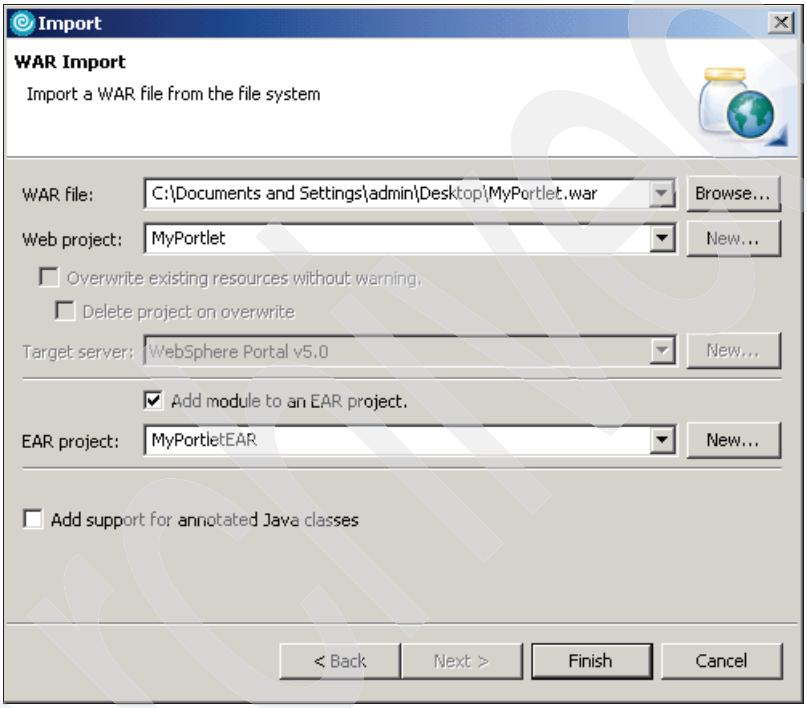


Figure 2-9 WAR Import window

Portlet Project wizard

Portlet projects are used for developing portlet applications in Rational Application Developer. To create a portlet application, first create a portlet project using the Portlet Project wizard.

The Portlet Project wizard is a very powerful tool that automatically assembles a framework for a portlet project containing all the resources that are necessary for testing, debugging, or deploying a portlet.

To use the Portlet Project wizard, do the following:

1. Select **File** → **New** → **Project**.
2. Select **Portlet Project** and click **Next**.
3. On the first screen in the wizard, enter a project name. You can also specify an alternate project location by clicking the **Browse** button.
4. If you do not want to create the initial portlet definitions in the project, clear the **Create a portlet** checkbox. Typically, a portlet does not need to be created when importing a WAR file.
5. Click the **Show Advanced** button to see more options (see Figure 2-10 on page 69):

   The advanced options allow changes to be made to the project's J2EE settings and runtime server environment. The Servlet version specifies the version of Servlet and JSP specifications to be included in your portlet.

   Tip: Use the 2.2 Servlet version if importing a WebSphere Portal V4.x project WAR file. Note that features such as Servlet filters and life cycle event listeners are not supported if this level is chosen.

   Choosing a Servlet version also determines the choice of target servers that appear in the drop-down list. When choosing a server, do not accidentally select any WebSphere Application Server options.

   a. Deselect the **Add module to an EAR project** option only if you do not intent to deploy the portlet. Name an EAR project according to the name of the enterprise application project (EAR) that the portlet project should be associated with during deployment. All portlet applications associated with a single EAR project will run on a single session on a WebSphere Portal Server. You should use the same EAR project for portlet projects that are related.

      The context root is used as the top-level directory in the portlet project when the portlet is deployed. It must not be the same as ones used by other projects.

   b. Ensure that the **Add support for annotated Java classes** checkbox is selected if using model annotations to generate code in portlet projects.

   c. Click **Next** to continue with the Portlet Project wizard or **Finish** to generate a portlet project based on the defaults associated with a Basic IBM API portlet project.

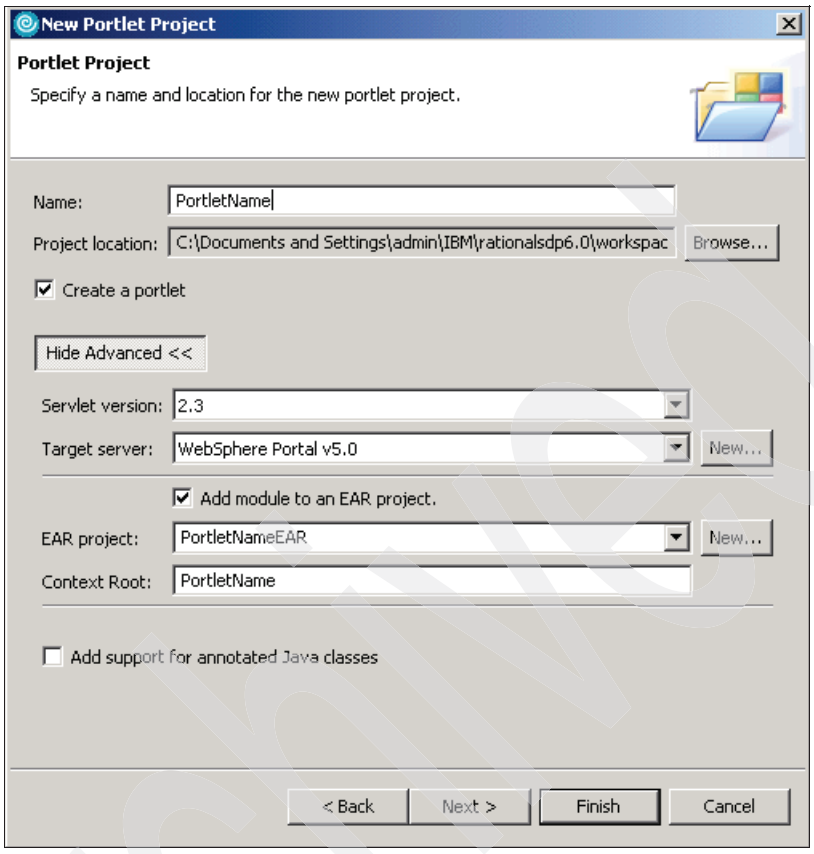


Figure 2-10 Portlet Project wizard

d. On the following screen, select a portlet type that is appropriate for the portlet project. There are four types of portlets (see Figure 2-11 on page 70):

- **Empty portlet**: Creates a portlet application that extends the PortletAdapter class with minimum code included. This option is selected if importing a project from a WAR file or when customizing empty portlet projects from scratch.
- **Basic portlet**: Creates a basic portlet application that extends the PortletAdapter class comprised of a complete concrete portlet and concrete portlet application. It contains a portlet class, sample JSP files that are used when rendering the portlet, and a sample Java bean.
- **Faces portlet**: Creates a Faces portlet application based on Java Faces technology.

- **Struts portlet**: Creates a Struts portlet application based on Java Struts technology.

e. When finished selecting option on this screen, click **Next**.

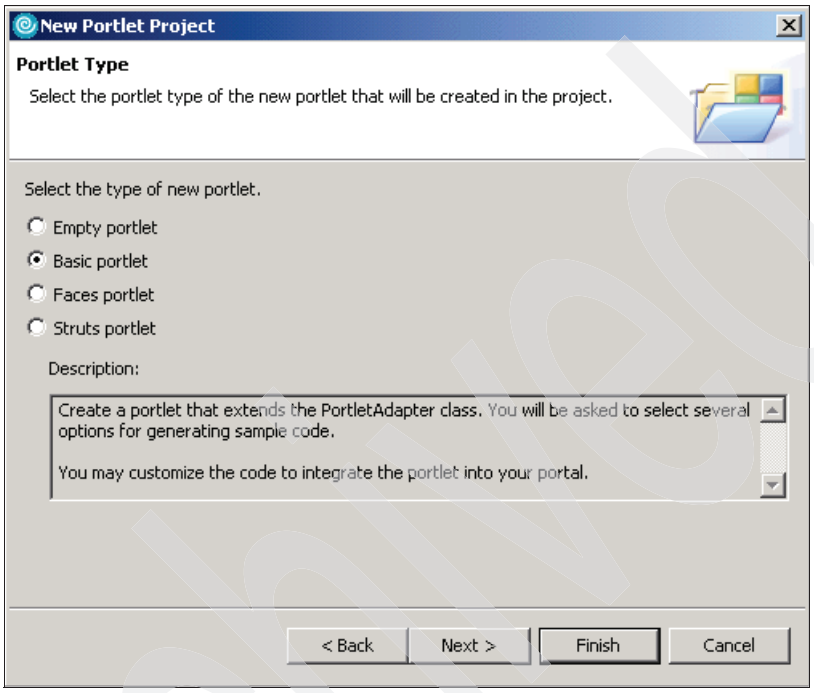


Figure 2-11 Portlet type window

When creating a Faces portlet, you will be presented with the following window.

Archive

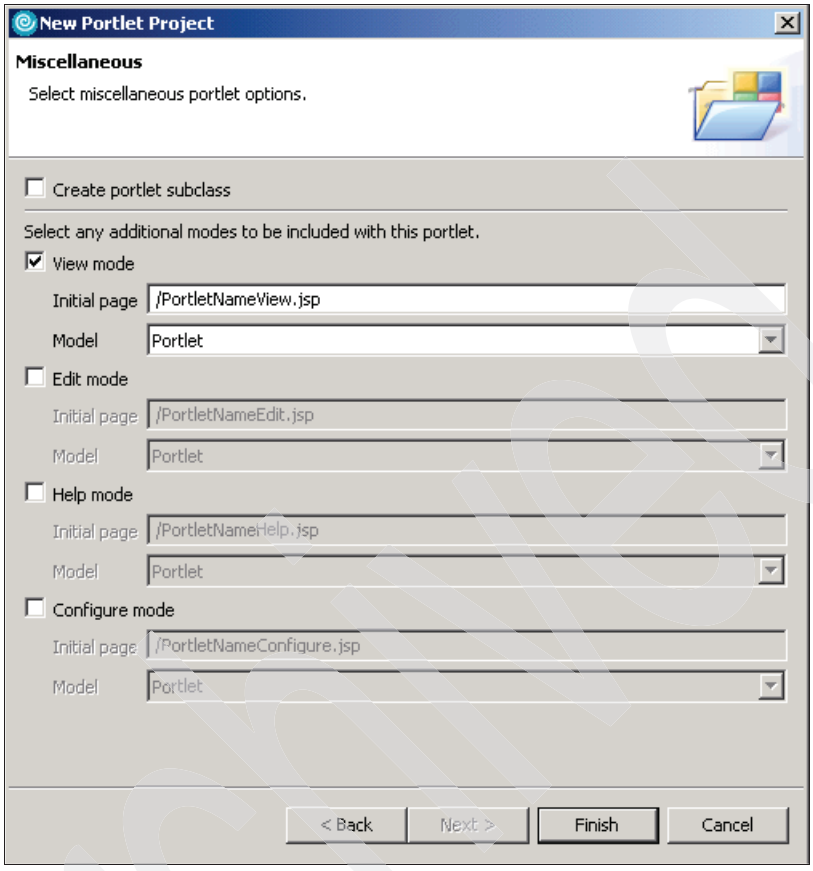


Figure 2-12 Faces portlet miscellaneous window

When creating a Struts portlet, you will be presented with the following window.

Archived

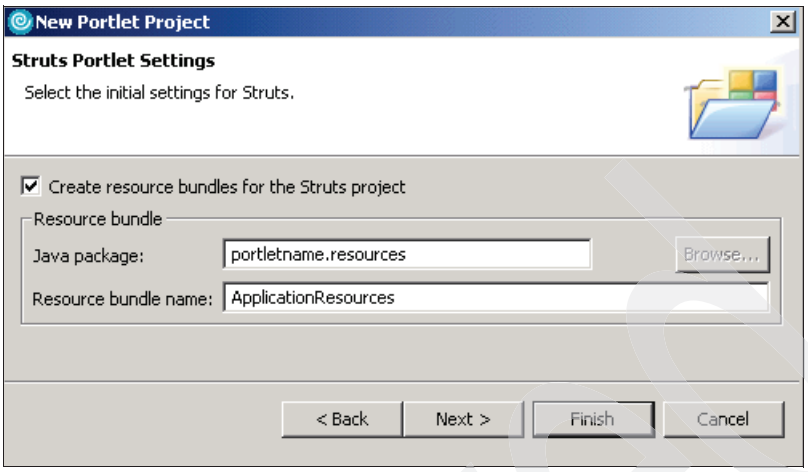


Figure 2-13 Struts portlet settings window

When creating a Basic portlet, you will be presented with the following window. It allows you to select features that provide additional functionality in the portlet application. Select features as necessary. Deselect the **Web Diagram** check box if you are creating a Basic or Empty portlet. Select **JSP Tag Libraries** to include the functionality of this technology in the portlet project.

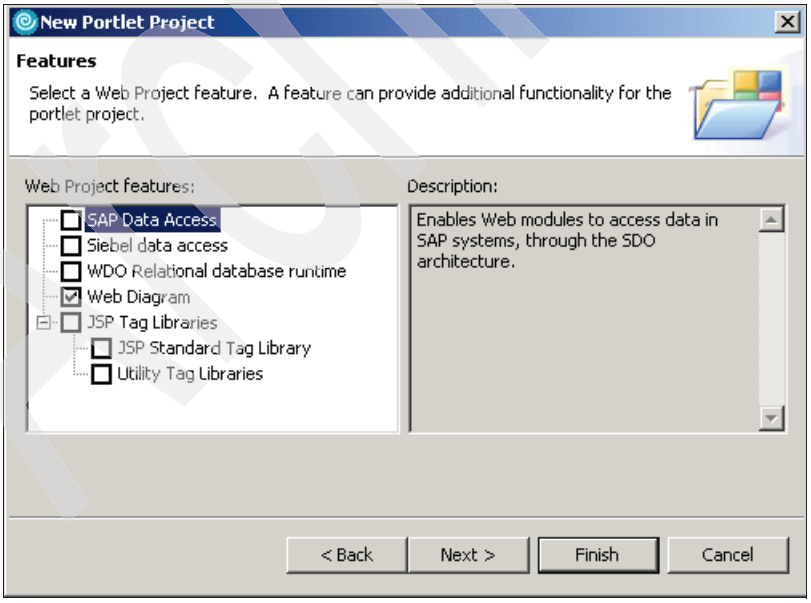


Figure 2-14 Portlet features window

In the Portlet Settings window, update or add any general portlet settings. The application name is the name of the portlet application as used to manage it by the portal administrator. To update this name after generating your portlet project, use the deployment descriptor editor to modify portlet.xml. Modify the Display name of each concrete portlet application.

The portlet name is the name of the portlet. It is also used by the portal administrator. It also can be updated used the deployment descriptor editor and modifying the Display name of each concrete portlet.

The default locale specifies a default locale to use if the client locale cannot be determined. You can add supported locale using the deployment descriptor editor and adding a locale to each concrete portlet.

The portlet title appears in the portlet title bar. To update this in the future, you can use the deployment descriptor editor to modify the title of each concrete portlet.

Change code generation options can be used to change the package and class prefixes.

Click **Next** to continue. If creating an empty portlet, the Miscellaneous screen as seen in Figure 2-18 on page 77 will be shown. If creating a Basic portlet, the Event Handling screen will be shown.

Archived

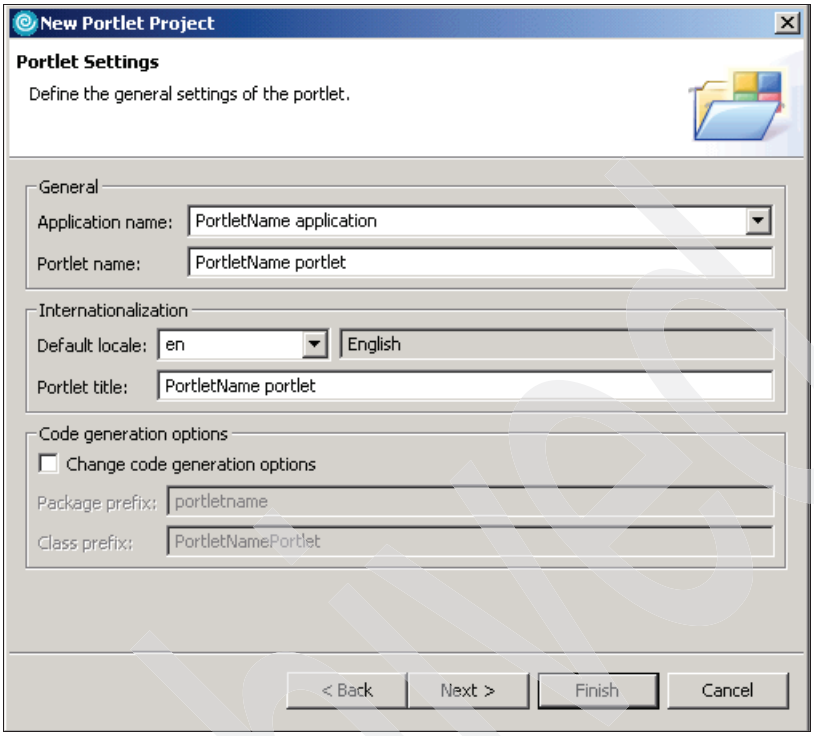


Figure 2-15 Portlet settings window

On this screen, you have the ability to optionally add event handling capabilities to the portlet application. An action event is sent when an HTTP request is received that is associated with a portlet action. The *Add action listener* option implements the ActionListener interface to handle action events. The *Add form sample* option generates code to demonstrate action event handling with a simple form example.

Cooperative portlets provide a model for declaring, publishing, and sharing information with each other using the WebSphere Portal property broker. Cooperative portlets are only available when the Servlet level is 2.3. Cooperative portlets can run on WebSphere Portal V5.x servers. Create a portlet application that extends the PortletAdapter class. *Enable cooperative target* adds a sample WSDL file so that the Click-to-Action target can receive input properties. If you select this option with the *Add form sample* option in the *Action Event* handling section of this screen, the generated portlet project will be enabled as a Click-to-Action receiver. It is also possible to create an action handler and form and customize the WSDL file as required. The *Add Click-to-Action sender portlet sample* adds a simple Click-to-Action sender portlet that is useful to test receiver function and provides sample code. The *Enable cooperative source*

adds the Click-to-Action tag library directive for JSP files of the Click-to-Action source portlet.

Message events can be sent from one portlet to others if the recipient portlets are placed on the same portal page as the sending portlet. To get a Java class that implements the MessageListener interface, select the **Add message listener** option. The *Add message sender portlet sample* generates a sample message sender portlet.

To add a function showing events received by listeners in View mode, select **Add event log viewer**. To select this option, you need to add at least one of the event listeners. The option to *Add edit panel* allows you to change the default maximum event count while in Edit mode.

Click **Next** to continue with the Portlet Project wizard.

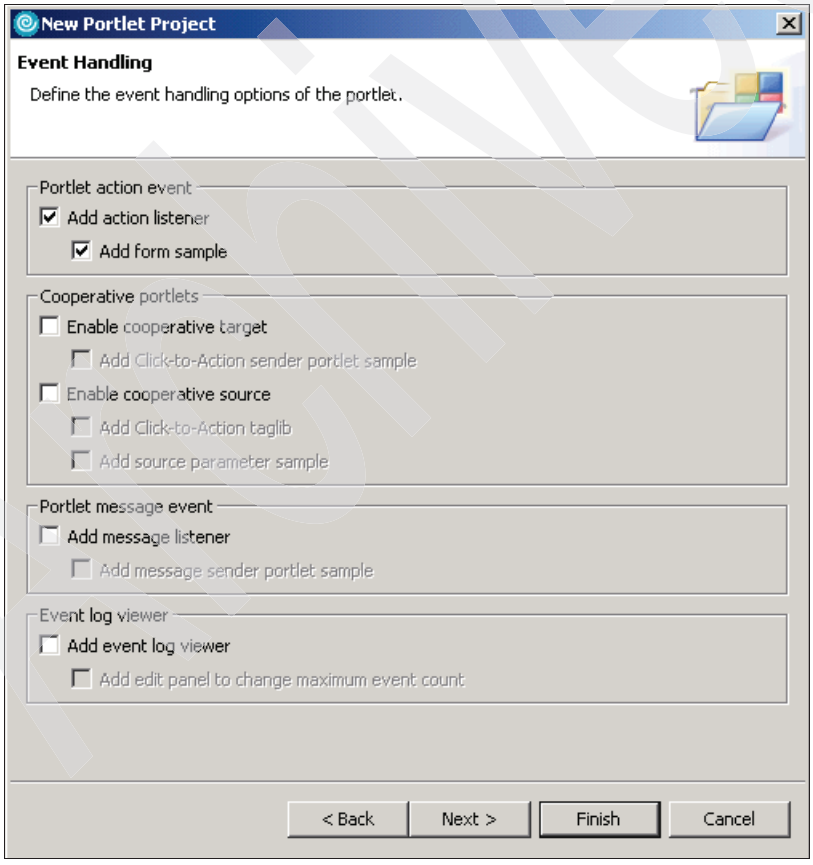


Figure 2-16 Event handling window

Use the Single Sign-On screen to add sample code to support credential vault handling which is used to safely store credentials that are used in portlet authentication. Portlets written to extract users' credentials from the credential vault can hide the login challenge from the user. A portlet private credential vault clot stores user credentials that are not shared among portlets. A shared credential vault slot shares user credentials among all a user's portlets. The administrative credential vault slot allows each user to store their confidential information for accessing administrator-defined resources such as Lotus Notes® databases. A system credential vault slot stores system credentials where the actual confidential information is shared among all users and portlets.

The slot name defines the name of the credential vault slot to store and retrieve the credentials. The **Show password** option allows a password to be displayed on the screen while in View mode.

Click **Next** to continue with the Portlet Project wizard.

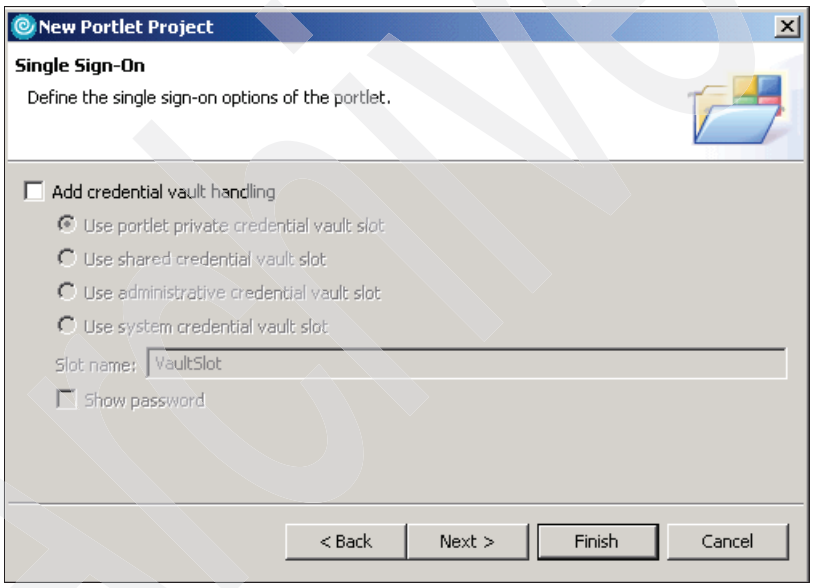


Figure 2-17 Single sign-on window

The Miscellaneous window allows other supported markup languages and portlet modes to be selected. For more information about markup languages, see "Choosing markup languages" on page 56. For more information about modes, see "Portlet modes" on page 50.

Archive

Click **Finish** to generate the new portlet project. You may be presented with a an option to switch to the Web perspective to work on this project. Click **Yes** if the Confirm Perspective Switch is shown.

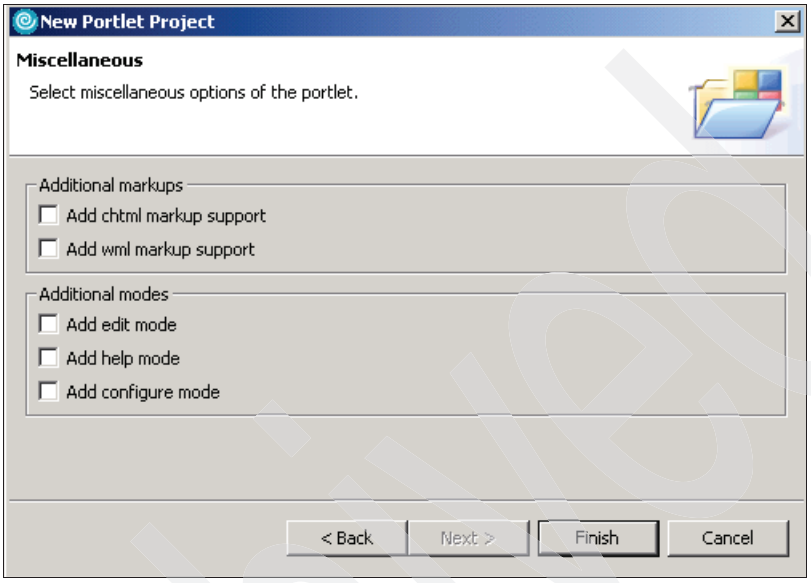


Figure 2-18 Supported markups and modes window

Web perspective

The Web perspective combines views and editors that assist you with Web application development. This perspective is used to edit the project resources, such as HTML and JSP files, and deployment descriptors that make up the portlet project.

Page Designer

Page Designer is an editor for HTML, XHTML, JSP, and Faces JSP files. It provides three representations of each file: Design, Source, and Preview. Each of these provides a different way to work with a file while it is being edited. You can switch between these by clicking the tabs at the bottom of the editor (see Figure 2-19 on page 78):

- **Design:** The design page provides a visual environment to create and work with a file while viewing its elements on the page.
- **Source:** The source page enables you to view and directly work with a file's source code.

- **Preview**: The preview page shows you how the current page is likely to look when viewed in an external Web browser. Previewing dynamic content requires running the portlet or portal page on a local or remote test server.

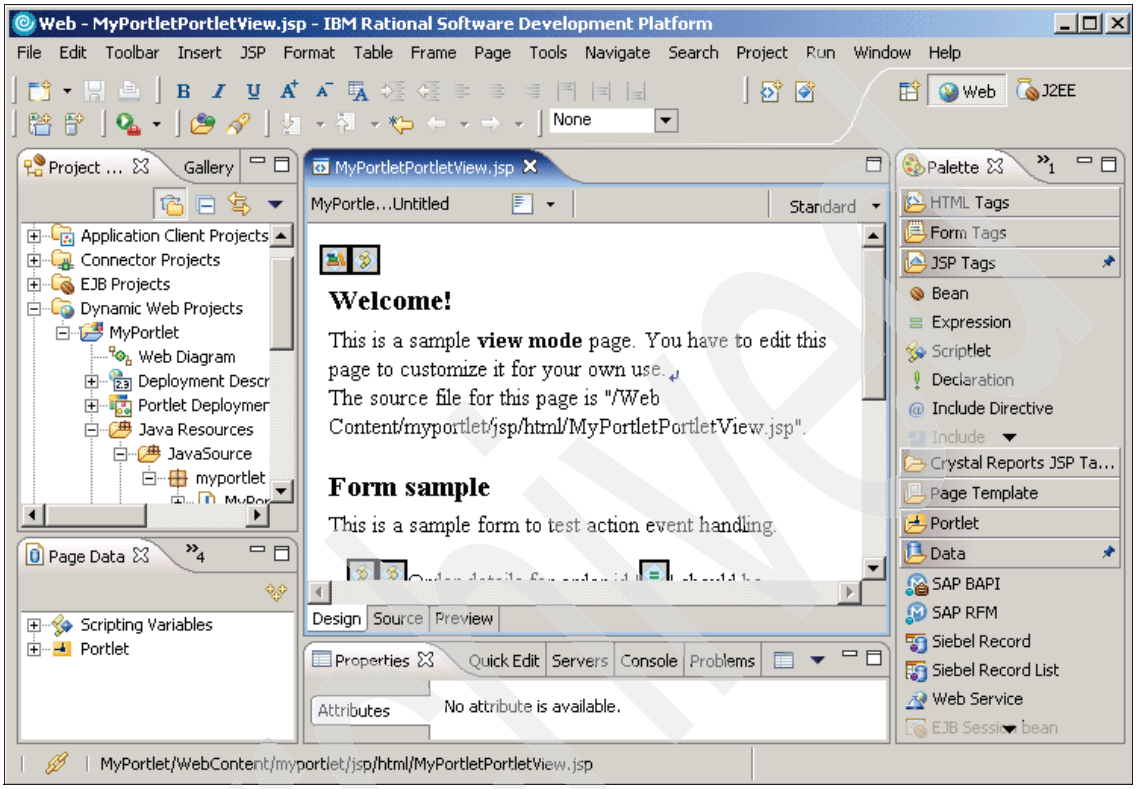


Figure 2-19 Page Designer showing the design page

2.2.5 Portal tools for testing and debugging portlets

Once development is underway, you will need to test and debug your applications. Rational Application Developer provides many ways for you to do this. When defining a remote (server attach) server for testing, debugging or profiling a portlet project, you must create and configure the server.

Portal Server configuration

Portlet tools provide an additional type of server configuration, called the portal server configuration, which contains the server configuration information needed to publish your portlet application on a WebSphere Portal machine. After it is published, your target portlet will appear on the test page and the debug page of your WebSphere Portal. Source-level debugging is also supported.

Remote server test

When developing portlet projects, you have the option of testing and debugging on a remote server or in a local test environment (as described in the next section). To test portlets on a remote WebSphere Portal Server, you will use this feature.

Before testing portlets with a Server Attach server, you may need to configure the remote server. See the section titled *Preparing WebSphere Portal for remote testing and debugging* in the product help for more information. The configuration steps detailed in this section are required when performing any of the following tasks.

- Testing or debugging with multiple users to the same remote server.
- Testing or debugging a JSR 168 portlet on WebSphere Portal 5.0.2.
- Debugging to a remote Server Attach server.
- Testing or debugging to a remote server behind a firewall.
- Testing or debugging to a remote server running Linux®.

Important: If multiple users are testing portlets to the same portlet server, ensure that the UIDs of the portlets are unique. Otherwise, when the portlet is installed on the portlet server, it may replace the original portlet using that UID.

- For the IBM portlet API, modify the UID using the portlet deployment descriptor editor.
- For the JSR 168 portlet API, the UID is constructed using the ID attribute of the portlet-app element.
- If the ID attribute is not specified, the UID is generated automatically using the login user ID and project name.

To use the remote server testing feature, do the following:

1. Right-click your portlet project and select **Run** → **Run on Server**.
2. To use an existing server, select **Choose an existing server** and choose a **WebSphere Portal Server Attach** server from the list.
3. To define a new external test server, you will need to use the New Server Wizard to configure it. See the section titled *Configuring remote servers for testing portlets* in the product help.
4. Click **Finish**.

   After the server starts and the portal is deployed, the Web browser opens to the URL of the portal application on the external server.

Note: An XML exception occurs and the server attach fails to start if the project name, the file name, the file directory structure or the user ID for WebSphere Portal login name is excessively long.

To correct this, shorten the length of the filename, the file directory structure or the User ID for WebSphere Portal login at the WebSphere Portal Server Attach server configuration.

WebSphere Portal Test Environment

Rational Application Developer includes the WebSphere Portal Test Environment to locally test and debug portlet applications.

The WebSphere Portal Universal Test Environment allows you to locally test and debug portlets developed with the Portal Tools from within the Rational Application Developer workbench. This is similar to running a Java servlet webapp in the WebSphere (Application Server) Test Environment.

The test environment is a WebSphere Portal runtime built on top of the WebSphere Test Environment. By default, the test environment uses Cloudscape as the portal configuration database. This can be configured to use DB2 UDB or Oracle.

When using the WebSphere Portal Test Environment, the server is running against the resources that are in the workspace. It supports adding, changing, or removing a resource from the portlet project without needing to be restarted or republished for these changes to be reflected.

To run your project in the WebSphere Portal Test Environment, right-click the portlet project and select **Run** → **Run on Server**. The Server Selection dialog is displayed. You may either choose to run the application on an existing server or manually define a new server.

To define a new local test server, perform the following steps:

1. Choose the option to **Manually define a server.**
2. Select **WebSphere Portal V5.0 Test Environment** from the list of server types.

   Note: You must have installed the WebSphere Portal V5.0 Test Environment when installing IBM Rational Application Developer for this option to be available.

3. Click **Next**.

4. On the WebSphere Server Configuration Settings page, select one of the following values:
   - Select **Use default port numbers** and set the HTTP port number to use the default HTTP port (9081).
   - Select **Use consecutive port numbers** and set the First port number to use port numbers other than the default numbers used by WebSphere Application Server.

     This setting causes the Test Environment to use sequential port numbers, starting with the number you specify. You must specify a port number that begins a range of port numbers that are not being used by another application. This option allows you to have an external Portal server or WebSphere Application Server running on your system, and allows the Test Environment to use different port numbers. You can also configure the Test Environment server's HTTP port numbers by editing the server configuration, as explained below.
5. Click **Finish**.

   Additional options for local servers can be viewed and changed by double-clicking the server in the Servers view. This opens the server configuration editor. You can change any of the settings that were defined previously. In addition, the Portal tab has several additional settings that can be changed to suit your individual configuration.

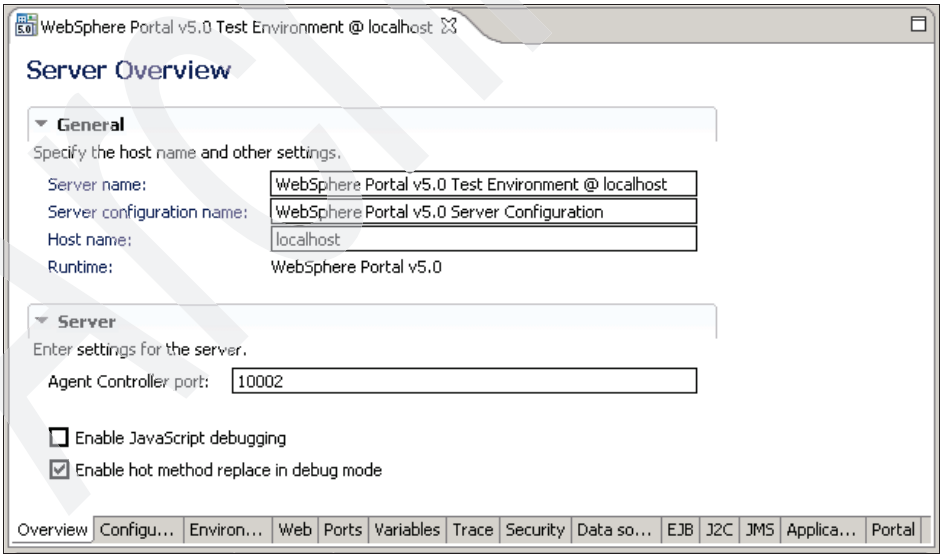


Figure 2-20 Server configuration editor

When you test a portlet on the local test environment server, the default theme and skin are used.

The test environment does not support features that rely on WebSphere Enterprise Edition (personalization and asynchronous rendering of portlets) or LDAP. Transcoding is also not enabled by default. It must be enabled to use a WML device emulator when developing portlets for mobile phones and other devices. See “Choosing markup languages” on page 56 for instructions on how to do this.

When testing or debugging, you may experience the following limitations:

- Help mode does not function correctly in the test environment while using the internal browser. Using an external browser corrects this issue.
- Single sign-on using LDAP is not supported when using the local test environment. LDAP is supported when testing portlets by remotely attaching to another WebSphere Portal server.
- You cannot create new portal users while in debug mode. Use the normal mode to create users.
- Portlet modifications are not previewed correctly when the portlet has been cached by the browser. Logging out and back in to the portal server corrects this.
- If using Linux, you may not be able to start the test environment server without the appropriate user permissions. Users need full permissions on `<STUDIO_HOME>/runtimes/portal_v50/cloudscape/wps50.`

2.2.6 Portal tools for deploying and managing portlets

When development is complete, these tools will help you to load your completed portlet project onto a WebSphere Portal Server.

Exporting WAR files

Exporting a portlet project to a WAR file allows you to install it on a WebSphere Portal server. To export a WAR file for a portlet project, do the following:

1. Right-click the portlet project and select **Export** → **WAR file**. The Export wizard opens.
2. On the WAR Export page, select a destination directory for the WAR file. Enter a name for the WAR file or accept the default.
3. Select the **Export Source files** checkbox to include source files in the WAR file. When deploying to a WebSphere Portal Server, you do not need to include the source files. If you were exporting a WAR file to continue development on another machine, you would want to select this option.

4. Select the **Overwrite existing file** option to replace an existing WAR file with the same name.
5. Click **Finish**.

Install the WAR file on the WebSphere Portal server by using the WebSphere Portal administrative tools.

Remote Server Deploy

Remote Server Deploy is a function that allows portlets developed for a WebSphere Portal V5.0 Server to be deployed in an automated fashion.

This functionality is not available for WebSphere Portal V5.1 Servers. To deploy portlets to a WebSphere Portal V5.1 server, you must export portlet projects to WAR files, and then install them to WebSphere Portal V5.1 using the WebSphere Portal administration interface. The process of exporting WAR files is described in “Exporting WAR files” on page 82.

To deploy a portlet project to a WebSphere Portal server, follow the steps below.

1. Right-click the portlet project and select **Deploy Portlet**. The Deploy Portlet wizard opens.
2. Select an existing server from the list or create a new one. Then click **Next**.
3. On the Portlets page, define these options for Portlet Overwriting:
   a. Select **Automatically overwrite portlets to replace existing portlets without warning**.
   b. Select the **Update** or the **Remove & Deploy** option.
      i. Use the **Update** option to install the portlet, but preserve any customization data that was added in the configure or Edit modes.
      ii. Use the **Remove & Deploy** option to remove and reinstall the portlet. During the removal process, all customization data is also removed, and portlets are removed from any pages where they were already placed. The install process only installs portlets, but does not restore customization data nor place portlets on pages. Use this option if you want to clean up portlet settings, or your portlet is not compatible with the old version.
4. Click **Finish**. Do not interrupt the deployment process.

Archived

Note: An XML Exception occurs and the server attach fails to start if the project name, the filename, the file directory structure or the User ID for WebSphere Portal login name is excessively long. To correct this, shorten the length of the filename, the file directory structure or the User ID for WebSphere Portal login at the WebSphere Portal Server Attach server configuration.

Portal administration

Administrative portlets can be enabled in the server configuration by using the Portal Server Configuration editor described in “Portal Server configuration” on page 78.

You can use the administrative portlets to configure advanced options when running portal and portlet projects.

There are several limitations to using the administrative portlets. You cannot install portlets using the administration portlets. In addition, any changes that are made are reset to the default values the next time the Test Environment is started.

It is recommended that you only use this option when necessary. It affects the performance of the Test Environment.

To debug portlets in a particular layout, use the test and debug options of a portal project, not the administration portlets in the test environment.

2.2.7 Enterprise Application Integration Portal tools

Rational Application Developer also includes some tools to help you with Enterprise Application Integration with SAP and Siebel.

Service Data Objects and tools

Service Data Objects (SDO), the JSR 235 standard, is a new model for representing data, accessing persistent data, and passing data between tiers. It provides a single, consistent interface to any data source.

The JSF tools for SDO in IBM Rational Application Developer provide minimal or zero coding for building dynamic data-bound JSPs.

IBM has included SDO mediators for applications including SAP and Siebel that are supported on WebSphere Portal V5.1 Servers.

SDO mediators are added to portlets through drag-and-drop from the Palette view and the Page Data view.

Archived

Business Process Portlet development tools

The Portal Tools in IBM Rational Application Developer V6.0 also include support for Business Process Execution Language (BPEL)-based business process portlet development. These portlets are supported on WebSphere Portal V5.1 servers.

To use these tools, process designers develop business processes by using the BPEL-editor and test them in the WebSphere Test Environment.

You can then import the resultant business processes as JAR files to develop and compile task processing portlets using the remote server attach function for testing and debugging portlets.

2.2.8 Coexistence and migration of tools and applications

When installing multiple version of IBM development software and working with portal and portlet projects developed with different versions of development software, there are some important issues to consider.

WebSphere Studio and Rational Application Developer

WebSphere Studio Application Developer and IBM Rational Application Developer can coexist with regards to the Portal Toolkit 5.0.x on WebSphere Studio 5.x.

Portlet Projects (Portal Toolkit 5.0.2.2 and above)

Portlet projects completed using the Portal Toolkit V5.0.2.2 will be migrated automatically to Rational Application Developer V6.0 Portal Tools by either migrating the Portal Toolkit workspace or importing the project using the Project Interchange feature.

During migration of Portal Toolkit V5.0.2.2 projects, some additional changes take place:

- The target server is set to WebSphere Portal V5.0, if no target server is set to the project.
- The portlet build path is corrected
- A portlet project nature is added.

Portlet projects (Portal Toolkit earlier than V5.0.2.2)

If migrating portlet projects from earlier versions of Portal Toolkit (prior to V5.0.2.2), the best practice is to export your portlet projects to WAR files and then import the WAR files into new portlet projects within IBM Rational Application Developer V6.0.

Archive

Manually migrate your portlet projects by following these directions:

1. Export the existing project to a WAR file, and include its source files.
   a. Right-click the project and select **Export**.
   b. Select **WAR file** and **Export source files** and click **Finish**.
2. Import the portlet WAR file into a new portlet project:
   a. In the Portal Tools for Rational Application Developer V6.0, create a new empty portlet project.
      i. Select **File** → **New** → **Project** → **Portal** → **Portlet Project** or **Portlet Project (JSR 168)**.
      ii. Deselect **Create a portlet**.
      iii. Click **Show Advanced**.
      iv. If you are importing a WebSphere Portal V4.2 portlet, select **2.2** as the servlet version.
      v. Select **WebSphere Portal V5.0** as the target server, and click **Finish**.
   b. Import the WAR file to this new empty portlet project.
      i. Select **File** → **Import**.
      ii. Select **WAR file** and specify the WAR file from the portlet project that you exported.
      iii. Select the newly created empty portlet project.
      iv. Select **Overwrite existing resources without warning**.
      v. Do not select **Delete project on overwrite**.
      vi. Delete the TLD file.

      It is recommended that you delete the portlet TLD file from the project if it exists. Otherwise, you will get a warning message when you rebuild the project. Leaving it may cause a problem when the portlet project is deployed to WebSphere Portal and the TLD file of the portlet is different from the file in the server.

Note: If you are migrating a WebSphere Portal V4.2 portlet, you will need to migrate this migrated portlet project to WebSphere Portal V5.x. Backward compatibility of portlet projects is not supported.s

2.3 Portal development scenario

To gain an understanding of the portal development process, this scenario demonstrates how the Portal Tools can be used to create a portal site.

The portal development scenario is organized into the following tasks:

- Preparing for the sample
- Adding and modifying a portal page
- Creating and modifying two portlets
- Adding portlets to a portal page
- Running the project in the test environment

Note: The sample code described in this chapter can be completed by following along in the procedures documented. Alternatively, you can import the sample Portal code provided in the `c:\6449code\portal\Portal.zip` Project Interchange file. For details refer to Appendix A, "Additional material" on page 1003.

When importing the Project Interchange file, we found some errors when using the IBM WebSphere Portal V5.0.2 Test Environment due to issues related files outside of the scope of the Project Interchange packaging (specifically, themes and related JSPs).

2.3.1 Preparing for the sample

Prior to working on the portal development scenario, ensure you have prepared the environment by installing the Portal Tools and WebSphere Portal Test Environment (V5.0.2 or V5.1).

Installing the Portal Tools

For details on installing the Portal Tools as component of the Rational Application Developer installation, refer to Chapter 3, "Portlet development platform sample installation" on page 101.

Installing the WebSphere Portal Test Environment

For the scenario in this chapter, you can install either the V5.0.2 or V5.1 WebSphere Portal Test Environments (sample applies to both).

For more information about installing the WebSphere Portal Test Environments refer to the following:

- IBM WebSphere Portal V5.0.2 Test Environment:

  Refer to Chapter 3, "Portlet development platform sample installation" on page 101

- IBM WebSphere Portal V5.1 Test Environment:

  Refer to Chapter 3, "Portlet development platform sample installation" on page 101.

Installing the Rational Application Developer V6 interim fix

We recommend that you install the latest Rational Application Developer interim fixes. For details refer to Chapter 3, “Portlet development platform sample installation” on page 101.

Starting Rational Application Developer

To begin, start the IBM Rational Application Developer workbench. By default, click **Start** → **Programs** → **IBM Rational** → **IBM Rational Application Developer V6.0** → **Rational Application Developer**.

Once Developer is open, you will begin using the Portal Tools to develop a portal site as instructed below.

2.3.2 Creating a portal project

To create a portal project, do the following:

1. Select **File** → **New** → **Project**.
2. When the New Project dialog appears, select **Portal Project** and then click **Next**.
3. If prompted with the dialog displayed in Figure 2-21, click **OK** to enable portal development capabilities.

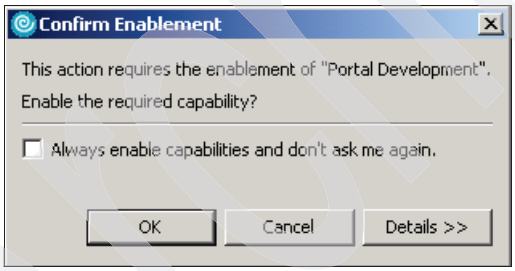


Figure 2-21 Enable portal development

4. When the Portal Project dialog appears, enter `MyPortal` in the Name field, click **Show Advanced** to see more options (we accepted the defaults), and then click **Next**.
5. When the Select Theme dialog appears, select the desired theme. For example, we select the **Corporate** theme and then clicked **Next**.
6. When the Select Skin dialog appears, select the desired skin. For example, we selected the **Outline** skin and then clicked **Finish**.

   This will generate the framework for the portal site.

Archived

7. If prompted to change to the Web perspective as seen in Figure 2-22, click **Yes**.

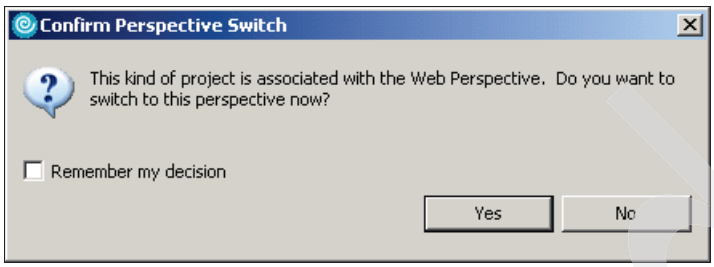


Figure 2-22 Confirm perspective switch

2.3.3 Adding and modifying a portal page

This section describes how to add and modify a portal page for a portal site.

To add a new portal page, do the following:

1. Drag-and-drop the **Page** button from the Palette and place it in the same column as the existing *Page1* was created (see Figure 2-23 on page 90).
2. Click **New Page**.
3. Select the **Title** tab from the Properties view at the bottom of the window.
4. Change the page names.

   Change the names of the pages *Page1* and *New Page* to `Top Page` and `Bottom page` respectively.

Archived

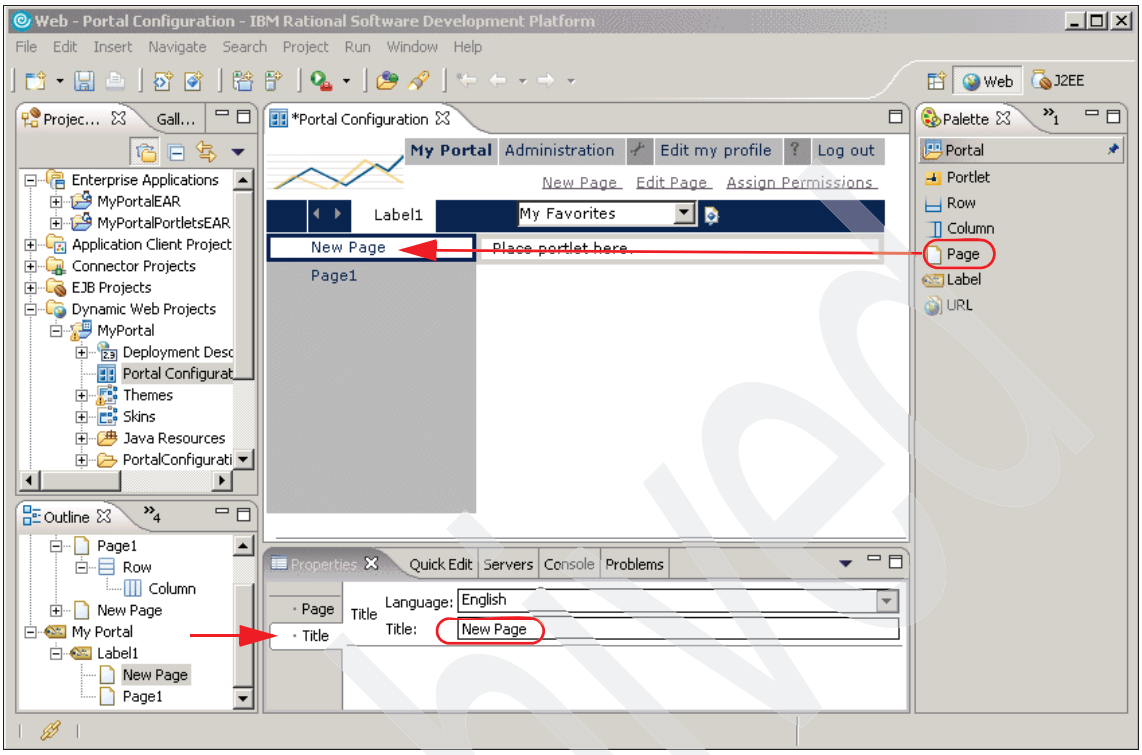


Figure 2-23 Insert a new page and modify title

5. Add a label to the page (see Figure 2-24 on page 91).

   a. Drag and drop the **Label** button from the Palette view, and place it to the right of the existing *Label1*.

   b. Drag and drop the **Page** button from the Palette view onto the *New Label* to add a new page on which to place portlets.

Archive

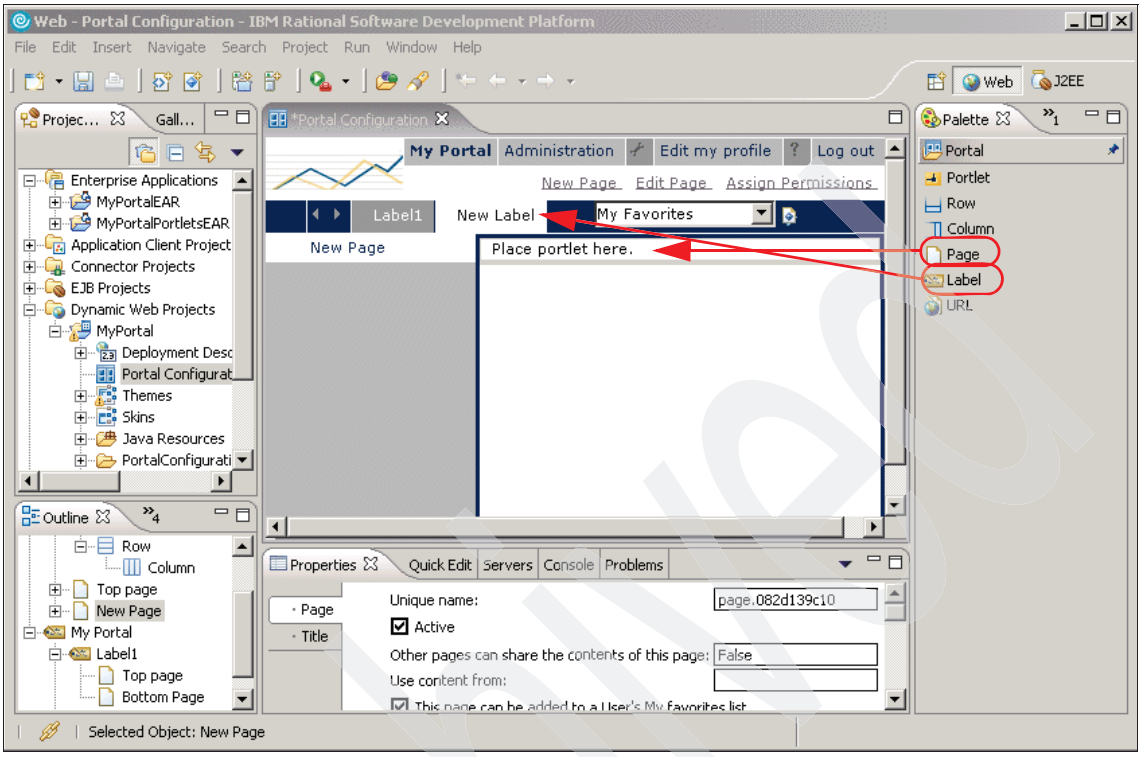


Figure 2-24 Add a new label and page

6. Change the label names (see Figure 2-25).

   In the same way that the titles of pages are modified, change the names of the two labels on the portal site. Name the right label `Right Label` and the left label `Left Label`.

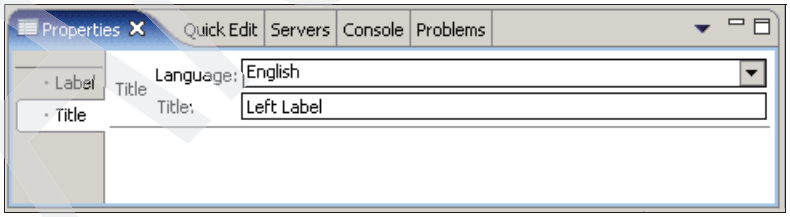


Figure 2-25 Changing label titles

7. Click **File** → **Save All** to save all the changes you have made.
8. By adding labels and pages, you are able to alter the navigational structure of the portal. You can also view an outline view of this structure by looking at the

Outline view which appears in the lower left corner of the workbench (see Figure 2-26).

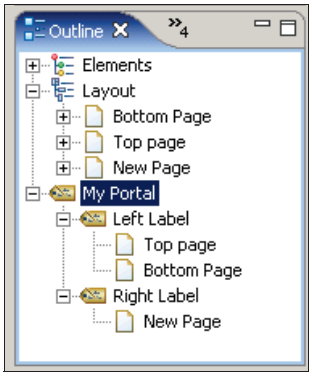


Figure 2-26 Outline view

2.3.4 Creating and modifying two portlets

Now that the portal site and its navigational structure have been defined, we can add content. Content is added to portals by placing portlets on each of the pages. We will create two portlet projects for our example in this section.

Creating the first portlet

To create the first portlet, do the following:

1. Click **File** → **New** → **Project**.
2. When the New Project dialog appears, select **Portlet Project** and click **Next**.
3. Enter `Basic Portlet` in the Name field and click **Next**.
4. When the Portlet Type dialog appears, select the **Basic portlet** type and click **Next**.
5. When the Features dialog appears, uncheck **Web Diagram** and click **Next**.
6. When the Portlet Settings dialog appears, we accepted the default portlet settings and click **Next**.
7. When the Event Handling dialog appears, do the following and then click **Next**:
   - Uncheck **Add form sample**
   - Uncheck **Add action listener**
8. When the Single Sign-on dialog appears, accept the default values for credential vault handling and click **Next**.

Archived

9. When the Miscellaneous dialog appears, check **Add Edit mode** on the miscellaneous settings page, and click **Finish** to generate your portlet code.

   The portlet's View mode JSP is now displayed in the workbench to be edited.

10. Expand Dynamic **Web Projects** → **MyPortal** in the Project Explorer view (see Figure 2-27).

    Under this directory are all the resources associated with the portlet including the supporting JSP files, Java classes, and the portlet's deployment descriptor. You can double-click any resource to edit it in its default editor.

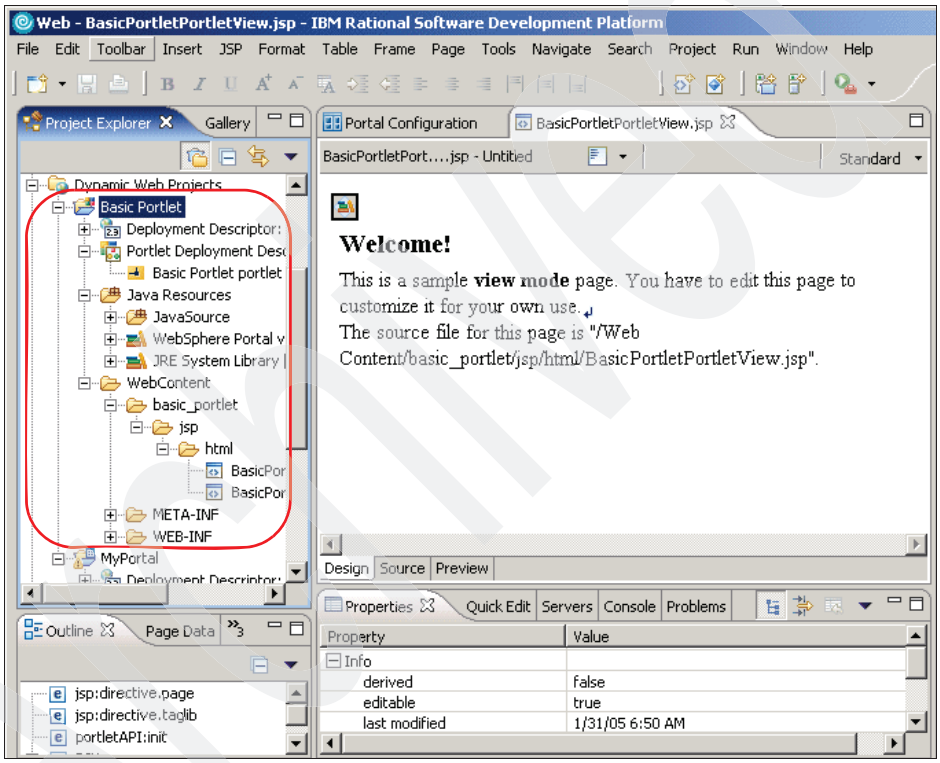


Figure 2-27 Basic Portlet project in the workbench

Creating the second portlet

Now, create a second portlet for the portal site. This portlet will process a form and display the results.

1. Select **File** → **New** → **Project**.
2. When the New Project dialog appears, select **Portlet Project**, and click **Next**.
3. Enter `Form Portlet` in the Name field and click **Next**.

4. When the Portlet Type dialog appears, select **Basic portlet** and click **Next**.
5. When the Features dialog appears, uncheck **Web Diagram**, check the **JSP Tag Libraries**, and then click **Next**.
6. When the Portlet Settings dialog appears, we accepted the default portlet settings, and clicked **Next**.
7. When the Event Handling dialog appears, accept the defaults on the event handling screen. The **Add action listener** and **Add form sample** options should be selected. Click **Next**.
8. When the Single Sign-on dialog appears, accept the default values for credential vault handling and click **Next**.
9. When the Miscellaneous dialog appears, accept the default and click **Finish** to generate your portlet code.
10. In the `FormPortletPortletView.jsp` file that is displayed on your screen, delete Welcome!
11. Figure 2-28 displays a sample View mode page. You have to edit this page to customize it for your own use.

    The source file for this page is as follows, leaving only the form sample to be displayed on this page:

    `/Web Content/form_portlet/jsp/html/FormPortletPortletView.jsp`

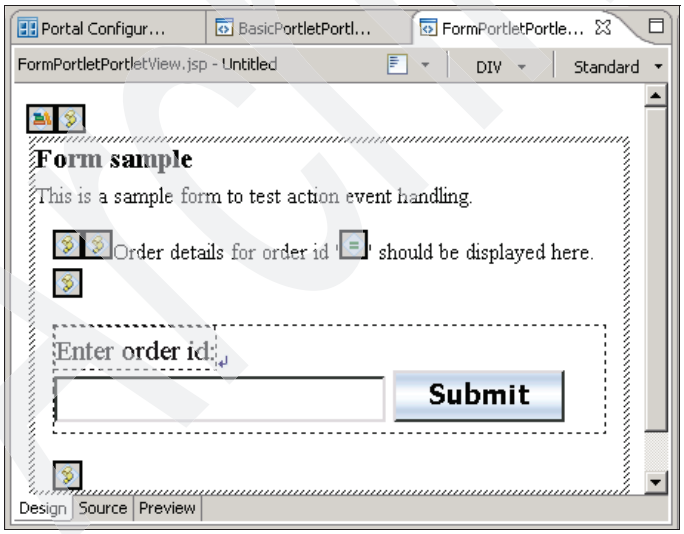


Figure 2-28 Modified second portlet

12. Click **File** → **Save All** to save all the changes made to the portlet projects.

2.3.5 Adding portlets to a portal page

Now return to the Portal Configuration editor used in 2.3.3, “Adding and modifying a portal page” on page 89 to add portlets to a portal page.

1. Expand **Dynamic Web Projects** → **MyPortal**.
2. Double-click **Portal Configuration** to open in the editor.
3. Add portlets to the *Left Label* of the *Top Page*.
   a. Select the **Left Label** of the *Top Page*.
   b. Drag and drop the **Column** button from the Palette View into the area of the *Top Page* that says **Place portlet here**.

      By doing this, the layout of the page is changed to accommodate two portlets side-by-side as seen in Figure 2-29.

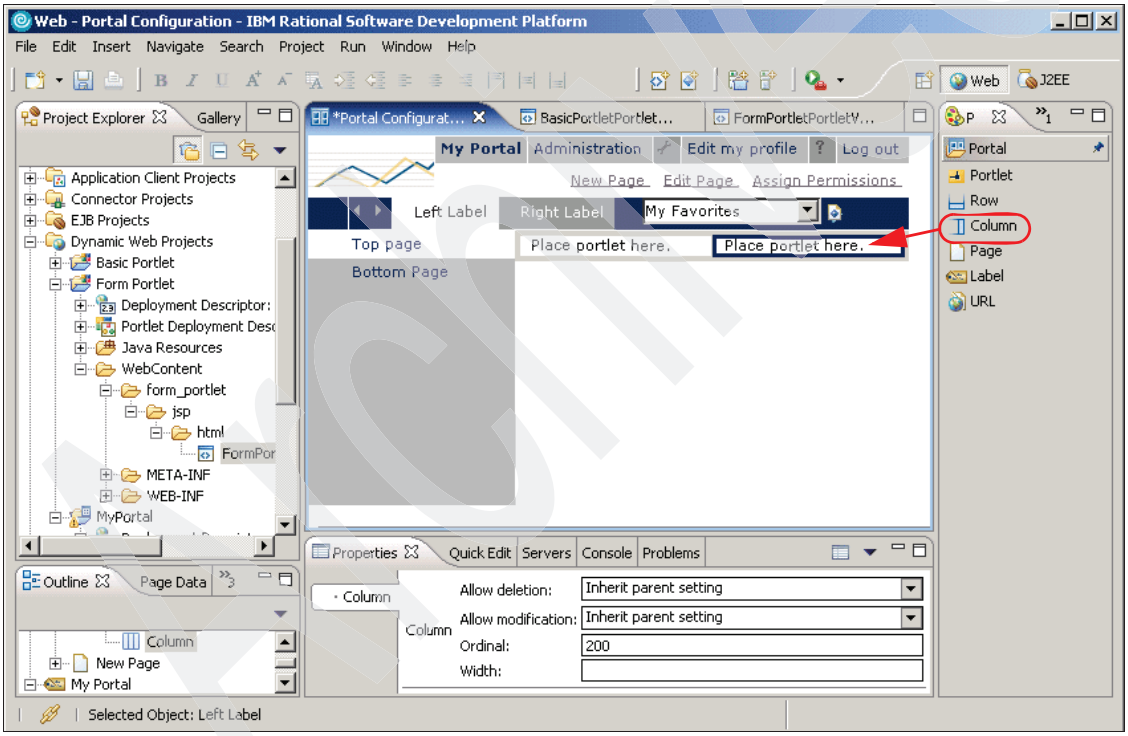


Figure 2-29 Adding a column

   c. Right-click the left column and click **Insert Portlet** → **As Child**.
   d. Select the **Basic Portlet portlet** as seen in Figure 2-30 on page 96, and click **OK**.

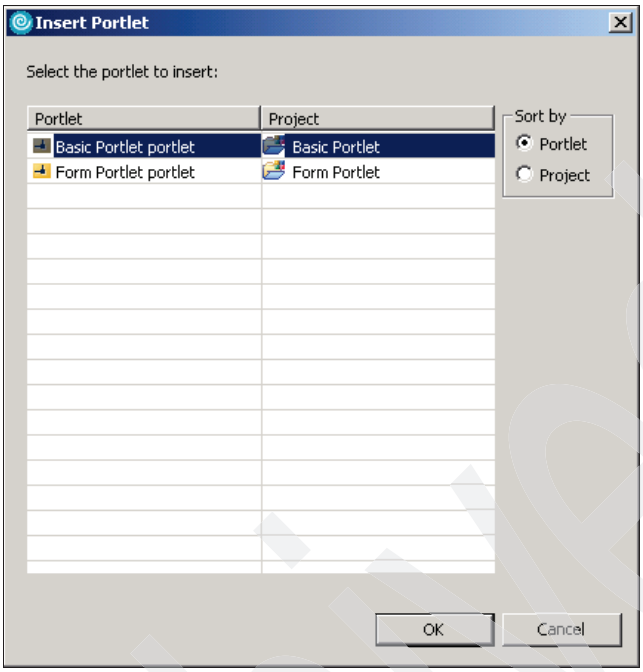


Figure 2-30 Select portlet to insert

4. Add portlets to the *Left Label* of the *Bottom Page*.
   a. Select the **Left Label** of the *Bottom Page*.
   b. Drag and drop the **Column** button from the Palette View into the area of the *Bottom Page* that says **Place portlet here**.
   c. Right-click the right column and click **Insert Portlet** → **As Child**.
   d. Select the **Form Portlet portlet** and click **OK**.
5. Perform the same action to insert the **Basic Portlet portlet** to the Left Label of the Bottom Page (see Figure 2-31 on page 97).

Archive

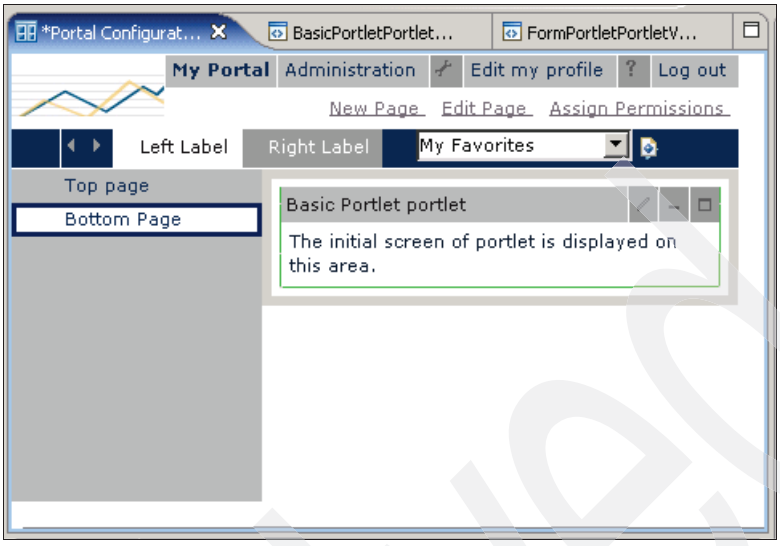


Figure 2-31 Basic portlet on the bottom page of the left label

6. Now click the **Right Label**.
7. Insert the `Form Portlet` onto this page (see Figure 2-32).

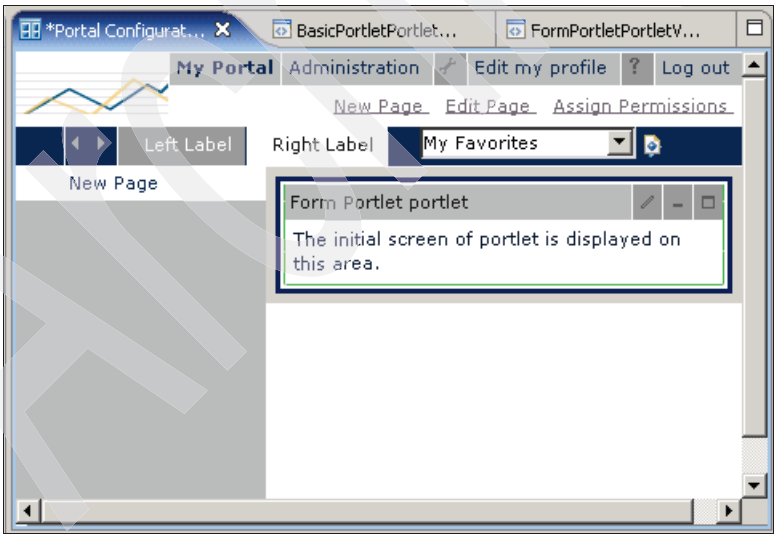


Figure 2-32 Inserting the form portlet into the right label new page

8. Click **File** → **Save All** to save all the changes made to your portal site.

2.3.6 Running the project in the test environment

Now you can run and test the project in the WebSphere Portal Test Environment. This section assumes that you have not previously defined a WebSphere Portal Test Environment server and will configure a server for you as part of the procedure.

1. Open the Web perspective.
2. Expand **Dynamic Web Projects**.
3. Right-click **MyPortal**, and select **Run** → **Run on Server** as seen in Figure 2-33.

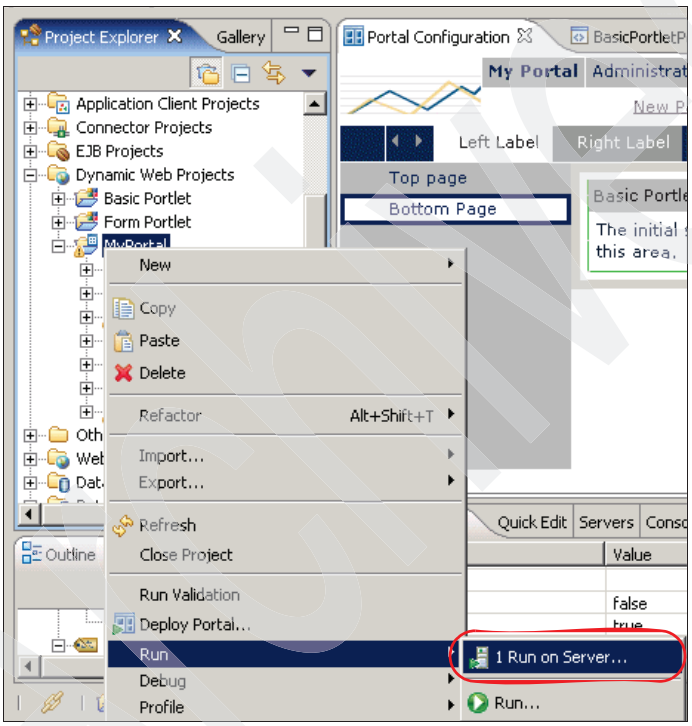


Figure 2-33 Run on server

4. When the Define a New Server dialog appears, select **Manually define a server**, and select the desired WebSphere Portal Test Environment (V5.0 or V5.1). For example, we selected **WebSphere Portal V5.0 Test Environment** and clicked **Next**.
5. When the WebSphere Server Configuration Settings dialog appears, we accepted the default port (9081) and clicked **Next**.

6. When the Add and Remove Projects dialog appears, select each of the following projects and click Add:
   - **Form PortletEAR**
   - **Basic PortletEAR**
   - **MyPortalEAR**
7. When done adding the projects to the Configured projects column, you should have four projects (MyPortalEAR, Form PortletEAR, Basic PortletEAR, MyPortalPortletsEAR) associated with your MyPortal project so that they can run on the server. Click **Finish**.
8. Click **OK** if you receive the Repair Server Configuration dialog window (see Figure 2-34). This indicates that your portlets will be added to the server so that they run in your portal project.

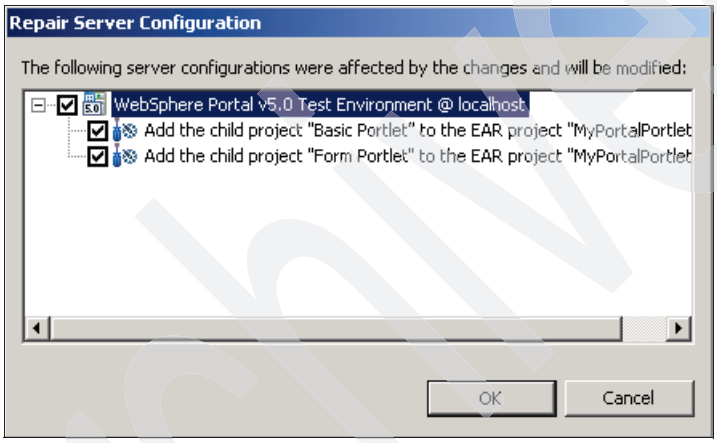


Figure 2-34 Repair Server Configuration dialog

9. The server will now start, and your portal site will load in the Web browser as seen in Figure 2-35 on page 100.

   Test the portal site.

   a. Navigate the portal site using the labels and page links.

   b. Enter Edit mode on the Basic Portlet by clicking Edit **Page**.

   c. Submit a value using the form.

Archive

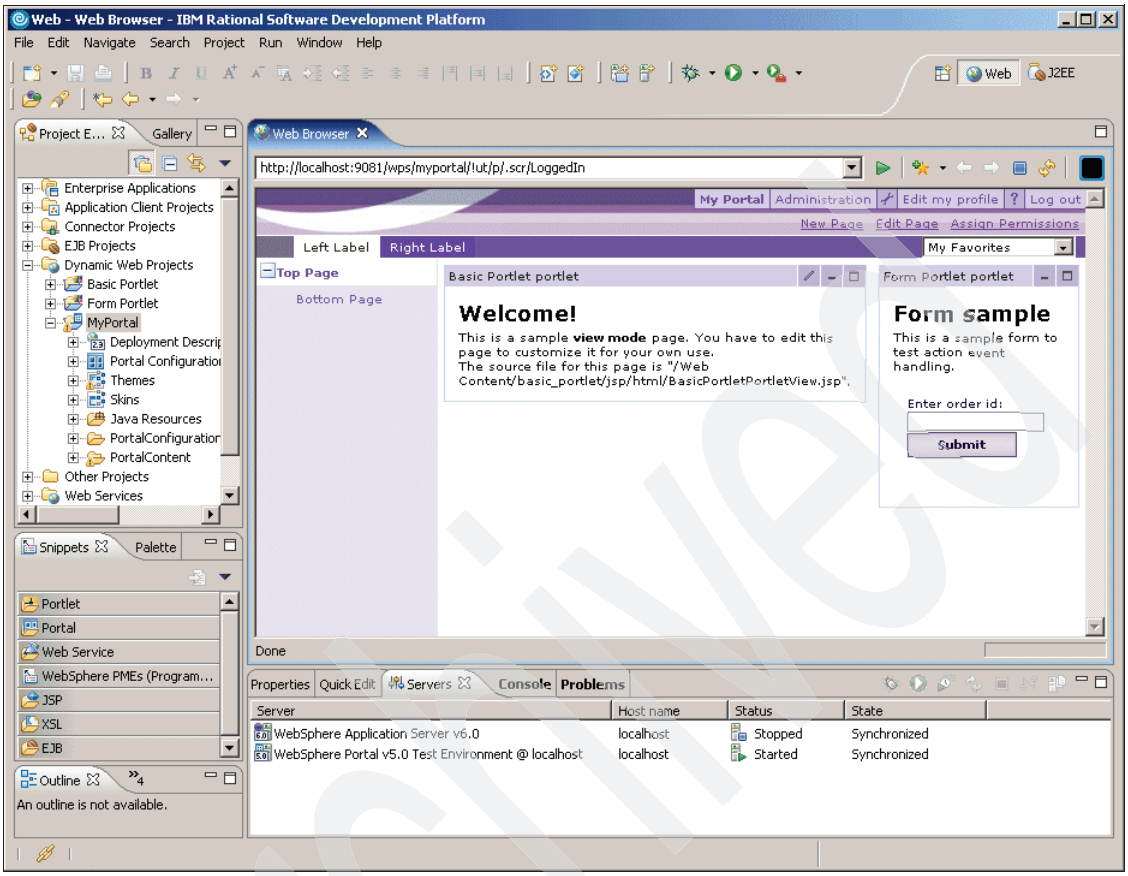


Figure 2-35 My Portal project in Web browser within IBM Rational Application Developer

Archive

3

Portlet development platform sample installation

This chapter illustrates a sample portlet development software installation by describing the steps necessary to prepare a single computer running Windows XP Professional SP1 for WebSphere application development and testing. The steps include:

1. Installation of Rational Application Developer V6.0 and Portal Tools
2. Installation of WebSphere Portal V5.1 Test Environment
3. Configuration of WebSphere Portal V5.1 Test Environment in Rational Application Developer V6.
4. Optional installation of WebSphere Test Environment V5.1

© Copyright IBM Corp. 2005. All rights reserved.

3.1 Prerequisites

The following software and hardware requirements must be met in order to successfully install and use Rational Application Developer V6 for portlet application development.

3.1.1 Hardware requirements

The following hardware must be present before you install Rational Application Developer:

- Pentium® III 800 MHz processor minimum (Higher is recommended)
- 768 MB RAM minimum (1 GB RAM is recommended)
- 3.5 GB of disk space for a full installation
- Display resolution of 1024 x 768 or higher
- TCP/IP stacks must be installed to use the test environments

3.1.2 Software requirements

One of the following operating systems must be present before you install Rational Application Developer:

- Windows XP Professional with Service Packs 1 and 2
- Windows 2000 Professional with Service Packs 3 and 4
- Windows 2000 Server with Service Packs 3 and 4
- Windows 2000 Advanced Server with Service Packs 3 and 4
- Windows Server 2003 Standard Edition
- Windows Server 2003 Enterprise Edition
- Red Hat Enterprise Linux Workstation, version 3.0 (all service packs)
- SuSE Linux Enterprise Server (SLES) version 9 (all service packs)

More information about software requirements can be found in the install.html file on the Rational Application Developer installation disk 1 or online at:

`http://www.ibm.com/developerworks/rational`

3.2 Rational Application Developer and Portal Tools

Prepare the following CDs prior to installing Rational Application Developer V6 and the Portal Tools:

- Rational Application Developer - Disk 1
- Rational Application Developer - Disk 2
- Rational Application Developer - Disk 3
- Rational Application Developer - Disk 4

Archived

Follow these steps to install this product:

1. Insert installation disk 1. If autorun is enabled on your system, the installation launchpad program automatically opens. If autorun is disabled on your system, run **`launchpad.exe`** from the CD to display the Rational Software Development Platform Launchpad.

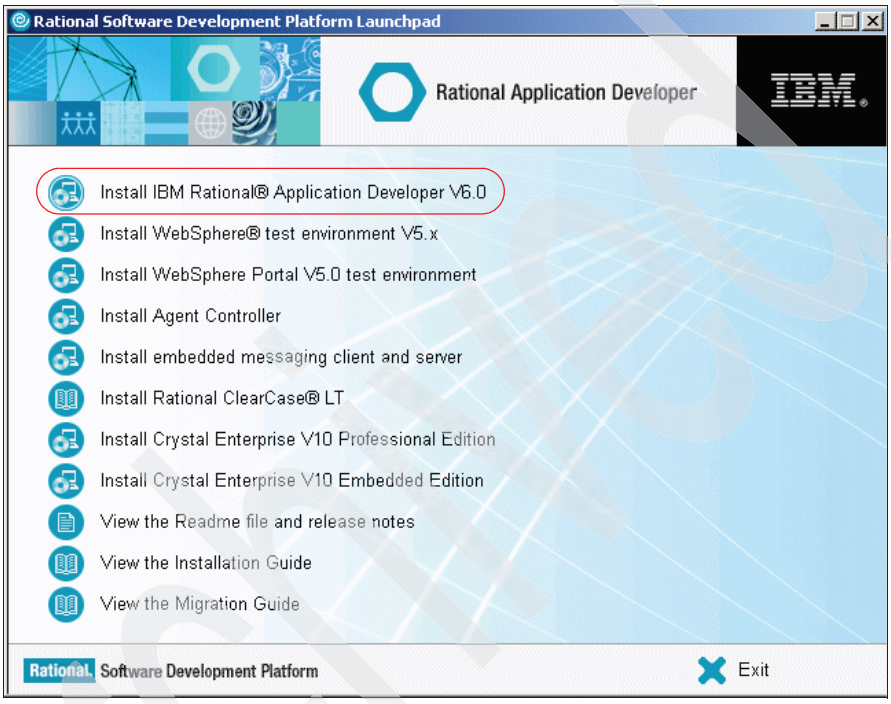


Figure 3-1 Rational Software Development Platform Launchpad

2. Select **Install Rational Application Developer V6.0**.
3. Click **Next** to continue once the installation program opens.
4. Accept the license agreement, and click **Next** to continue.
5. Accept the default installation path: C:\Program Files\IBM\Rational\SDP\6.0\.

Archived

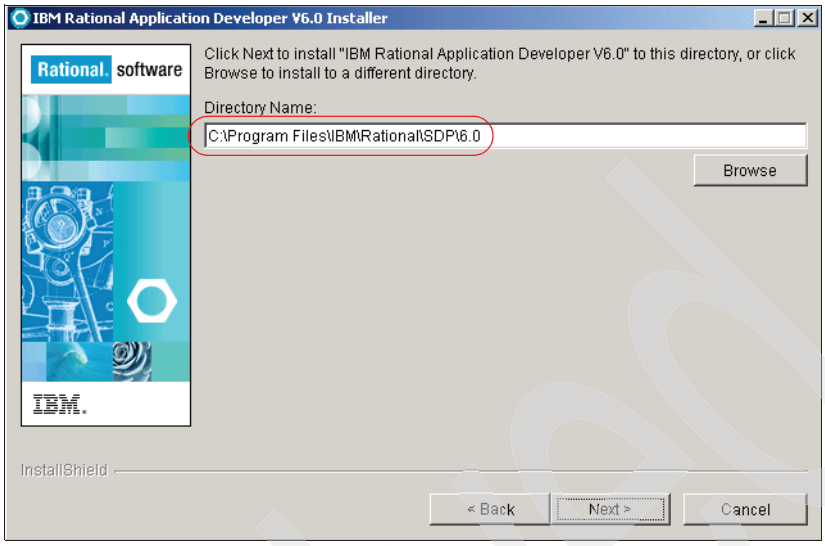


Figure 3-2 Installation directory window

6. In the features window, you may deselect **IBM WebSphere Application Server V6.0 Integrated Test Environment**. It is not required for portlet application development and is not a prerequisite for the WebSphere Portal V5.1 Test Environment. You may wish to leave this feature selected if you will be developing applications for WebSphere Application Server V6.0. You can run the installation program again in the future to add this functionality.

   WebSphere Application Server V5.1 will be automatically installed when installing the WebSphere Portal V5.1 Test Environment as shown in 3.3, "WebSphere Portal V5.1 Test Environment" on page 106.

   Also select **Portal Tools** under the Additional Features heading. This option is required for portlet development. Click **Next** to continue.

Archive

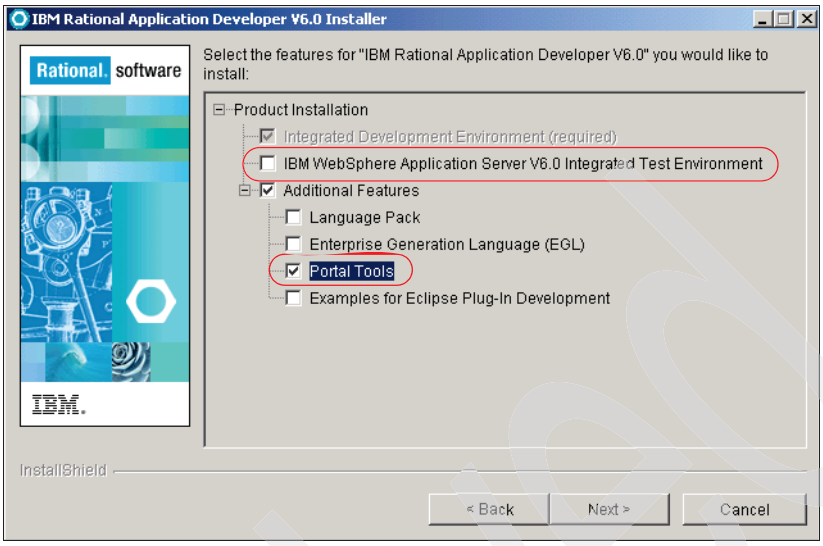


Figure 3-3 Features window

7. In the summary information window, click **Next** to continue with the installation.
8. When prompted, insert disks 2, 3, and 4 and click **OK** after each to continue the installation.
9. Click **Next** to continue after this installation has been completed.
10. You may deselect the **Launch Agent Controller Install** option. Agent Controller is a daemon that allows client applications to launch and manage local or remote applications and provides information about running applications to other applications. You must install Agent Controller before you can use profiling tools to profile your applications, logging tools to import remote log files, component testing tool to run test cases, runtime analysis tools for probe insertion, code coverage and leak analysis, and tools for remote application testing on WebSphere Application Server version 5.0 or 5.1. It is not a required component for portlet development.

> **Note:** Agent Controller can be installed after the installation of Rational Application Developer by selecting **Install Agent Controller** from the launchpad, as shown in Figure 3-1.

More information about this tool can be found online at:

`http://www.ibm.com/developerworks/rational`

Click **Finish** to complete this installation.

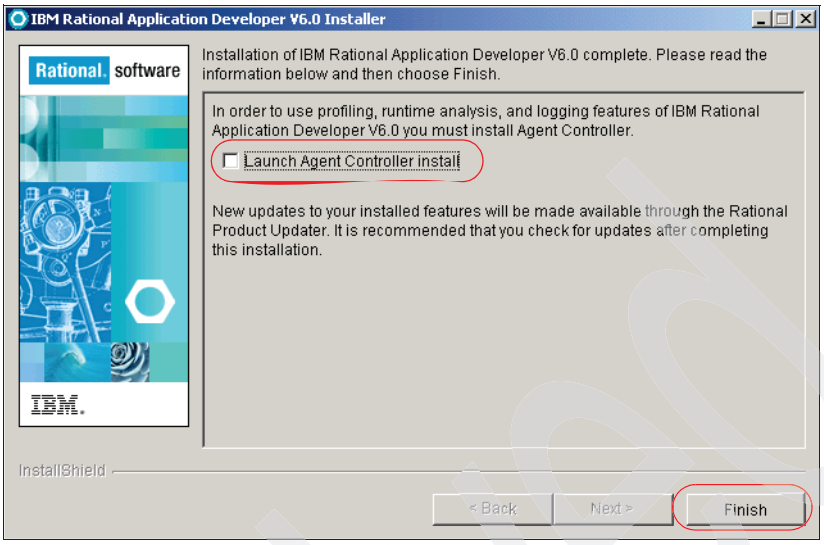


Figure 3-4 Agent Control installation window

3.3 WebSphere Portal V5.1 Test Environment

Prepare the following CDs prior to installing WebSphere Portal V5.1 Test Environment for Rational Application Developer V6. These CDs are bundled with Rational Application Developer V6. They are also shipped with WebSphere Portal V5.1.

- WebSphere Portal V5.1 - Setup
- WebSphere Portal V5.1 - Disk 1-1
- WebSphere Portal V5.1 - Disk 1-2
- WebSphere Portal V5.1 - Disk 1-15
- WebSphere Portal V5.1 - Disk 2
- WebSphere Portal V5.1 - Disk 3

Follow these steps to install this product:

1. Insert the Setup disk. If autorun is enabled on your computer, the installation program will load. If it is not, browse the CD and run **`install.bat`**.
2. Select the installation language to be used with the installation program. Click **OK** to continue.
3. In the next window, you may open and read the InfoCenter or click **Next** to continue the installation.
4. Accept the license agreement and click **Next** to continue the installation.

5. Select the **Test Environment** installation, and click **Next** to continue.

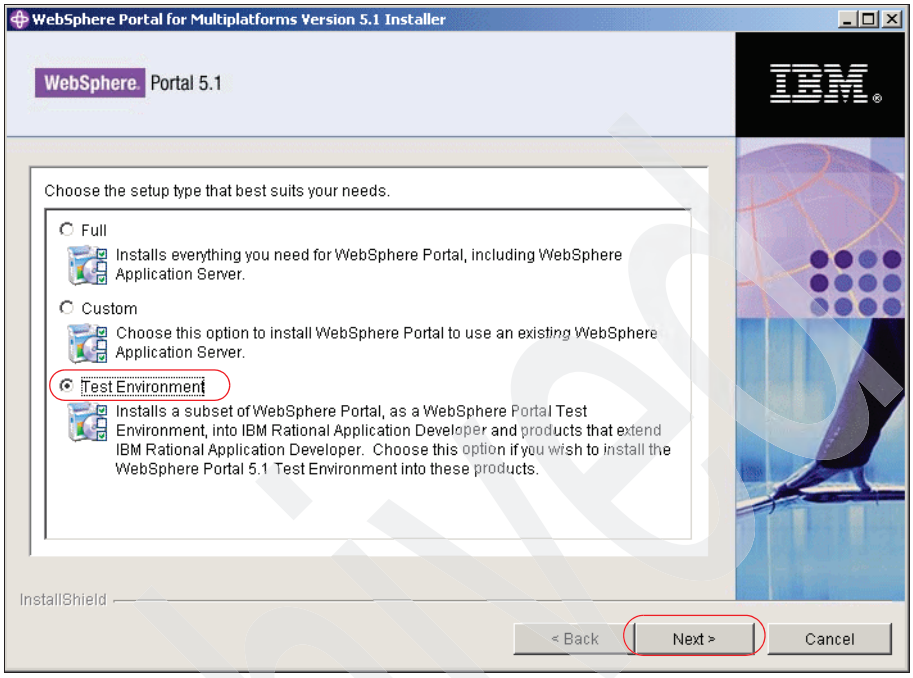


Figure 3-5 WebSphere Portal V5.1 installation window

6. Click **Next** to continue after ensuring that no instances of WebSphere Application Server or WebSphere Portal are running on the local machine.
7. Accept the default installation directory for WebSphere Application Server. Click **Next** to continue.

Archive

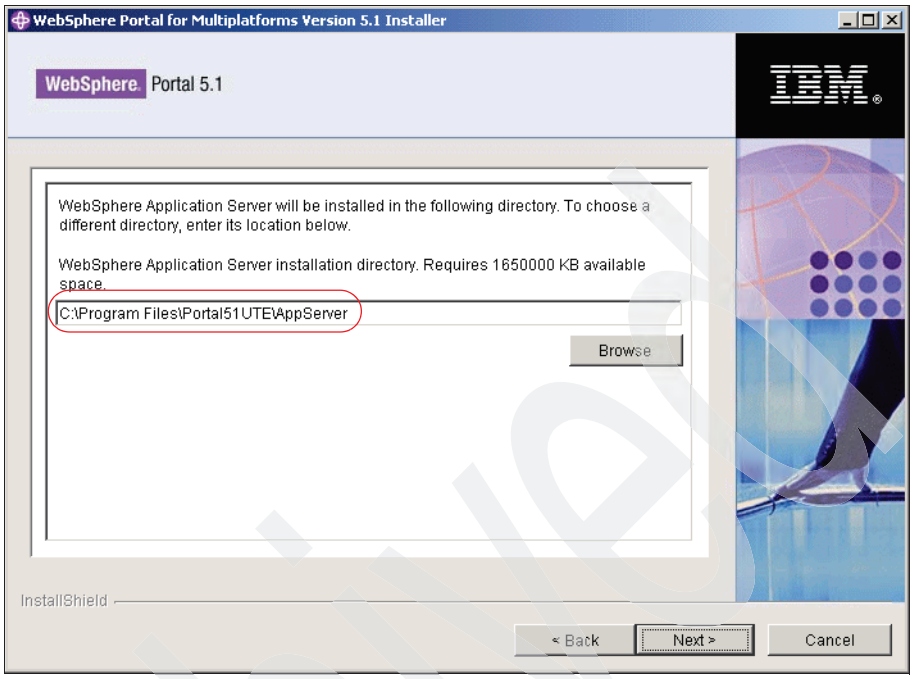


Figure 3-6 WebSphere Application Server installation directory

8. Accept the default installation directory for WebSphere Portal. Click **Next** to continue.

Archive

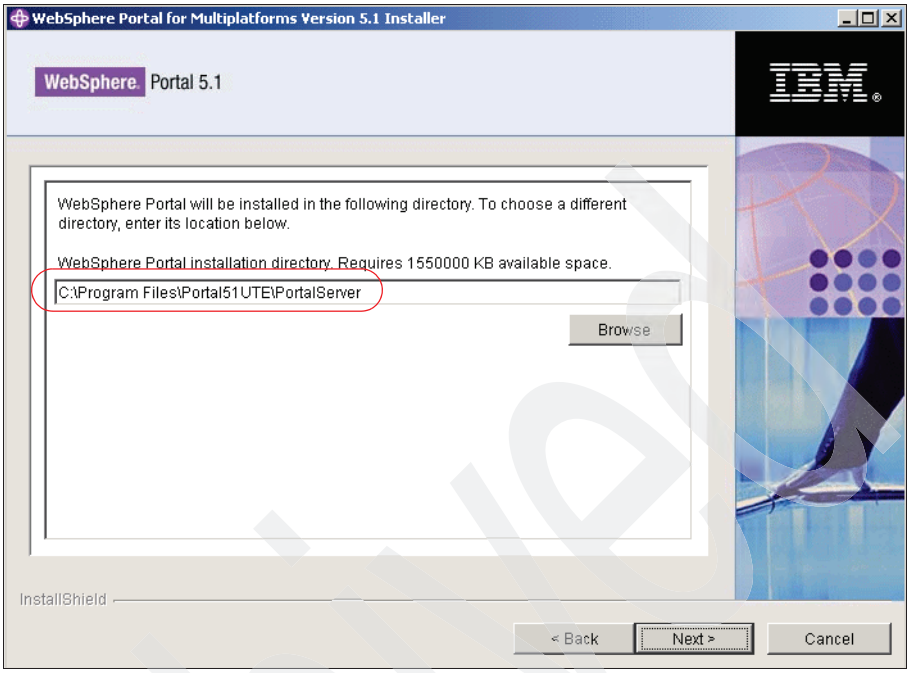


Figure 3-7 WebSphere Portal installation directory

9. Enter information for a WebSphere Portal administrative user and password. For example, enter a username and password of `wpsadmin`. Click **Next** to continue.

Archive

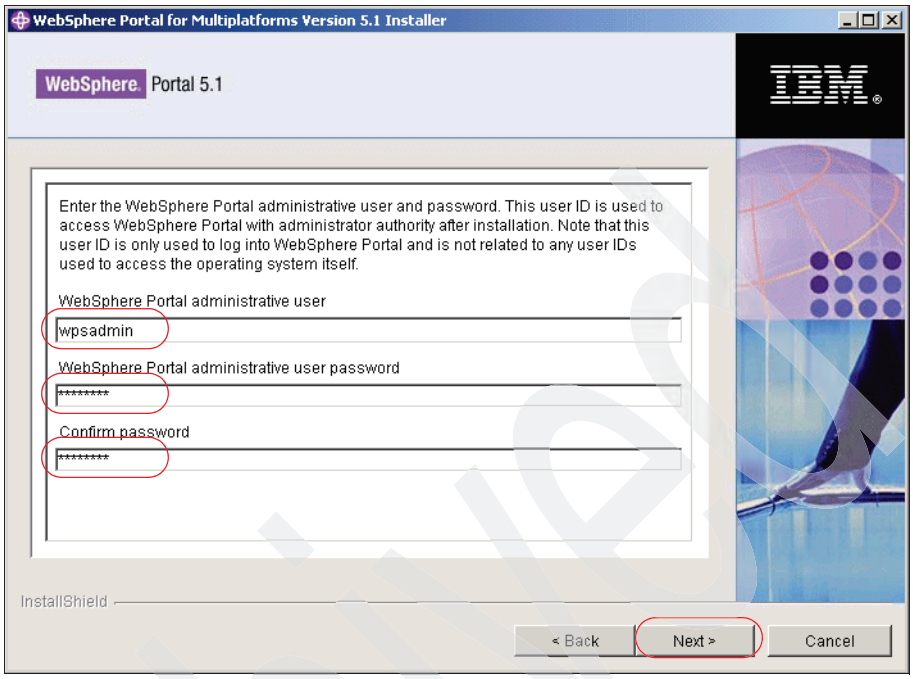


Figure 3-8 WebSphere Portal administrative user and password window

10. Click **Next** to begin the installation.
11. When prompted, insert CDs 1-2, 1-1, 1-15, 2, and 3. Click **Next** after each to continue the installation.
12. Click **Finish** in the final installation window to complete the installation.

3.4 Configuration of the Test Environment

To use the WebSphere Portal V5.1 Test Environment in Rational Application Developer, you must configure it.

Follow these instructions to configure this product:

1. Run Rational Application Developer by clicking **Start** → **Programs** → **IBM Rational** → **IBM Rational Application Developer V6.0** → **Rational Application Developer**.
2. Click **Window** → **Preferences**.

Archive

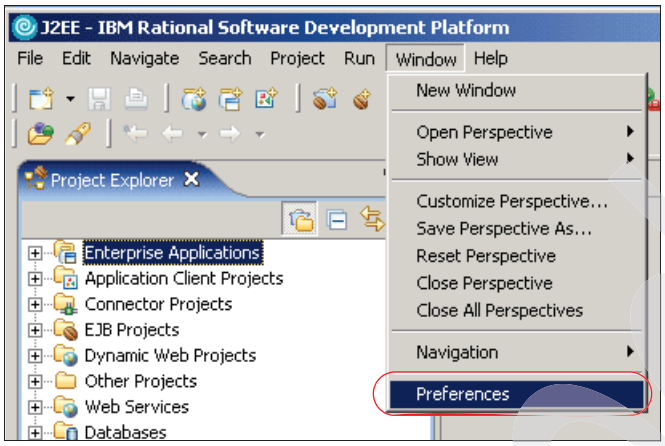


Figure 3-9 Rational Application Developer preferences

3. Expand the Server category and select **Installed Runtimes**. Select the **WebSphere Portal V5.1** stub. Then click the **Edit** button.

Archived

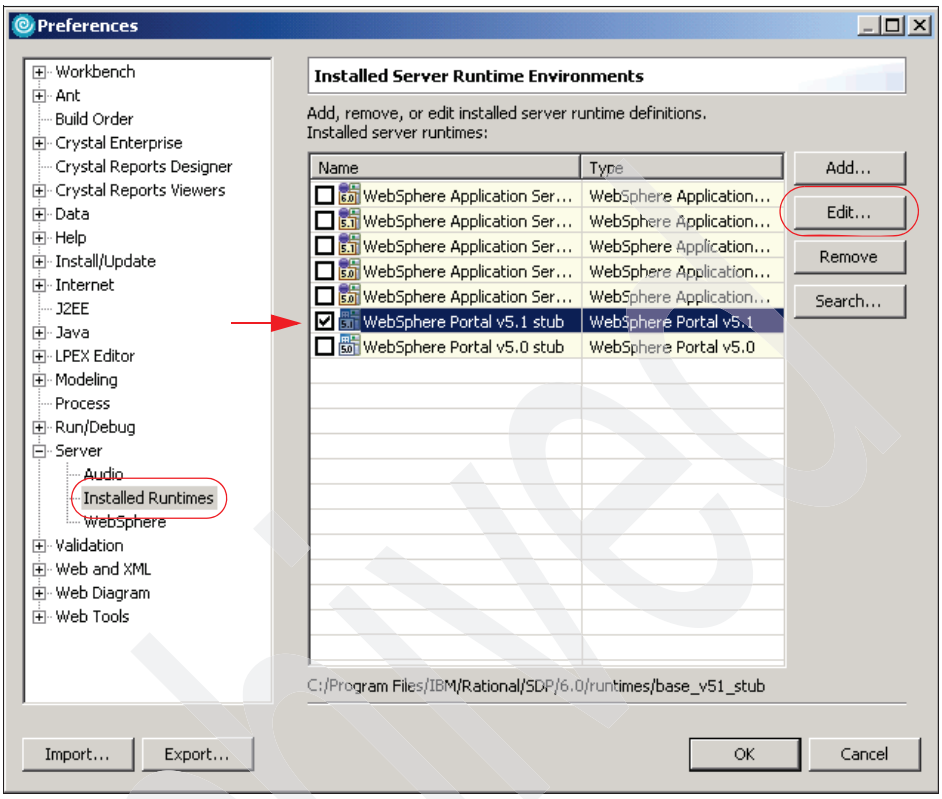


Figure 3-10 Installed Server Runtime Environments window

4. Change the following information:
   - **Name:**
     `WebSphere Portal V5.1`
   - **WebSphere Portal Location:**
     `C:\Program Files\Portal51UTE\PortalServer`
   - **WebSphere Application Server Location:**
     `C:\Program Files\Portal51UTE\AppServer`

Archive

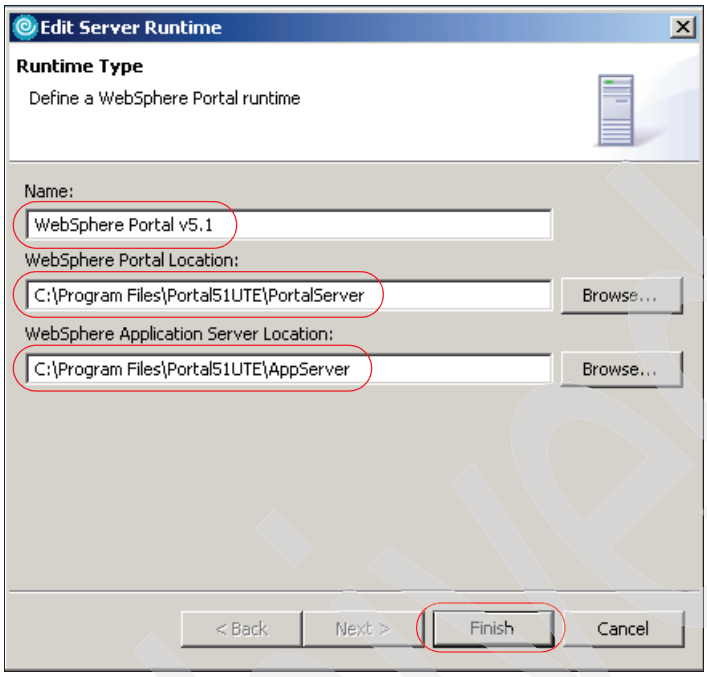


Figure 3-11 Changing server profile

5. Click **Finish** and then **OK** to save these changes.

3.5 WebSphere Test Environment V5.1 (optional)

Occasionally, you may wish to test some application functionality by running a component of your application within WebSphere Application Server instead of within the WebSphere Portal Test Environment. To do this, you will need to install the WebSphere Test Environment. Prepare Rational Application Developer - Disk 1 and Rational Application Developer - WebSphere Test Environment V5.x Disk 1 and follow these steps:

1. Insert Rational Application Developer - Disk 1. If autorun is enabled on your system, the installation launchpad program automatically opens. If autorun is disabled on your system, run `launchpad.exe` from the CD to display the Rational Software Development Platform Launchpad. This launchpad is shown in Figure 3-1 on page 103.
2. Select **Install WebSphere test environment V5.x**. You will be prompted to insert WebSphere Test Environment V5.x Disk 1. Do so and click **OK**.
3. Click **Next** to navigate to the next window.

4. Accept the license agreement and click **Next**.
5. Select **WebSphere Application Server 5.1** from the Integrated Test Environments category and click **Next** to continue.

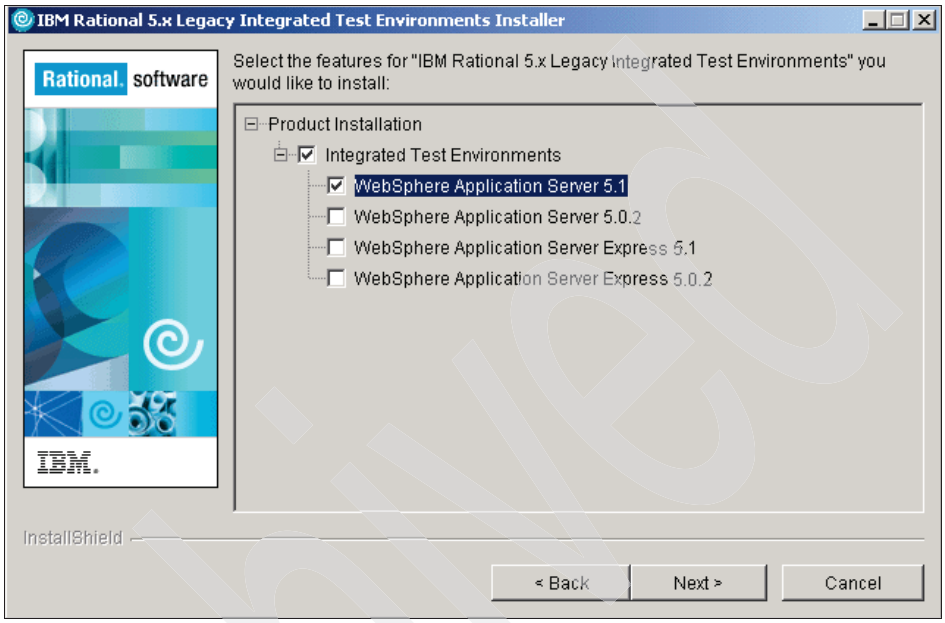


Figure 3-12 Integrated test environments selection window

6. Click **Next** to begin installation of the test environment.
7. Click **Finish** to complete installation of the test environment.

No additional configuration is needed within Rational Application Developer for the WebSphere V5.1 Test Environment to function.

Archive

4

IBM Portlet API

This chapter provides details about 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 IBM 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 IBM 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.

Archived

© Copyright IBM Corp. 2005. All rights reserved.

4.1 IBM portlets

IBM portlets are Web applications that runs in the context of the WebSphere Portal Server. They inherit from the javax.servlet.http.HttpServlet class and as such are treated as servlets by the application server. The portlet is executed inside a Web container managed by the application server. In the IBM 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.

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.

4.2 IBM 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 4-1 on page 117.

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.

Archived

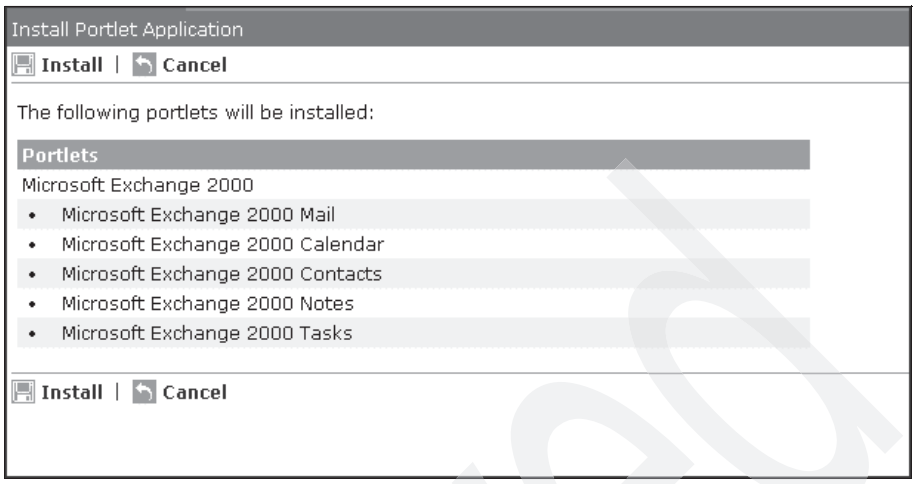


Figure 4-1 A Portlet application with multiple portlets

4.3 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, you can return some content. To allow a portlet to return

Archived

content in a minimized state you will need to update the skin to allow it. 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.

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 a button, an action event is generated and sent to the registered listener. The Portlet API also provides 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.

4.4 Portlet modes

The following portlet modes are supported in the IBM Portlet API:

- **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.

4.5 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.

Archived

Maximized	The portlet view is maximized and takes over the entire body of the portal replacing all the other portal views.
Minimized	By default only the portlet title bar is visible inside the portlet page. You can modify the skin do display the portlets content in a minimized state. When doing this you test for the state before generating content. If the state is minimized you should only generate minimal content.

4.6 Core objects

Portlets are descendents of HttpServlets and as such inherit much of the basic functionality from that class. However, as illustrated in 4.3, “Servlets versus portlets” on page 117, 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:

http://www-128.ibm.com/developerworks/websphere/zones/portal/

4.6.1 Hierarchy

The abstract class Portlet descends from the HttpServlet interface as illustrated in Figure 4-2. Note that the package structure indicates the portlet belongs to the org.apache.jetspeed.portlet package. It is important to understand that the IBM Portlet API and the Jetspeed API are not the same, or even compatible at this time.

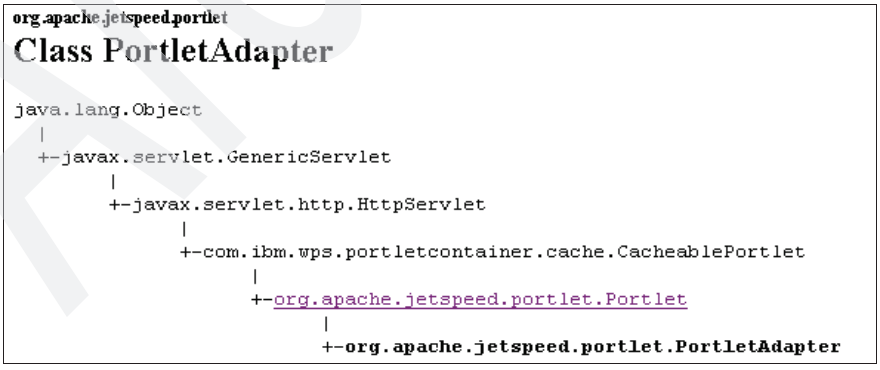


Figure 4-2 Portlet hierarchy

4.6.2 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.

4.6.3 PortletAdapter

This class is provided as a default implementation of the Portlet class. It is recommended that your portlet extends 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.). The service method in PortletAdapter will not call doGet or doPost and therefore should not be used in your portlet. In addition to the methods of the Portlet class, this class defines several additional methods.

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 4-1 illustrates the usage of these methods.

Example 4-1 Setting and Accessing the concrete portlet variable

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

4.6.4 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 4-2 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 returns the previous mode visited by the user.

Example 4-2 Working with the PortletRequest

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

4.6.5 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.

- **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 4.6.18, "PortletURI" on page 133.

Archive

Example 4-3 Working with the PortletResponse

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

4.6.6 PortletSession object

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*.

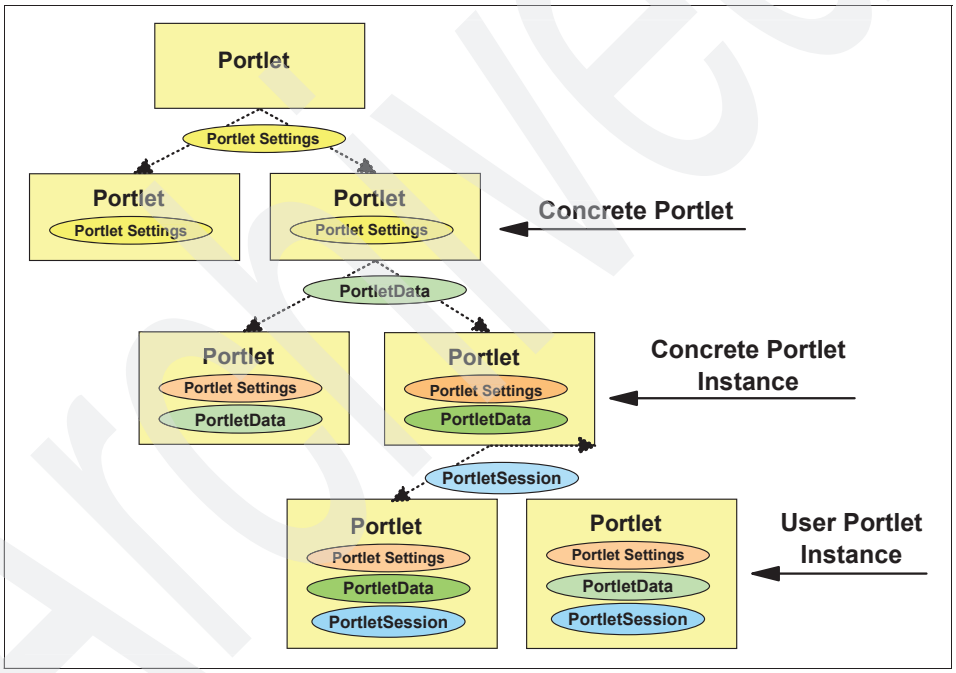


Figure 4-3 The portlet parameterization

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 4-4 on page 125.

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 4-4 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 should be serializable. To be serializable, the class needs to implement the Serializable interface. Object being placed in the session should be serializable in case the session needs to sent to another JVM in the case of failover or load balancing. This will prevent a NotSerializableException.

4.6.7 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 4-5. Figure 4-4 on page 126 illustrates the result of most of the methods of the client object when requested via Internet Explorer and a Nokia WAP emulator.

Example 4-5 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.

Archive

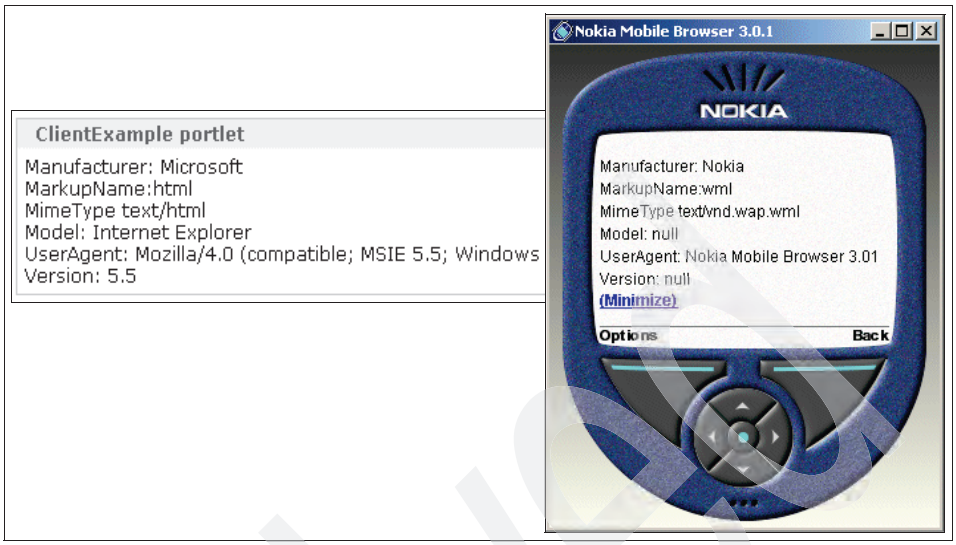


Figure 4-4 Client Information displayed on various clients

4.6.8 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 4-6 on page 127.

- **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 about the PortletContexr, refer to 4.6.9, "PortletContext object" on page 127.

Example 4-6 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();
```

4.6.9 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 4-7.

Example 4-7 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 4-8 on page 128 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 about NLS, see Chapter 12, "Internationalization" on page 373.

Example 4-8 Including a JSP

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

4.6.10 PortletSettings object

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.*

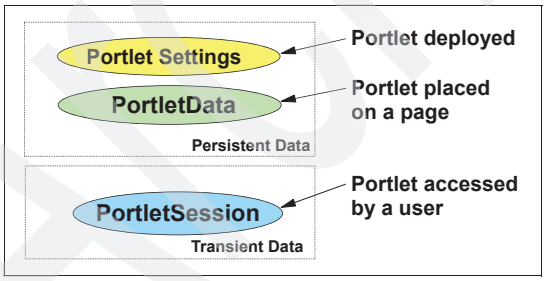


Figure 4-5 Portlet parameterization objects

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

Archived

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 4.6.11, “PortletApplicationSettings object” on page 129.

Example 4-9 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();
```

4.6.11 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 4-10 Working with PortletApplicationSettings

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

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

4.6.12 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.

Example 4-11 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);
```

4.6.13 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 4-12 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 4-12 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 4-13 on page 132.

Example 4-13 Log object

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

4.6.14 PortletException

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

4.6.15 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 not 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.

4.6.16 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 4-14 on page 133 illustrates this approach. Minimized, Normal and Maximized are defined as constants in the PortletWindow.State class.

Archived

Example 4-14 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.
}
```

4.6.17 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 4-15 illustrates accessing both basic and extended attributes.

Example 4-15 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

4.6.18 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.

4.7 Portlet life cycle

This section will explain the portlet life cycle and when certain objects become available. The basic life cycle of each portlet is displayed in Figure 4-6. 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 4.8, “Listeners” on page 137.

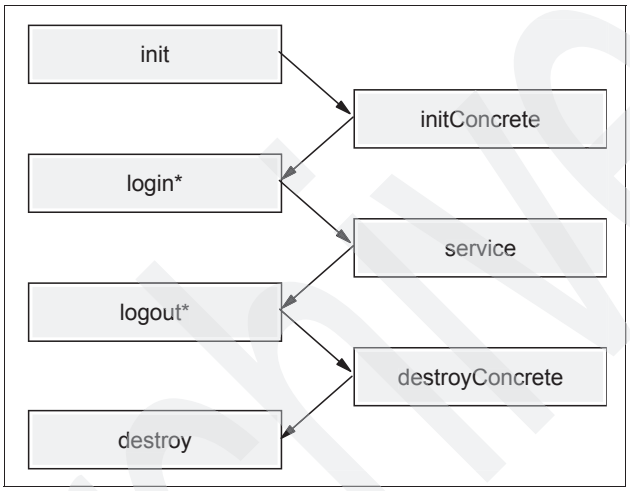


Figure 4-6 Basic portlet life cycle

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 4-7 on page 135.

- init()
- initConcrete()

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

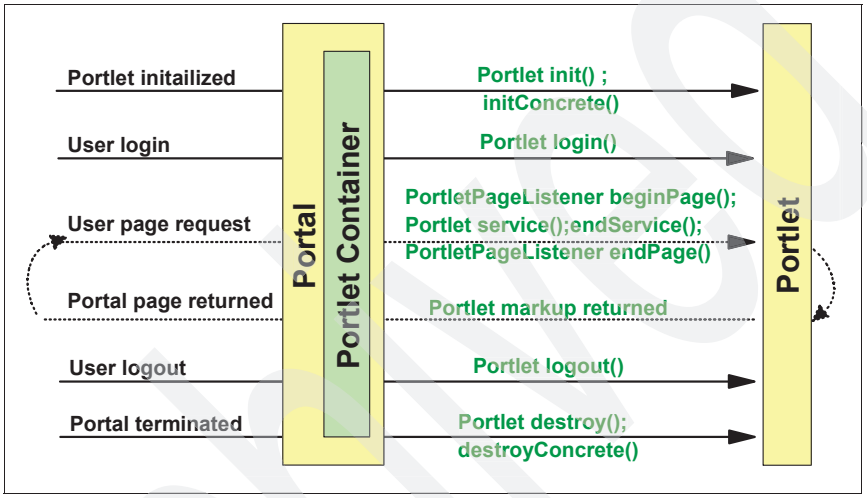


Figure 4-7 Portlet life cycle

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.

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 doXXX 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.

4.8 Listeners

The event model of the IBM 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 unsolicited and unexpected events.

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

4.8.1 PortletTitleListener

This listener allows you to dynamically set the title of the portlet. This listener requires the single method as shown in Example 4-16. 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 4-16 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() + ")");
   }
```

4.8.2 PortletPageListener

This interface provides the opportunity to add content to the top and bottom of the aggregated page. Example 4-17 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 4-8 on page 139.

Example 4-17 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 4-8 on page 139.

Archive

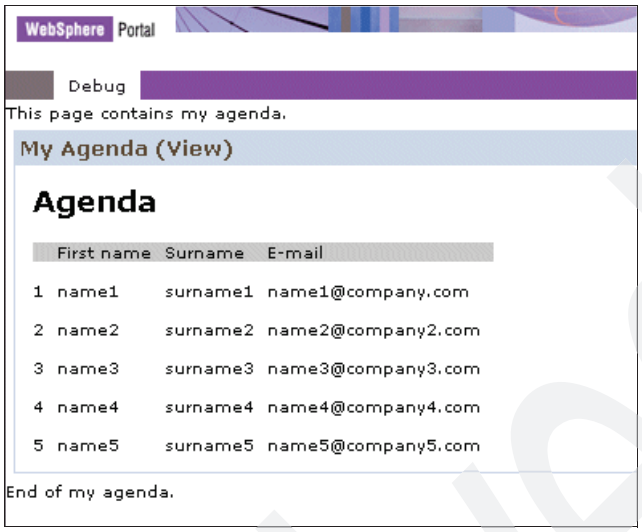


Figure 4-8 beginPage and endPage placements

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.

4.8.3 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 4-18 on page 140. Figure on page 134 illustrates where in the life cycle of the portlet these methods are called. The functionality of the login and logout methods is detailed in 4.7, “Portlet life cycle” on page 134.

Archived

Example 4-18 PortletSessionListener methods

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

4.8.4 WindowListener

The WindowListener is no longer supported in WebSphere Portal as of version 5.1.

Note: The WindowEvent interface has also been removed in WebSphere Portal V5.1; 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.

Abstract

Example 4-19 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 { }
```

4.8.5 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.

4.8.6 PortletApplicationSettingsAttributesListener

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

4.9 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 about sending messages, see 4.10, “Attribute storage summary” on page 145.

When the Portal server services a request, it acts in two distinct phases. The first phase is the event processing phase. All events, including 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.

4.9.1 ActionListener

The org.apache.jetspeed.portlet.event.ActionListener interface defines a single method to be implemented as illustrated in Example 4-20.

Example 4-20 ActionListener Interface

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

4.9.2 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 4-21.

Example 4-21 Working with the ActionEvent

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

4.9.3 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

Archived

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 4-22.

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.

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 4-22.

Example 4-22 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 4-23 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 and you should use the Simple Action string instead, as illustrated in Example 4-23.

Example 4-23 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);
}
```

4.9.4 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

4.10 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 that you 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

4.11 Portlet JSPs

When designing your portlet applications, you will generally use the MVC Model 2. 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.

- 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.

4.11.1 Portlet tag library

The portlet.tld is located in the <WP-ROOT>app\wps.ear\wps.war\WEB-INF\tld directory. Example 4-24 illustrates referencing the tag library at the beginning of a JSP.

Example 4-24 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 4-25 on page 148 illustrates this approach.

Archived

Example 4-25 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 4-26 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 4-25 illustrates the use of this tag. This tag requires that a name attribute be specified.

- `URIParameter <portletAPI:URIParamter name=“string” value=“string”/>`

  This tag is only used when creating a PortletURI object. Example 4-26 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 4-27 Retrieving a single PortletData attribute

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

Archived

- 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 4-28 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 4-28 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 4-29 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 4-29 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 4-30 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 4-31 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 4-32 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 4-32 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 4-33 illustrates the usage and result of this tag.

Example 4-33 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 4-34 on page 151 would successfully locate the image and create a fully qualified URL.

Archive

Example 4-34 Creating a fully 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

Example 4-35 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>
```

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 4-36 illustrates this.

Example 4-36 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 4-37. All conditions must evaluate to true for the if tag to return true.

Example 4-37 Evaluating multiple attributes

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

Archived

```
</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 about writing to the log, see 4.6.13, “PortletLog object” on page 131. Example 4-38 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 4-38 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).

  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`.

4.11.2 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.

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. When aggregation is complete, the page is returned to the user.

Archived

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.

4.12 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.

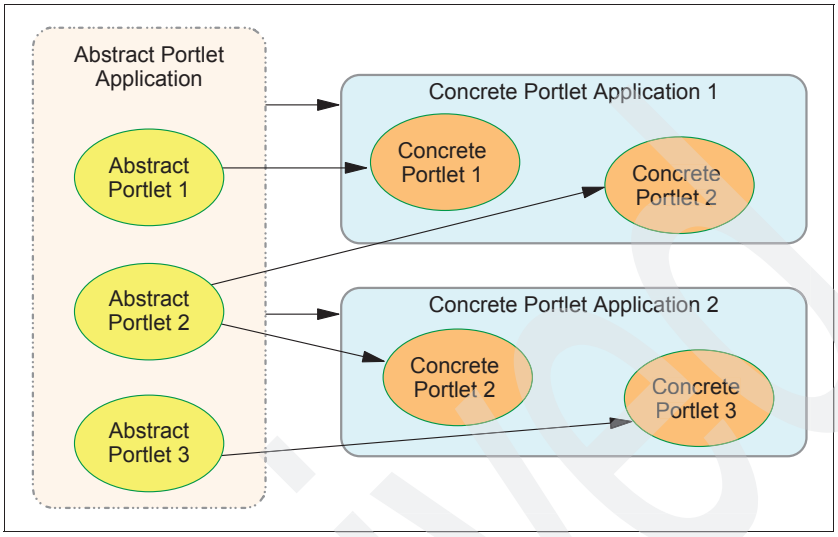


Figure 4-9 Abstract and Concrete relationship in a single portlet.xml

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 4-10 on page 157.

Archive

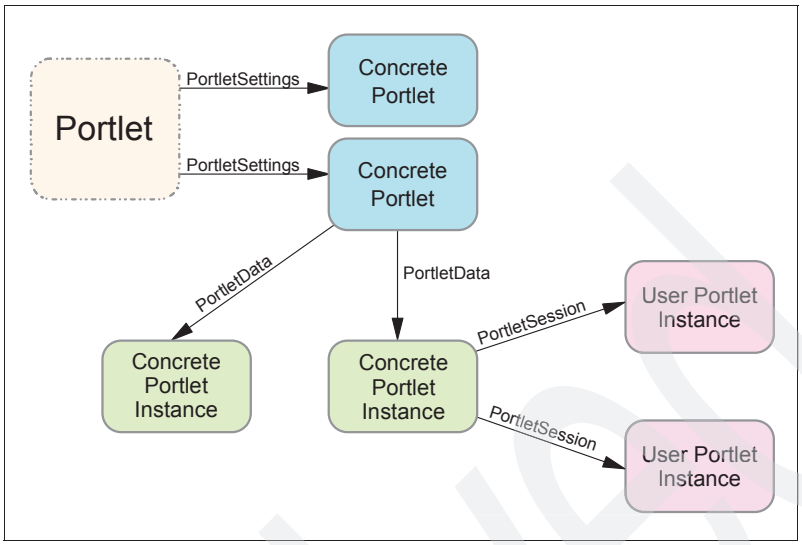


Figure 4-10 Portlet parameterization

4.12.1 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 4-39 provides an example web.xml document. To keep this section simple, the deployment descriptors will define only a single portlet in the application.

Example 4-39 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 about parameters see "Parameter summary" on page 168.

- **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 168.

- **param-value** *required*

  The String value held by the parameter.

Archived

- **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 168.

- **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.

- **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.

Archived

4.12.2 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 4-40 displays a simple portlet.xml.

Example 4-40 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>
            <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>
```

Archived

```
            <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.

  - **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 171. 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.

Archived

- **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.

  - **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 4-41 where `Servlet_1` is the ID defined in the `<servlet id>` tag of the web.xml.

Example 4-41 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.

- **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:

Archived

- **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.

  - **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

Archived

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 4.12.5, “UID guidelines” on page 171 for more information about 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 4.12.3, “Parameter summary” on page 168.

- **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 4-42.

Example 4-42 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.

Archive

- **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 168.

- **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.

4.12.3 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

Parameter Name	Location	Visibility	Programmatic Access	Runtime Accessibility
Config-Param	portlet-xml concrete portlet definition	Individual Concrete portlets	PortletSettings.getAttribute()	Read/Write

Archived

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

The relationship between servlets, abstract portlets and concrete portlets is best described in Figure 4-11. Note that some required elements have been omitted for clarity.

```
<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 4-11 web.xml and portlet.xml relationship

Archived

4.12.5 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

4.13 Resources

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

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

- For help via a news group, visit `http://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

Archived

5

A first portlet application

This chapter provides a sample scenario for creating and testing the simplest example of a portlet project using the IBM Portlet API, the Hello World example. You will use the New Portlet Project wizard to create a framework for your application. You will then run your application in the test environment. After running the portlet, you will modify the source code of the portlet and verify your changes. These activities will allow you to understand the techniques used to develop portlet projects.

Note: The portlet application described in this chapter has the following characteristics:

- Portlet API: IBM Portlet API
- Application type: MVC

© Copyright IBM Corp. 2005. All rights reserved.

5.1 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 and its components can be seen in Figure 5-1.

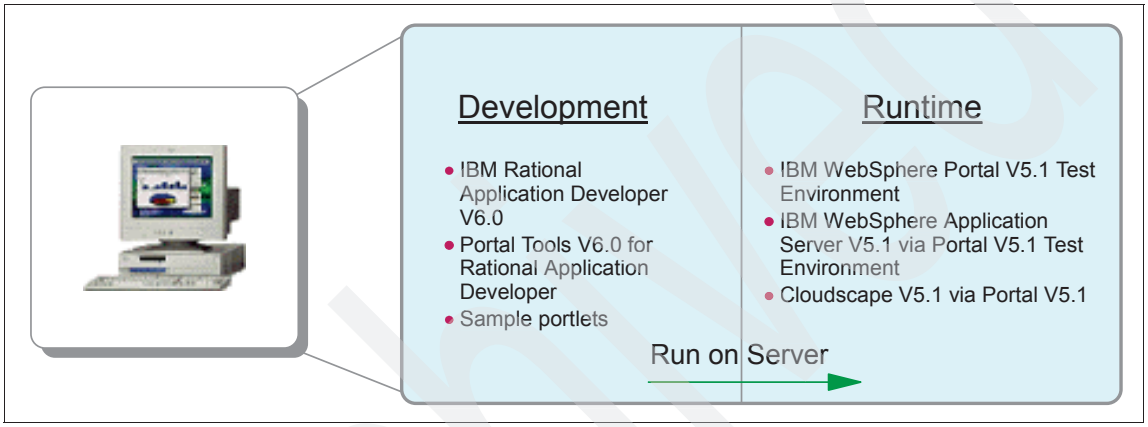


Figure 5-1 Development and runtime environments

5.2 Creating the portlet project

You will start by creating a portlet application using IBM Rational Application 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 5-2 on page 175 illustrates the MVC portlet application. The portlet application also includes a Configure.jsp, but it is not shown here.

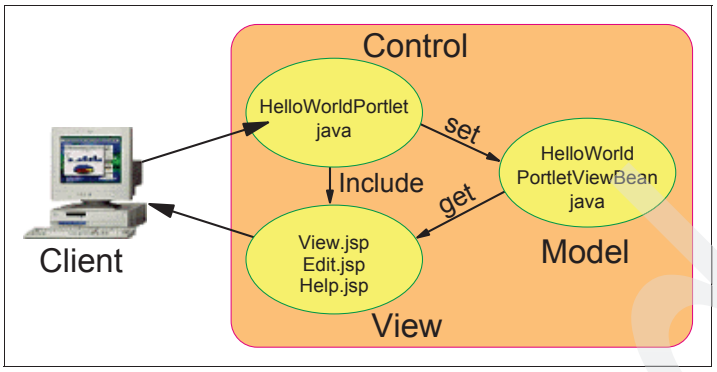


Figure 5-2 HelloWorld portlet application

You will start by creating a portlet application using IBM Rational Application 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 Portal V5.1 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.

5.2.1 Using the Portlet Project wizard

To set up the framework for your portlet project, follow these instructions:

1. Open the IBM Rational Application Developer by clicking **Start** → **Programs** → **IBM Rational** → **IBM Rational Application Developer V6.0** → **Rational Application Developer**. If you are prompted to select a workspace, click **OK** to use the default workspace directory.
2. Select **File** → **New** → **Project**.

Archived

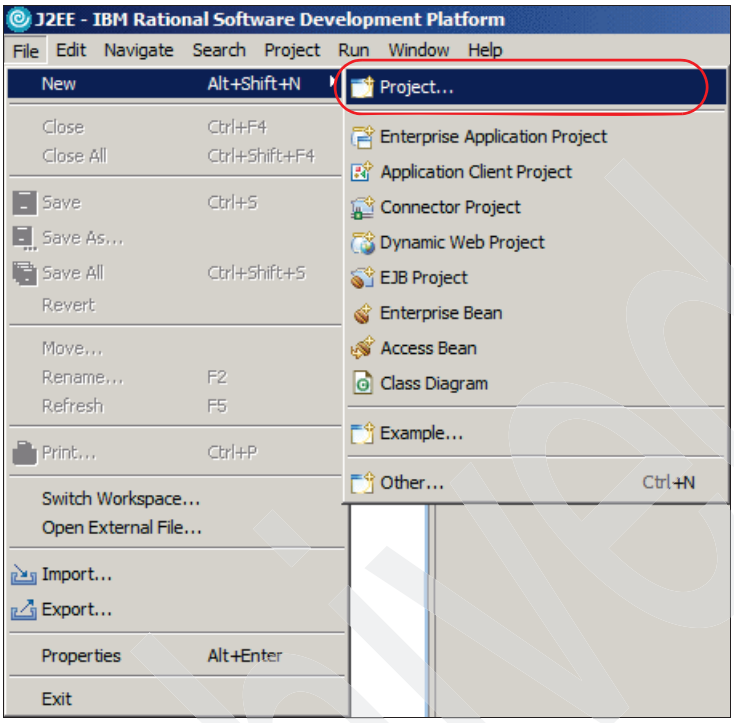


Figure 5-3 Starting a new project

3. Select **Portlet Project** from the list, and click **Next**.

Archive

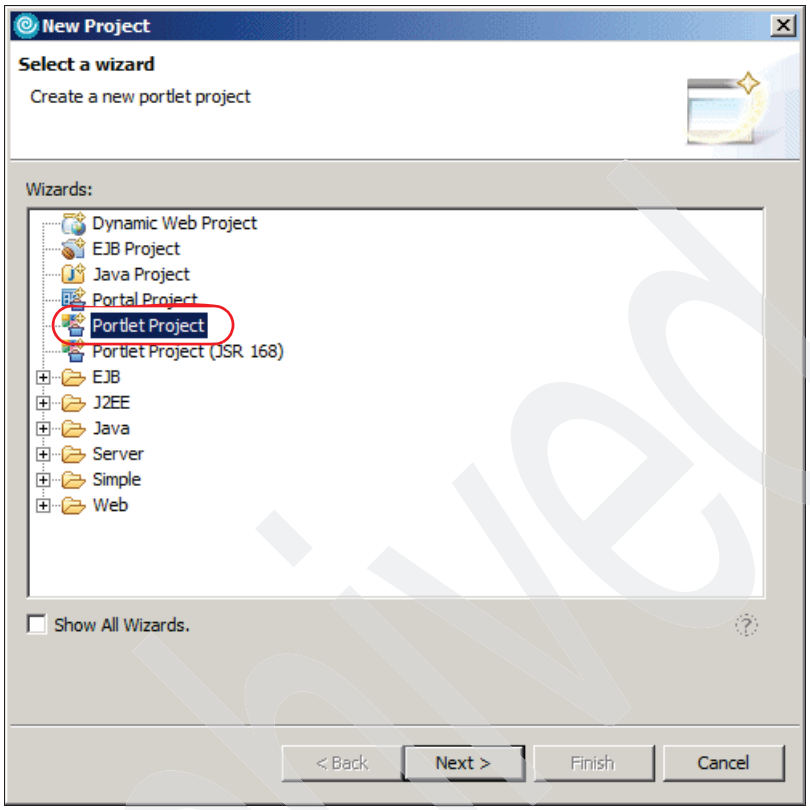


Figure 5-4 Selecting a portlet project

4. If you receive a dialog to confirm the enablement of the Portal Development Tools, click **OK**.

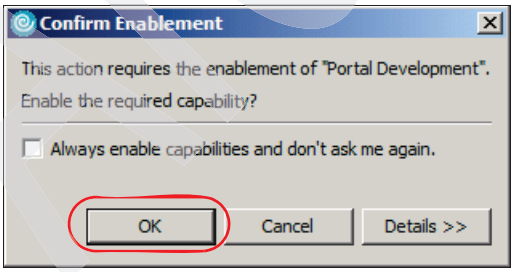


Figure 5-5 Enable portal development

Archived

5. In the Portlet Project window, name the project `HelloWorld`. Click the **Show Advanced** button, and select the **WebSphere Portal V5.1** target server. Click **Next** to continue the wizard.

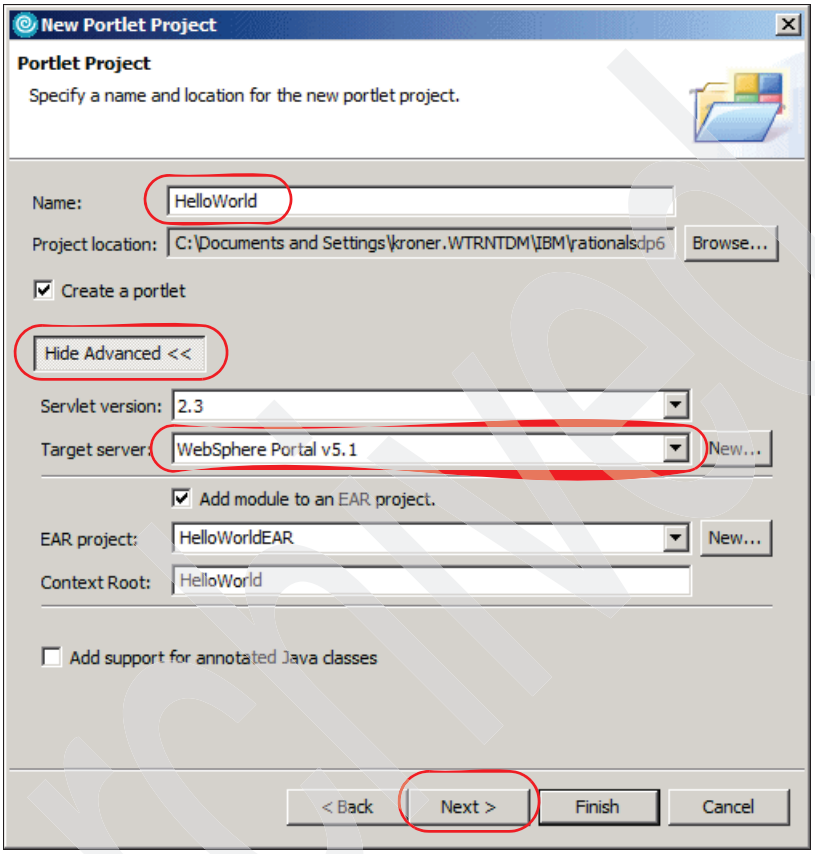


Figure 5-6 Portlet Project window

6. Select the **Basic Portlet** type, and click **Next** to continue.

Archived

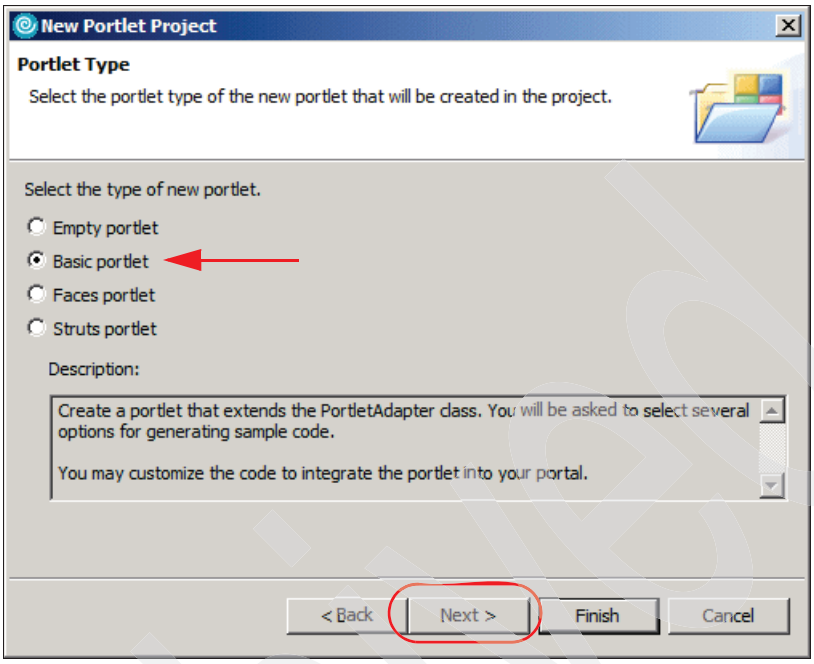


Figure 5-7 Portlet type window

7. On the Features window, deselect **Web Diagram** and select **JSP Tag Libraries** to add these to your project. Click **Next** to continue.

Archived

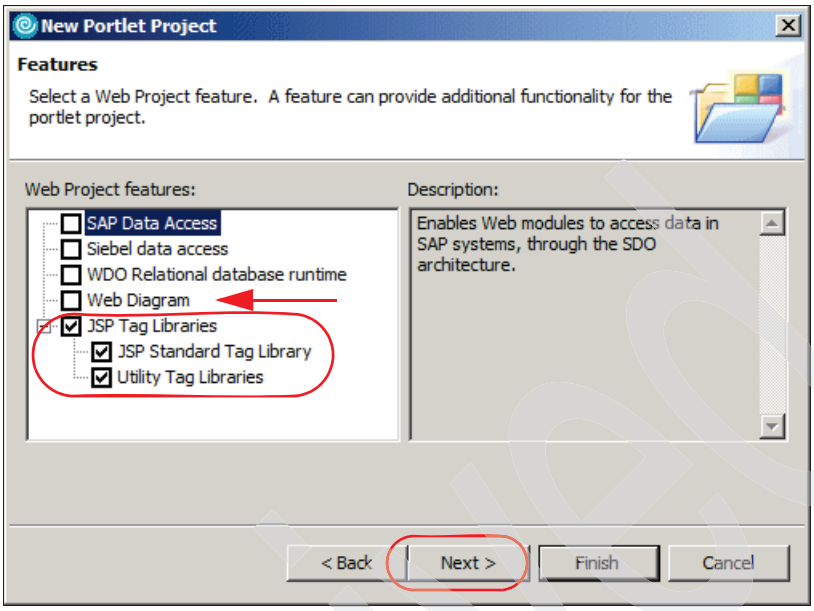


Figure 5-8 Project features window

8. Accept the default settings on the Portlet Settings window. Click **Next** to continue.

Archive

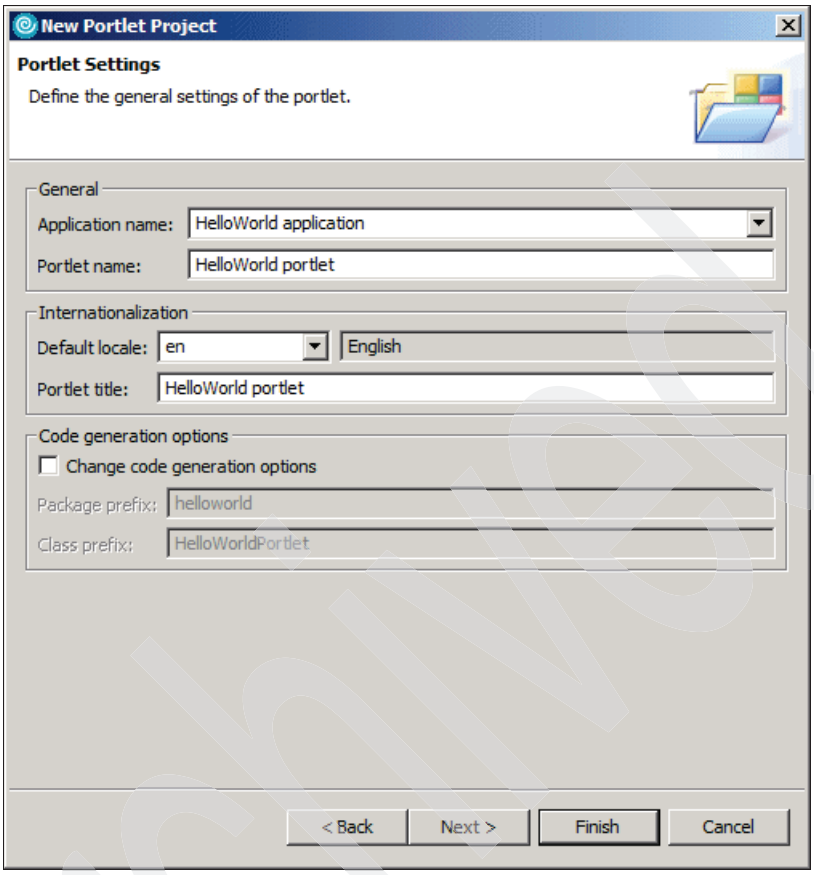


Figure 5-9 Portlet Settings window

9. Deselect **Add action listener** and **Add form sample** from the Event Handling window. Click **Next** to continue.

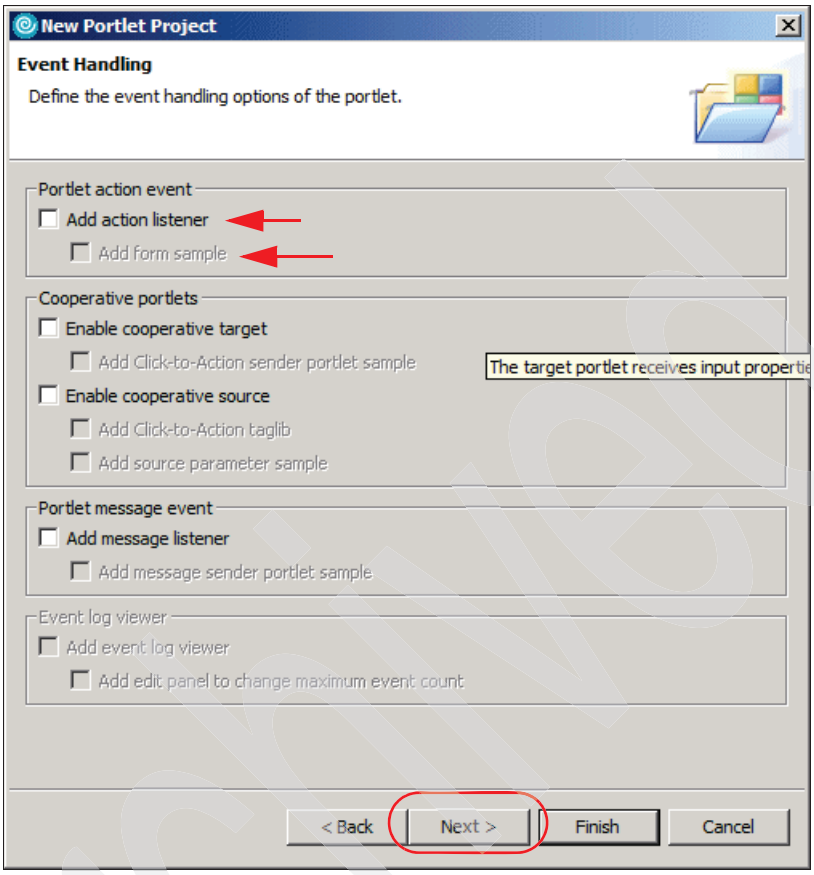


Figure 5-10 Event handling window

10. Since the portlet project does not require credential vault handling, accept the default settings on the Single Sign-On window. Click **Next** to continue.

Archive

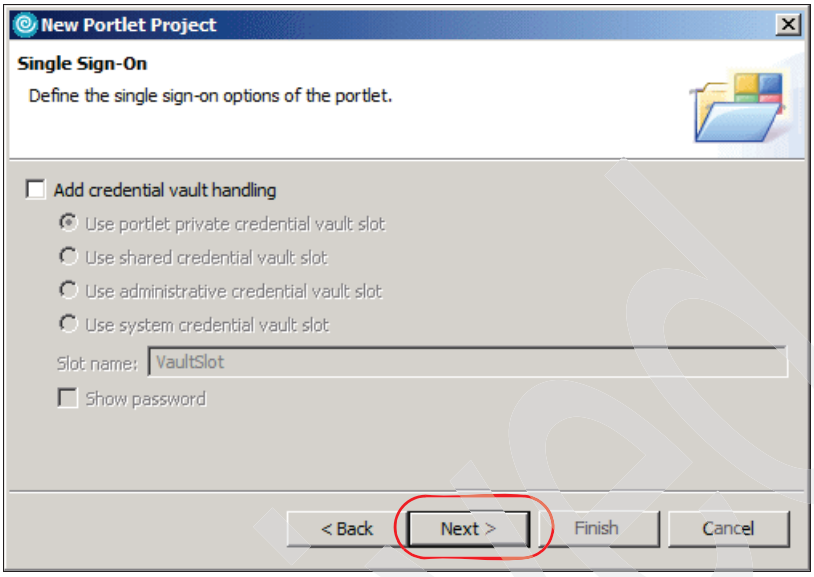


Figure 5-11 Single Sign-On window

11. On the Miscellaneous window, select **Add Edit mode**, **Add Help mode**, and **Add configure mode**. Then click **Finish** to generate your project.

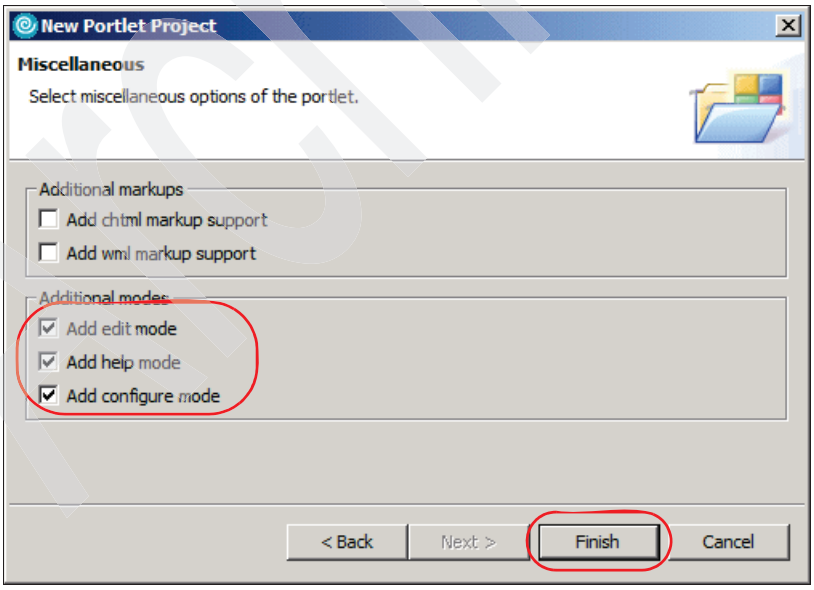


Figure 5-12 Miscellaneous window

12. If you are prompted to switch to the Web perspective to work on this project, answer **Yes**.

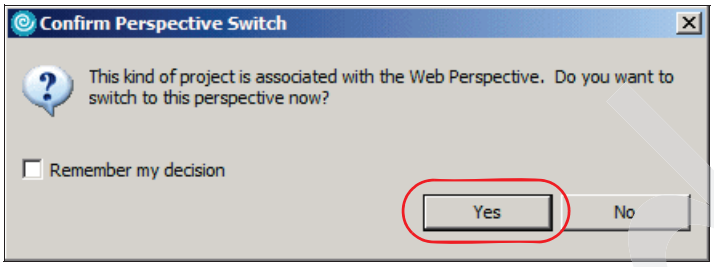


Figure 5-13 Confirm perspective switch dialog

13. The workbench will now show your newly created portlet project in the Web perspective.

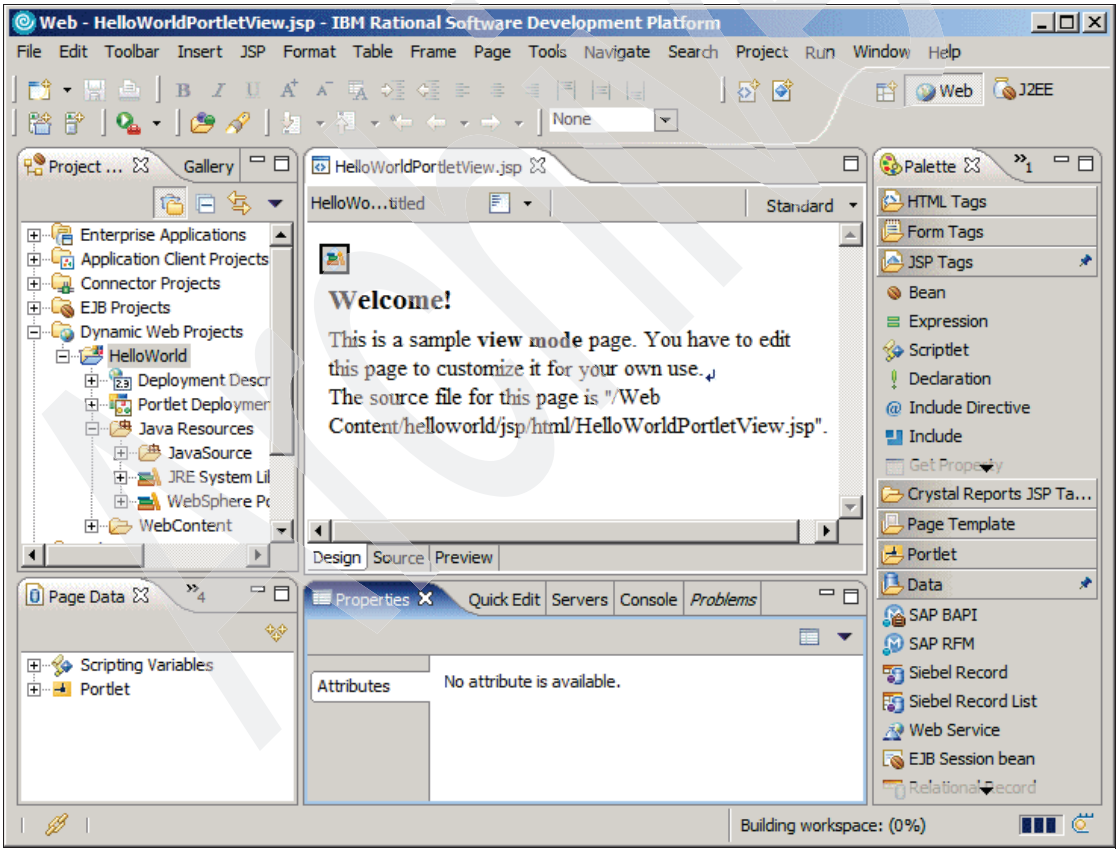


Figure 5-14 Workbench overview

14. The files for your project can be seen in the Project Explorer view on the left-hand side of the workbench.

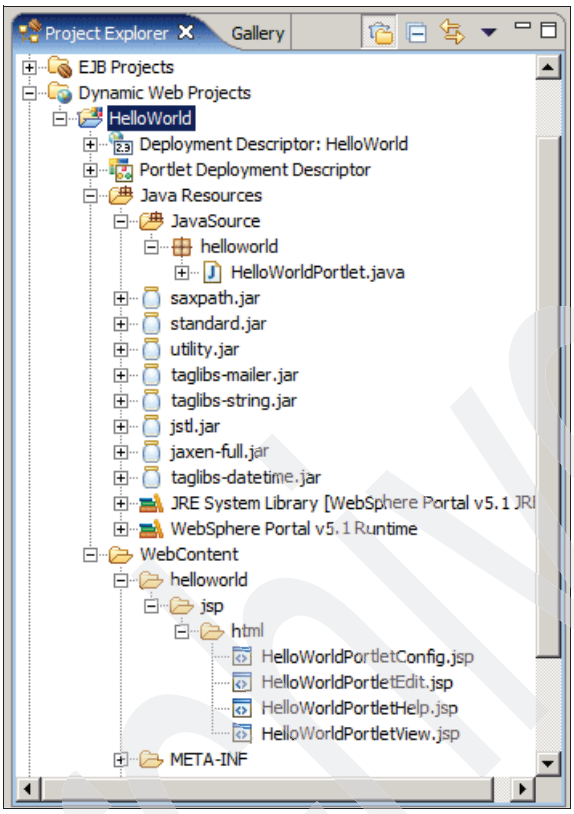


Figure 5-15 Project explorer view

5.3 Configuring the test environment

To run this project in the WebSphere Portal V5.1 Test Environment, you will need to create a new test server. To do this, follow these instructions:

1. In the Servers view at the bottom of the workbench, right-click and select **New → Server**.

Archived

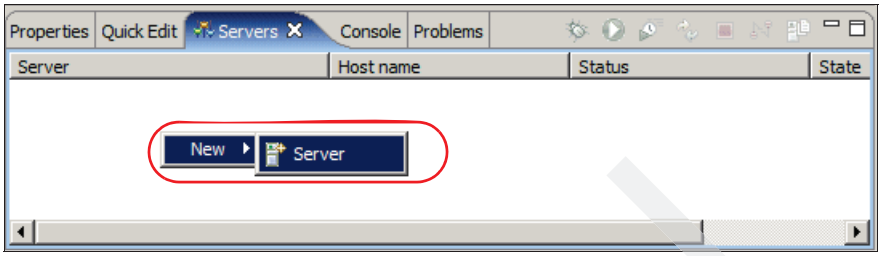


Figure 5-16 Adding a new server

2. In the Define a New Server window, choose the **WebSphere Portal V5.1 Test Environment** and click **Next**.

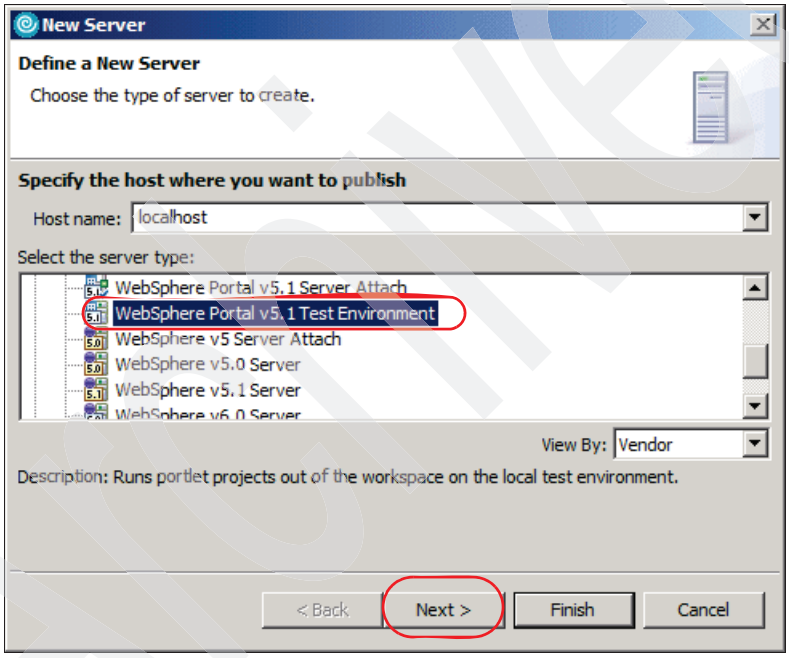


Figure 5-17 Defining a new server

3. Accept the default port of 9081 and click **Next**.

Archived

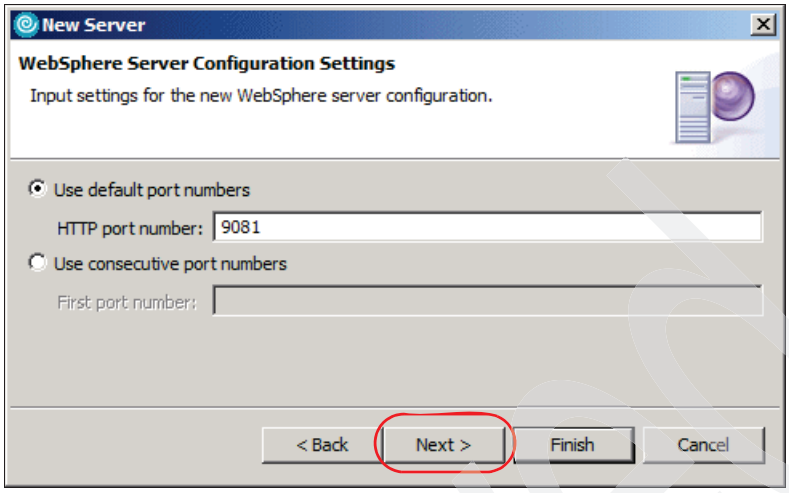


Figure 5-18 Server port selection

4. On the Portal Server Settings window, accept the defaults and click **Next**.

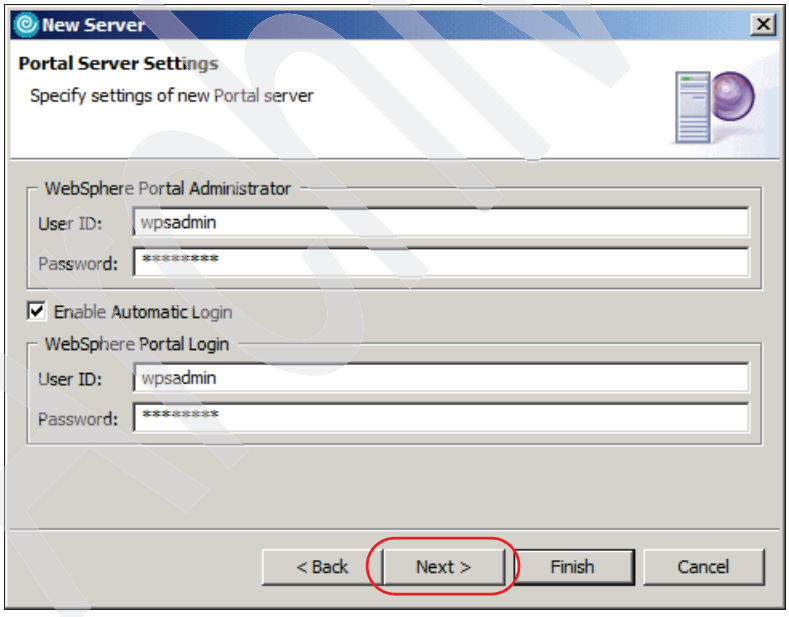


Figure 5-19 Portal Server Settings window

5. On the Projects window, add HelloWorld to this server. Then click **Finish** to configure your new test environment.

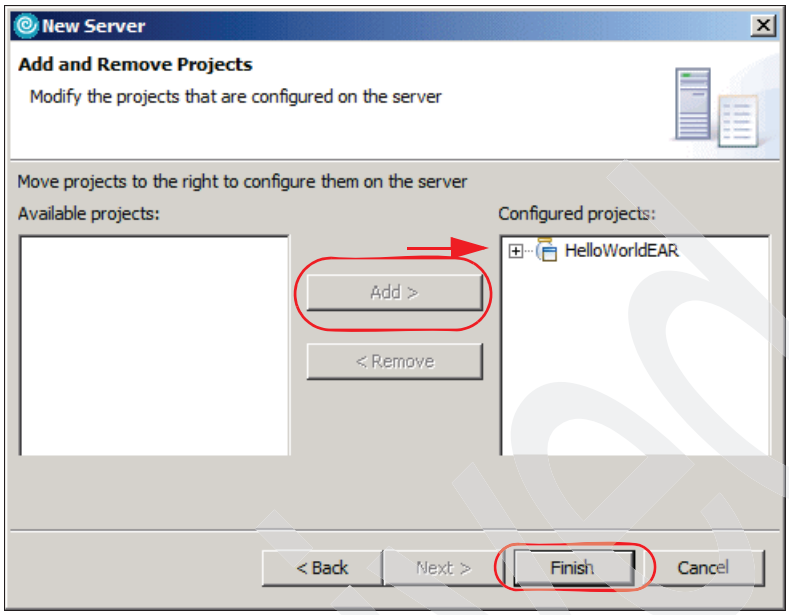


Figure 5-20 Add and Remove Projects window

5.4 Running the portlet project

To run your portal project on the local test environment server, follow these steps:

1. To run your project, right-click it in the Project Explorer view.

Archived

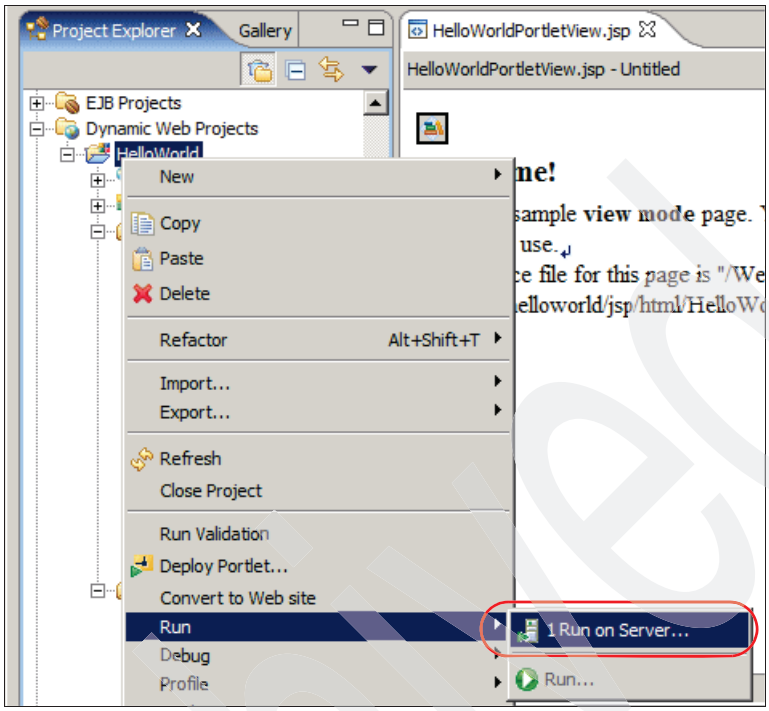


Figure 5-21 Run your project

2. Select the existing **WebSphere Portal V5.1 Test Environment** and click **Next**.

Archived

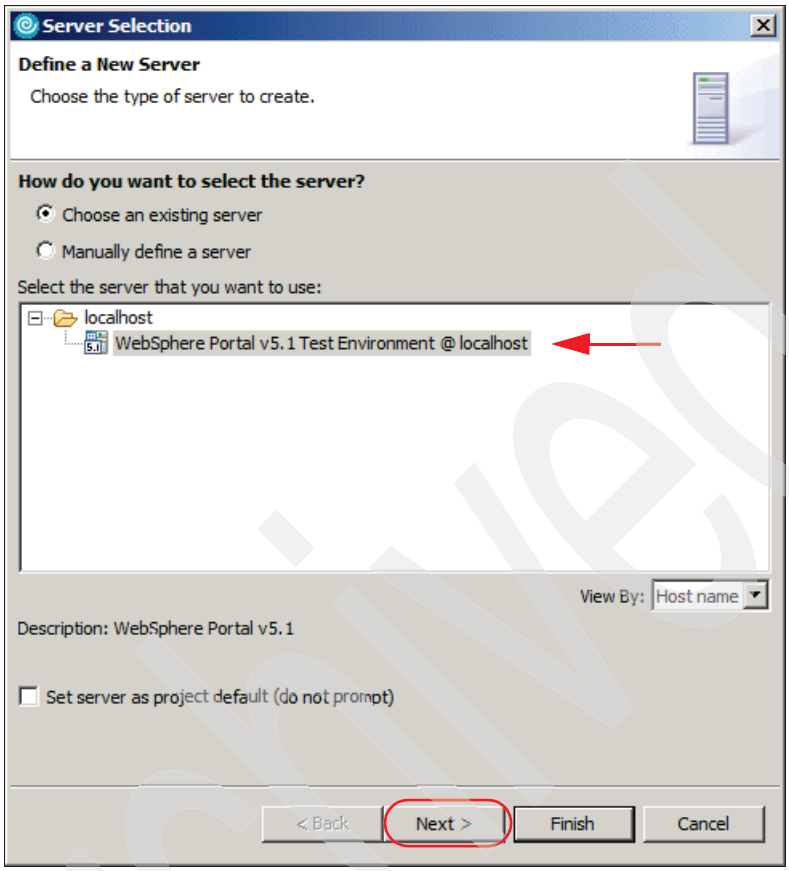


Figure 5-22 Define a New Server

3. Confirm that HelloWorld is configured to run on this server. Click **Finish** to run the project.

Archived

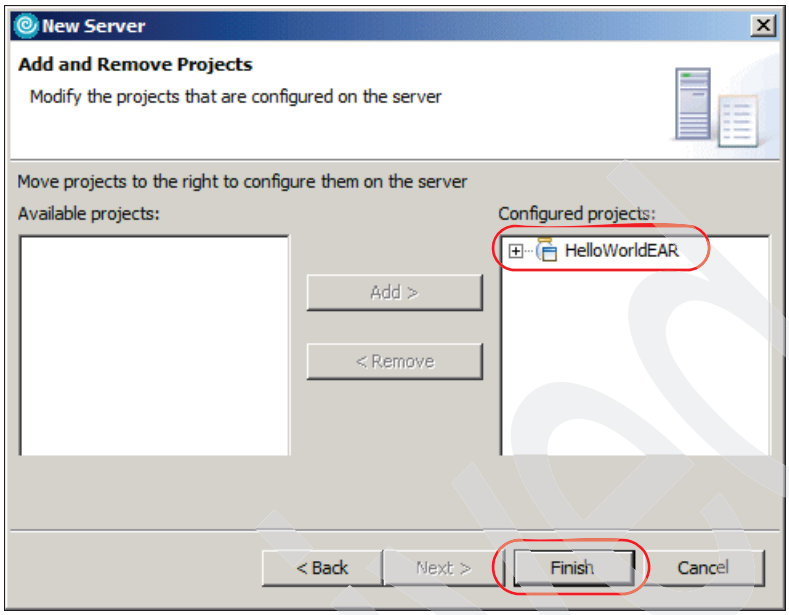


Figure 5-23 Add and remove projects window

4. Your portlet will load in the Web browser configured in the workbench. By default, it will run in View mode, click the icons in the upper right of the portlet to enter configure mode, Edit mode, and Help mode. Configure mode is only available to Portal administrators to change portlet settings.

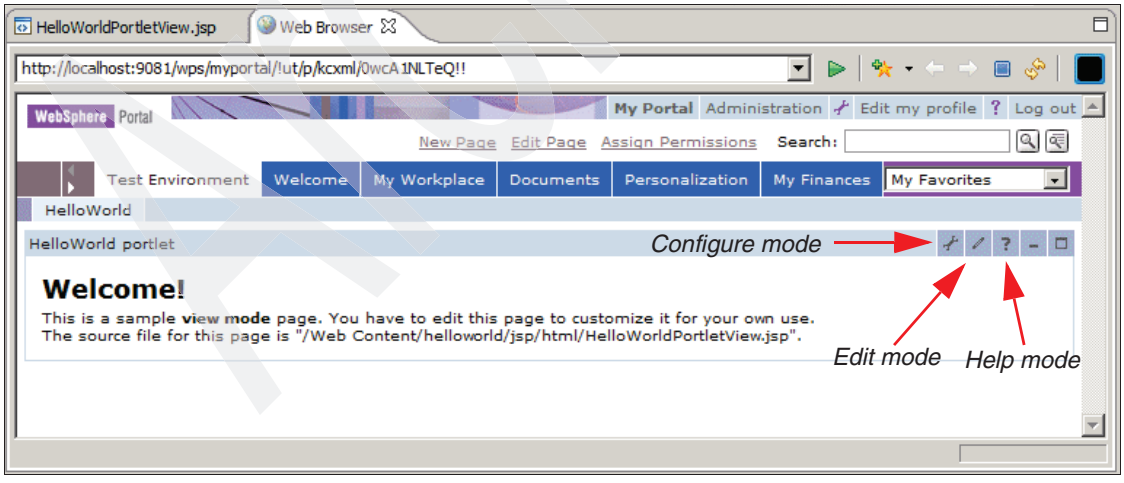


Figure 5-24 Portlet running in View mode with icons highlighted

5.5 Modifying the portlet project and verifying changes

You will now make some changes to your portlet and verify them in the test environment.

5.5.1 Changing the JSP used for the View mode

Follow these steps to see how the JSPs can be modified to show dynamic information:

1. Double-click **HelloWorldPortletView.jsp** from the /WebContent/helloworld/jsp/html/ directory in the Project Explorer view.

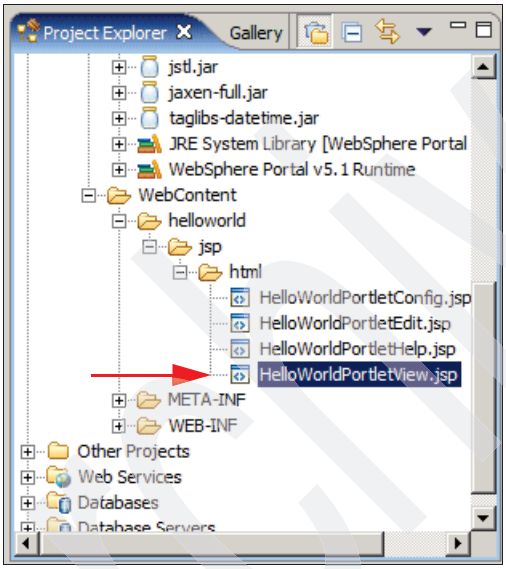


Figure 5-25 View.jsp

2. Click the **Source** tab at the bottom of the editor to see the source code.

Archived

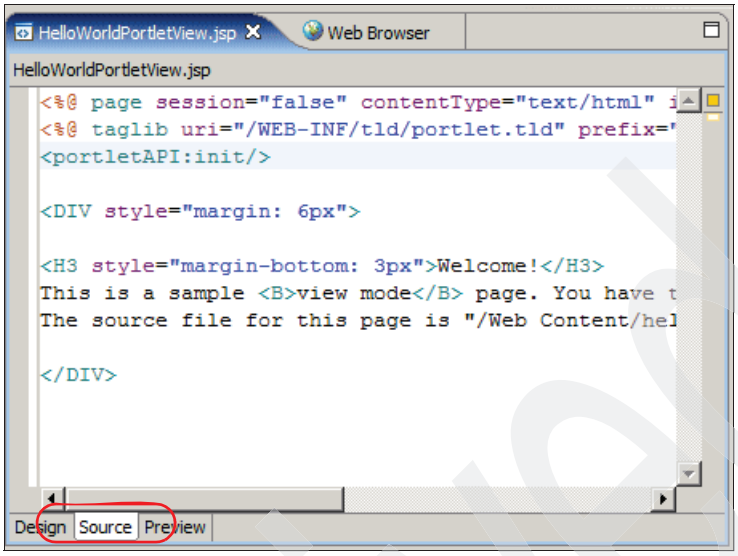


Figure 5-26 Editing JSP source

3. Add the following source code to the HelloWorldPortletView.jsp.

Example 5-1 View JSP source edit

```
<%@ page session="false" 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()%>
...
```

4. Save all of your changes by clicking **File** → **Save All**.
5. Again, right-click your project in the Project Explorer view. Then click **Run** → **Run on Server**.

Archived

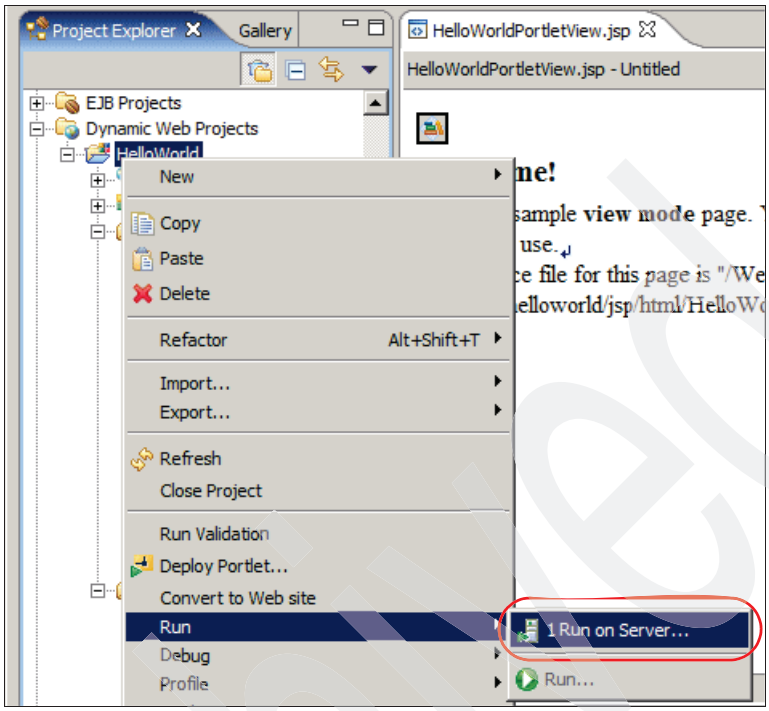


Figure 5-27 Run on server

6. Choose your existing **WebSphere Portal V5.1 Test Environment**. You can click **Finish** to run the project in the Web browser since you have already configured this server to work with this project.

Archived

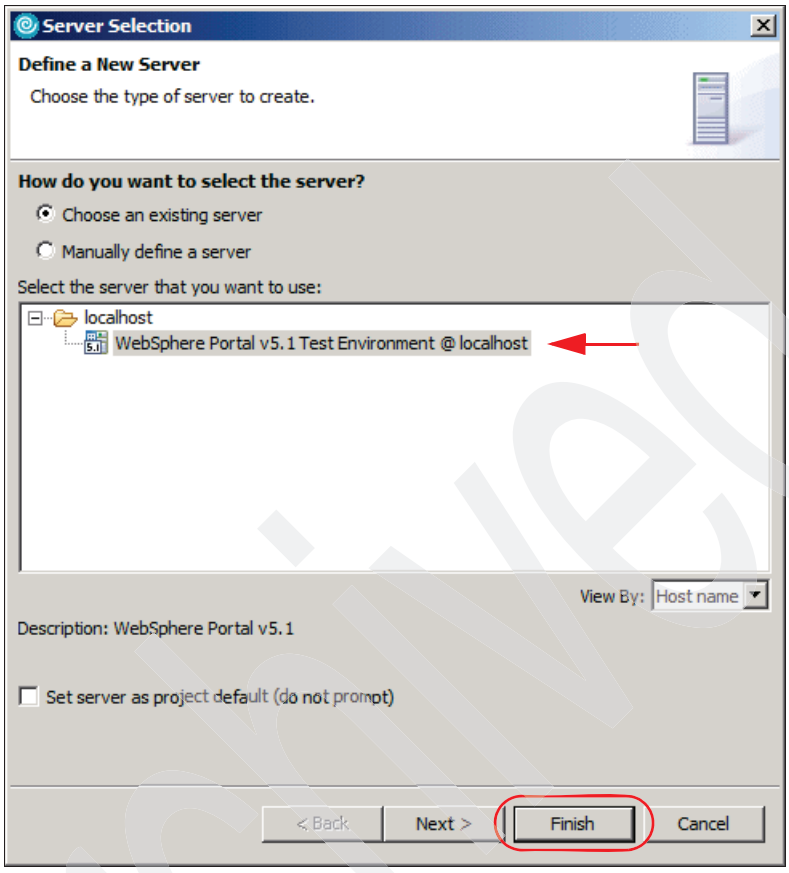


Figure 5-28 Running your project in the test environment

7. Observe your changes in the Web browser.

Archived

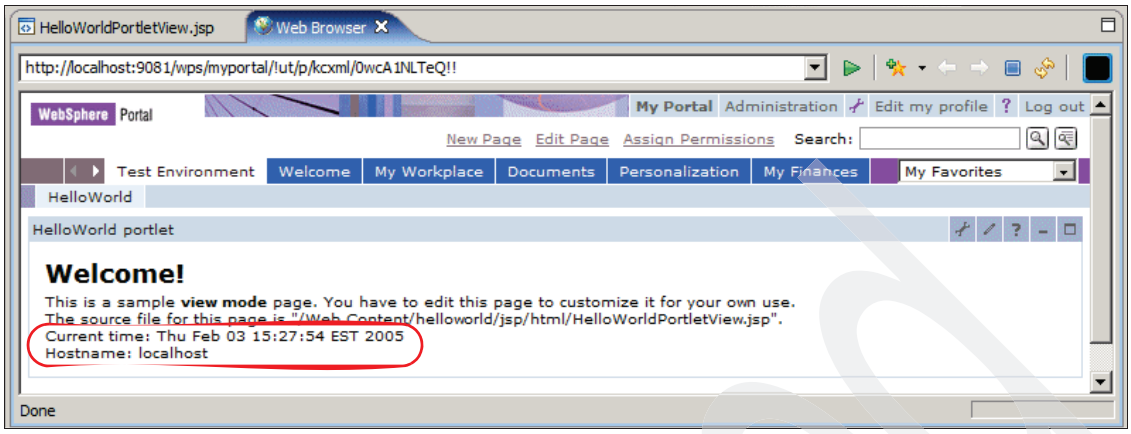


Figure 5-29 Changes in the View JSP

5.5.2 Adding a JavaBean

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 source files used to invoke the JSPs to render content are located in the /Java Resources/JavaSource/helloworld/ folder of your project as seen in the Project Explorer view.

Archived

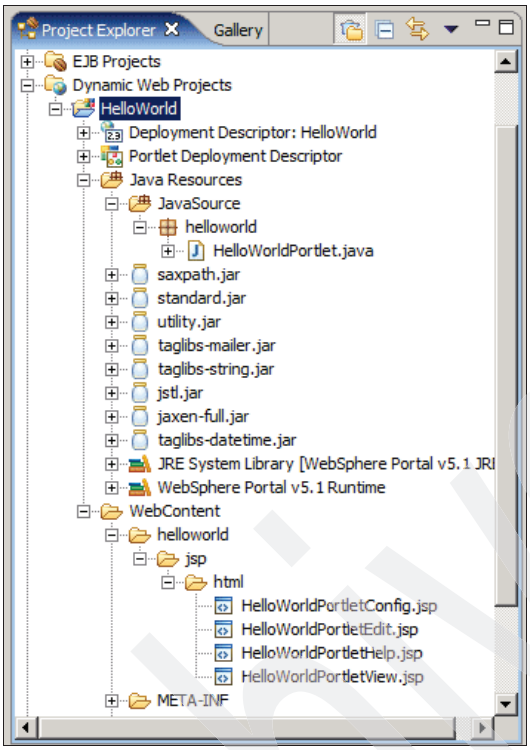


Figure 5-30 Project Explorer view

2. Right-click the helloworld package and select **New** → **Class**.

Archived

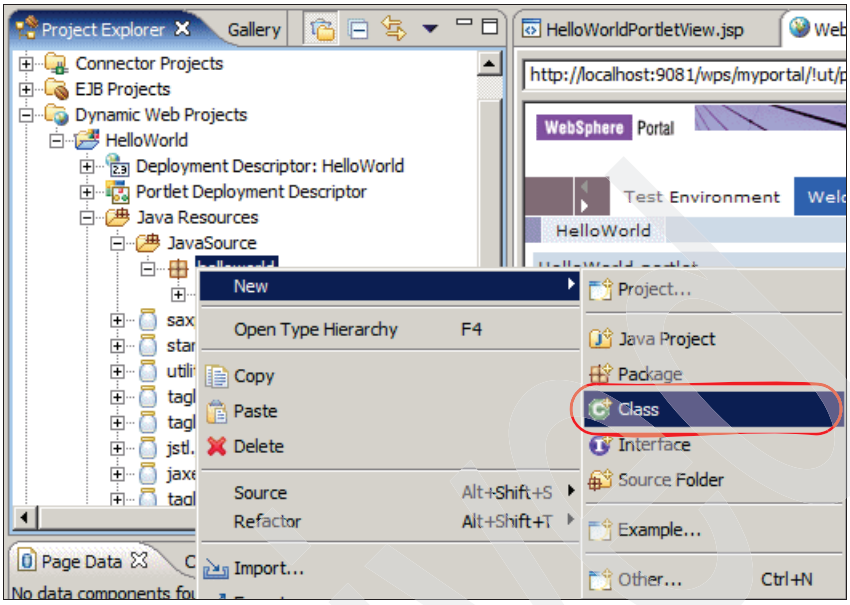


Figure 5-31 Adding a new class

3. Name the class `HelloWorldPortletViewBean`. Click **Finish** to add it to your project.

Archive

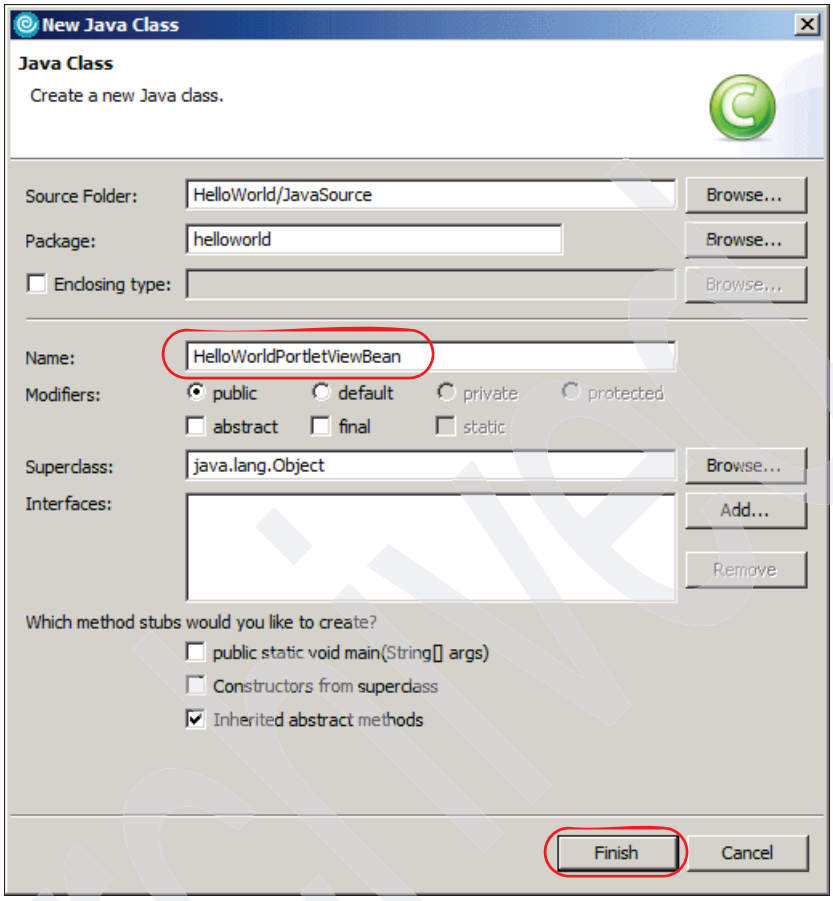


Figure 5-32 Naming the new class

4. The HelloWorldPortletViewBean.java file will now appear under the /Java Resources/JavaSource/helloworld/ folder. Double-click this file for editing. Modify the source code according to the following example. When you are finished, select **File** → **Save All** to save your changes.

Example 5-2 JavaBean code

```
public class HelloWorldPortletViewBean {

   private String myName = "";

      public String getMyName() {
      return myName;
   }
      public void setMyName(String myName) {
```

```
        this.myName = myName;
    }
}
```

5. 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.

Example 5-3 doView() method code modification

```
    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 the 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);
        }
```

6. 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 5-4 View JSP code modification

```
<jsp:useBean id="HelloWorldPortletViewBean"
class="helloworld.HelloWorldPortletViewBean" scope="request"></jsp:useBean>

<%@ page session="false" 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>
```

7. Again save your changes by clicking **File** → **Save All.**
8. Again, right-click the **HelloWorld** project, and select **Run** → **Run on Server**. Click **Finish** on the Server Selection window to run it on the existing test environment server. Your changes will be shown in the Web browser.

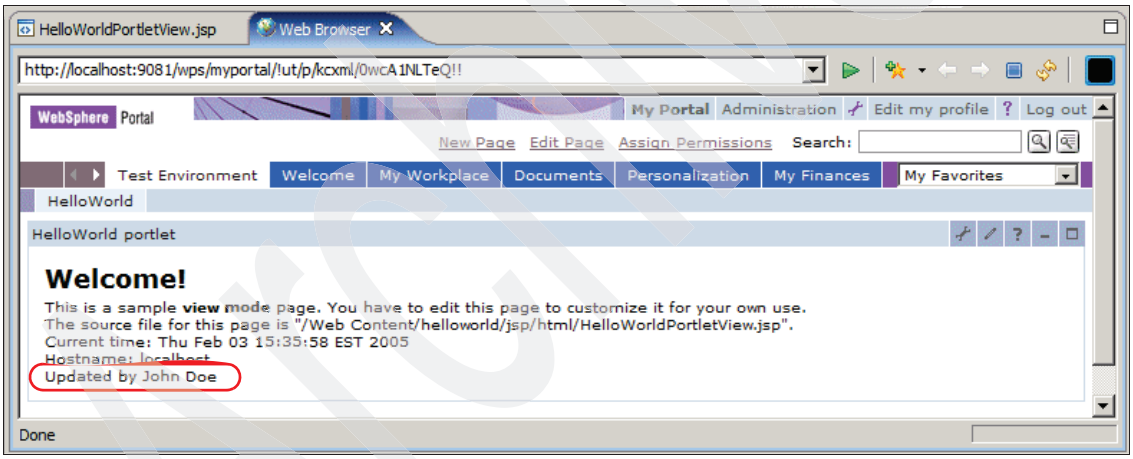


Figure 5-33 Portlet with changes

Archived

Archived

6

IBM Portlet API portlet development

This chapter introduces you to the action event handling capabilities of portlet applications. Action events occur when a user interacts with a portlet, usually by submitting a form. To receive these events, an action event listener must be implemented in the portlet class. In addition, the actionPerformed() method must be added to process the action event.

This chapter covers a scenario that will allow you to understand the techniques used to develop portlets that process action events.

In this chapter, you will find the following topics:

- How action events are processed
- Creating a basic portlet to implement an action event using IBM portlet API
- Code samples
- Running a portlet in the WebSphere Portal V5.1 Test Environment

Note: The portlet application described in this chapter has the following characteristics:

- Portlet API: IBM Portlet API
- Application type: MVC

© Copyright IBM Corp. 2005. All rights reserved.

6.1 About action events

Portlet events contain information about an event to which a portlet might need to respond. Action events are 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 or submits a form.

Portlet actions can be one of two types:

- **PortletAction objects**: The PortletAction object has been deprecated in favor of simple portlet action strings. It is maintained in the Portlet API to support existing portlets that use PortletActions. You should always use simple portlet actions instead of PortletAction objects.
- **Simple portlet action Strings**: Actions created as simple actions can be executed multiple times, enabling a user's back button to work. Links created with simple portlet actions are represented in the URL rather than in the session. Therefore, portlets with simple actions can be placed on an anonymous page where no session exists. Simple portlet actions are associated with action events by using the getActionString() method.

  Simple portlet actions are not available in the Portlet API prior to WebSphere Portal Version 4.2. 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 6-1.

Example 6-1 Check PortletAPI version

```
if ( (getPortletConfig().getContext().getMajorVersion() <= 1) &&
       (getPortletConfig().getContext().getMinorVersion() <= 1) ) {
   // simple actions not supported
} else {
   // simmple actions supported
}
```

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. The ActionListener interface provides the actionPerformed() method, to which an ActionEvent is passed. This event listener is implemented directly in the portlet class. The listener can access the PortletRequest from the event and respond using the PortletRequest or PortletSession attributes. In order to notify other portlets of an event, the listening portlet may choose to send messages.

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 6-2.

Example 6-2 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.

Example 6-3 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

Archived

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 6-4.

Example 6-4 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 6-5 displays the code for adding a parameter. Be aware that parameters set via the PortletURI are not passed in the traditional HTML syntax. Example 6-5 shows how parameters are added to the URI.

Example 6-5 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.

6.2 Development 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.

The development and runtime environments are illustrated in Figure 6-1.

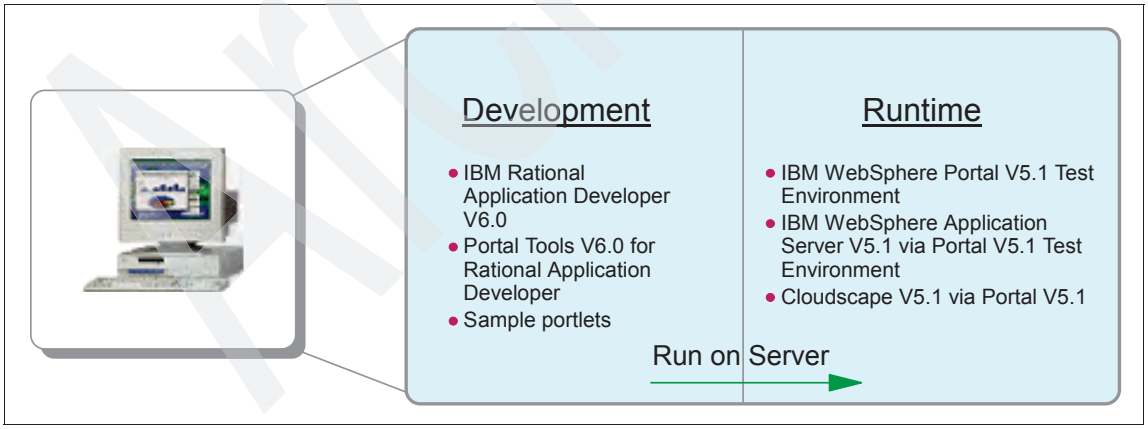


Figure 6-1 Development environment

Archive

We will start by creating a portlet application using IBM Rational Application Developer. This portlet is again based on the Model-View-Controller (MVC) design pattern.

In this scenario, you will create a portlet with support for action events using the Portlet Project wizard. To send an ActionEvent you must associate a PortletAction with the HTTP request. The PortletAction is normally linked with URLs or buttons in an HTML form to provide a way for portlet programmers to implement different processing actions for different user input or interaction.

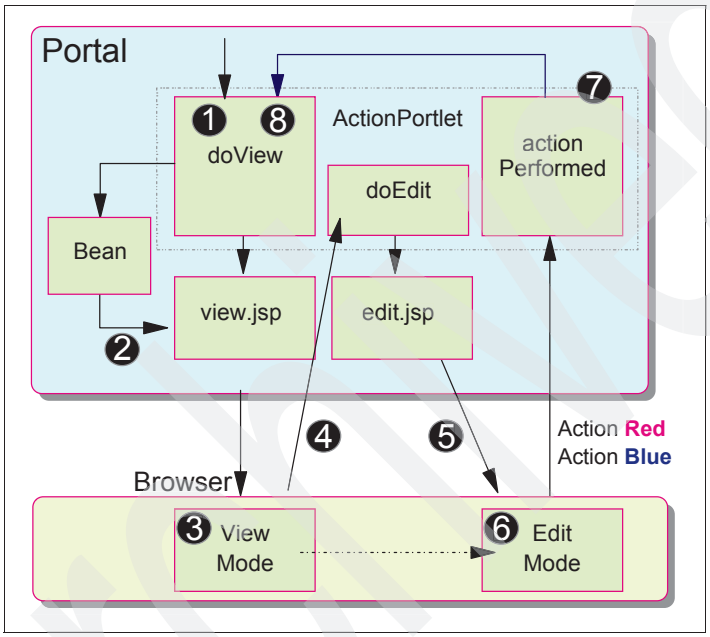


Figure 6-2 Event handling scenario

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.Starting your portlet development project

6.3 Creating the portlet project

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. Follow these steps:

1. Open the IBM Rational Application Developer by clicking **Start** → **Programs** → **IBM Rational** → **IBM Rational Application Developer V6.0** → **Rational Application Developer**. If you are prompted to select a workspace, click **OK** to use the default workspace directory.
2. If you have developed a portlet project before, you can select **File** → **New** → **Portlet Project**. Otherwise, select **New** → **Project**.

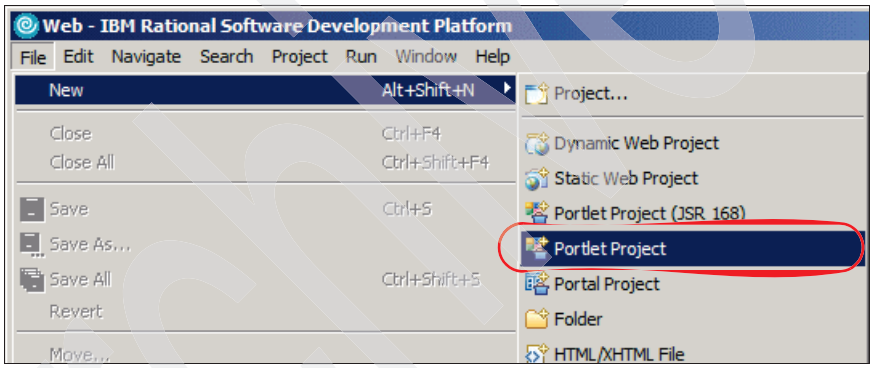


Figure 6-3 Starting a new portlet project

3. Name the project `ActionEvent`. Click the **Show Advanced** button and select WebSphere Portal V5.1 as the target server. Then click **Next**.

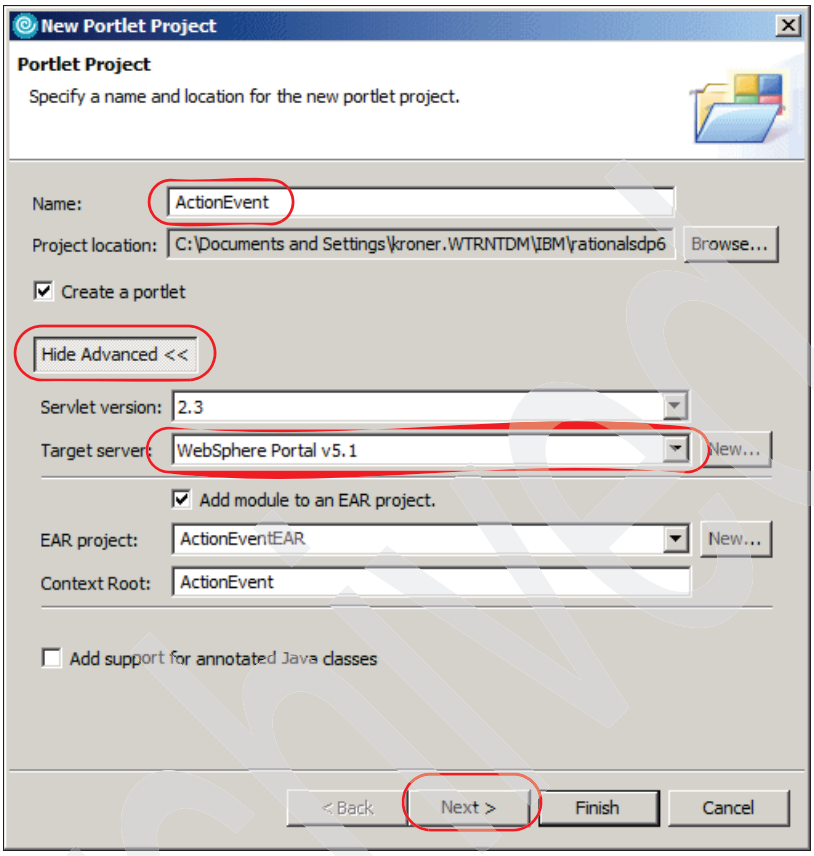


Figure 6-4 Portlet project settings window

4. Choose the **Basic Portlet** type. Click **Next** to continue.
5. On the Features window, deselect **Web Diagram** and select **JSP Tag Libraries**. Then click **Next** to continue.

Archive

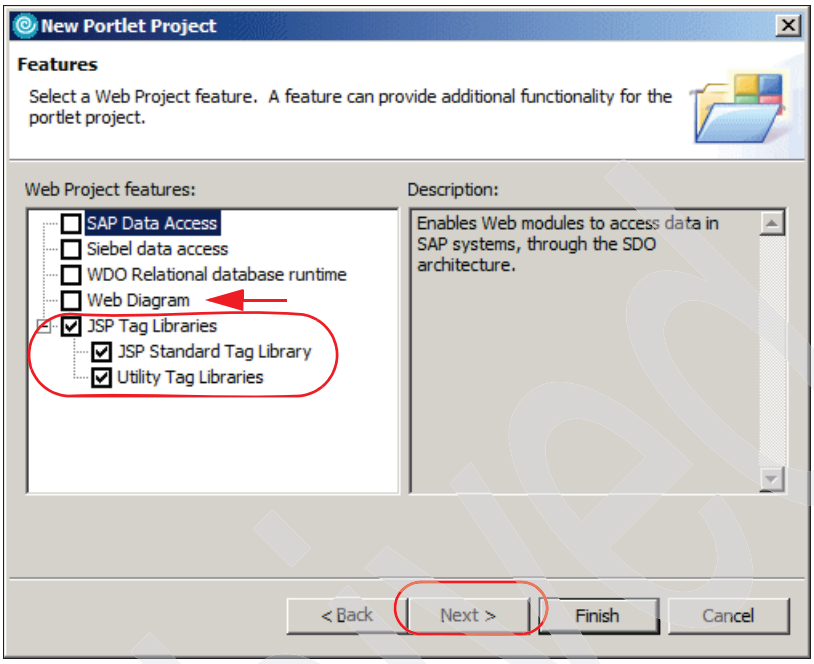


Figure 6-5 Adding JSP Tag Libraries

6. Accept the default settings on the Portlet Settings window. Click **Next** to continue.
7. On the Event Handling window, deselect **Add form sample**. Since you will be using the action event listener, leave this option selected. Click **Next** to continue.

Archive

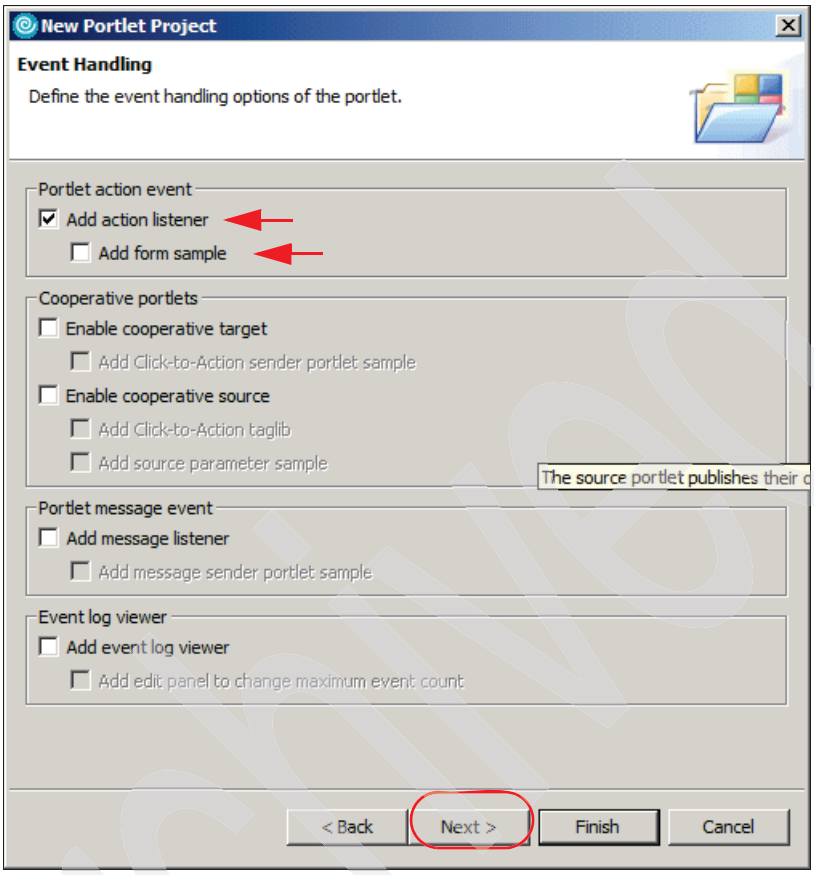


Figure 6-6 Event handling window

8. Since you will not be using the credential vault in this project, accept the defaults on the Single Sign-On window and click **Next**.
9. Click **Add Edit mode** on the Miscellaneous window. You will be using it in this scenario. Click **Finish** to generate your portlet project.

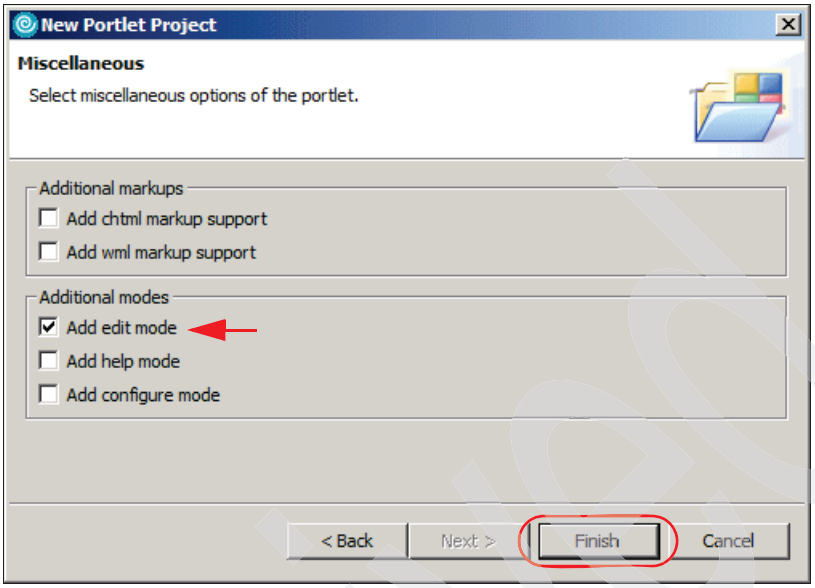


Figure 6-7 Add Edit mode

10. Your project will display in the Project Explorer view on the left-hand side of the workbench.

Archived

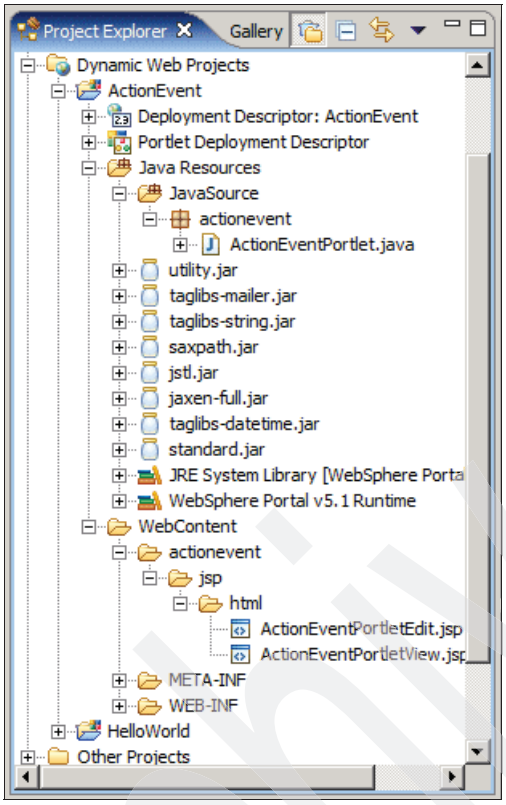


Figure 6-8 Project explorer view

6.4 Configuring your project in the test environment

To configure your project to work with an existing WebSphere Portal V5.1 Test Environment, follow these steps.

1. Right-click **WebSphere Portal V5.1 Test Environment** in the Servers view at the bottom of the workbench. Select **Add and remove projects**.

Archived

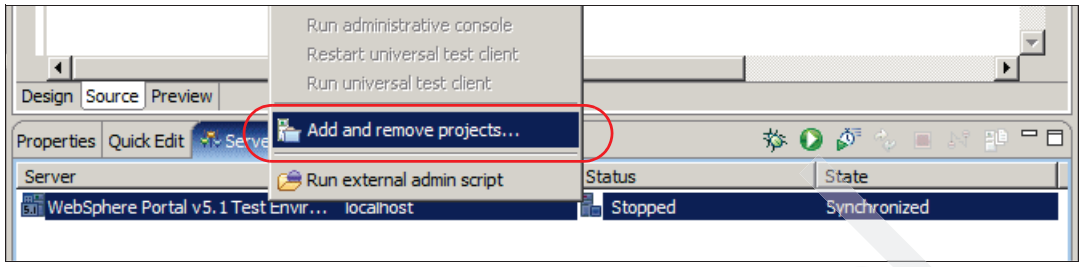


Figure 6-9 Modifying the server configuration

2. Use the Add and Remove buttons to remove HelloWorldEAR from the test environment and to add ActionEventEAR to the test environment. Click **Finish** to finish this configuration.

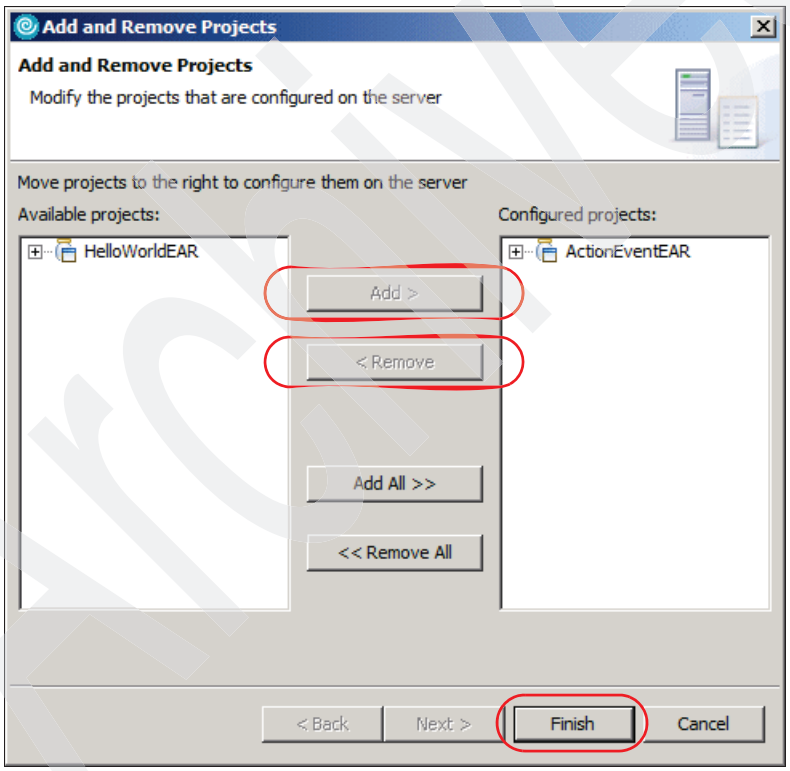


Figure 6-10 Server configuration changes

Archived

6.5 Examining and modifying the source code

You will now add a pair of buttons to the Edit.jsp so that a user can interact with the portlet to generate an action event. Follow these instructions:

1. Double-click the **ActionEventPortletEdit.jsp** located in the /WebContent/actionevent/jsp/html/ folder. It will open in its editor. 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. Modify the source code in this file so it matches the source code shown in the following example.

Example 6-6 ActionEventPortletEdit.jsp

```
<%@ page session="false" contentType="text/html"
import="org.apache.jetspeed.portlet.*,actionevent.*" %>
<%@ taglib uri="/WEB-INF/tld/portlet.tld" prefix="portletAPI" %>
<portletAPI:init/>

<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>
```

Archived

2. After making these updates, click **File** → **Save All** to save your changes. You may see errors listed in the Problems view. The changes you will make in other files will correct these errors.
3. Next, you will make changes to the ActionEventPortletView.jsp file. Open it by double-clicking the file which is located in the /WebContent/actionevent/jsp/html/ folder. Make the changes highlighted in the following example.

Example 6-7 ActionEventPortletView.jsp

```
<%@ page session="false" 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>
```

4. Click **File** → **Save All** to save your changes.
5. Next, you will make the changes to ActionEventPortlet.java. Open this file for editing by navigating to the /Java Resources/JavaSource/actionevent/ folder and double-clicking it.
6. 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 6-8 ActionEventPortlet.java

```
...
public class ActionEventPortlet extends PortletAdapter implements
ActionListener {
```

```
    public static final String VIEW_JSP         =
"/actionevent/jsp/ActionEventPortletView.";     // JSP file name to be rendered
on the view mode
...
    //  Add strings corresponding to the actions
    public static final String ACTION_RED       = "ACTION.RED";
    public static final String ACTION_BLUE      = "ACTION.BLUE";
...

   public void init(PortletConfig portletConfig) throws UnavailableException {
      super.init(portletConfig);
...

}
```

b. Next, you will edit the doView method to send content to the JSP to render.

Example 6-9 ActionEventPortlet.java

```
...
public void doView(PortletRequest request, PortletResponse response) throws
PortletException, IOException {

      // 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. Remove the existing action string handler, and add the code from the following example.

Example 6-10 ActionEventPortlet.java

```
public void actionPerformed(ActionEvent event) throws PortletException {
      if( getPortletLog().isDebugEnabled() )
```

```
        getPortletLog().debug("ActionListener - actionPerformed called");
    // ActionEvent handler
    String actionString = event.getActionString();
    PortletRequest request = event.getRequest();
    // 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 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){}
    }

    if(actionString.equalsIgnoreCase(ACTION_BLUE)){

        // Create the string of HTML to be rendered
        String value = "Action <FONT color=\"#0000ff\">BLUE</FONT>";

        // 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){}
    }
}
```

7. Click **File** → **Save All** to save your changes.

6.6 Running your project in the test environment

It is now time to run your project in the test environment to see your changes. Follow these instructions:

Archived

1. Right-click the **ActionEvent** project in the Project Explorer view. Then select **Run** → **Run on Server**.

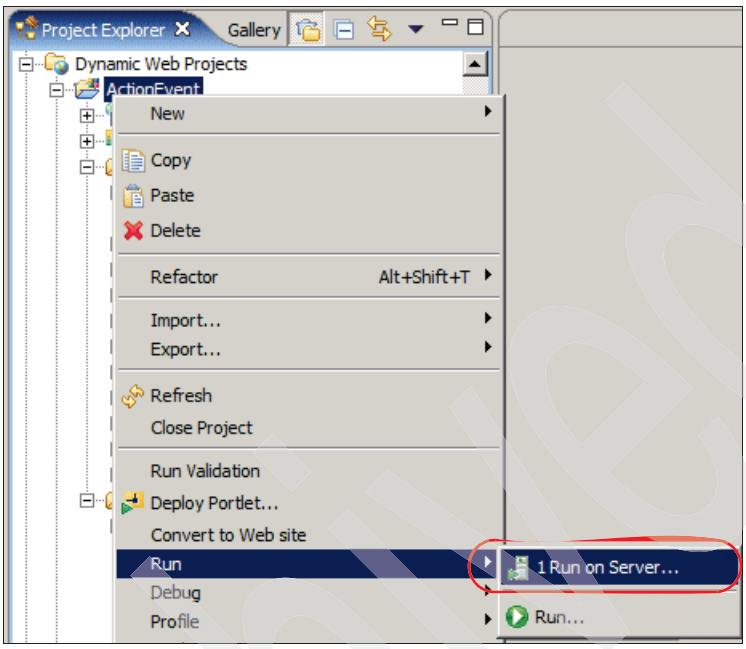


Figure 6-11 Running your project

2. Select your existing **WebSphere Portal V5.1 Test Environment** server and click **Next**.

Archived

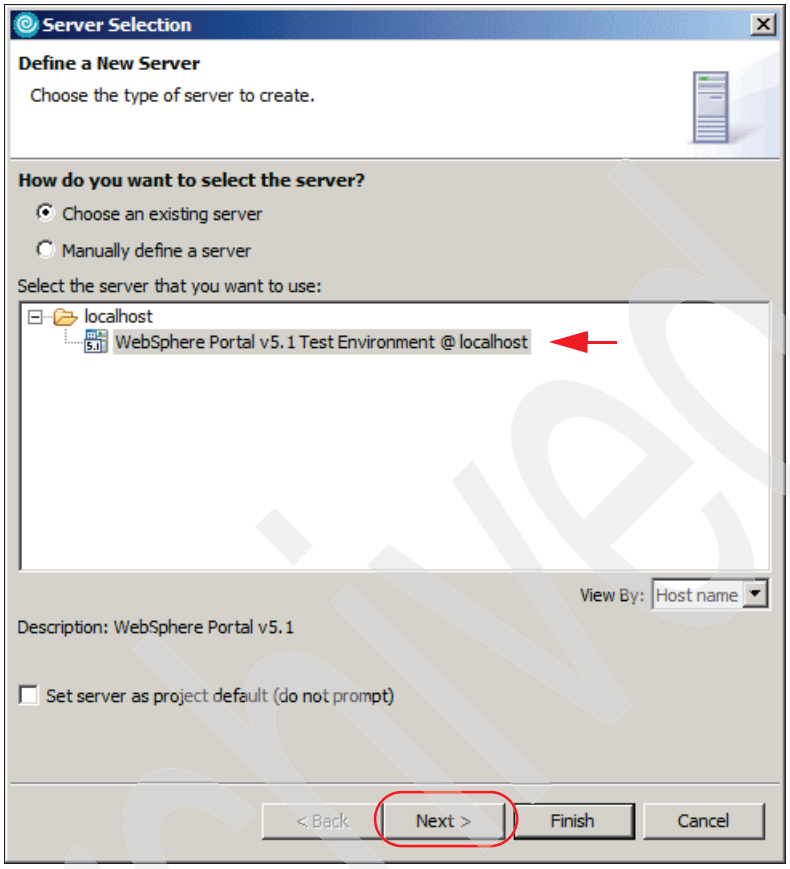


Figure 6-12 Select your server

3. Confirm that the ActionEventEAR project is the only one configured to run in the test environment. Click **Finish** to run your project.

Archived

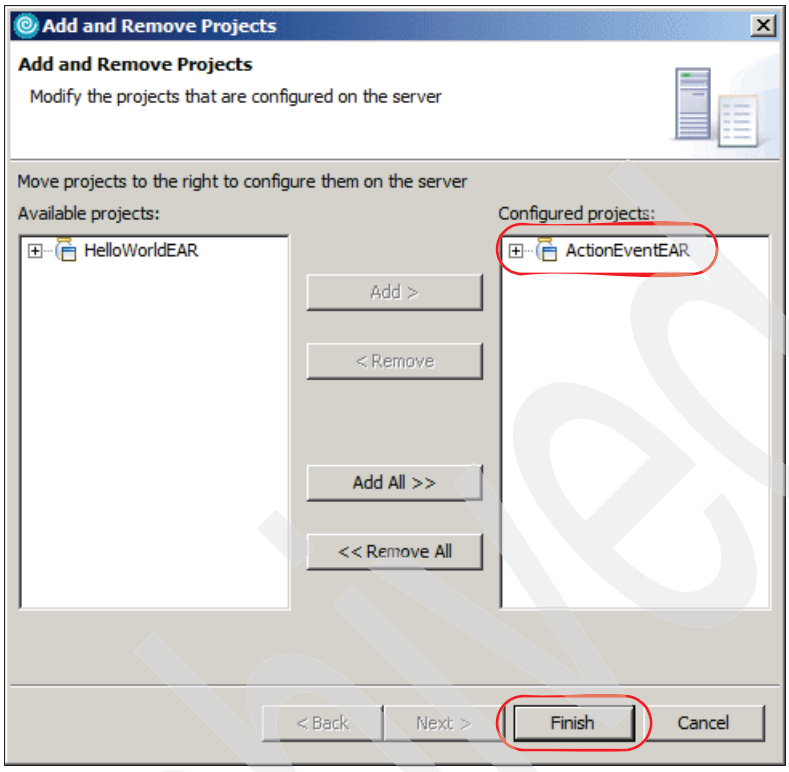


Figure 6-13 Confirm configuration

4. Your project will run in the Web browser. Since no action has been made, a message is displayed to indicate this. Enter Edit mode to choose an action by click the Edit mode icon.

Archived

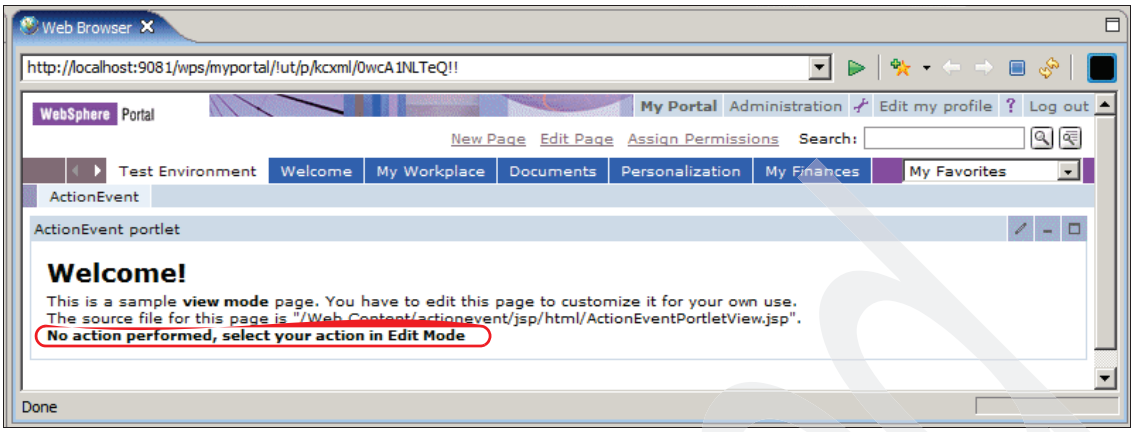


Figure 6-14 No action performed

5. Now select either the **Red Action** or **Blue Action** button.

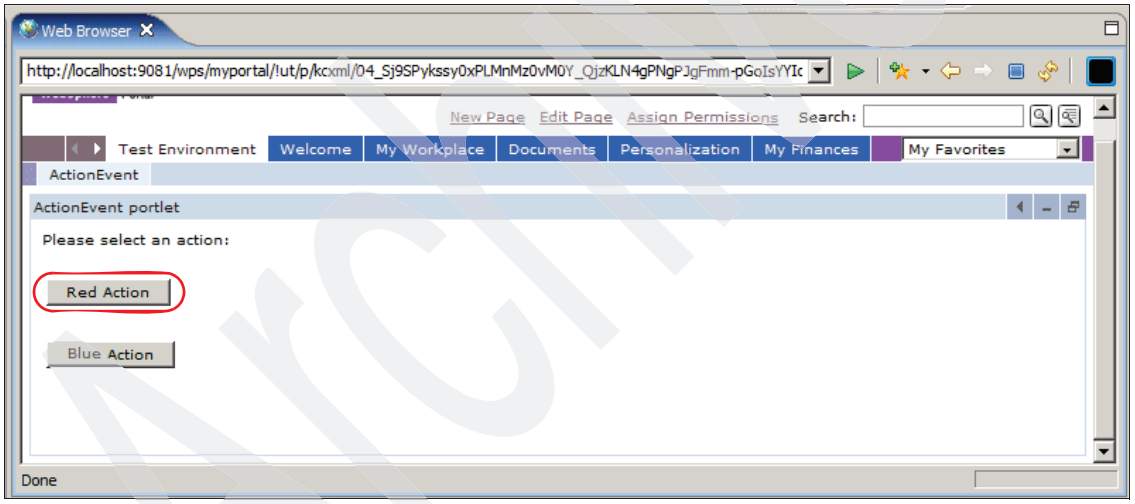


Figure 6-15 Selecting an action

6. A message will be displayed according to which button you select. You may enter Edit mode repeatedly to select an action.

Archived

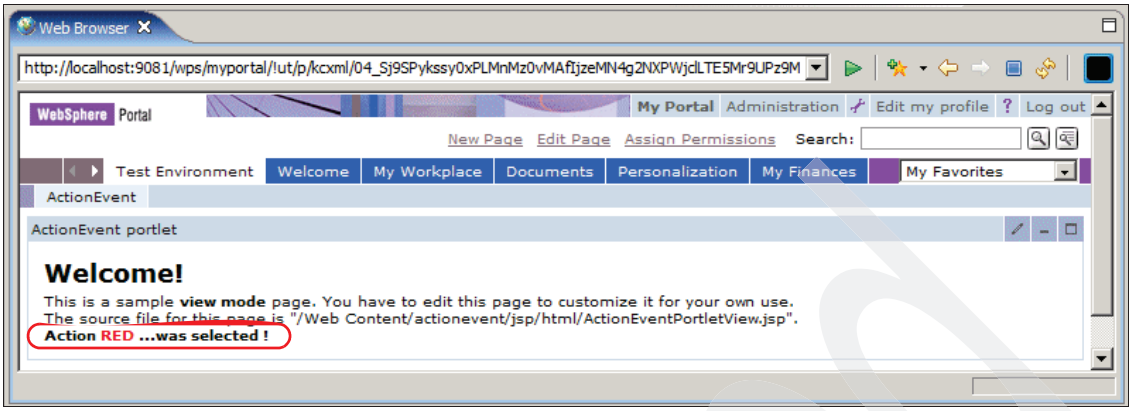


Figure 6-16 Red action selected

Archived

7

Portlet messaging

Portlet messaging is provided for portlet applications using the IBM Portlet API. This chapter describes what portlet messaging is and the objects you will need to work with when messaging between portlets.

- MessageListener
- MessageEvents
- DefaultPortletMessage
- PortletMessage

Archived

© Copyright IBM Corp. 2005. 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 the same 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. Portlets that are deployed as Web services cannot send or receive messages.

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 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.

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

Note: Sharing data between two or more portlets can be accomplished using cooperative portlets. Cooperative portlets subscribe to the WebSphere Portal property broker by publishing properties that they can share. Property Broker should be used instead of messaging.

7.2 MessageListener

The org.apache.jetspeed.portlet.event.MessageListener interface must be implemented by the portlets receiving a message. 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.

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 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 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.

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.PortletMessage;
public class AgendaMessage implements PortletMessage {
   private AgendaBean entry;

   public AgendaBean getEntry() {
      return entry;
   }
```

```
   public void setEntry(AgendaBean entry) {
      this.entry = entry;
   }
}
```

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();
      PortletRequest request = event.getRequest();
      // Add action string handler here
      if ( actionString != null && actionString.startsWith(ACTION_DETAILS)) {
         //get the entry selected from the actionString
         String opc = actionString.substring(actionString.indexOf("=")+1);
         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.setEntry(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 {
      if( getPortletLog().isDebugEnabled() )
          getPortletLog().debug("MessageListener - messageReceived called");
      // MessageEvent handler
      PortletMessage msg = event.getMessage();
      // Add PortletMessage handler here
      if( msg instanceof AgendaMessage ) {
```

```
            AgendaBean detailEntry = ((AgendaMessage)msg).getEntry();
            // 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.

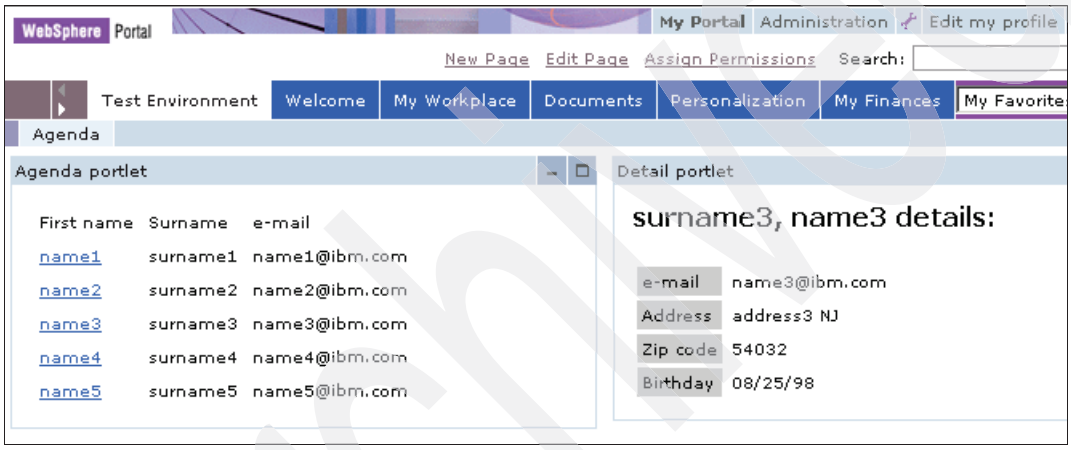


Figure 7-1 Receiving a custom message with an entry of the agenda

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 6, “IBM Portlet API portlet development” on page 203) 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.

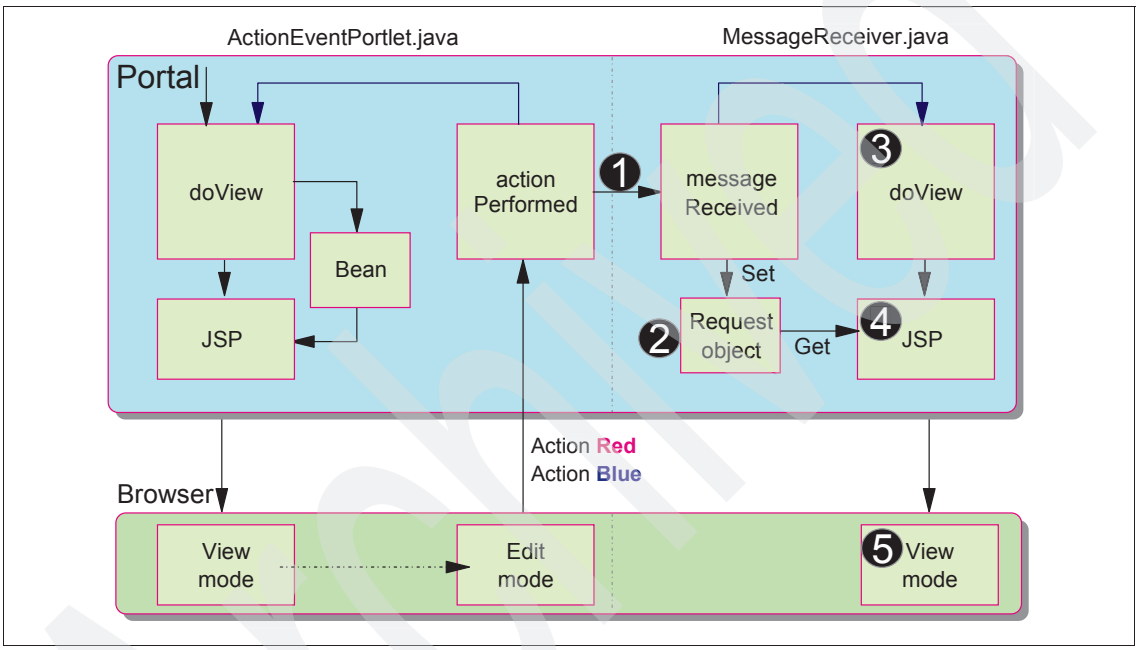


Figure 7-2 Message Event handling scenario

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.

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 IBM RAD is not running, start the IBM Rational Application Developer by clicking **Start** → **Programs** → **IBM Rational** → **IBM Rational Application Developer V6.0** → **Rational Application Developer**.
2. If the Test Environment is still running, stop the server by invoking **Servers** (make sure you switch to the Web perspective), right-click **WebSphere Portal 5.1 Test Environment** 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 Resources/Java Source/actionevent/ folder) as illustrated in Example 7-7.

Example 7-7 Modifying implementation of actionPerformed() methods

```
...
   public void actionPerformed(ActionEvent event) throws PortletException {
      if( getPortletLog().isDebugEnabled() )
         getPortletLog().debug("ActionListener - actionPerformed called");
      // ActionEvent handler
      String actionString = event.getActionString();
      PortletRequest request = event.getRequest();
      // 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 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 ade){}
      }

      if( actionString.equalsIgnoreCase(ACTION_BLUE) ) {
         // Create the string of HTML to be rendered
         String value = "Action <FONT color=\"#0000ff\">BLUE</FONT>";

         // 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 ade){}
      }

   }
...
```

4. Save and close the ActionEventPortlet.java file.
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 /WebContent/actionevent/jsp/html/ directory.

Archived

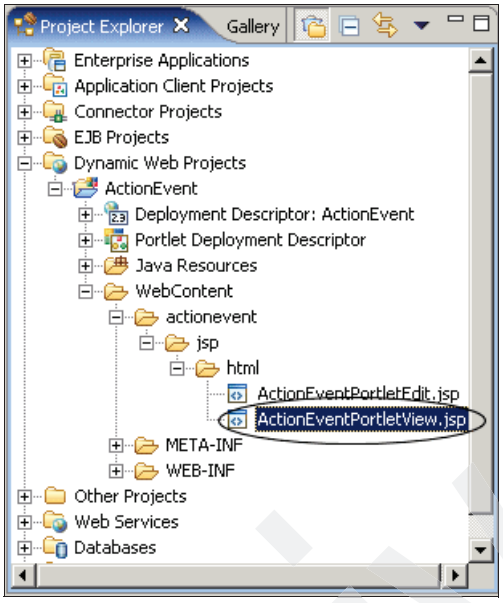


Figure 7-3 Project Explorer View

6. Insert the text highlighted in Example 7-8.

Example 7-8 Adding the code to broadcast PortletMessage

```
<%@ page session="false" 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 ! and this
information was broadcasted as a message.</B>
<% } %>
```

Archived

```
</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 the wizard to create the target portlet to receive the message sent by ActionEventPortlet.java.

1. Click **File** → **New** → **Portlet**. If you do not see this option, select **File** → **New** → **Other** and then select **Portlet** under **Portal** folder and click **Next** to continue.
2. In the next window, enter the following information:
   - Project: select **ActionEvent** from the list.
   - Default name prefix: `MessageReceiver`
   - Select the type of new portlet: `Basic portlet`

Archived

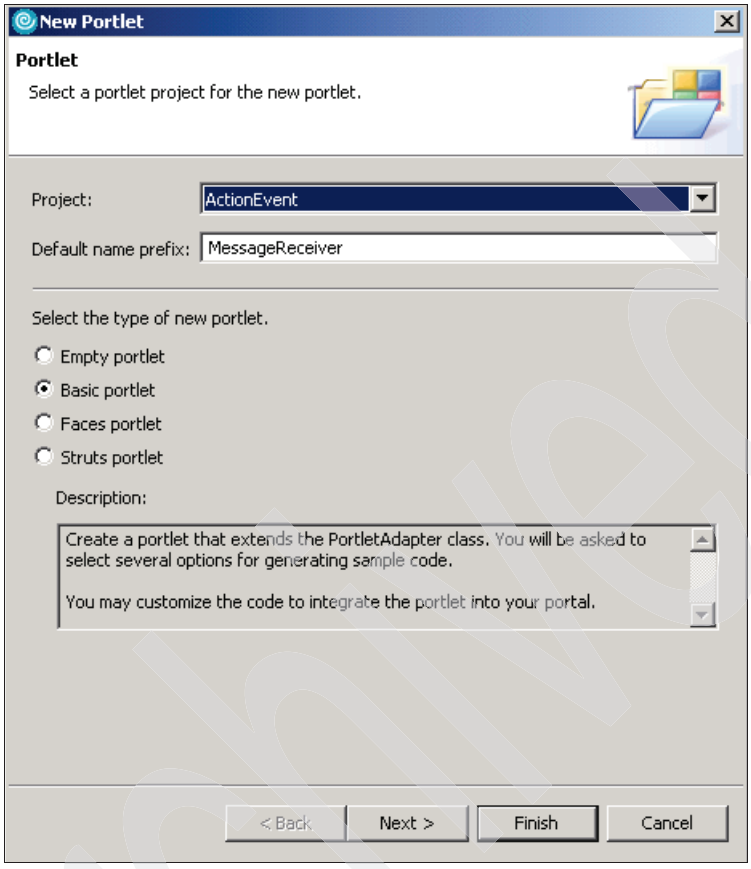


Figure 7-4 Adding a portlet

3. Click **Next**.
4. Examine and accept the default values in the portlet settings window and click **Next**.
5. Uncheck the **Add form sample** and the **Add action listener** boxes. Check the **Add message listener** box to add the messageReceived method. Click **Next**.

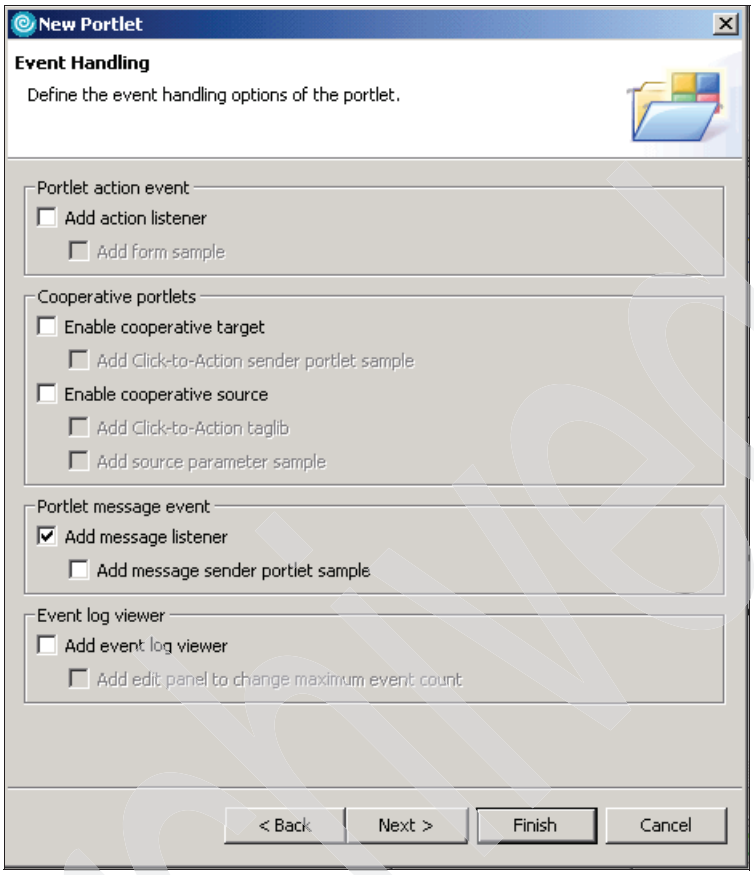


Figure 7-5 Adding a message listener

6. Do not check the **Add credential vault handling** box (not required in this sample scenario). Click **Next**.
7. Leave the options for markups and modes unchecked. Click **Finish** to add the portlet to your project.
8. You will now see the new portlet files in the Project Explorer panel.

Redbooks

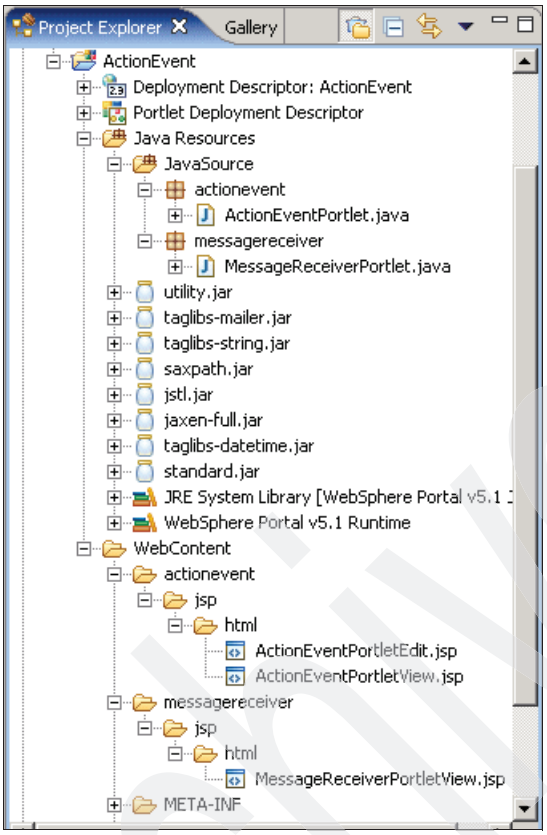


Figure 7-6 Project Explorer panel

9. Open the MessageReceiverPortlet.java file located in the /Java Resources/JavaSource/messagereceiver/ folder by double-clicking it.

10. 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.

Example 7-9 MessageReceiverPortlet.java

```
...
   public void messageReceived(MessageEvent event) throws PortletException {
       if( getPortletLog().isDebugEnabled() )
```

Archived

```
            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
        }
    }
...
```

11. When you are done, save the file and exit.

12. Now that you can receive the message, you will need to modify the MessageReceiverPortletView.jsp to display the message to the user. To edit this file, open the /WebContent/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 MessageReceiverPortletView.jsp file (message receiver portlet)

```
<%@ page session="false" 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/MessageReceiverPortletView.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>
```

13. 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 (MessageReceiverPortlet.java). Follow these steps:

1. Right-click the ActionEvent project and select **Run** → **Run on Server**.
2. In the Define new server window select **Choose an existing server** and select **WebSphere Portal V5.1 Test Environment**.
3. Click **Next.**
4. Be sure that the ActionEventEAR project is configured to run in the test environment (it appears in the right panel).
5. Click **Finish**. 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 7-7. 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.

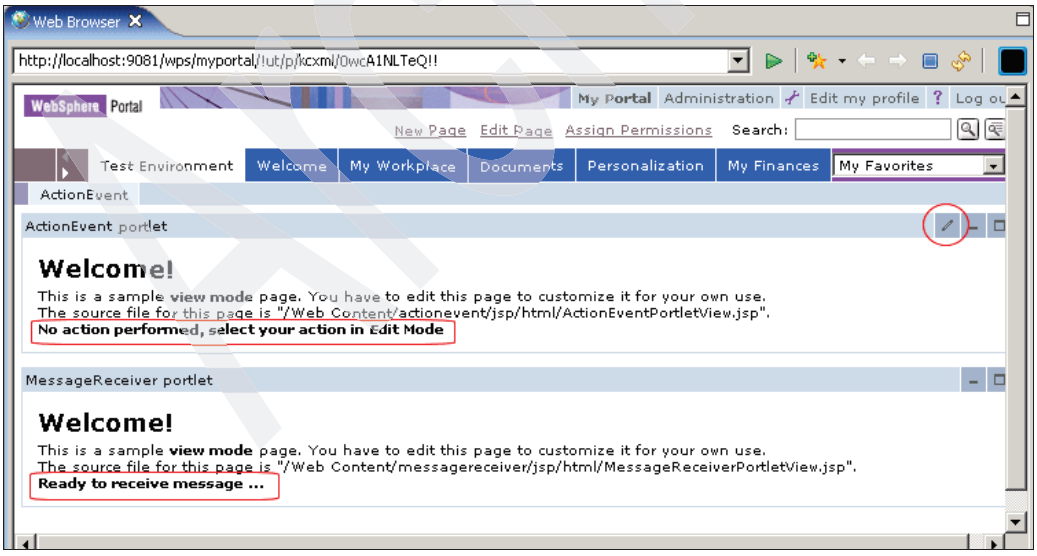


Figure 7-7 Running the messaging project in the Portal Server Test Environment

6. Choose the Edit mode of ActionPortlet. 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.

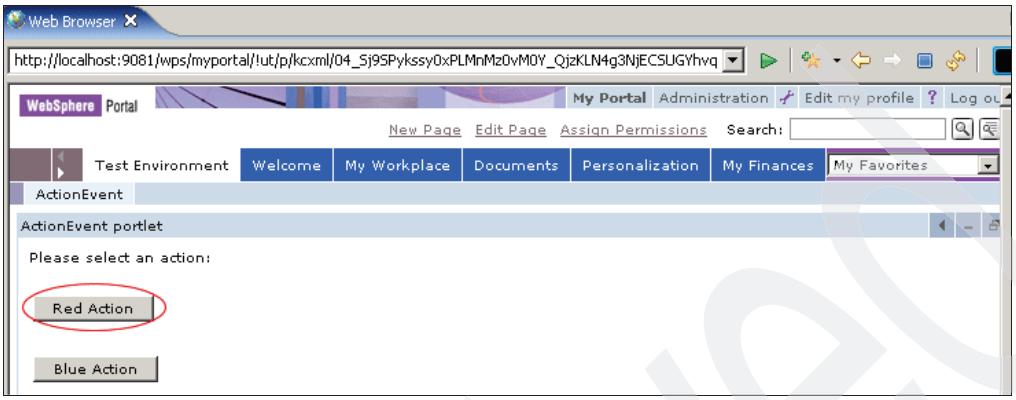


Figure 7-8 Creating an action and broadcasting the message

7. You will see the value shown in both the ActionEvent portlet and the MessageReceiver portlet.

   Note: In summary, you have seen how the IBM portlet API implements message events which can be useful for passing data between portlets that need to be notified of other portlets's actions and events. This is a very useful feature of the API when building portlet applications that contain multiple portlets.

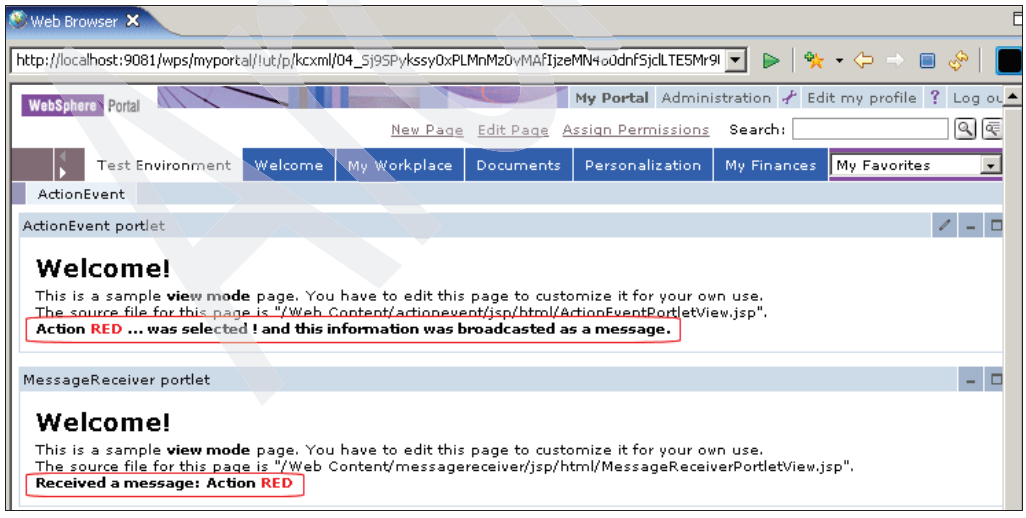


Figure 7-9 Red action and Red message broadcast

8. You can enter Edit mode again and select the **Blue Action** button.The results will again be displayed accordingly.

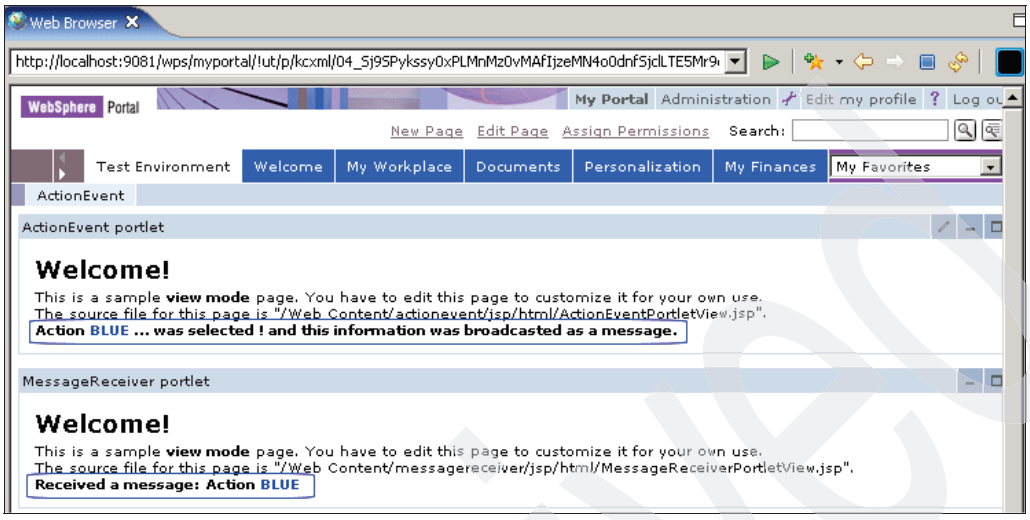


Figure 7-10 Blue action and message broadcast

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 project:
   a. Select **File** → **New** → **Portlet Project**.
   b. Enter `Message` as the project name and select **WebSphere Portal V5.1** as Target server.
   c. In the event handling page, check only the option **Add message listener**.
   d. Click **Finish**.
2. Open the MessagePortlet.java file located in the /Java Resources/JavaSource/message/ folder and modify the messageReceived() method to receive the broadcast 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.

Archive

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. You need to configure the new project to run in test environment. Right-click WebSphere Portal V5.1 Test Environment server and select **Add and remove projects**. Select MessageEAR in the available projects panel and click **Add**.

Archived

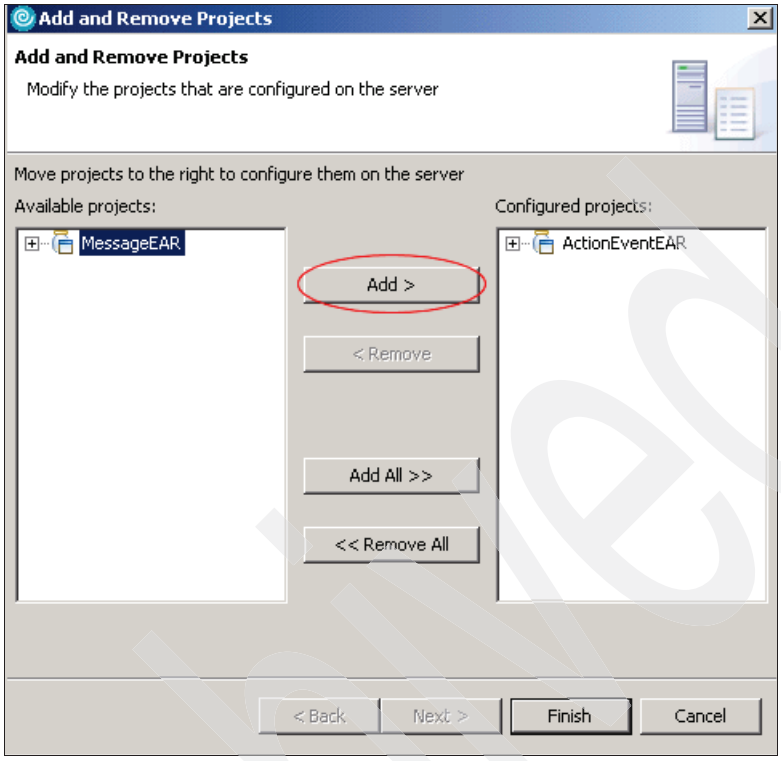


Figure 7-11 Add and remove projects window

5. Click **Finish**.
6. Before starting the server open the server properties by double-clicking in WebSphere Portal 5.1 Test environment server. Click **Portal** panel and check **enable base portlets for portal administration and customization** option.

   Note: Portlets that send and receive message must be on the same page. By default, when you configure more than one project on the server they will appear in different pages. You will need to customize the ActionEvent page adding the new Message portlet.

Archived

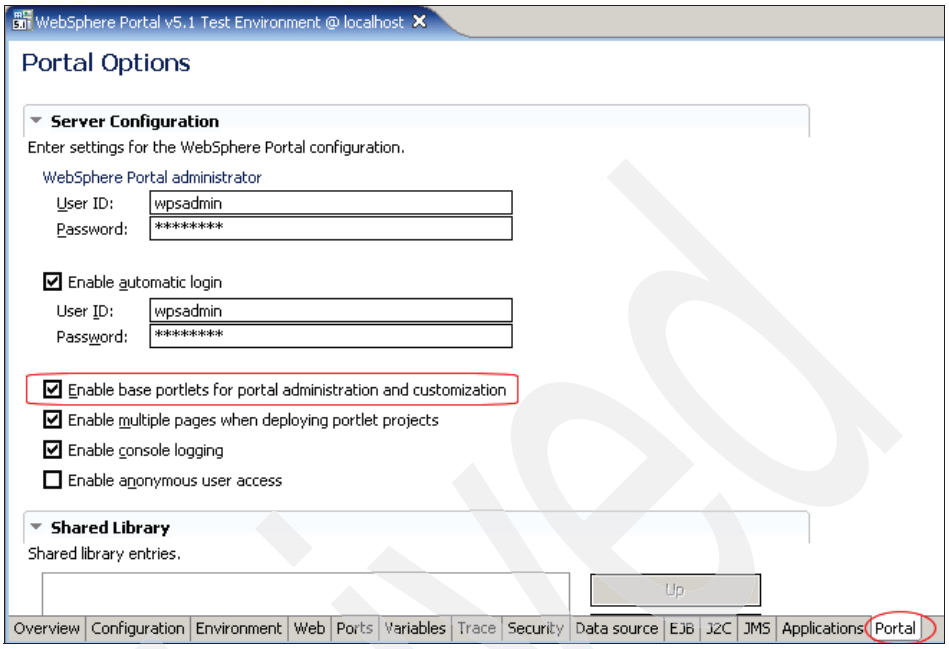


Figure 7-12 Portal options

7. Save and close the server options window.
8. Right-click the ActionEvent project or Message project and choose **Run** → **Run on Server**.
9. When the Web browser appears, you will see the applications in different pages.

Archived

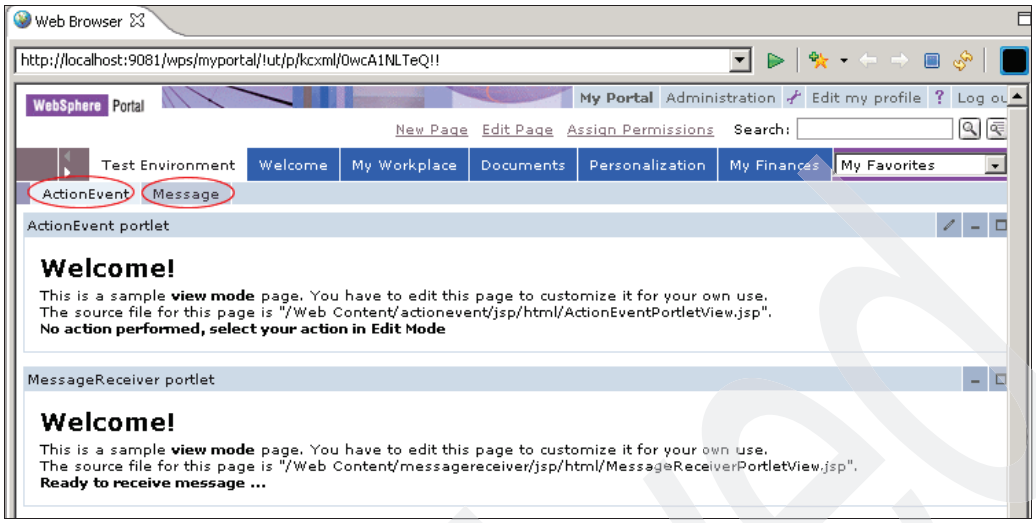


Figure 7-13 Running two portlet applications

10. To customize your page, select **Administration** in the portal theme. You need to log in as an administrator user to see this option.

11. In Administration window, select on the left panel **Manage Pages** under **Portal User Interfaces.** On the right panel, you will see options to work with your pages. Select **My Portal** and then **Test Environment**. You will see a page as show in Figure 7-14.

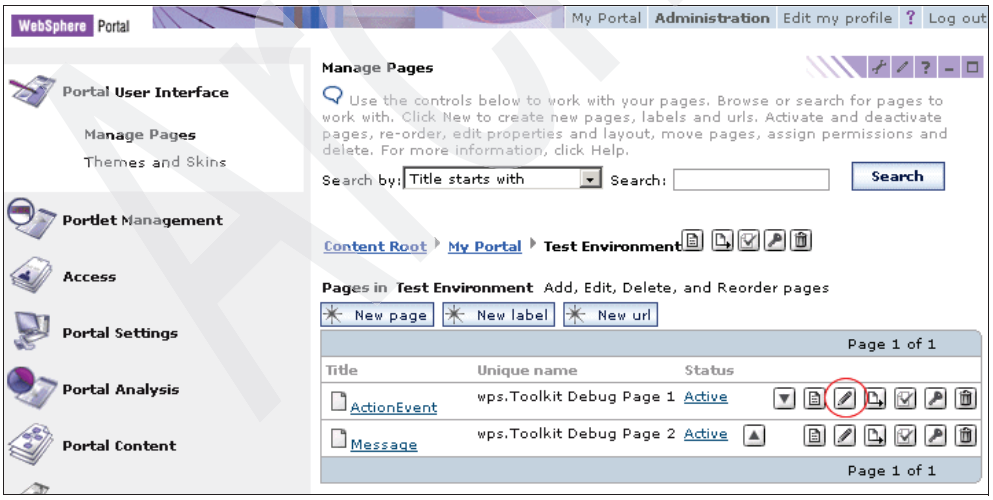


Figure 7-14 Manage pages

12. Click the pencil icon to edit page layout.

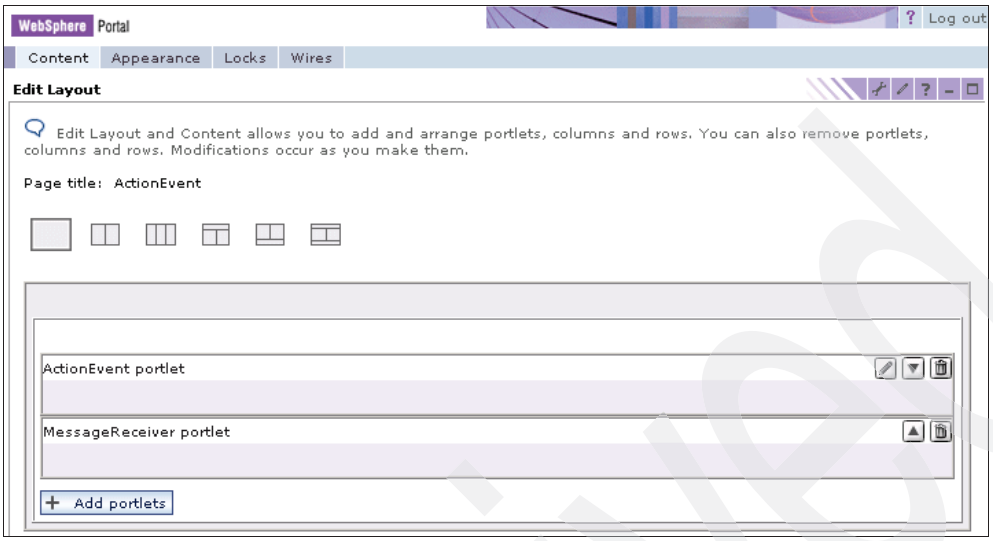


Figure 7-15 Edit page layout

13. Click **Add portlet** button. A new window appears with all portlets that have been installed. Fill the search file with the name of the portlet you are looking for, Message in our example, and click **Search**

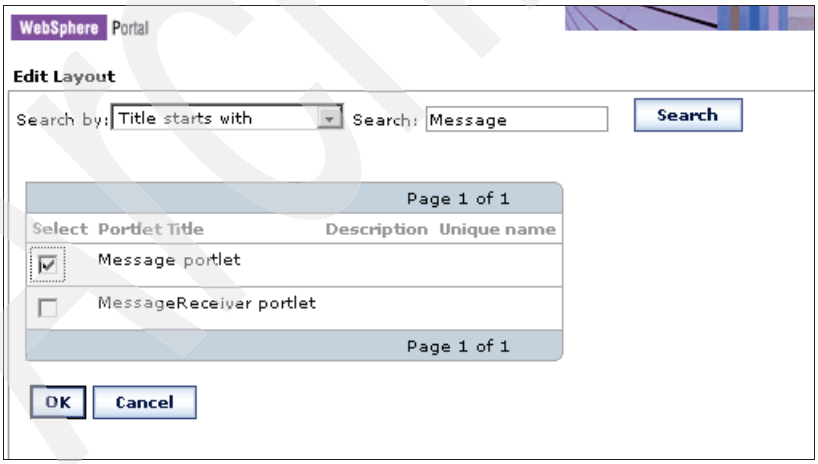


Figure 7-16 Adding a portlet to a page

14. In the results of your search you will see the Message portlet, check it and click **OK**.

15. Now you will see the Message portlet in ActionEvent page layout.
16. Click **DONE**.
17. Click **My Portal** to test the application.
18. You will see the tree portlets on ActionEvent page.

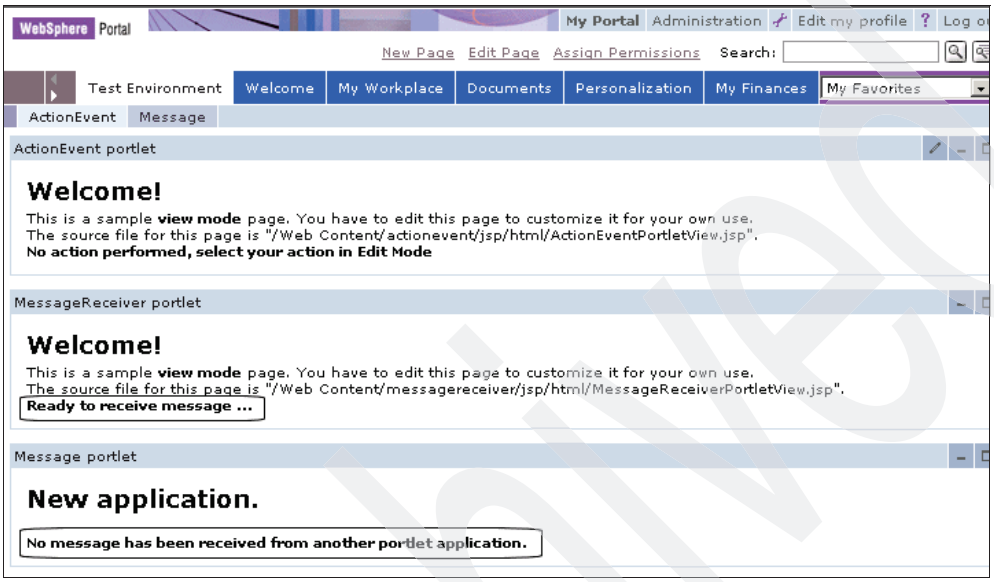


Figure 7-17 Before sending a broadcast message to all portlets

19. 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.

Archive

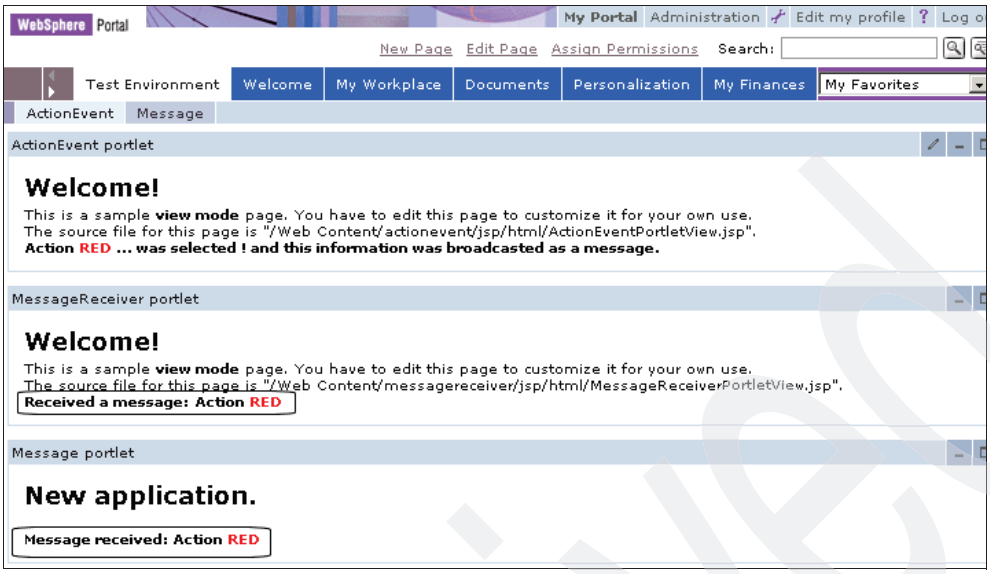


Figure 7-18 After sending a broadcast message

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 Resources/JavaSource/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.

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.

Archived

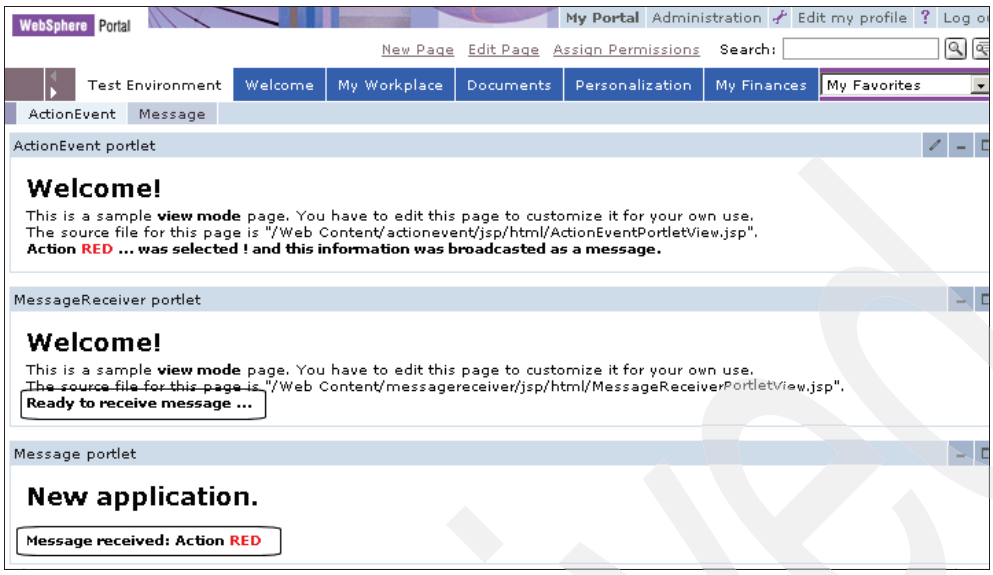


Figure 7-19 After sending a message to a specific portlet

Archived

8

JSR 168 API

JSR 168 is new portlet specification created to be a standard for portlet development. IBM WebSphere Portal provides support for portlets developed using this API. In this chapter, the following topics are discussed:

- JSR 168 overview
- JSR 168 comparison to servlets
- JSR 168 portlet modes
- JSR 168 portlet window states
- Core JSR 168 objects
- JSR 168 portlet life cycle
- JSR 168 portlet caching
- Listeners
- Deployment descriptors
- JSR 168 limitations in WebSphere Portal

Archived

© Copyright IBM Corp. 2005. All rights reserved.

8.1 JSR overview

JSR 168 is a specification from the Java Community Process for the standardization of portlets. The specification was developed to provide interoperability for running portlets on any vendor's implementation of the JSR 168 portlet container. The specification was approved in October of 2003. For more information about the Java Portlet Specification, see the JSR 168 specification:

`http://developers.sun.com/prodtech/portalserver/reference/techart/jsr168/`

WebSphere Portal starting with version 5.0.2.1 provides a runtime environment for both the IBM Portlet API, which is based on org.apache.jetspeed.portlet interfaces and JSR 168 compliant portlets.

The basic concepts are similar between the two APIs; however, several differences do exist.

Note: Portlets of either API can be placed on the same portal page. However, a portlet cannot mix classes and methods from both packages although the JSR 168 API can take advantage of some of the IBM specific features, including Cooperative portlets and portlet services like the Credential Vault.

8.1.1 Number of portlet instances

The portlet deployment descriptor, also called portlet.xml, defines how the portlet container creates the portlet instance. For a single system which is not clustered there is only one portlet object per portlet definition. In a clustered environment (marked distributable in the web.xml), only one portlet object will be instantiate per portlet definition per VM.

8.1.2 Portlet windows

Portlets are defined in the portlet.xml. The portlets may contain preferences and default values. These values are used to create the PortletPreferences object. The portlet entity has the PortletPreferences attached to it. A portlet window is the representation of a portlet entity on a page, including navigational and session state attached to it.

The portletPreferences object may be customized by the user or by the administrator. Administrators can globally change the parameters while in Config mode. The changes will affect all of the portlet instances on all pages. If a user modifies a portletPreference, it will only affect their PortletPreferences object.

The user will use the administrator's portletPreferences object until they set their own; after that, changes made by the administrator will not be reflected in the users portletPreferences.

As an example, a portlet is placed on a page that displays your name. The administrator has access to the Edit and Configuration modes. The user will only have access to the Edit mode. The administrator modifies the name using the Configuration mode to enter his name. The user then logs in and sees the administrator name. The user changes the name in Edit mode, adding their own name. Now the user will always see their own name even if the administrator updates it in Config mode. This user has their own portlet window representing this portlet.

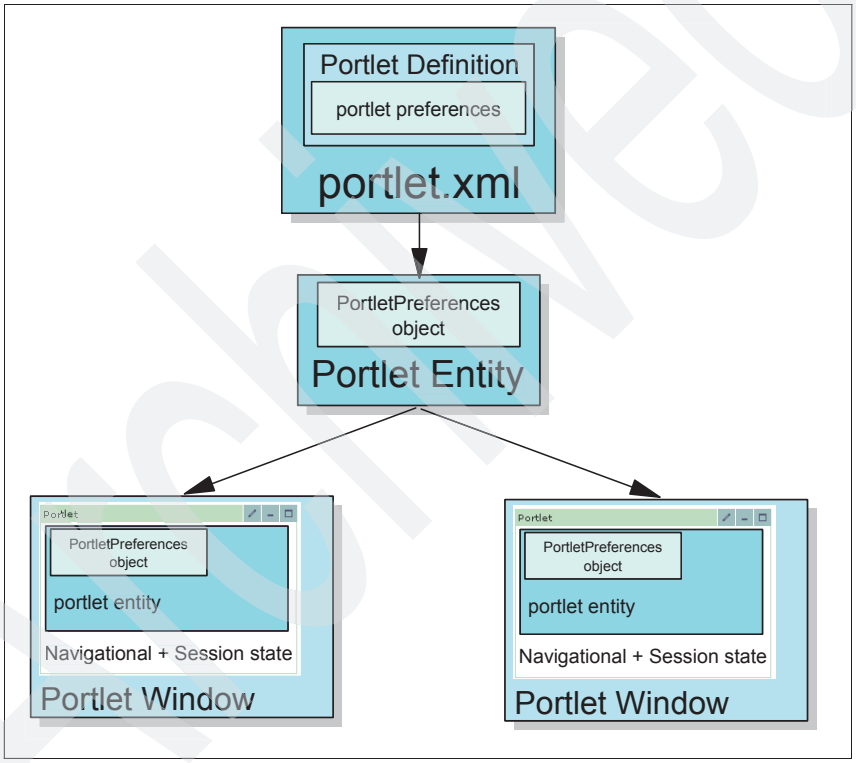


Figure 8-1 Portlet window

8.1.3 Thread safety

The PortletRequest and PortletResponse objects are not guaranteed to be thread safe. These object should only be used in the scope of the processAction

and render methods. You should not reference the PortletRequest or PortletResponse in any object outside these methods.

WebSphere Portal server will handle multiple concurrent requests to the same portlet request handling methods on multiple threads. If the request handling methods modify class variables, these variables will also be modified for all instances of the portlet. This can cause unexpected problems in your portlets that are difficult to debug. Use method local variables to avoid this problem.

8.2 JSR 168 comparison to servlets

In the IBM Portlet API, portlets are servlets and extend the HttpServlet class. The IBM Portlet API also implements or extends many of the servlet interfaces including the HttpServletRequest, HttpServletResponse, and HttpSession. The JSP 168 API does not extend or implement any of the Servlet classes or interface. However, JSR 168 portlets are modeled after the servlet specification.

JSR 168 portlets do leverage many features of the Servlet specification including deployment, classloading, life cycle, session management and request dispatching.

Unlike servlets, portlets can only generate markup fragments. The portal server will aggregate all portlets and themes and skins to create the complete page. Portlets therefore cannot set character set encoding for the response. They also cannot set HTTP header responses.

The life cycle of servlet and portlets also differ and will be discussed in a later section, please see.“JSR 168 Portlet life cycle” on page 280.

Portlets are loaded with the same classloader used to load the Web application. This allows the ServletContext and the PortletContext to share attributes. This also allows the HttpSession to share attributes with the PortletSession. The opposite is true also, the PortletSession can share attributes with the HttpSsession.

8.3 JSR 168 portlet modes

Portlets will perform different tasks and create different output depending on the function they are currently performing. Modes will allow the portlets to provide different function for the task that is required. JSR 168 supports three modes, View, Edit, and Help. JSR 168 also supports custom modes. IBM WebSphere Portal at the time of this writing only supports the custom mode Configuration.

Portlet can change the mode programmatically in the processAction method. You can also specify the mode when creating an action or render link. See “interface javax.portlet.PortletURL” on page 261.

View

The View mode is used for displaying content reflecting the current state of the portlet. View mode may included multiple screens the user can navigate. The doView() method of the GenericPortlet class is invoked for this mode. All portlets are required to support the View mode.

Edit

The Edit mode is used for customizing the behavior of the portlet by modifying the PortletPreferences object. As with View mode the Edit mode may contain multiple screens for navigation. The doEdit() method of the GenericPortlet class is invoked for this mode. Portlets are not required to support the Edit mode.

Help

Help should be used to provide the user with information about the portlet. This could be generic or it could provide context-sensitive help. The doHelp() method of the GenericPortlet class is invoked for this mode. Portlets are not required to support the Help mode

Custom-portlet-mode

Note: WebSphere Portal will only support the custom portlet mode Configuration.

Configuration mode is used to globally update a portlet configuration. In Configuration mode an administrator can update the read-only PortletPreferences. Changes are reflected in all occurrences of the portlet on all pages. The doDispatch() method of the GenericPortlet class needs to be overridden to handle this custom mode. The code in Example 8-1 was generated from Ration Application Developer when custom mode was selected in the portlet creation process.

Example 8-1 Overridden doDispatch() method to handle the custom Config mode

```
protected void doDispatch(RenderRequest request, RenderResponse response)
                                    throws PortletException, IOException {
      if( !WindowState.MINIMIZED.equals(request.getWindowState()) ){
         PortletMode mode = request.getPortletMode();
         if( CUSTOM_CONFIG_MODE.equals(mode) ) {
            doCustomConfigure(request, response);
            return;
```

```
            }
        }
        super.doDispatch(request, response);
    }
```

The doDispatch() method calls the doCustomConfigure() method of the portlet class if the current mode is represented by the CUSTOM_CONFIG_MODE variable, else it calls the doDispatch of GenericPortlet class to handle all other modes.

8.4 JSR 168 Portlet window states

Window state determines how much space the portlet will have to render its content. The portlet container will provide the current window state to the portlet to allow the portlet the display the appropriate content. JSR 168 supports three Window states, NORMAL, MAXIMIZED, and MINIMIZED.

Like portlet mode the Window state can be programmatically changed in processAction() while processing an action request. The state can also be requested while creating a render or action URL.

NORMAL

Normal indicates that the portlet will be provided equivalent space on the page as other portlets. This portlet could also be the only portlet on the page. The Show layout tools in WebSphere Portal could be used to further customize the layout of the portlet on the page.

MAXIMIZED

Maximized state indicates that the portlet will be the only portlet displayed on the page.

MINIMIZED

The default implementation of the GenericPortlet class and the IBM supplied skins, are to show only the portlet title bar when in the minimized state. The skins will not invoke the render method and therefore will not display any of the portlets output. You can modify the control.jsp for the skin as seen in Example 8-2 on page 257. You will also need to override the doDispatch() method to get different results as seen in Example 8-3 on page 257.

Archive

Note: Example 8-2 shows the portletState solo. This is to support portlets written using the IBM Portlet API. Solo is not a supported window state in the JSR 168 API.

Example 8-2 Modifying control.jsp in the skin

```
<wps:if portletState="Normal,Maximized,Minimized,Solo">
```

Example 8-3 Overriding doDispatch

```
protected void doDispatch(RenderRequest request, RenderResponse response)
       throws PortletException, IOException {
   if(request.getWindowState() == WindowState.MINIMIZED){
      response.setContentType(request.getResponseContentType());
      response.getWriter().println("Portlet is minimized!<br />");
   } else {
   super.doDispatch(request, response);
   }
}
```

Custom window states

Note: Custom Windows states are part of the JSR 168 specification and is mentioned here. WebSphere Portal at the time of this writing does not support custom Window states. Unlike the IBM portlet API, JSR 168 does not have a solo window state either.

The custom state would be defined in the portlet deployment descriptor using the custom-window-state element. Portlets could use the getSupportedWindowStates() method of the PortalContext to obtain the window states supported by the portal server. Unsupported states will be ignored.

8.5 Core JSR 168 objects

This section will describe the core object used when developing a JSR 168 portlet. We will also discuss some important method of each object.

8.5.1 interface javax.portlet.Portlet

All portlets must implements this interface directly or by extending a class that implements this interface. This interface defines the following methods.

void destroy()

This method is called when the portlet is being removed from service. Use this method to release any resources or write persistent state. Once the destroy method has been called no request can be sent to this portlet. If needed, a new portlet object will be instantiated to serve new requests. If there is a RuntimeException thrown during the destroy method, the destroy is considered to have completed successfully. The portlet object is eligible for garbage collection after the destroy method has completed.

void init(PortletConfig config)

The init method is called upon the first request to the portlet to indicate that the portlet is being placed into service. The init must successfully complete before the portlet is allowed to service any requests. The init is called only once per portlet object and should be used to load any expensive resources (such as database connections). A unique PortletConfig object is passed into the init method. This PortletConfig object is per portlet definition. You can gain access to the initialization parameters and the resource bundles from the PortletConfig object. You can also get the PortletContext from the PortletConfig. The PortletContext will provide information about the portlet's runtime environment.

The init method can throw two different exceptions. An UnavailableException or a PortletException. If either type of exception is thrown then the portlet is not put into service and the destroy method is not called.

The UnavailableException may specify a minimum amount of time that the portlet will be unavailable. If a time is specified then the portlet cannot be put back into service until at least that amount of time has expired.

void processAction(ActionRequest request, ActionResponse response)

The processAction method is called in response to an action request. URLs generated by the portlet using RenderResponse.createActionURL() or the ActionURL JSP tag will generate an action URL causing this method to be invoked. This method should be used to update the portlets state based on the action request parameters. Two objects are passed in, an ActionRequest and an ActionResponse.

The ActionRequest provides access to the parameters, window state, portlet mode, portlet context, portlet session and the PortletPreferences object. During the action request, the portlet may issue a redirect to a different URL.

While processing an action request, the portlet window state and the portlet mode may be changed. This change would be reflected in the next render phase.

Archived

It should not be assumed that the portlet will change mode or window state as WebSphere Portal server may override the change. Changes to the window state and mode are made through the ActionResponse object.

Code in this method should be written to handle concurrent execution from multiple threads.

void render(RenderRequest request, RenderResponseresponse)

The render method is invoked for all render requests. During the render request, the portlet generates content based on its current state (View, Edit, Help, or Configuration). The render method are passed two objects. A RenderRequest object and a RenderResponse object. The RenderRequest provides access to the render parameters, window state, portlet mode, portlet context, portlet session and the PortletPreferences object. The RenderRequest does not have access to the parameters passed into the ActionRequest unless they have been explicitly added during the action request using the ActionResponse.setRenderParameter() method.

The RenderResponse is used to display content, it can either use the RenderResponse Writer or it can delegate to a JSP or Servlet using the PortletRequestDispatcher.

Code in this method should be written to handle concurrent execution from multiple threads.

8.5.2 class javax.portlet.GenericPortlet

The GenericPortlet class implements the Portlet interface and does provide default implementations of the methods. All portlets should extend the GenericPortlet class rather that implementing the Portlet interface directly. The GenericPortlet implements many of the methods and reduces the workload when developing portlet.

java.lang.String getInitParameter(java.lang.String name)

Returns the value of the initialization parameter. If the parameter does not exist null will be returned

java.lang.Enumeration getInitParameterNames()

Returns all the initialization parameters names in an Enumeration of Strings. If there are no initialization parameters defined then an empty Enumeration is returned.

Archived

javax.portlet.PortletContext getPortletContext()

Returns the PortletContext for the portlets portlet application. Please see “interface javax.portlet.PortletContext” on page 263 for more information about the PortletContext class.

java.lang.String getName()

Returns the name of the portlet defined by either the administrator or the application deployment descriptor.

java.util.ResourceBundle getResourceBundle(java.util.Localelocale)

Portlets may define basic information to be used in the title bar. This information may include title, short-title and keywords. This information can be added directly in the portlet deployment descriptor or it may be added in an external ResourceBundle. If this information is located in an external ResourceBundle, the portlet deployment descriptor must provide the name of the ResourceBundle. Portal will search the values in the ResourcesBundle first and then it will look for the resources inline. If the resource cannot be found, an empty String will be returned. The render method of the GenericPortlet uses this method to get the portlet title.

void doDispatch(RenderRequest request, RenderResponseresponse)

The doDispatch method is called from the render method. The doDispatch will determine the portlet mode and invoke the correct method. If using a custom mode, you should override the doDispatch method to test for your custom mode. If the request is not for your custom mode invoke super.doDispatch to allow GenericPortlet do dispatch the correct method.

void doEdit(RenderRequest request, RenderResponseresponse)

When the portlet mode is Edit, the doEdit method will be invoked by doDispatch.

void doHelp(RenderRequest request, RenderResponseresponse)

When the portlet mode is Help, the doHelp method will be invoked by doDispatch.

Archived

void doView(RenderRequest request, RenderResponseresponse)

The doview method is invoked by doDispatch when the portlet mode is View.

PortletConfig getPortletConfig()

This method returns the PortletConfig object for this portlet. Please see “interface javax.portlet.PortletConfig” on page 277 for more information about the PortletConfig class.

java.lang.String getPortletName()

This returns the name of the portlet.

java.lang.String getTitle(RenderRequest request)

This returns the portlet title.

void processAction(ActionRequest request, ActionResponseresponse)

The processAction method is invoked for all action requests for the portlet.

void render(RenderRequest request, RenderResponseresponse)

The render method sets the portlet title and then invokes the doDispatch() method.

8.5.3 interface javax.portlet.PortletURL

The PortletURL are used to create URLs that reference the portlet. There are two type of PortletURLs, ActionURL and RenderURL. By using the RenderResponse.createActionURL, RenderResponse.createRenderURL or the JSP tags, the portlet developer can create the different types of URLs. When a render URL is encountered, the render method is invoked and it turn invokes the appropriate doXXX method. When an action URL is encountered, the processAction method is invoked. Only action URLs should be used for forms. Render and Action parameters are different. Parameters set for an action URL (parameters in an HTML form) are not available to the render method unless they are explicitly added using the setRenderParameter of the ActionResponse class. Some important methods of this interface are listed below.

Archived

void setParameter(java.lang.String name, java.lang.Stringvalue)

This method is used to add parameters. Parameter added for a render URL will only be available during the render request. Parameters added for an action URL will only be available for the action request unless they are specifically added using ActionResponse.setRenderParameter method.

void setParameter(java.lang.String name, java.lang.String[]values)

This method will have the same characteristics as the method above, only that it should be used when there are multiple values for the parameter.

void setParameters(java.util.Map parameters)

This method also adds parameters, and will have the same characteristics as the two methods listed above. This is used when the name value pairs of the parameters are in a Map. The keys in the Map must be of type java.lang.String and the values must be of type java.lang.String[] (String array)

void setPortletMode(PortletMode portletMode)

This method is used to change the portlet mode. If the mode is not a supported mode by the portlet then a PortletModeException will be thrown.

void setWindowState(WindowState windowState)

This method is used to change the portlet window state. If the window state is not a supported state a WindowStateException will be thrown

java.lang.String toString()

This method will return a URL in the correct form for the portal server. See Example 8-4.

Example 8-4 Using an actionURL in a link

```
<%
PortletURL actionurl = renderResponse.createActionURL();
%>
<a href="<%= actionurl.toString() %>">ActionURL</a><BR>
```

void setSecure(boolean secure)

This method is part of the PortletURL interface, however, it is not supported by WebSphere Portal. The security level will always be same as the current request.

8.5.4 interface javax.portlet.PortletContext

The PortletContext provides the portlet with information about the portlet application that it is running in. If the portlet application is not in a distributed environment, there is only once PortletContext object instantiated for each portlet application deployed. If the application is in a distributed environment (marked distributable in the web.xml) then there will be one instance per JVM.

You can log events, get resources for the portlet application, initialization parameters, you can also store and retrieve attributes that all the portlets, JSPs and servlet of the application can access. You will use the PortletContext object to obtain a request dispatcher for including other JSPs and servlets. The PortletContext and the ServletContext are closely coupled. The initialization parameters for both the PortletContext and the ServletContext are defined in the web.xml. The PortletContext attributes are stored in the servlet context and are accessible in any JSP or servlet. Attributes that are stored in the servlet context are also available to all portlets.

The following methods of the PortletContext class will have the same implementation as the ServletContex:

- getAttribute
- getAttributeNames
- getInitParameter
- getInitParameternames
- getMimeType
- getRealPath
- getResource
- getResourcePaths
- getResourceAsStream
- log
- removeAttribute
- setAttribute

The portlet specific methods of the PortletContext interface are:

int getMajorVersion()

Returns the major version of the JSR 168 Portlet API that is supported by the Portal server.

int getMinorVersion()

Returns the minor version of the JSR 168 Portlet API that is suported by the Portal server.

Archive

java.lang.String getPortletContextName()

This returns the display-name element defined in the web.xml. This represents the name of the portlet application for this PortletContext.

Example 8-5 getPortletContextName()

```
web.xml
<web-app id="WebApp_ID">
   <display-name>MyPortlet</display-name>
....
</web-app>

Code in portlet
String portletDisplayName = getPortletContext().getPortletContextName();
System.out.println("Display name: " + portletDisplayName);

Output in console
Display name: MyPortlet
```

8.5.5 interface javax.portlet.PortletRequest

The portlet request object will contain information about the client request. The request will also include parameters, the portlet mode, the window state, session, and access to the portlat context. PortletRequest defines commonly used functionality for ActionRequest and RenderRequest.

Parameters received during an action request will not be sent to the render request unless they are explicitly added using the setRenderParameters or setRenderParameters of the ActionResponse class. This can only be done in the processAction method.

If the render request follows an action request as part of the same request, the parameters sent in the render request will be the render parameters set during the action request.

A portlet will only be able to see parameters in its own request. Parameters set for other portlets will not be visible.

Parameter and attribute names beginning with javax.portlet are reserved and should not be used.

Request properties are used for portal specific properties. These properties may include http headers.

Example 8-6 Code to get the properties and values of a request

```
Enumeration enum = request.getPropertyNames();
```

Archived

```
while(enum.hasMoreElements()){
   String name=(String)enum.nextElement();
   Enumeration values = request.getProperties(name);
   StringBuffer valueBuffer = new StringBuffer();
   while(values.hasMoreElements()){
   //seperate values with a ';'
   valueBuffer.append((String)values.nextElement() + "; "); }
   System.out.println("Name: " + name + ", Value: " + valueBuffer.toString());
}
```

Example 8-7 Output from the above code while running a sample portlet

```
Name: accept-encoding, Value: deflate; gzip;
Name: connection, Value: Keep-Alive;
Name: referer, Value:
http://localhost:9081/wps/myportal/!ut/p/kcxml/04_Sj9SPykssy0xPLMnMz0vM0Y_QjzKL
N4g39DEDSUGYpvqRaGLGmEKOCBFvfV-P_NxU_QD9gtzQOIhyROUAATozOQ!!/delta/base64xml/L0
lJWWtpaWxDbEEhIS9JRGpBQUFUQUFNSkFBTXdzaXNwc1lBISEvNElVR1JZUWxHamdJLzZfMF8xTDYvN
18wXzFSRQ!!;
Name: host, Value: localhost:9081;
Name: accept-language, Value: en-us;
Name: user-agent, Value: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0);
Name: cookie, Value: JSESSIONID=0OOO4jBOKukifHc_3h_7QBn9D-z:-1;
Name: wps.markup, Value: html;
Name: accept, Value: */*;
```

The following methods should be used for getting and setting request parameters:

java.lang.String getParameter(java.lang.String name)

Returns the value of the request parameter or null if it does not exist.

java.util.Enumeration getParameterNames()

Returns all the parameter names in the form of an Enumeration of Strings or null if no parameters exist.

java.lang.String[] getParameterValues(java.lang.String name)

Return a String array of the parameters values. If the parameter does not exist null will be returned.

java.lang.Map getParameterMap()

Returns a Map of all the parameters. The keys are Stings and the values are stored in String [] (String arrays). If there are no parameters, an empty Map will be returned.

Archived

The following methods should be used for getting and setting request attributes.

java.lang.Object getAttribute(java.lang.String name)
Return an object of the named attribute or null if the object does not exist

java.util.Enumeration getAttributeNames()
Returns an Enumeration of the names of all the attribute.

void setAttribute(java.lang.String name, java.lang.Objectvalue)
This method is used to add an attribute.

void removeAttribute(java.lang.String name)
This method removes the stored named attribute.

Here is a list of other important method of the PortletRequest class.

java.lang.String getContextPath()
The path returned starts with '/' character but it does not end with a '/'. This method should be used when including resources such as images

Example 8-8 Code to get the context path of the portlet

```
<%= portletResponse.encodeURL(renderRequest.getContextPath() +
"/images/myImage.jpg") %>
```

boolean isPortletModeAllowed(PortletMode mode)
This method will true if the mode is defined in the portlet deployment descriptor. If the mode is not defined it will return false.

PortletMode getPortletMode()
This method will return the current portlet mode.

boolean isWindowStateAllowed(WindowState state)
If the window state is defined in the deployment descriptor this will return true, else it will return false.

WindowState getWindowState()
Returns the current window state of the portlet.

Archived

java.lang.String getProperty(java.lang.String name)

Returns the value of the named property. If the property does not exist, null is returned. Only the first value is returned if there are more than one value. The getProperties() method should be used for multiple values. An IllegalArgumentException is thrown is the name is null.

java.util.Enumeration getProperties(java.lang.String name)

Returns an Enumeration of Strings representing all the values of the named property. An empty Enumeration is returned if there are no properties. An IllegalArgumentException is thrown if the name is null.

java.util.Enumeration getPropertyNames()

Returns an Enumeration of Strings containing all the properties for this request. An empty Enumeration is returned if there are no properties.

java.lang.String getRemoteUser()

This method will return the user who is logged in. If the user is not logged in null will be returned. Even with security disabled this method will return the user information, although with security disabled the returned name may not be human-readable. With security enabled a more readable user name will be returned.

boolean isUserInRole(java.lang.String role)

Note: At the time of this writing this method is unsupported and will always return false. All <security-role-ref> elements in the portlet.xml are ignored.

javax.security.Principal getUserPrincipal()

Returns the Principal object of the current user who is logged in. If security is not enabled, this method will always returns null.

8.5.6 interface javax.portlet.ActionRequest

The ActionRequest interface extends the PortletRequest interface and is used during an action request. The ActionRequest object is passed into the processAction method. The ActionRequest object is in scope only during execution of the processAction method.

The ActionRequest interface adds the additional ability to get the input stream of the request. Access to the input stream is useful when a file is uploaded from a form.

Archived

Example 8-9 Form is a JSP using multipart encoding to upload a file.

```
<form method="POST" action="<portlet:actionURL />"
enctype="multipart/form-data">
<input type="file" name="<%= UploadPortlet.FILE_NAME %>"><br />
<INPUT name="<%=UploadPortlet.FILE_SUBMIT%>" value="Upload" type="submit"/>
</form>
```

java.io.BufferedReader getReader()

This returns a BufferedReader containing the body of the HTTP request as character data. The character data returned will be of the same type as the character encoding of the body. If the character encoding is not supported, an UnsupportedEncodingException will be thrown.

When the data is of type application/x-www-form-urlencoded, portal has already processed the form and the parameters will be available via request parameters. An IllegalStateException will be thrown if the data is of type `application/x-www-form-urlencoded.`

Either the getReader or the getPortletInputStream may be called for the ActionRequest object but not both. If the getPortletInputStream has been called, an ensuing call to getReader will throw an IllegalStateException.

java.io.InputStream getPortletInputStream()

An InputStream containing the HTTP request body in the form of binary data will be returned.

When the data is of type application/x-www-form-urlencoded, portal has already processed the form and the parameters will be available via request parameters. An IllegalStateException will be thrown if the data is of type `application/x-www-form-urlencoded.`Either the getPortletInputStream or the getReader may be called for the ActionRequest object but not both. If the getreader has been called, an ensuing call to getPortletInputStream will throw an IllegalStateException.

java.lang.String getContentType()

Returns the MIME type of the HTTP request body. If the type is not known, null is returned.

java.lang.String getCharacterEncoding()

Returns the character encoding used in the body of the HTTP request or null if one was not specified.

void setCharacterEncoding(java.lang.String enc)

This method will set the character encoding for the returned input of the getReader method. This must be called before the call to getReader. An UnsupportedEncodingException is thrown if the encoding passed in is unsupported or unknown. An IllegalStateException will be thrown if this method is called after getReader().

int getContentLength()

Returns the length of the input stream bytes. If the length is unknown a -1 will be returned.

Example 8-10 Parsing a file using the getReader() method

```
if(request.getContentType().lastIndexOf("multipart/form-data") != -1) {
   BufferedReader in = request.getReader();
   String line;
   while ((line = in.readLine()) != null) {
   if((line.indexOf("Content-Disposition") != -1) ||
   (line.indexOf("Content-Type") != -1)){
   /*
   Place your parsing code here.
   The first few lines will contain the filename and content type
   Example:
   ----------------------------7d5352d210102
Content-Disposition: form-data; name="FileName"; filename="C:\MyFile.txt"
Content-Type: text/plain
<This would be a blank line>
<The rest of the file>

All request parameters would then be printed
Example:
----------------------------7d5352d210102
Content-Disposition: form-data; name="UserName"

Hailey
*/
}
else if(line.length() > 0)
   //Line is part of the file
}
out.println(request.getPortletSession().getId());
in.close();
}
```

Archived

8.5.7 interface javax.portlet.RenderRequest

The RenderRequest interface extends the PortletRequest. The RenderRequest does not define any additional functionality. The RenderRequest object is scoped to the render method.

8.5.8 interface javax.portlet.PortletResponse

The portlet response object contains information to be returned to the portlet during a request. Examples of this include redirection, portlet mode change, title, content, etc. The portlet response object is passed into the render and the processAction methods.

The portlet response provides a way to add or update the response properties.

void setProperty(java.lang.String key, java.lang.String value)

This method will replace the value of the existing key. If the key is being passed in is null, an IllegalStateException will be thrown.

void addProperty(java.lang.String key, java.lang.String value)

This method will add a property. Once again an IllegalStateException will be thrown if the key being passed in is null.

java.lang.String encodeURL(java.lang.String path)

This method should be used to encode URLs when including other resources in the portlet like images, servlets, JSPs, and other files. The returned String will be the encoded URL required by portal. If the URL does not need to be encoded then the original String will be returned.

The path for this method must be either an absolute URL:

```
renderResponse.encodeURL(“http://localhost:9081/wps/myportal/myportlet/
images/myimage.gif”);
```

or a root relative path using the getContextPath method of the RenderRequest object.

Example 8-11 Code to get the root relative path

```
<%= renderResponse.encodeURL(renderRequest.getContextPath() +
"/images/myimage.gif") %>
```

8.5.9 interface javax.portlet.ActionResponse

The ActionResponse interface extends the PortletResponse interface. The ActionResponse is used during an action request and is passed into the processAction method. ActionResponse includes added functionality for changing the portlet mode, changing the portlet window, setting render parameters, and redirecting to another URL. The ActionResponse object is only in scope during the processAction method.

void sendRedirect(java.lang.String location)

This method redirects the user to the specified URL. This method will only accept an absolute URL or a root relative URI. An IllegalArgumentException is thrown if a relative URL is passed in. If this method is called after setRenderParameter, setRenderParameters, setwindowState, or setPortletMode then an IllegalStateException is thrown and the redirection is not executed.

void setPortletMode(PortletMode portletMode)

This method is used to change the portlet mode. If a mode is selected that is not supported by the portlet then a PortletModeException is thrown. You can use the isPortletModeAllowed method of the PortletRequest object to determine if this is a supported portlet mode. If this is called after the sendRedirect method an IllegalStateException is thrown. Please see “JSR 168 portlet modes” on page 254 for more information about portlet modes.

void setWindowState(WindowState windowState)

This method is used to change the window state of the portlet. If a window state is selected that is not supported then a WindowStateException will be thrown. You can use the isWindowStateAllowed method of the PortletRequest object to determine if the window state is supported. If this is called after the sendRedirect method then an IllegalStateException is thrown. Please see “JSR 168 Portlet window states” on page 256 for more information.

void setRenderParameter(java.lang.String key,java.lang.String value)

Parameters set using this method will be used in all subsequent render requests until the portlet is targeted with a new request. If there are no parameters set using this method then there will be no render parameters for the RenderRequest.

This method is overloaded for render parameters that contain a String[] (String array) for the values.

void setRenderParameter(java.util.Map parameter)

This method will have the same function as the method listed above. This method accepts a Map of key value pairs where the key must be of type String and the value must be of type String[] (String array).

8.5.10 interface javax.portlet.RenderResponse

The RenderResponse interface extends the PortletResponse interface. This object is passed to the render method. This object is used to set the title of the portlet and generate content by either obtaining a writer or delegating to a JSP or Servlet. The scope of this object is only for the render method.

Some important method of the RenderResponse interface. Please see the JSR 168 API for a complete listing of methods.

void setContentType(java.lang.String type)

This method is used to set the content type. An IllegalArgumentException will be thrown if the type is not a valid type return by the getResponseContentType method of the PortletRequest object.

If the getWriter or getPortletOutpuStream are called before this method then an IllegalStateException is thrown.

java.io.PrintWriter getWriter()

This method should be used for writing content to the response. The setContentType should be called before this method. An IllegalStateException will be thrown if the getPortletOutputStream has already been called.

java.io.OutputStream getPortletOutputStream()

This method is also used for generating content to the response. Like the getWriter, the setContent should be called before this method. An IllegalStateException will be thrown if the getWriter has already been called.

int getBufferSize()

The buffer size for the response is returned. A zero will be returned if no buffer is used.

void setBufferSize(int size)

This method will set a requested buffer size for the response body. The buffer size will be at least as large as the requested size. It could be larger. This method needs to be called before any content is written.

Archived

void reset()

This will clear all the content on the buffer and the properties that were set. This will throw an IllegalStateException if the response has already been committed. You can use the isCommitted() method to determine if any bytes have been written to the client.

void resetBuffer()

This method is the same as the reset, however this will not clear the properties. This will also throw an IllegalStateException if the response is committed. You can use the isCommitted() method to determine if any bytes have been written to the client.

void flushBuffer()

Invoking this method will force the content in the buffer to be written to the client. The response is considered to be commited after a call to this method.

boolean isCommited()

Returns true if the response has been committed else it returns false.

void setTitle(java.lang.String title)

This method is used for setting the title of the portlet. Currently WebSphere Portal does not support this method.

java.lang.String getNamespace()

This method returns the namespace of the portlet. This String should be prefixed to elements to make them unique on a page. JavaScript variable, functions, and form fields should be prefix with the String returned by this method.

PortletURL createRenderURL()

Creates a URL pointing to the portlet. This URL will cause a render request.

Please see “interface javax.portlet.PortletURL” on page 261 for information about the PortletURL object.

PortletURL createActionURL()

Creates a URL pointing to the portlet. This URL will cause an action request.

Please see 8.5.3, “interface javax.portlet.PortletURL” on page 261 for information about the PortletURL object.

Archived

8.5.11 interface javax.portlet.PortalContext

The PortalContext will provide information about the portal server that is invoking the portlet. This object can be obtained from the PortletRequest object.

java.lang.String getPortalInfo()

This method returns information about the server

Example 8-12 String returned from the getPortalInfo() method

```
IBM WebSphere Portal/5.1
```

java.lang.String getProperty(java.lang.String name)

Returns the value of the portal property with the given name. Null will be returned if no property of that name is found.

java.util.Enumeration getPropertyNames()

Returns an Enumeration of all the portal property names. An empty Enumeration is returned if there are no portal properties.

java.util.Enumeration getSupportedPortletModes()

Returns an Enumeration of PortletMode objects that the portal server supports.

java.util.Enumeration getSupportedWindowStates()

Returns an Enumeration of WindowState objects that are supported by the portal server.

8.5.12 interface javax.portlet.PortletPreferences

The PortletPreferences object is used to provide a customized view of the portlet to the user. Preferences are stored as name value pairs. They can be either defined in the portlet deployment descriptor or programmatically. If the parameters are defined in the deployment descriptor, they have the option of being read only. Read only parameters can only be updated by the administrator while in Config mode. The administrator can also modify the parameters using the admin portlets. If the parameters are added programmatically they are not considered to be read-only. Users can modify parameters only while in the Edit mode and only parameters that are not read only. Changes made in Config mode by the administrator will affect all instances of the portlet on all pages.

Example 8-13 Preference parameter tag in the portlet deployment descriptor

```
<preference>
   <name>My Preference Parameter Name</name>
```


Archived

```
   <value>My Preference Parameter Value</value>
</preference>
```

java.util.Map getMap()

This method returns a Map of all the preferences. The keys will be of type String and the Values will be of type String[] (String array). An empty Map will be returned if there are no preferences.

java.util.Enumeration getNames()

Returns an Enumeration of all the preference keys that have a value. An empty Enumeration will be returned if there are no keys.

java.lang.String getValue(java.lang.String key, java.lang.Stringdef)

This method returns the first preference found matching the key parameter. If the preference has multiple values, only the first value will be returned. If there are no values found for this preference the default parameter passed in will be returned as the value. If the key is null then an IllegalArgumentException will be thrown.

java.lang.String[] getValues(java.lang.String key, java.lang.String[] def)

This method will have the same functionality as the method listed above with the exception that this is used when there are multiple values for the preference.

boolean isReadOnly(java.lang.String key)

Use this method to determine if the preference is read only. Only Admistrators in CONGIF mode can modify read only preferences. An IllegalArgumentException is thrown if the preference does not exist.

void reset(java.lang.String key)

This method will reset the preference to the default value if there is one specified. If there are no default values, this preference will be removed. If the preference cannot be modified, a ReadOnlyException will be thrown. An IllegalArgumentException will be thrown if the preference does not exist.

void setValue(java.lang.String key, java.lang.String value)

This method is used to modify the value for the preference. The value can be null but the key must be a valid String. An IllegalArgumentException will be thrown if

Archived

the key is null, the key is too long, or the value is too long. A ReadOnlyException will be thrown if the user does not have the right to modify this preference for this request.

void setValues(java.lang.String key, java.lang.String[] values)

This method has the same functionality as the method listed above with the exception that this method should be used when there are multiple values for the preference.

void store()

This method must be called for new or update preferences to be stored in persistence. This needs to only be called once for all of the values to be stored. Store does not have to be called for each preference. If store() does not throw an exception, it is assumed that the save completed. If the save cannot be completed in the backend, an IOException will be thrown. This method can also throw a ValidtorException if the parameter cannot be validated by the PreferenceValidator class specified in the deployment descriptor. See interface javax.portlet.PreferencesValidator for more information.

An IllegalStateException is thrown if this method is called inside the render method.

8.5.13 interface javax.portlet.PreferencesValidator

Portlets that wish to validate preferences should create a class that implements PreferencesValidator interface. This class should then be added to the portlet deployment descriptor.

Example 8-14 PreferenceValidator defined in the portlet deployment descriptor

```
<preferences-validator>myportlet.myportletPreferencesValidator
</preferences-validator>
```

The validate method of this class will be called by the store method. If the validate class throws a Validator exception the store will not complete and it will propagate the exception to the portlet.

void validate(PortletPreferences preferences)

If the preferences are not valid then this method should throw a ValidatorException. If the preferences are valid this method should finish gracefully. It is the responsibility of the portlet developer to add a error description to the ValidatorException if needed.

Archive

Example 8-15 Validate method generated by Rational Application Developer

```
public void validate(PortletPreferences preferences) throws ValidatorException
{
   Collection failedKeys = new ArrayList();
   for( Enumeration names=preferences.getNames(); names.hasMoreElements(); ) {
      String name = names.nextElement().toString();
      if( name.startsWith(".") ) continue;
      String value = preferences.getValue(name, "");
      //validates that the preferences do not start or end with white space
      //validates that the preferences start with http: or https:
      if( !value.equalsIgnoreCase(value.trim()) ||
      !(value.startsWith("http:")||value.startsWith("https:")) ) {
      failedKeys.add(name);
      }
      }
      if( !failedKeys.isEmpty() ) {
         throw new ValidatorException("One or more preferences do not comply
         + with the validation criteria",failedKeys);
   }
}
```

8.5.14 interface javax.portlet.PortletConfig

The portlet config object is used during initialization of the portlet. From the PortletConfig object you can obtain the PortletContext and the resource bundle used for setting the title. Unlike the PortletContext object, the PortletConfig will use the init parameters defined in the portlet.xml and not the init parameters defined in the web.xml.

The portlet may define some basic information in the deployment descriptor that is used for the title bar. The information may also be used for the categorization of the portlet. This information can be directly stored in the deployment descriptor or it may be stored in an external resource bundle. Resource bundles will be discussed in a later section but was addressed here to acknowledge that the portlet config provides access to this information.

java.lang.String getInitParameter(java.lang.String name)

This will return the named initialization parameters defined in the portlet deployment descriptor.

Example 8-16 Init parameter defined in the portlet deployment descriptor

```
<init-param>
   <name>My init param</name>
   <value>My init param value</value>
```

```
</init-param>
```

If the named parameter does not exist then null is returned. An IllegalArgumentException is thrown if the named parameter is null.

java.util.Enumeration getInitParameters()

Returns an Enumeration of Strings for all the initialization parameters defined by the portlet. An empty Enumeration is returned if there are no initialization parameters defined.

PortletContext getPortletContext()

Returns the PortletContext object. Please see “interface javax.portlet.PortletContext” on page 263 for more information about the PortletContext interface.

java.lang.String getPortletName()

This returns the name of the portlet as defined by the administrator or the name defined in the portlet deployment descriptor.

Example 8-17 Portlet name in the portlet deployment descriptor

```
<portlet-name>myPortlet</portlet-name>
```

java.lang.ResourceBundle getResourceBundle(java.util.Locale)

Returns the resource bundle for the locale specified. The resource bundle can be defined as external or it can be inline text in the portlet.xml.

8.5.15 interface javax.portlet.PortletSession

The portlet session object provides session tracking for all requests coming from the same client. All portlets of the same portlet application will share a portlet session for that user. Portlets of other portlet application will not be able to access this session object and therefore will not be able to share attributes.

PortletSession provides two static variables to be used to when adding attributes. The two variables representing the scopes are APPLICATION_SCOPE and PORTLET_SCOPE.

Objects stored in the APPLICATION_SCOPE are shared among all portlets of the same portlet application. Attributes stored as PORTLET_SCOPE are available only to the portlet, although the attribute is not completely hidden it is

Archived

namespaced to the specific portlet. The PortletSessionUtil class should be used to decode portlet attributes stored in the PORTLET_SCOPE when called through the HttpSession of the HttpSessionBindingListener. Please see “Listeners” on page 284 for more information.

The PortletSession and the HttpSession can share attributes. This means that servlets and JSPs will have access to attributes stored by the PortletSession at APPLICATION_SCOPE. The PortletSession will also have access to attributes stored by servlets and JSPs.

Example 8-18 Sharing session attributes

```
Code in the portlet
request.getPortletSession().setAttribute("name", "Hailey Anne",
PortletSession.APPLICATION_SCOPE);
PortletRequestDispatcher rd =
getPortletContext().getRequestDispatcher("/MyServlet");
rd.include(request,response);

Code in MyServlet
String name = request.getSession().getAttribute("name");
```

The following methods of the PortletSession are based on the HttpSession:

- getCreationTime
- getId
- getLastAccessedTime
- getMaxInactiveInterval
- invalidate
- isNew
- setMaxInactiveInterval

The following methods of PortletSession are also based on HttpSession methods:

- getAttribute
- setAttribute
- removeAttribute
- getAttributeNames

PortletSession does overload the previous methods to provide access for the different storage scopes of the portlet session (PORTLET_SCOPE and APPLICATION_SCOPE).

java.lang.Object getAttribute(java.lang.String name, int scope)

Returns the named object bound in session to the given scope. If no object is found, null is returned.

Archived

java.lang.Enumeration getAttributeNames(int scope)

Returns an Enumeration of type String that contains all the attribute names of the specified scope.

void removeAttribute(java.lang.String name, int scope)

Removes the named attribute from the specified scope.

void setAttribute(java.lang.String name, java.lang.Object value, int scope)

This method will bound the named object to the specified scope. If an attribute with the given name already exist in the scope it is updated with the new object. If the object implements HttpSessionBindingListener is will be notified. This will be discussed further in the listeners section.

Note: Attribute names starting with javax.portlet are reserved and should not be used.

8.6 JSR 168 Portlet life cycle

The life cycle of a portlet closely resembles the life cycle of a servlet. There are, however, some differences. The following section will describe the life cycle of a portlet from beginning to end.

WebSphere portal server will mange the life cycle of the portlet by using the methods of the Portlet class.

8.6.1 Instantiation

The portlet will be instantiated during startup. The portlet will be loaded using the same classloading that is used for loading the Web application.

8.6.2 Initialization

The init() method of the Portlet class is called when the portlet is first requested. Portal will pass a unique Portletconfig object to the init method. During initialization any one time setup for the portlet should be completed. This includes expensive operations such as database connections.

There are two exceptions that may be thrown during initialization. If either exception is thrown, the portlet will not be placed into service and the destroy() method will not be called.

Archived

UnavailableException

The UnavalableException may indicate a specific time out value. If WebSphere portal server wishes to attempt to place this portlet back into service then it must at least wait for the specified time to expire.

PortletException

All RuntimeException thrown during initialization are treated as PortletExceptions. If PortletException is thrown the portal server does not need to wait any amount of time before attempting to place the portlet into service.

8.6.3 Request handling

JSR 168 portlets use a two phase processing model. The phases are split between a render phase and an action phase.

The portlet can generate a link to itself by using a PortletURL object. There are two types of PortletURL, ActionURL and a RenderURL. Using the RenderResponse object you can create a URL of either type or you can also use the JSP tags, actionURL and renderURL.

During request handling a portlet may throw one of the following exceptions

PortletException

This exception indicates that error in processing has occurred. All RuntimeExceptions are treated as a PortletException. If this is thrown from the processAction() method, all actions on the ActionResponse object will be ignored and the render method for this portlet will not be call during this request. The render method for all other portlets on the page will be invoked. If the portlet is cached, the cached version may be returned instead of invoking the render method.

PortletSecurityException

This exception indicates that the user does not have the proper rights for the portlet.

UnavailableException

This exception indicates that the portlet is unavailable to process the request. Like in the init method, the UnavailableException may indicate a time value that the portlet will be unavailable for.

Render phase

A RenderURL will cause the portlet to enter the render phase, this is also known as a render request. The render phase is used for displaying content in the

portlet. During a render request, the render method for all portlets on the page will be invoked, unless the portlet is cached. The render method may not be call and the cached content of the portlet could be returned. The processAction method will not be invoked during a render request. The render phase is also entered when a request is made for the page that the portlet resides.

Action phase

An actionURL will cause the portlet to enter the action phase, also known as action request. The action phase is typically used to modify the state of the portlet or to process a task. During the action phase the processAction() method of the portlet is called. After the process action completes, the render method is invoked for all portlets on the page, unless the portlet is cached. The render method may not be called and the cached content of the portlet could be returned. Action parameters are passed into the processAction method via the ActionRequest object. These parameters can represent a field in an HTML form or they can be parameters explicitly set by the programmer.

While in the action phase, the portlet mode and the window state may be changed. Although they can be programmatically changed, the portal server can override the change for any reason including the user may not have the required rights. The user may also be redirected to another URL during the action phase by using a request dispatcher.

Action parameter are not passed to the render phase unless they are explicitly specified. You should use the setRenderParameter or setRenderParameters method of the ActionResponse to pass parameters to the render phase. The parameters will be out of scope after the render method completes.

8.6.4 End of service

When WebSphere portal server is removing the portlet from service, the destroy() method of Portlet is invoked. This gives the portlet an opportunity to clean up resources obtained in the init method and save the persistent state. WebSphere portal server will wait until all threads serving a request for this portlet have completed. If this does not occur in a timely manner the portlet will be removed from service. The destroy method may throw a RuntimeException. If this happens the destroy is considered to have completed successfully and the portlet is removed from service.

Once the portlet is removed, the portlet object is eligible for garbage collection. There cannot be any further requests for this portlet unless a new portlet object is instantiated.

8.7 Portlet caching

Caching improves performance by reducing the load on the server. Caching is per user per portlet, a portlet cached for one user will not be used for another user. A JSR 168 portlet must use the portlet.xml to enable caching.

Example 8-19 Portlet caching in the portlet.xml

```
<portlet>
   ...
      <expiration-cache>600</expiration-cache>
   ...
</portlet>
```

The expiration cache is defined in seconds. In the example above, the contents of the portlet should be cached for 10 minutes. A value of zero indicates that the portlet caching is disabled. A -1 value will cause the portlet cache to never expire. Any value above 0 will be interpreted as the number of seconds the content should be cached for.

If the expiration-cache is defined in the portlet.xml, the portlet may programmatically change the expiration time. The request property *expiration-cache* should be used to modify the caching values.

Example 8-20 Programmatically changing the expiration time-out value

```
RenderResponse.setProperty(
    PortletResponse.EXPIRATION_CACHE,
    (new Integer(600)).toString() );
```

While the portlet content is cached and the portlet is not a target of a request, the cached content will be used during the render phase. If there is a request for the portlet, the cached content will be disregarded and the render method of the portlet will be invoked.

8.7.1 Remote cache

JSR 168 portlets can indicate how a page is cached on a remote proxy server.

The ibm-portlet-portal-ext.xmi will need to be modified in order to enable remote portlet caching. The <remote-cache-scope> element is used to specify if the cache is SHARED or NON_SHARED.

Example 8-21 ibm-portlet-portal-ext.xmi

```
<?xml version="1.0" encoding="UTF-8"?>
<portlet-app xmlns="http://www.ibm.com/xml/ns/portlet/portlet-app_1_0_ext.xsd"
```

Archived

```
version="1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ibm.com/xml/ns/portlet/portlet-app_1_0_ext.xsd
http://www.ibm.com/xml/ns/portlet/portlet-app_1_0_ext.xsd" >
  <anonymous-session>true</anonymous-session>
  <!-- Use the portlet name in the portlet.xml for the href -->
  <portlet href="PORTLET_NAME_FROM_PORTLET_XML">
    <remote-cache-scope>SHARED</remote-cache-scope>
    <remote-cache-dynamic>true</remote-cache-dynamic>
  </portlet>
</portlet-app>
```

Portlets can get a persistent HttpSession object on an unauthenticated page by specifying true in the <anonymous-session> element. With a value of false, the HttpSession is lost after each request. The default value for the anonymous-session element if false if the element is not present.

You can also specify the setting <remote-cache-dynamic>. This element determines if the portlet will publish remote cache information. A setting of false will improve performance by indicating that portal does not need to wait for the remote cache to be published.

Like local cache, you can also programmatically modify the setting for remote cache.

Example 8-22 Programmatically modifying remote cache settings

```
RenderResponse.setProperty( RemoteCacheInfo.KEY_SCOPE, “SHARED”);
RenderResponse.setProperty( RenderResponse.EXPIRATION_CACHE, (new
Integer(600)).toString());
```

8.8 Listeners

JSR 168 portlets do not define any portlet listeners, instead they use servlet listeners. Servlet listeners should be implemented in the portlet environment like they are in the servlet environment. The listeners should be added in the web.xml or to any object that needs to be informed when being added or removed from the session should implement HttpSessionBindingListener. Detailed description of servlet listeners are outside the scope of this book, but we will briefly describe them here.

8.8.1 HttpSessionBindingListener

Object that require notification when being added or removed from the portlet session should implement this interface. This interface defines two methods,

valueBound and valueUnbound. Both of these methods are passed a HttpSessionBindingEvent object. From the HttpSessionBindingEvent object you can get access to the name and value of the object that has been added or removed from the session. You can also get the session object. When using attribute that are store in the portlet scope, you should use the PortletSessionUtil class to decode the attribute names.

8.8.2 ServletContextListener

The ServletContextListener can be used for the PortletContext. This listener contains contextInitialized and contextDestroyed methods. An object of type ServletContextEvent is passed into the initialized and destroyed methods. From this object you can get access to the ServletContext. The contextInitialized method will be called when the PortletContext is created and contextDestroyed will be call when the PortletContext is destroyed.

8.8.3 ServletContextAttributeListener

This interface defines three methods, added, removed and replaced. These methods are to be used when attributes have been added, removed, and replaced. A ServletcontextAttributeEvent is passed to these three methods. From this object you can get the name and value of the attribute that has been added, updated or removed.

8.8.4 HttpSessionListener

The HttpSessionListener provides two methods sessionCreate and sessionDestroyed. These are invoked when the session has been created and destroyed. A HttpSessionEvent object is passed into these method. From the HttpSessionEvent you can get the HttpSession.

8.8.5 HttpSessionAttributeListener

Like the ServletContextAttributeListener, the HttpSessionAttributeListener defines the same three methods attributeAdded, attributeRemoved, and attributeReplaced. These methods are pass a HttpSessionBindingEvent object. You can get the name and value of the attribute that has been added, removed or replaced.

The following examples will show the web.xml, the listener class and a bean implementing the HttpSessionBindingListener.

Example 8-23 web.xml

```
<web-app id="WebApp_ID">
```

```
...
   <listener>
      <listener-class>listener.MyListeners</listener-class>
   </listener>
...
</web-app>
```

Example 8-24 Class implementing ServletContextListener, ServletContextAttributeListener,HttpSessionListener, and HttpSessionAttributeListener

```
public class MyListeners implements ServletContextListener,
   ServletContextAttributeListener, HttpSessionListener,
   HttpSessionAttributeListener {
//ServletContextListener methods.
public void contextInitialized(ServletContextEvent sce) {
   System.out.println("Servlet Context initialized.");
}
public void contextDestroyed(ServletContextEvent sce) {
   System.out.println("Servlet Context destroyed");
}
//ServletContextAttributeListener methods.
public void attributeAdded(ServletContextAttributeEvent scae) {
   System.out.println("Servlet context attribute added: " + scae.getName());
}
public void attributeRemoved(ServletContextAttributeEvent scae) {
   System.out.println("Servlet context attribute removed: " + scae.getName());
}
public void attributeReplaced(ServletContextAttributeEvent scae) {
   System.out.println("Servlet context attribute replaced: " +
   scae.getName());
}

//HttpSessionListener methods.
public void sessionCreated(HttpSessionEvent hse) {
   System.out.println("HttpSession created");
}
public void sessionDestroyed(HttpSessionEvent hse) {
   System.out.println("Httpsession destroyed.");
}

//HttpSessionAttributeListener methods.
public void attributeAdded(HttpSessionBindingEvent hsbe) {
   System.out.println("HttpSession added: " + hsbe.getName());
}
public void attributeRemoved(HttpSessionBindingEvent tbia) {
   System.out.println("HttpSession attribute removed: " + tbia.getName());
}
public void attributeReplaced(HttpSessionBindingEvent hsbe) {
   System.out.println("HttpSession attribute replaced: " + hsbe.getName());
```

Archived

```
}
}
```

Example 8-25 Bean implementing HttpSessionBindingListener

```
public class SessionBean implements HttpSessionBindingListener {
   public void valueBound(HttpSessionBindingEvent hsb) {
      System.out.println("SessionBean added to HttpSession");
   }
   public void valueUnbound(HttpSessionBindingEvent hsb) {
      System.out.println("SessionBean removed from HttpSession");
   }
}
```

8.9 Deployment descriptors

Deployment descriptors provide information to the server about the application. Two deployment descriptors are required for portlets, web.xml and portlet.xml. The web.xml (Web deployment descriptor) will be used to define all non portlet resources. Compared to the IBM portlet API, JSR 168 does not define a servlet so you do not have to define a servlet in the web.xml. Remember that in JSR 168 the portlet do not extend the HttpServlet interface like they do in the IBM API. Therefore portlets are not servlets and are not required to go into the web.xml. You can, however, use this to define other resources. The following items would be set in the web.xml.

- <description> tag should contain the portlet application description
- <display-name> tag should contain the portlet application name
- <servlet> tag should be used to define any new servlets.
- <security-role> will define security roles for the portlet security role mapping

Example 8-26 Web deployment descriptor

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app id="WebApp_ID">
   <display-name>My Portlet</display-name>
    <description>This is my portlet</description>
   <welcome-file-list>
      <welcome-file>index.html</welcome-file>
      <welcome-file>index.htm</welcome-file>
      <welcome-file>index.jsp</welcome-file>
      <welcome-file>default.html</welcome-file>
      <welcome-file>default.htm</welcome-file>
      <welcome-file>default.jsp</welcome-file>
```

```
   </welcome-file-list>
</web-app>
```

Please see the servlet specification for complete details on the Web deployment descriptor.

The portlet.xml is used to define all portlet related resources. You should use Rational Application Developer to create and modify your portlet.xml. Rational Application Developer contains tools for inserting and removing elements and also to verify that the portlet.xml is valid. We will discuss each of the elements in the portlet.xml to give a complete understanding.

8.9.1 Portlet.xml declaration

All portlet deployment descriptors need to declare the xml version and encoding.

<?xml version="1.0" encoding="UTF-8"?>

8.9.2 portlet-app - *required, can occur only once*

The portlet-app element is the root element for the portlet deployment descriptor. This has a required attribute “version” that specifies the version of the xml schema that this portlet.xml conforms with.

Example 8-27 portlet-app tag. This should all appear on one line

```
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
version="1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd
http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd">
```

Attributes

- version -

  This attribute is required.

- id -

  This attribute is optional.

Elements

(The elements are required to be listed in the order they are presented)

- **portlet**

  Can occur zero or more times. This element defines the portlet. The portlet name must be unique within the portlet application.

Archived

- **custom-portlet-mode**

  Can occur zero or more times. This element defines any custom portlet modes for the portlet application. The name of the portlet-mode defined must be unique for the portlet application.

- **custom-window-state**

  Can occur zero or more times. This element defines any custom window states for this application. The name of the window-state must be unique within the portlet application.

  Note: WebSphere Portal does not provide support for any custom window states at this time.

- **user-attributes** -

  Can occur zero or more times.

- **security-constraint** -

  Can occur zero or more times.

  Note: WebSphere portlet does not currently support the security-constrain tag.

8.9.3 portlet - *can occur zero or more times*

The portlet tag contains information used for initializing the portlet.

Example 8-28 portlet tag in the portlet.xml

```
<portlet>
      <portlet-name>MyPortlet</portlet-name>
      <display-name>MyPortlet portlet</display-name>
      <display-name xml:lang="en">MyPortlet portlet</display-name>
      <portlet-class>myportlet.MyPortletPortlet</portlet-class>
      <init-param>
         <name>wps.markup</name>
         <value>html</value>
      </init-param>
      <expiration-cache>0</expiration-cache>
      <supports>
         <mime-type>text/html</mime-type>
         <portlet-mode>view</portlet-mode>
         <portlet-mode>edit</portlet-mode>
         <portlet-mode>help</portlet-mode>
         <portlet-mode>config</portlet-mode>
      </supports>
```

Archived

```
   <supported-locale>en</supported-locale>
   <resource-bundle>myportlet.nl.MyPortletPortletResource</resource-bundle>
   <portlet-info>
      <title>MyPortlet portlet</title>
   </portlet-info>
   <portlet-preferences>
      <preference>
         <name>.MyPortletPortletEditKey</name>
         <value>Read-Write Value</value>
      </preference>
      <preference>
         <name>.MyPortletPortletConfigKey</name>
         <value>Read-Only Value</value>
         <read-only>true</read-only>
      </preference>
   </portlet-preferences>
</portlet>
```

Attributes

- **id**

  The id attribute is optional.

Elements

- **description -** *can occur zero or more times*

  Attribute:

  - **xml:lang**

    This attribute indicates the language used in the description

  This element is used in describing the portlet. This should include information about the portlet.

- **portlet-name -** *must occur only once*

  The portlet name must be unique within the portlet application. The portlet-name can only occur once for each portlet element.

- **portlet-class -** *must occur only once*

  The portlet class must be a fully qualified class or interface. The portlet-class can only occur once for each portlet element.

- **init-param** - *can occur zero or more times*

  Attribute:

  - **id**

    The id attribute is optional.

Archived

Elements:

– **description -** *can occur zero or more times*

Attributes:

• **xml:lang**

This attribute indicates the language used in the description

This element is used to describing the initialization parameters.

– **name** - *must occur only once*

The name element specifies the name of the initialization parameter

– **value** - *must occur only once*

The value element specifies the value for this initialization parameter

► **expiration-cache** - *can occur either zero or one time*

This elements defines caching for this portlet. The parameter indicates the time in seconds after which the portlet output expires. -1 indicates that the output never expires.

► **supports** - *must occur one or many times*

All portlets are required to support View mode. The supports element must occur at least once, containing the View mode. WebSphere portlet will only support the following modes View, Edit, Help, and Configuration. Configuration is the only custom mode supported by WebSphere Portal.

Attribute:

– **id**

The id attribute is optional.

Elements:

– **mime-type** - *must occur only once*

This element is used to define the MIME type for the portlet mode. The MIME type may contain wildcard character '*', like "text/*" or "*/*, or it can explicitly define the type like "text/html".

– **portlet-mode** - *can occur zero or more times*

Since all portlets must support View mode it is not required to add it. However, it is recommended that it is added for readability. The portlet-mode must be a valid mode.

► **supported-locale** - *can occur zero or more times*

Indicates the locale that this portlet supports.

Archived

You can choose to have a resource bundle with zero or one portlet-info elements. You can only define one resource-bundle for each portlet application. If you do have a resource bundle with the inline portlet-info element, portal will look for the value in the resource bundle first. If it does not find the value there but finds it inline, it will add the inline element to the resource bundle.

- **resource-bundle** - *must occur only once*

  This element will contain the filename for the resource bundle. The filename should be the fully qualified class name and it should be located in /WEB-INF/classes/nls subdirectory of the WAR. All of the locale-specific resource bundles append the locale to the filename (for example, MyPortlet_es.properties for Spanish).

  To correctly display the portlet title, there must be a corresponding supported-locale element for each resource bundle locale. If the locale is missing from the supported-locale element, the default locale will be used for the portlet title.

  There are three constrains for the resource bundle

  – javax.portlet.title

    This is the title of the portlet to be displayed in the title bar. This may be overridden programmatically of by the administrator.

  – javax.portlet.short-title

    This is a short title for the portlet. This is used in cases where device cannot display the full title.

  – javax.portlet.keywords

    Keywords may be used by portal when searching for portlets. The keywords must be separated by commas (‘,’).

Example 8-29 Resource bundle in portlet.xml

```
<resource-bundle>nls.SamplePortlet</resource-bundle>
```

Example 8-30 Resource bundle examples

```
# English Resource Bundle
#
# Filename: SamplePortlet.properties
javax.portlet.title=Sample Portlet
javax.portlet.short-title=sample
javax.portlet.keywords=sample,test

# Spanish Resource Bundle
#
```

Archive

```
# Filename: SamplePortlet_es.properties
javax.portlet.title=EjemploMi Portlet
javax.portlet.short-title=ejemplo
javax.portlet.keywords=ejemplo, prueba
```

- **portlet-info** - *can occur only once*

  The portlet info is used to provide the title, short-title, and keywords for the default locale.

  Attribute:

  - **id**

    The id attribute is optional.

  Elements:

  - **title -** *must occur only once*

    This is the title of the portlet to be displayed in the title bar. This may be overridden programmatically of by the administrator.

  - **short-title** - *can occur zero or one time*

    This is a short title for the portlet. This is used in cases where device cannot display the full title.

  - **keywords** - *can occur zero or one time*

    Keywords may be used by portal when searching for portlets. The keywords must be separated by commas (‘,’).

  If the resource bundle does not exist, the portlet-info must occur only once.

- **portlet-info** - *can occur only once*

  The portlet info is used to provide the title, short-title, and keywords for the default locale.

  Attribute:

  - **id**

    The id attribute is optional.

  Elements:

  - **title -** *must occur only once*

    This is the title of the portlet to be displayed in the title bar. This may be overridden programmatically of by the administrator.

  - **short-title** - *can occur zero or one time*

    This is a short title for the portlet. This is used in cases where device cannot display the full title.

Archivo

– **keywords** - *can occur zero or one time*

Keywords may be used by portal when searching for portlets. The keywords must be separated by commas (‘,’).

- **portlet-preferences** - *can occur zero or more times*

The portlet-preferences element will store the portlet preferences. Portlet preferences are used to store persistent data for the user.

Attribute:

– **id**

The id attribute is optional.

Elements:

– **preference** - *can occur zero or more times*

The preference element will store the name and value pairs for the preferences. Please refer to “interface javax.portlet.PortletPreferences” on page 274 for more information.

Attribute:

• **id**

The id attribute is optional.

Elements:

• **name** - *must occur once*

The name element represents the name of the preference.

Example 8-31 Using the preferences in the portlet.xml and code

```
<portlet-preferences>
   <preference>
      <name>test</name>
      <value>value for test preference</value>
      <read-only>true</read-only>
   </preference>
</portlet-preferences>

The following code will retrieve the value for test.

PortletPreferences preferences = RenderRequest.getPreferences();
String value = preferences.getValue("test", "No value found");
```

• **value** - *can occur zero or more times*

You can have multiple values for the preference. Multiple values will be returned as a String[] (String array).

- **read-only** - *can occur zero or one time*

  Read only will take a boolean value of either true or false. True indicates that this preference can only be updated by the administrator during Configuration mode.

– **preference-validator** - *can occur zero or one time*

  The preference-validator class must use the fully qualified class name. This class must implement javax.portlet.PortletValidator. The validate() method will be invoked during a call to the store() method of the PortletPreference class.

► **security-role-ref** - *can occur zero or more times*

Is used in conjunction with the web.xml.

Note: WebSphere portal version 5.1 will ignore all security-role-ref tags as they are currently not supported.

Attribute:

– id

  This id attribute is optional.

Elements:

– **description** - *can occur zero or more times*

  Attribute:

  - **xml:lang**

    This attribute indicates the language used in the description.

  This element is used to describing the security role references.

– **role-name** - *must only occur once*

  This defines the name for the security role.

– **role-link** - *can occur zero or one time*

  The role-link element refers to a defined security role. The role-link element must contain the name of one of the security roles defined in the security-role elements of the web.xml.

8.9.4 custom-portlet-mode - *can occur zero or more times*

The custom-portlet-mode is used to define modes for the portlet. Currently WebSphere portal only supports the Configuration custom mode.

Archived

Attribute

- **id**

  The id attribute is optional.

Elements

- **description -** *can occur zero or more times*

  Attributes:

  - **xml:lang**

    This attribute indicates the language used in the description.

  This element is used to describing the custom portlet mode.

- **portlet-mode** - *must occur only once*

  The portlet-mode names are not case sensitive. They must be unique within the portal application.

8.9.5 custom-window-state - *can occur zero or more times*

The custom-window-state is used to define window states for the portlet. Currently WebSphere portal does not supports any custom window states.

Attribute

- **id**

  The id attribute is optional.

Elements

- **description -** *can occur zero or more times*

  Attribute:

  - **xml:lang**

    This attribute indicates the language used in the description

  This element is used to describing the custom window.

- **window-state**- *must occur only once*

  The window-state names are not case sensitive. They must be unique within the portal application.

8.9.6 user-attribute - *can occur zero or more times*

Portlets may need to behave differently based on the user. The user attributes allow you to access user attributes. The following table shows the user attributes

Archived

supported by WebSphere portlet. The second column shows the corresponding attribute name in Member Manager (wmmAttributeName).

Table 8-1 Supported user attributes

User attribute name	Member Manager equivalent
user.gender	ibm-gender
user.employer	o
user.department	ou
user.jobtitle	ibm-jobTitle
user.name.prefix	ibm-personalTitle
user.name.given	givenName
user.name.family	sn
user.name.middle	ibm-middleName
user.home-info.telecom.telephone.number	homePhone
ser.home-info.online.email	u ibm-otherEmail
user.business-info.postal.street	street
user.business-info.postal.stateprov	stateOrProvinceName
user.business-info.postal.postalcode	postalCode
user.business-info.postal.country	countryName
user.business-info.telecom.telephone.number	telephoneNumber
user.business-info.telecom.fax.number	facsimileTelephoneNumber
user.business-info.telecom.mobile.number	mobile
user.business-info.telecom.pager.number	pager
user.business-info.telecom.online.email	ibm-primaryEmail

A map containing all of the user attribute information can be retrieved using a call to the getAttribute of the request object, see Example 8-32 on page 298. If the user is unauthenticated then a null will be returned. If, however, the user is authenticated and the there are no user attributes defined, an empty Map will be returned. The Map will contain the user attributes mapped during deployment and they will be in the form of a String name value pair. Security must be enabled to retrieve most of the user information.

Archived

Example 8-32 Getting user attribute information.

```
Map userinfo = (Map)request.getAttribute(RenderRequest.USER_INFO);
       if(userinfo != null){
          for (Iterator it=userinfo.entrySet().iterator(); it.hasNext(); ) {
               Map.Entry entry = (Map.Entry)it.next();
               String key = (String)entry.getKey();
               String value = (String)entry.getValue();
               System.out.println("Name: " + key + ", Value: " + value);
           }

       }
```

Attribute

- id

  The id attribute is optional.

Elements

- **description -** *can occur zero or more times*

  Attribute:

  - **xml:lang**

    This attribute indicates the language used in the description

  This element is used to describing the user attributes.

- **name** - *must occur only once*

  Defines the name of the user attribute.

8.9.7 security-constraint - *can occur zero or more times*

Once again, this tag is not supported by WebSphere portal at the time of this writing. Please consult the WebSphere Portal Infocenter for more information.

Attribute

- id

  The id attribute is optional.

Elements

- **display-name** - *can occur zero or more times*

  Rational Application Developer will use the display-name to name this security constraint in the IDE. This name does not have to be unique.

Archived

Attribute:

– **xml:lang**

This attribute indicates the language used in the description

- **portlet-collection** - *must occur only once*
  – **portlet-name** - *can occur once or many times*

  The portlet name must be a valid portlet defined in the portal application.
- **user-data-constraint -** *must occur only once*

  Attribute:

  – **id**

  The id attribute is optional.

  Elements:

  – **description -** *can occur zero or more times*

  Attribute:

  • **xml:lang**

  This attribute indicates the language used in the description.

  This element is used to describe the user data constraints.

  – **transport-guarantee** - *must occur only once*

  The transport-guaranteeType specifies the type of communication between client and portlet. This value can only be one of the following, NONE, INTEGRAL, or CONFIDENTIAL.

  NONE indicates the portlet does not require guarantees for the transport. INTEGRAL indicates that the data between the portlet and client cannot be changed while in transit.

  CONFIDENTIAL indicates the data cannot be observed while in transit. This ensures that the data cannot be observed by other parties.

  INTEGRAL or CONFIDENTIAL usually indicate SSL is required.

8.10 JSR 168 limitations in WebSphere Portal

The following limitations apply in this release of WebSphere Portal:

- The JSR 168 specification does not define portlets filters. The portlet filter provided as a portal service in WebSphere Portal cannot be used with JSR 168 portlet.

Archived

- The setsecure() method of the PortletURL is part of the PortletURL interface, However, it is not supported by WebSphere Portal. The security level will always be same as the current request.
- The isUserInRole() method of the portlet request at the time of this writing this is unsupported and will always return false. All <security-role-ref> elements in the portlet.xml are ignored.
- The getUserPrincipal() method of the portlet request always returns null if application server security is not enabled.
- The getRemoteUser method will return the user who is logged in. If the user is not logged in, null will be returned. Even with security disabled this method will return the user information, although with security disabled the returned name may not be human-readable. With security enabled a more readable user name will be returned.
- The <security-constraint/> element in the portlet.xml is not supported at the time of this writing.
- User information supported a limited number of user attributes. Please see “user-attribute - can occur zero or more times” on page 296 for more information.
- RenderResponse.setTitle(java.lang.String title) method is currently not supported.

Archived

9

JSR 168 portlet development

In this chapter, you will use Rational Application Developer to create a JSR 168 portlet. The following tasks are included:

- Use the wizard to create a JSR 168 portlet project
- Examine the generated portlet
- Update the generated portlet
- Deploy the portlet to run on the Portal Test Environment
- Execute the portlet

Note: The portlet application described in this chapter has the following characteristics:

- Portlet API: JSR 168
- Application type: MVC

© Copyright IBM Corp. 2005. All rights reserved.

9.1 Overview

The JSR 168 sample portlet shown in this chapter will allow you to implement the following:

- You will enter your name and select the color that it is displayed in.
- You will also be able to add new colors to use.
- A user with administration rights will be able to change the greeting that appears before your name.
- You will use the PortletPreferences object to store the colors and greeting.
- A JavaBean, stored in the PortletSession, will store the currently selected color, name, and greeting.
- The PortletPreferencesValidator will be used to validate the color before adding it to the portletpreferences object.
- The portlet will show the render and action phases of the JSR 168 portlet API.
- It will also display the View, Edit, Config, and Help modes.

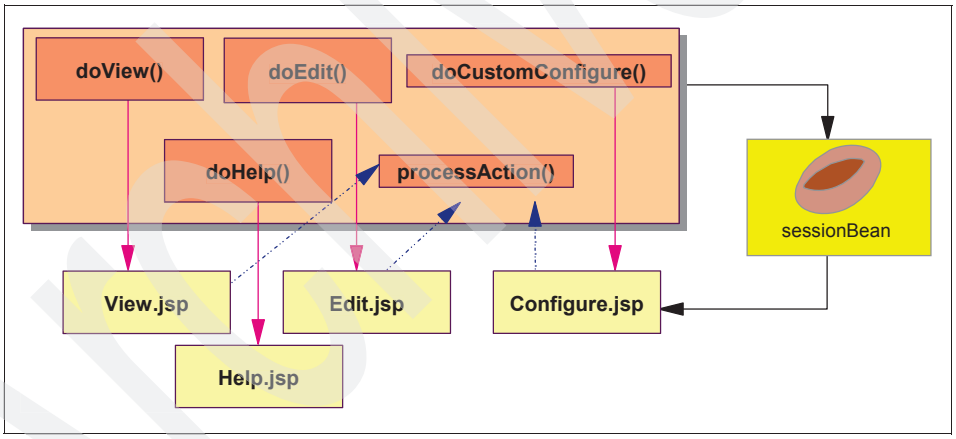


Figure 9-1 JSR 168 sample portlet

9.2 Creating a JSR 168 portlet project

In this section, you will create a new portlet project using Rational Application Developer. The portlet created by the wizard will be then manually updated for this scenario.

Archived

9.2.1 Creating a basic JSR 168 portlet

Execute the following steps to create the sample portlet:

1. Open Rational Application Developer if not already opened.
2. Select **File** → **New**
3. Select **Project**.
4. Select **Portlet Project (JSR 168)** to create a JSR 168 project.
5. An Auto Launch Configuration Change Alert window may pop up. If so, select **Yes** to continue.
6. If needed, select **OK** to confirm enablement of Portal Development tools.
7. Enter `MySimplePortlet` as the project name. Target server should be Portal V5.1. Select **Next**.

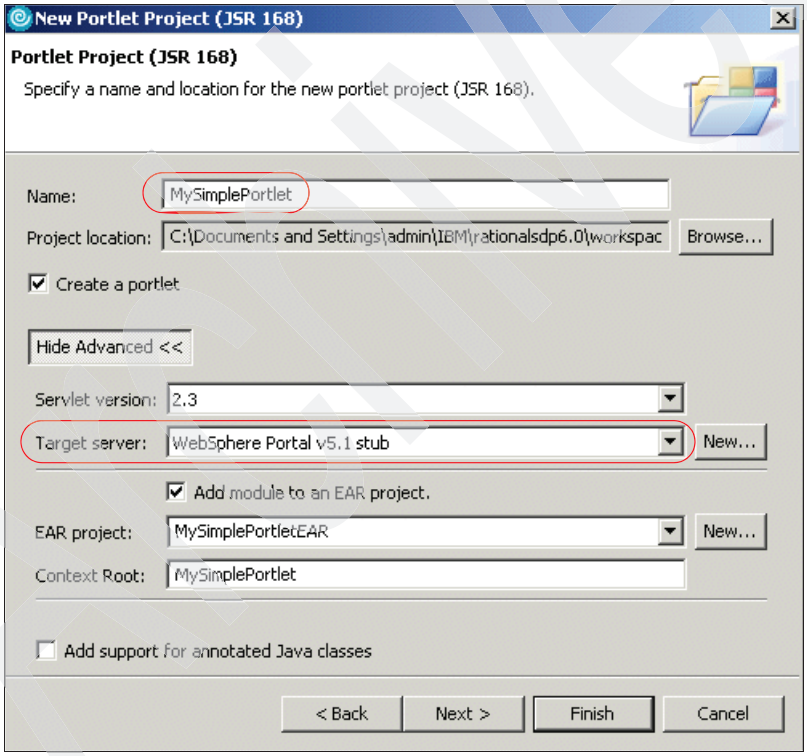


Figure 9-2 Enter the name and target server of your portlet

8. Select **Basic portlet (JSR 168)** and then click **Next**.

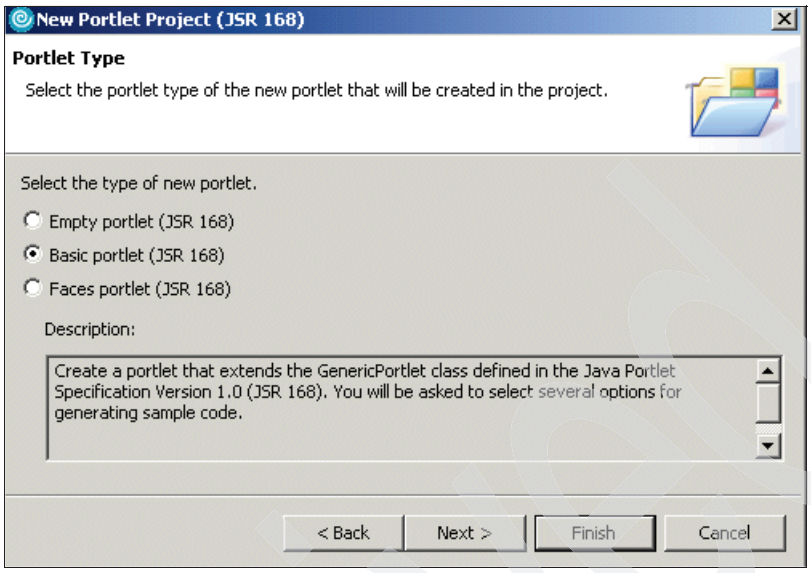


Figure 9-3 Selecting a basic JSR 168 portlet

9. Examine and take default values in the features window. Click **Next**.
10. Examine and take default values in the portlet settings window. Click **Next**.
11. In the Actions and Preferences window, select all four check boxes to add support for action handling and preferences handling. Notice that the Add Preferences validator box will be available when the Add preferences handling box is selected.

Archived

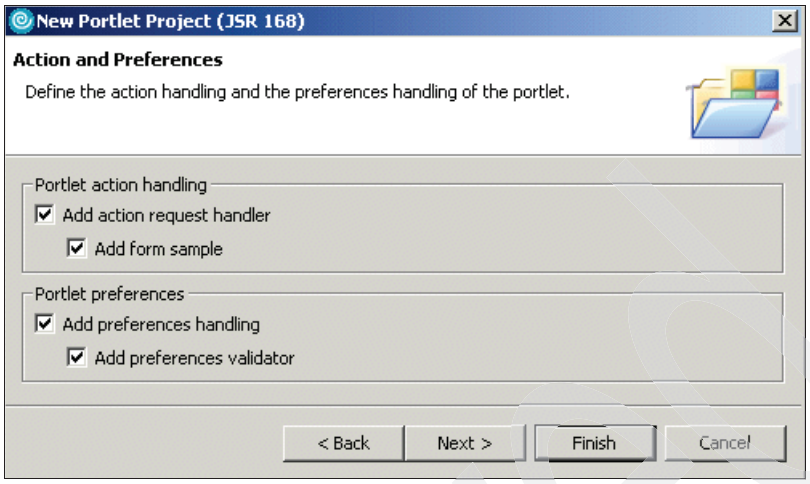


Figure 9-4 Actions and Preferences

12. Single Sign-On with Credential Vault will not be used. Click **Next**.
13. Select the check boxes to support Edit Mode, Help Mode, and Configure Mode. Select **Finish** to complete the creation of the portlet project.

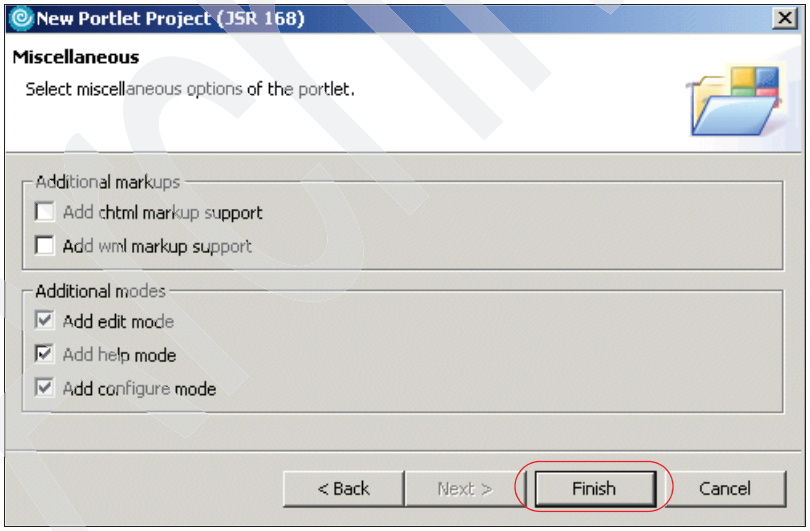


Figure 9-5 Additional modes

9.2.2 Examining the generated portlet

As an optional exercise, examine the generated portlet. For example:

1. Locate and display the web.xml.
2. Locate and display the portlet descriptor (portlet.xml).
3. Read the generated Java code for MySimplePortlet.java
   a. Identify what class the portlet is extended from.
   b. See what parameters are passed to the doView() method.
   c. See what parameters are passed to the processAction() method
   d. Review the doCustomConfigure() method
4. Review the generated session bean.
5. Display the generated JSPs for all modes (view, edit, help, configure).

9.3 Updating the generated portlet

In this section, you will update the components of the generated JSR 168 portlet in order to implement the sample portlet scenario as described in 9.1, “Overview” on page 302. For the sake of simplicity in your project, do the following:

1. Disable `Group Projects.` This can be accomplished by selecting the button that looks like a folder or by clicking the small triangle and deselecting **Group Projects** as shown in Figure 9-6.

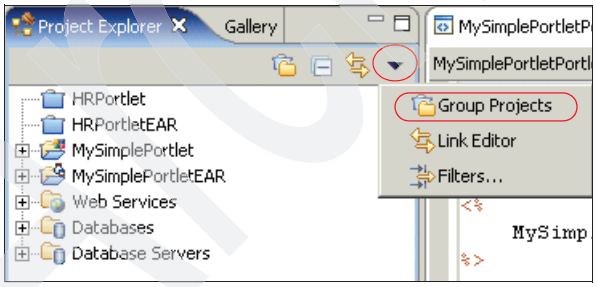


Figure 9-6 Group projects by type

2. In a similar way, select **Filters.** The Filters window will now be displayed.
3. Select the **Types of content** tab.
4. Deselect **WebService**, **Databases**, and **Database Servers** since these extensions will not be used in this project. Click **OK**.

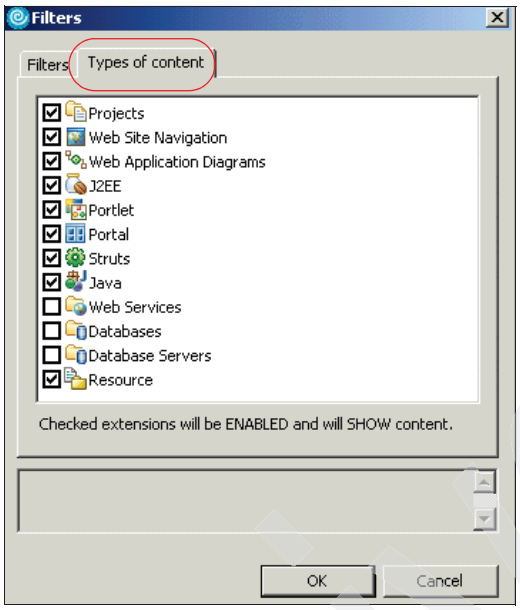


Figure 9-7 Filters window

9.3.1 Modifying the session bean

The generated session bean (MySimplePortletPortletSessionBean.java) is used to pass data to the JavaServer Pages (JSPs). In this section, you will modify this class as follows:

- Store the user name, color and greeting.
- Declare these class variables as private.
- Use getter and setter methods for saving and retrieving the data.

Note: Using private class variables allows you to control what values are set by using the setter methods.

Execute the following steps:

1. Expand the **MySimplePortlet** project by selecting the plus symbol next to it.
2. Expand the **Java Resources** and the **JavaSource** folders.
3. Finally, expand the **mysimpleportlet** package folder.

Archived

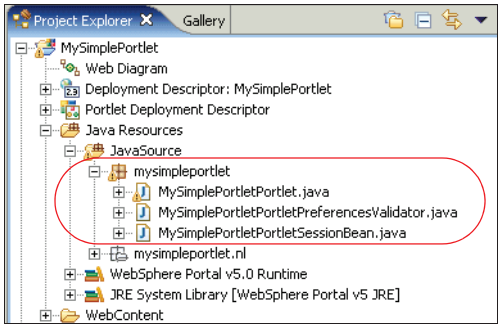


Figure 9-8 Java Source folder expanded

4. Open the class by double-clicking **MySimplePortletPortletSessionBean.java**.
5. Delete all the class variables and methods from the SessionBean class.
6. Create private Strings for userName, color, and greeting. The bean class should now look as illustrated in Example 9-1.

Example 9-1 MySimplePortletPortletSessionBean class

```
package mysimpleportlet;
public class MySimplePortletPortletSessionBean {
   private String userName = "";
   private String color = "";
   private String greeting = "";
}
```

7. Right-click in the MySimplePortletPortletSessionBean window and select **Source** then **Generate Getters and Setters**.

Archived

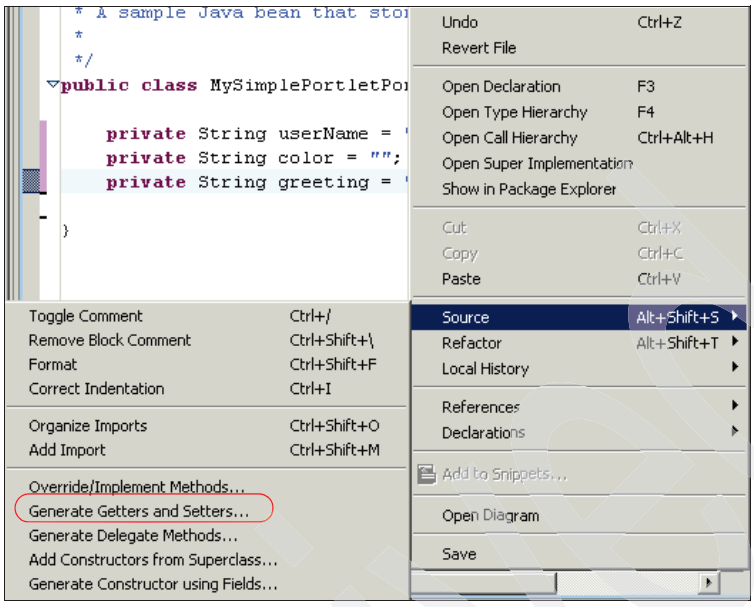


Figure 9-9 Generate Getters and Setters option

8. Check the boxes for color, userName, and greetings so that getters and setters will be generated for all three variables. Click **OK**.

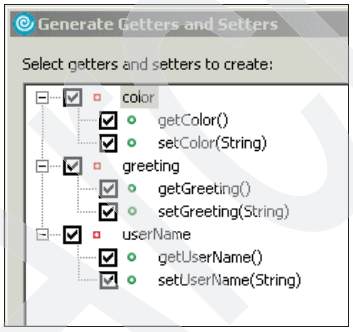


Figure 9-10 Getters and Setters

9. Save the file by using **Ctrl-s** or **File** → **Save**.

   Note: You may see errors in other classes after saving this file. You will correct these errors in a later step.

9.3.2 View mode

In this section, you will update the JSP for View mode and the processAction() method to satisfy the requirements of the sample scenario.

Modifying MySimplePortletPortletView.jsp (View mode)

In this section, you will modify the view JSP to display the greeting and name in the selected color. The sample form created will also be modified to select a color and receive a name.

Execute the following steps:

1. Expand the **WebContent** folder by clicking the plus symbol.
2. Expand the **mysimpleportlet** folder.
3. Expand **JSP** then **html**.
4. Double-click **MySimplePortletPortletView.jsp** to open it.
5. Replace the MySimplePortletPortletView.jsp with the code shown in Example 9-2.

Example 9-2 MySimplePortletPortletView.jsp

```
<%@ page session="false" contentType="text/html"
import="java.util.*,javax.portlet.*,mysimpleportlet.*" %>
<%@taglib uri="http://java.sun.com/portlet" prefix="portlet" %>
<portlet:defineObjects/>
<%
MySimplePortletPortletSessionBean sessionBean =
(MySimplePortletPortletSessionBean)renderRequest.getPortletSession().getAttribu
te(MySimplePortletPortlet.SESSION_BEAN);
%>

<DIV style="margin: 6px">

<% /******** This form will be updated later *********/ %>
<FORM method="POST" action="<portlet:actionURL/>">
<LABEL  for="<%=MySimplePortletPortlet.FORM_TEXT%>">Enter order id:</LABEL><BR>
<INPUT name="<%=MySimplePortletPortlet.FORM_TEXT%>" type="text"/>
<INPUT name="<%=MySimplePortletPortlet.FORM_SUBMIT%>" type="submit"
value="Submit"/>
</FORM>
<% /******** End of sample code *********/ %>
</DIV>
```

6. You will now get the userName, color, and greeting variables out of the sessionBean. You should consider the following issues for this sample scenario:

- Notice that you have previously defined the greeting and userName variable with an empty String in the sessionBean.
- The first time running the portlet, both the name and the greeting will be empty and therefore will not appear on the page.
- You will set a CSS style in the header to display the user name and greeting in the currently selected color. If the length of the color String is not longer than zero indicating that no color has been selected, we will use black for the color.
- The first time the portlet is called, there will not be a color selected so color black will be used as default.

7. Add the code highlighted in Example 9-3 to get the variables from the session bean.
8. Add also a heading element (H3) to display the variables obtained from the session bean.

> **Note:** You do not have to test for null for these variable because you set them as an empty String by default. However, for a production application, it is recommended testing for null.

Example 9-3 Getting variables from sessionBean

```
<%@ page session="false" contentType="text/html"
import="java.util.*,javax.portlet.*,mysimpleportlet.*" %>
<%@taglib uri="http://java.sun.com/portlet" prefix="portlet" %>
<portlet:defineObjects/>
<%
MySimplePortletPortletSessionBean sessionBean =
(MySimplePortletPortletSessionBean)renderRequest.getPortletSession().getAttribu
te(MySimplePortletPortlet.SESSION_BEAN);
    String name = sessionBean.getUserName();
    String greeting = sessionBean.getGreeting();
    String color = sessionBean.getColor();
    if((color == null) || (color.length() == 0)){
    color = "black";
    }
%>
<H3 style="margin-bottom: 3px; color:<%= color %>"><%= greeting + " " + name
%></H3>
```

9. Modify the form to get the user's name and the desired color.
   - You will now change the label and the input for the form to use the variable FORM_NAME.

- Next, you will add a select tag and use SimplePortletPortlet.FORM_COLOR variable as the name.
- Then add code to get the preferences.
- Once you have the preferences, you can look through all the preferences to fill the drop-down list for all the colors.
- In a later step you will be adding the three primary colors into the portlet descriptor (portlet.xml) preferences along with a greeting preference. Because of this, you do not need to test the preferences to determine if there are any colors.
- You do, however, have to test for the .greeting preference. You do this by testing if the preference first character starts with a "." character.
- If the color from the sessionBean matches the preference color then you mark the option tag as selected. It will be the selected color the next time the user views the page.

10. Replace the form in MySimplePortletPortletView.jsp with the form shown in Example 9-4.

Example 9-4 New updated form in the view JSP

```
<FORM method="POST" action="<portlet:actionURL/>">
<LABEL  for="<%=MySimplePortletPortlet.FORM_NAME%>">Enter your name:</LABEL><BR>
<INPUT name="<%=MySimplePortletPortlet.FORM_NAME%>" value="<%= name %>" type="text"/>
<SELECT name="<%= MySimplePortletPortlet.FORM_COLOR %>">
<%
PortletPreferences prefs = renderRequest.getPreferences();
// loop through all the preferences to get the colors.
// The three primary colors have been added to the portlet.xml so we do not
// need to test for no colors.
   Enumeration prefNames = prefs.getNames();
   while(prefNames.hasMoreElements()) {
   String thisColor = (String)prefNames.nextElement();
      if((thisColor != null) && (!thisColor.startsWith("."))){
      %>
      <option <% if(color.equalsIgnoreCase(thisColor)) { %>
      selected="selected" <% } %> value="<%= thisColor %>" >
      <%= thisColor %></option>
         <%
       }
       }%>
       </SELECT>
      <INPUT name="<%=MySimplePortletPortlet.FORM_SUBMIT%>" type="submit" value="Submit"/>
</FORM>
```

11. Save the file by using **Ctrl-s** or **File** → **Save**. Once again, there will be errors in the JSP and they will be fixed in the next step.

12.Optionally, preview the View mode JSP as illustrated in Figure 9-11.

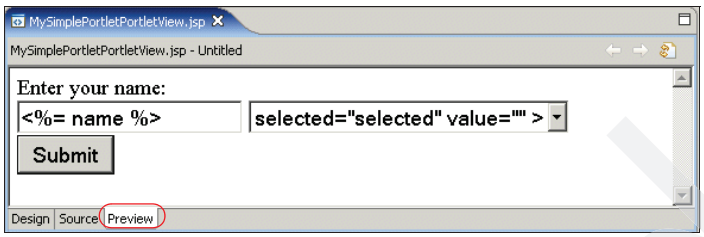


Figure 9-11 View mode JSP preview

Updating MySimplePortletPortlet to handle the new view form

You will need to update MySimplePortletPortlet.java code to support the changes you made in the JSP for View mode (MySimplePortletPortletView.jsp). The following issues should be considered:

- You changed the variable name in the form from FORM_TEXT to FORM_NAME.
- You also added the FORM_COLOR drop-down box.
- You will need to modify the portlet class for the new variables
- You also need to modify the actionProcess () method to handle the name and color from the form in MySimplePortletPortletView.jsp.

Follow these steps to update the Java code:

1. Expand the **MySimplePortletPortlet** class in the Project Explorer view of Rational Application Developer and select the **FORM_TEXT** field.

Archived

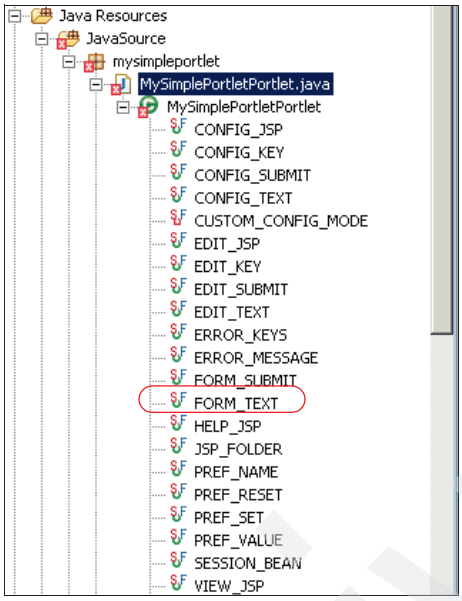


Figure 9-12 Project Explorer view

2. Right-click the **FORM_TEXT** field and select **Refactor** → **Rename**.
3. A dialog box will pop up allowing you to change the field name. Change the name to `FORM_NAME`. Click **OK**.

   Note: Be sure the check box for **Update References** is selected.

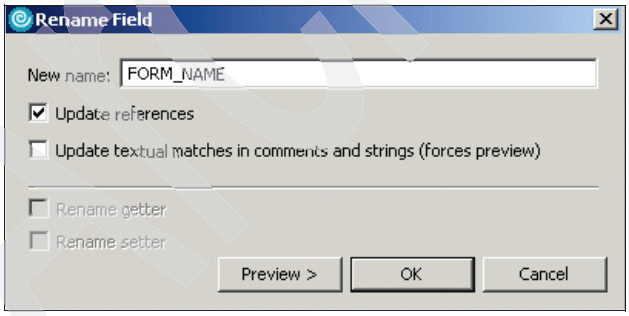


Figure 9-13 Renaming the field

4. If you get a warning, select **Continue**. This warning is a result of compilation errors in the portlet class. These errors will be corrected in a later step.

Archived

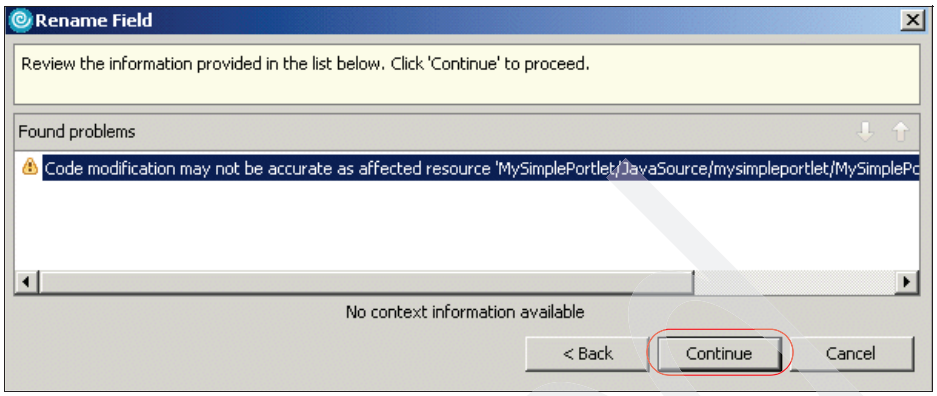


Figure 9-14 Warning message

5. All the references in the portlet class will now be updated to reflect the new name. We will now modify the value for this variable.
6. Open the portlet class by double-clicking the file.
7. Modify the FORM_NAME variable to look like Example 9-5. This should be all be on the same line. You changed the value to reflect the new name of the variable and you also modified the comment line.

Example 9-5 FORM_NAME variable

```
public static final String FORM_NAME = "MySimplePortletPortletFormName"; //
Parameter for the name of the user
```

8. Now add a new variable for the FORM_COLOR. Enter the line shown in Example 9-6. It should be all in one line and under the FORM_NAME line.

Example 9-6 New FORM_COLOR variable

```
public static final String FORM_COLOR = "MySimplePortletPortletFormColor";
// Parameter for the selected color
```

9. Locate the processAction() method. You will be using a session bean to store the user name and the selected color from the form. The processAction() method needs to be updated to handle the new fields of the form from MySimplePortletPortletView.jsp. Add the methods to set the variables in the sessionBean using the setter methods you created earlier. The code is shown in Example 9-7 on page 316.

Example 9-7 Submit action process

```
if( request.getParameter(FORM_SUBMIT) != null ) {
   // Set form text in the session bean
   MySimplePortletPortletSessionBean sessionBean = getSessionBean(request);
   if( sessionBean != null ) {
      sessionBean.setUserName(request.getParameter(FORM_NAME));
      sessionBean.setColor(request.getParameter(FORM_COLOR));
   }
}
```

10. Also, in the processAction() method, remove the complete if statements for PREF_RESET and PREF_SET. You will not be using these parameters. The processAction() method should now look as illustrated in Example 9-8.

Example 9-8 Processing actions in the processAction() method

```
public void processAction(ActionRequest request, ActionResponse response)
throws PortletException, java.io.IOException {
if( request.getParameter(FORM_SUBMIT) != null ) {
   // Set form text in the session bean
   MySimplePortletPortletSessionBean sessionBean = getSessionBean(request);
   if( sessionBean != null ) {
      sessionBean.setUserName(request.getParameter(FORM_NAME));
      sessionBean.setColor(request.getParameter(FORM_COLOR));
   }
}

if( request.getParameter(EDIT_SUBMIT) != null ) {
   PortletPreferences prefs = request.getPreferences();
   try {
      prefs.setValue(EDIT_KEY,request.getParameter(EDIT_TEXT));
      prefs.store();
   }
   catch( ReadOnlyException roe ) {
   }
   catch( ValidatorException ve ) {
   }
}

if( request.getParameter(CONFIG_SUBMIT) != null ) {
   PortletPreferences prefs = request.getPreferences();
   try {
      prefs.setValue(CONFIG_KEY,request.getParameter(CONFIG_TEXT));
      prefs.store();
   }
   catch( ReadOnlyException roe ) {
   }
   catch( ValidatorException ve ) {
```

Archive

```
    }
}
}
```

11. Save the file by using **Ctrl-s** or **File** → **Save**

    There should not be any errors in the portlet class at this point. You should also not have any errors in the view JSP. You may need to open the view JSP make a small change like adding a new blank line and save the file again to cause it to recompile.

    Note: You may see an error for the renderRequest object not being resolved. This object will be created during runtime by the <portlet:defineObjects/> tag. This error should be ignored at this time.

9.3.3 Edit mode

In this section, you will update the generated JSP for Edit mode and the processAction() method to satisfy the requirements of the sample scenario.

Modifying MySimplePortletPortletEdit.jsp (Edit mode)

You will now modify MySimplePortletPortletEdit.jsp to allow the user to add more colors by invoking the portlet Edit mode.

Execute the following steps:

1. Open MySimplePortletPortletEdit.jsp by double-clicking the file.
2. You start by leaving only the form for entering the new value and the form for returning to the View mode. The Edit mode JSP should now look as shown in Example 9-9.

   Note: As an alternative, you can change the JSP by replacing it with the code shown in Example 9-9.

Example 9-9 Edit JSP after deleting some source

```
<%@ page session="false" contentType="text/html"
import="java.util.*,javax.portlet.*,mysimpleportlet.*"%>
<%@taglib uri="http://java.sun.com/portlet" prefix="portlet" %>
<portlet:defineObjects/>
<DIV style="margin: 6px">
  <FORM ACTION="<portlet:actionURL/>" METHOD="POST">
    <LABEL for="<%=MySimplePortletPortlet.EDIT_TEXT%>">New Value</LABEL><BR>
    <INPUT name="<%=MySimplePortletPortlet.EDIT_TEXT%>" value="<%=value%>"
type="text"/><BR>
    <INPUT name="<%=MySimplePortletPortlet.EDIT_SUBMIT%>" value="Save"
type="submit"/>
  </FORM>
```

```
<FORM ACTION="<portlet:renderURL portletMode="view"/>" METHOD="POST">
   <INPUT NAME="back" TYPE="submit" VALUE="Back to view mode">
</FORM>
</DIV>
```

3. First, you will edit the field names of the label and input to show that these fields will hold the new color value as follows:
   a. Change the name of the label and input from `MySimplePortletPortlet.EDIT_TEXT` to `MySimplePortletPortlet.EDIT_COLOR`.
   b. Also change the value of the label tag from New Value to New Color.
   c. Delete the `value=”<%=value%>”` from the input tag.
   d. The form should now look as shown in Example 9-10.

Example 9-10 Updated form in edit JSP

```
<FORM ACTION="<portlet:actionURL/>" METHOD="POST">
<LABEL for="<%=MySimplePortletPortlet.EDIT_COLOR%>">New Color</LABEL><BR>
    <INPUT name="<%=MySimplePortletPortlet.EDIT_COLOR%>" type="text"/><BR>
    <INPUT name="<%=MySimplePortletPortlet.EDIT_SUBMIT%>" value="Save"
type="submit"/>
</FORM>
```

4. Display colors using an unordered list.
   a. Create a new line under the first division tag (`<DIV style=”margin: 6px”>`) and add the following heading element:

      ```
      <H3 style="margin-bottom: 3px">Available colors:</H3>
      ```

   b. The colors will be displayed in an unordered list. They will be displayed by their name and in the color they represent. For example, blue will be displayed in blue. Create a new line under line you just created and create an opening and closing list tag:

      ```
      <ul> </ul>
      ```

   c. The edit page should now look as illustrated in Example 9-11.

Example 9-11 Updated view page

```
<%@ page session="false" contentType="text/html"
import="java.util.*,javax.portlet.*,mysimpleportlet.*"%>
<%@taglib uri="http://java.sun.com/portlet" prefix="portlet" %>
<portlet:defineObjects/>
<DIV style="margin: 6px">
<H3 style="margin-bottom: 3px">Available colors:</H3>
<ul> </ul>
<FORM ACTION="<portlet:actionURL/>" METHOD="POST">
```

Archived

```
<LABEL for="<%=MySimplePortletPortlet.EDIT_COLOR%>">New Color</LABEL><BR>
    <INPUT name="<%=MySimplePortletPortlet.EDIT_COLOR%>" type="text"/><BR>
    <INPUT name="<%=MySimplePortletPortlet.EDIT_SUBMIT%>" value="Save"
type="submit"/>
</FORM>
<FORM ACTION="<portlet:renderURL portletMode="view"/>" METHOD="POST">
   <INPUT NAME="back" TYPE="submit" VALUE="Back to view mode">
</FORM>
</DIV>
```

5. You have already created the code to loop through the preferences in the View mode JSP (MySimplePortletPortletView.jsp). You will now use the code in that JSP to create a snippet that you can easily paste into other files rather that retyping the code each time.

   a. Open MySimplePortletPortletView.jsp.

   b. In the View mode JSP, highlight and copy to the clipboard the code shown in Example 9-12. To copy select **Ctrl-c** or use **Edit** → **Copy**.

Example 9-12 Selected code from View mode page (MySimplePortletPortletView.jsp)

```
PortletPreferences prefs = renderRequest.getPreferences();
   // loop through all the preferences to get the colors.
   // The three primary colors have been added to the portlet.xml so we do not
   // need to test for no colors.
   Enumeration prefNames = prefs.getNames();
   while(prefNames.hasMoreElements()) {
   String thisColor = (String)prefNames.nextElement();
   if((thisColor != null) && (!thisColor.startsWith("."))){
%>
   <option <% if(color.equalsIgnoreCase(thisColor)) { %>
      selected="selected" <% } %> value="<%= thisColor %>" >
      <%= thisColor %></option>
<%}
   }
```

   c. Notice that you did not select the opening and closing JSP scriptlet tags (<% and %>).

   d. You will also use this code snippet in MySimplePortletPortlet class and these tags will not be needed.

   e. When you create the snippet you will also remove the other scriptlet tags and replace the line with a variable.

6. Create a snippet from the previous code.

   Note: Snippets are sections of code that can be reused throughout your application.

Archive

a. Be sure you are in the Web Perspective and the **Snippets** view is selected as illustrated in Figure 9-15.

b. In the gray area below the Web Services button, right-click and select **Customize...**.

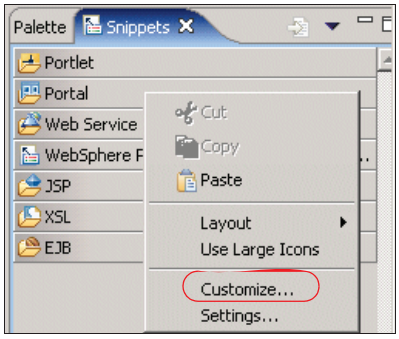


Figure 9-15 Select Customize

7. The Customize Palette window will now be displayed.

a. In the Customize Palette Window select **New** then **New Category.**

b. For the name, type `MySnippets` and then select **Apply.**

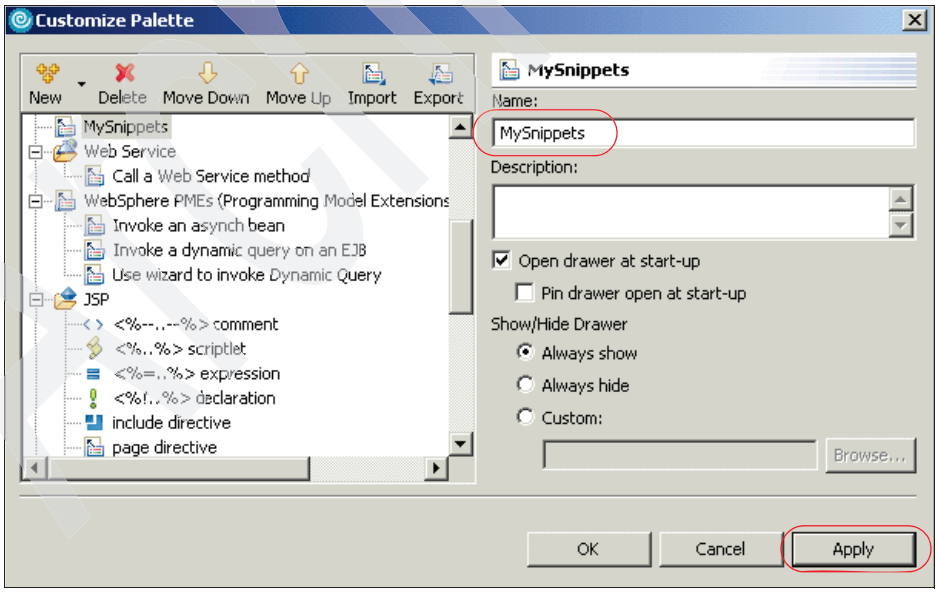


Figure 9-16 Customize Palette window

Archived

8. You should now see your new snippet on the left window.
   a. Highlight the snippet by clicking it and then select **New** → **New Item**.
   b. For the name, type `Color Preferences` and for the description, type `Loop through the colors in the preferences object`.
   c. You will now paste in the code you copied to the clipboard from the view JSP. Click in the Template Pattern window and select **Ctrl-V** to paste the code.

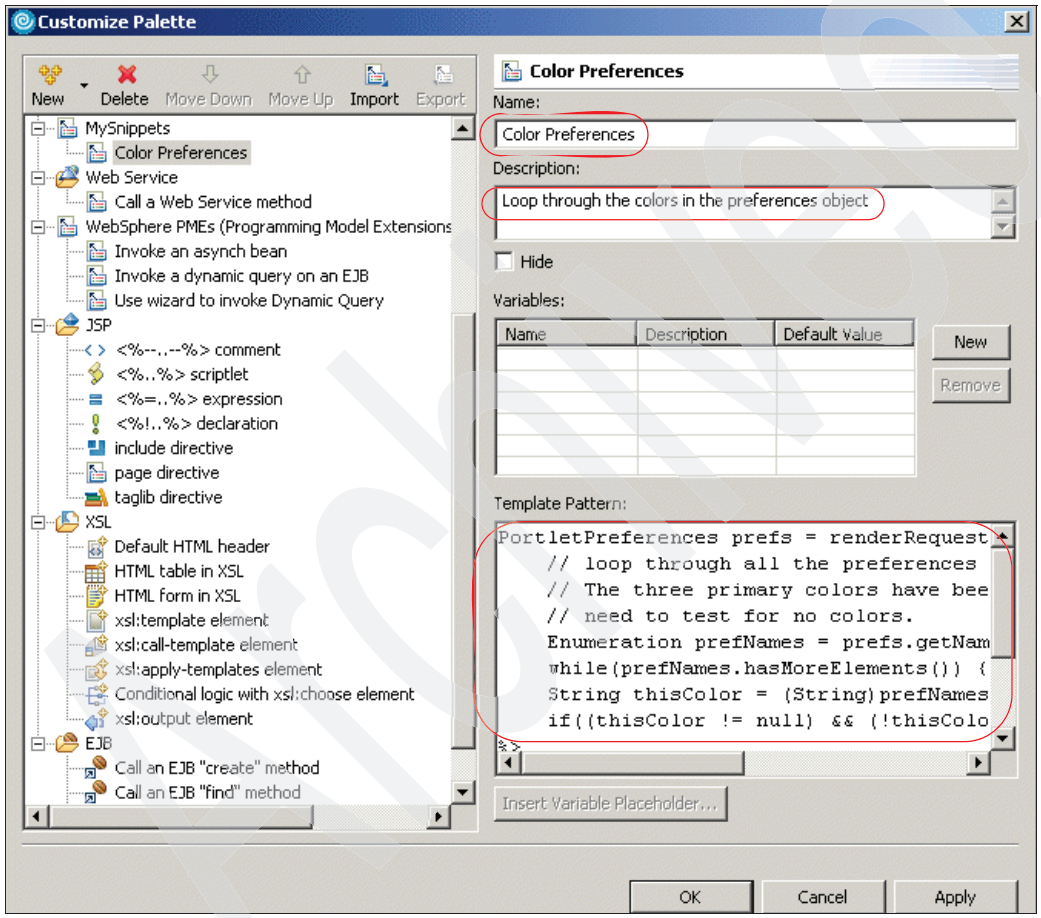


Figure 9-17 Template Pattern

9. You now need to modify the selected code to add the variable.
   a. Delete the following section in the template window:

```
%><option <% if(color.equalsIgnoreCase(thisColor)) { %>
selected="selected" <% } %> value="<%= thisColor %>" ><%= thisColor
%></option> <%
```

b. Under the variable section, select **New** and enter a name of `loopAction` and a description of `Loop Action`.

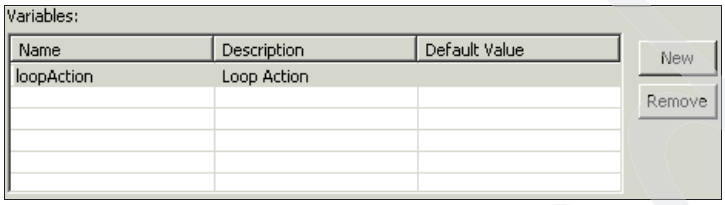


Figure 9-18 Snippet variables

c. Place your cursor on the line that you deleted in step a., “Delete the following section in the template window:” on page 321 and select the **Insert Variable PlaceHolder** button.

d. Double-click **loopAction** then select **Apply**.

e. When your Customize Palette window looks as shown in Figure 9-19 on page 323, select **OK**.

Archive

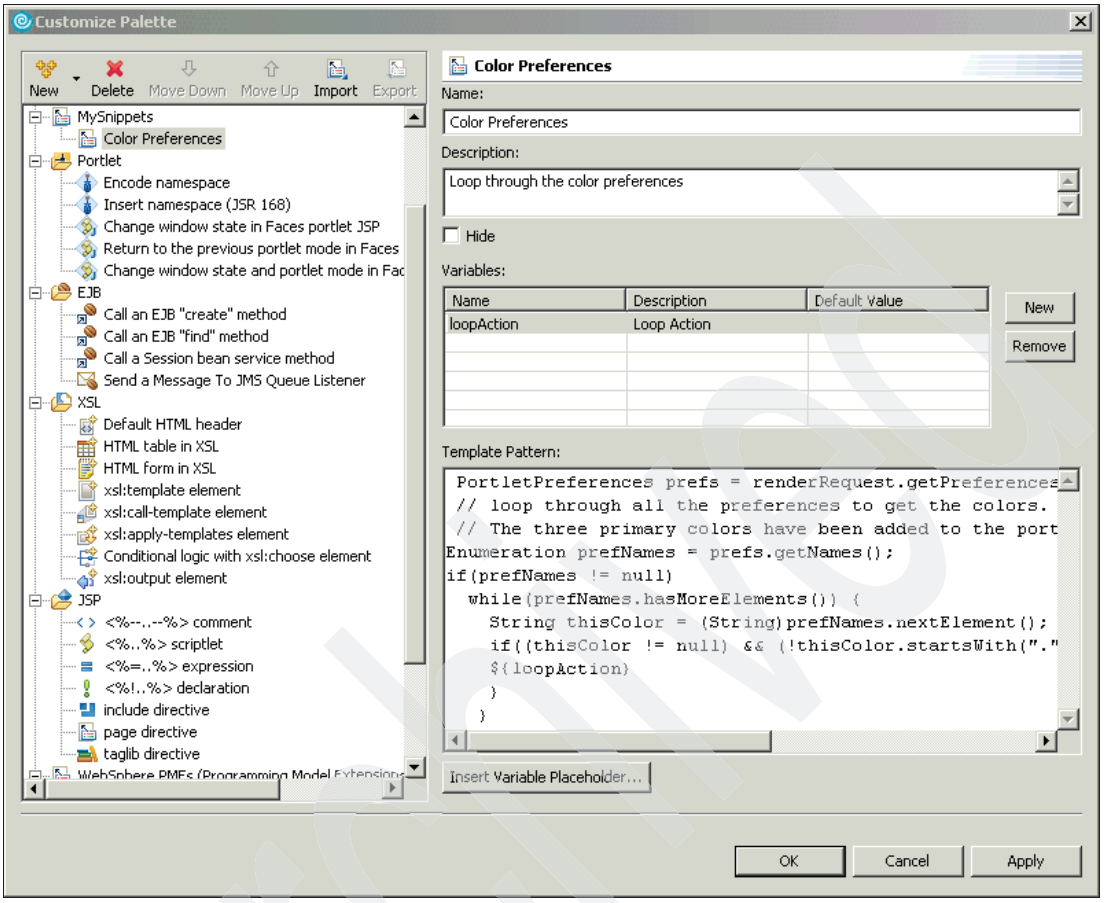


Figure 9-19 Completed customized palette window

10. Place your cursor in between the <ul> and tags in MySimplePortletPortletEdit.jsp and enter <% %> as highlighted in Example 9-13.

Example 9-13 Updated tags in edit page

```
<%@ page session="false" contentType="text/html"
import="java.util.*,javax.portlet.*,mysimpleportlet.*"%>
<%@taglib uri="http://java.sun.com/portlet" prefix="portlet" %>
<portlet:defineObjects/>
<DIV style="margin: 6px">
<H3 style="margin-bottom: 3px">Available colors:</H3>
<ul><% %> </ul>
```

11. Place you cursor in between the two <% %> tags you just created.

12. In the snippets view, double-click the **Color Preferences** snippet you just created.

13. Enter the following for the loopAction variable by placing the cursor in the value box next to the loopAction then select **Insert**.

```
%><li style="color:<%= thisColor %>"><%= thisColor %></li><%
```

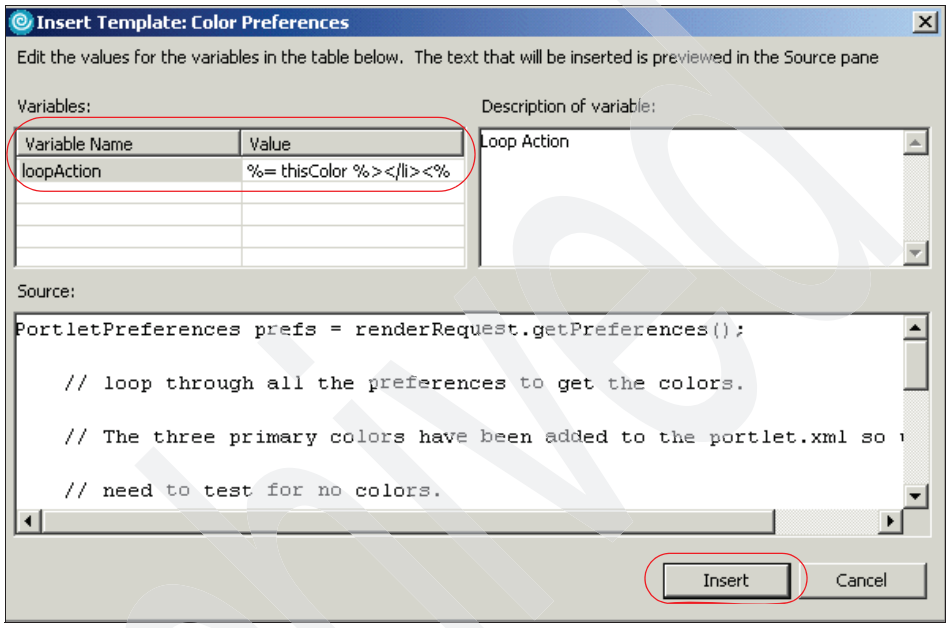


Figure 9-20 Insert loopAction variable

Note: This will create a list items for each color in the preferences. The style tag will display the color in the color it depicts.

14. You will now add a note in MySimplePortletPortletEdit.jsp informing the user that there are only 16 supported colors for this portlet. In a later step, you will modify the PortletPreferencesValidator to only accept these colors.

    a. Add a new line directly under the ending tag.

    b. Enter the following code:

```
<br />
<div style="color:maroon">
(Only the following colors are supported:<br>
aqua, black, blue, fuchsia, gray, green, lime, maroon, <BR>
navy, olive, purple, red, silver, teal, white, yellow)
</div>
```

15. Save the file by using **Ctrl-s** or **File** then **Save**.

> **Note:** Once again, you will see errors in your JSP. These will be fixed in a later step.

16. Optionally, use the click the **Preview** tab to see the Edit mode JSP as illustrated in Figure 9-21.



Figure 9-21 Edit mode JSP preview

Updating MySimplePortletPortlet class to handle the edit form

If not already done, open the portlet class by double-clicking it.

1. You will not be using the EDIT_KEY variable so it can be deleted.

Example 9-14 Delete EDIT_KEY line

```
public static final String EDIT_KEY = ".MySimplePortletPortletEditKey";     //
Key for the portlet preferences
```

2. Modify the EDIT_TEXT to EDIT_COLOR and also change the value:

```
public static final String EDIT_COLOR = "MySimplePortletPortletEditColor";
// Parameter name for the new color
```

3. Find the processAction() method and locate the if statement for the process of the edit form submit.

```
if( request.getParameter(EDIT_SUBMIT) != null ) {
   PortletPreferences prefs = request.getPreferences();
   try {
```

Archived

4. You will need to make sure that the new color is not null. First, you will create a new String named newColor to store the color retrieved from the EDIT_COLOR field of the form. You will then test to see is the newColor String is null. If the newColor String is not null trim any leading and trailing spaces but using the trim() method and you also set it to all lower case.
5. Modify the try block in the if statement to look like Example 9-15.

Example 9-15 Modified the if statement for the edit form

```
if( request.getParameter(EDIT_SUBMIT) != null ) {
   PortletPreferences prefs = request.getPreferences();
   try {
      String newColor = request.getParameter(EDIT_COLOR);
      if(newColor != null){
         newColor.trim().toLowerCase();
      }
      prefs.setValue(newColor,newColor);
      prefs.store();
   }
   catch( ReadOnlyException roe ) {
   }
   catch( ValidatorException ve ) {
   }
}
```

6. Save the file by using **Ctrl-s** or **File** → **Save**.

Note: You should not see any errors in the portlet class after saving. If MySimplePortletPortletEdit.jsp still contains errors, open the file add a blank line then save the file to force it to recompile. You should no longer see the errors.

9.3.4 Configure mode

In this section, you will update the generated JSP for configure mode and the processAction() method to satisfy the requirements of the sample scenario.

Modifying MySimplePortletPortletConfig.jsp (Configure mode)

You will edit the config JSP form for the greeting. Execute the following steps:

1. Delete everything after the first division tag (<DIV style="margin: 6px">) up to the form. Then delete everything after the form to the closing division tag (</DIV>). Your JSP should now look like Figure 9-16 on page 327.

   Note: As an alternative you can replace the wizard generated JSP with the JSP illustrated in Example 9-16 on page 320.

Example 9-16 Configuration mode JSP

```
<%@ page session="false" contentType="text/html"
import="javax.portlet.*,mysimpleportlet.*" %>
<%@taglib uri="http://java.sun.com/portlet" prefix="portlet" %>
<portlet:defineObjects/>

<DIV style="margin: 6px">

  <FORM ACTION="<portlet:actionURL/>" METHOD="POST">
    <LABEL for="<%=MySimplePortletPortlet.CONFIG_TEXT%>">New Value</LABEL><BR>
    <INPUT name="<%=MySimplePortletPortlet.CONFIG_TEXT%>" value="<%=value%>"
type="text"/><BR>
    <INPUT name="<%=MySimplePortletPortlet.CONFIG_SUBMIT%>" value="Save"
type="submit"/>
  </FORM>

</DIV>
```

2. You will add the code to get the current greeting from the sessionBean. Add the following lines to the Configure mode JSP above the opening division tag.

```
<%
//This should be on one line
   MySimplePortletPortletSessionBean sessionBean =
(MySimplePortletPortletSessionBean)renderRequest.getPortletSession().getAtt
ribute(MySimplePortletPortlet.SESSION_BEAN);
//The following code will be on new lines.
    String greeting = "";
    if(sessionBean != null)
    greeting = sessionBean.getGreeting();
%>
```

3. Change the label and input parameters of the form from MySimplePortletPortlet.CONFIG_TEXT to MySimplePortletPortlet.CONFIG_GREETING. Also change the value of the label from New Value to New greeting.
4. Change the value of the input from `<%= value %>` to `<%= greeting %>`
5. Save the file by using **Ctrl-s** or **File** → **Save**.
6. Optionally, preview the Configuration mode JSP.

Archived

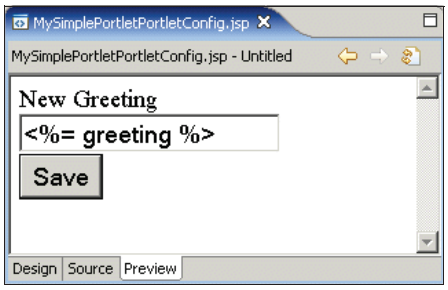


Figure 9-22 Configuration mode JSP preview

> **Note:** There might be some errors in your JSP. These errors will be fixed in a later step.

Updating MySimplePortletPortlet to handle the config form

You will also need to modify the portlet class to accept the greeting. Execute the following steps:

1. If not already opened, open the portlet class by double-clicking it.
2. Modify the CONFIG_TEXT to look like Example 9-17.

Example 9-17 CONFIG_GREETING variable

```
public static final String CONFIG_GREETING  =
"MySimplePortletPortletConfigGreeting";   // Parameter name for the greeting
```

3. Change the value of the CONFIG_KEY from `.MySimplePortletPortletConfigKey` to `.greeting`.
4. Locate the processAction() method and find the if statement for CONFIG_SUBMIT.

```
request.getParameter(CONFIG_SUBMIT) != null ) {
        PortletPreferences prefs = request.getPreferences();
        try {
```

5. Process the greeting message as follows:
   a. You will first create a String named greeting and store the value of the CONFIG_GREETING field from the form.
   b. You will then test if the greeting String is not null and if its length is greater than zero. If it is, you will store it in the preferences using the CONFIG_KEY
   c. You will then store the greeting in the session bean.

Archived

d. Your code should look like Example 9-18.

Example 9-18 Config form in processAction() method

```
if( request.getParameter(CONFIG_SUBMIT) != null ) {
   PortletPreferences prefs = request.getPreferences();
   try {
      String greeting = request.getParameter(CONFIG_GREETING);
      if((greeting != null) && (greeting.length() > 0)){
      prefs.setValue(CONFIG_KEY, greeting);
      prefs.store();
      //get greeting from preferences and set it.
      MySimplePortletPortletSessionBean sessionBean = getSessionBean(request);
      sessionBean.setGreeting(request.getPreferences().getValue(CONFIG_KEY,
      "Hello"));
       //set it session
       request.getPortletSession().setAttribute(SESSION_BEAN, sessionBean);
}
   }
   catch( ReadOnlyException roe ) {
      //this is read only and can only be updated by an admin
   }
   catch( ValidatorException ve ) {
   }
}
```

6. Save the file by using **Ctrl-s** or **File** → **Save**

> **Note:** There should not be any errors in this file. If you still have errors in MySimplePortletPortletConfig.jsp, open the JSP add a blank line and then save. This will force the JSP to recompile and remove the errors.

9.3.5 Updating the portlet descriptor (portlet.xml)

In this section, you will update the portlet descriptor (portlet.xml) to do the following:

- Add three primary colors as preferences
- You will also add the .greeting preference

Execute the following steps:

1. Open the portlet.xml by double-clicking the `Portlet Deployment Descriptor` in your project.
2. Select the **Portlets** tab and then select MySimplePortlet as seen in Figure 9-23 on page 330.

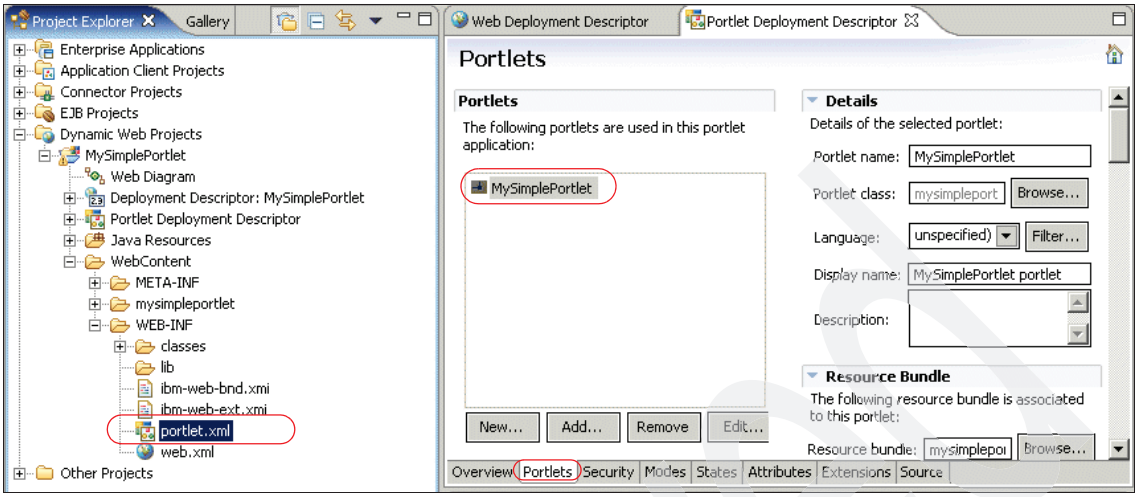


Figure 9-23 Portlet.xml

3. Scroll down until you see the *Persistent Preference Store* box as shown in Figure 9-24.
4. Clear all preferences as follows:
   a. Select all the preferences by holding down **Ctrl** and selecting each preference.
   b. Select **Remove**.

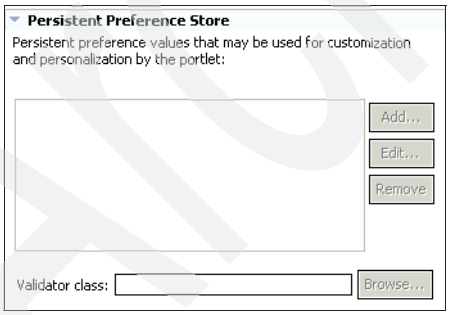


Figure 9-24 Persistent Preference Store

5. You will now add the preferences for the .greeting, red, yellow, and blue. For the .greeting, you will mark it as read-only.

   Note: Read only means that only an administrator in Configuration mode will be able to update this preference during runtime.

Archive

Execute the following steps:

a. Select **Add** to open the New Preference window.

b. Enter `.greeting` for the name.

c. Select **Add** to enter a value of `Hello`.

d. For the .greeting preference only: check the box for read-only.

e. Select **OK**.

f. Repeat steps a through e for the three colors (red, yellow, and blue). Use lower case for the color names and values. Do not select the read-only option for these preferences.

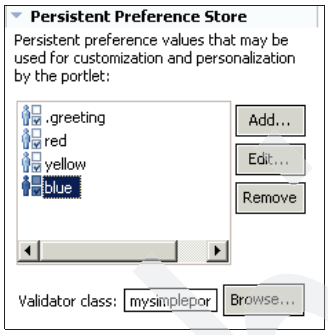


Figure 9-25 New Preference window

g. Save and close the portlet.xml by using **Ctrl-s** or **File** → **Save**.

> **Note:** There should not be any errors in any of the files.

6. Examine the portlet descriptor (portlet.xml) source and review the preferences you just entered.

9.3.6 Modifying the MySimplePortletPortletPreferenceValidator class

You will now update the MySimplePortletPortletPreferenceValidator class to only store the 16 supported colors.

1. Open the MySimplePortletPortletPreferenceValidator by double-clicking the class.
2. Delete the for loop in the validate method so that it looks like Example 9-19 on page 332.

Archived

Example 9-19 Validate method

```
public void validate(PortletPreferences preferences) throws ValidatorException
{
Collection failedKeys = new ArrayList();
if( !failedKeys.isEmpty() ) {
          throw new ValidatorException("One or more preferences do not comply
with the validation criteria",failedKeys);
       }
}
```

3. In the validate method, you will do the following:
   a. Create a String array to store all the valid colors.
   b. You will then loop through all the preferences (excluding the .greeting).
   c. If there are any colors that do not match a supported color, a ValidatorException will be thrown and the new color will not be stored.
   d. The code in your validate method should look like Example 9-20.

Example 9-20 New validate method

```
public void validate(PortletPreferences preferences) throws ValidatorException
{
String[] HTMLColors = new String[] { "aqua", "black", "blue", "fuchsia",
"gray", "green", "lime", "maroon", "navy", "olive", "purple", "red", "silver",
"teal", "white", "yellow" };
Collection failedKeys = new ArrayList();
PREFERENCES_LOOP:
   for( Enumeration names=preferences.getNames(); names.hasMoreElements(); ) {
      String name = names.nextElement().toString();
      if( name.startsWith(".") ) continue;
      String value = preferences.getValue(name, "");
      for(int i = 0; i < HTMLColors.length; i++){
         if(value.equalsIgnoreCase(HTMLColors[i])) continue PREFERENCES_LOOP;
      }
      failedKeys.add(name);
   }
   if( !failedKeys.isEmpty() ) {
      throw new ValidatorException("One or more preferences do not comply with
the validation criteria",failedKeys);
}
}
```

4. Save the file by using **Ctrl-s** or **File** → **Save**. There should not be any errors in this file.

Archived

9.4 Running the portlet

Now that you have created the portlet, you will deploy it and execute it on the local Portal test environment. For example, execute the following steps to deploy the portlet:

1. Run the portlet by right-clicking **MySimplePortlet** project in the Project Explorer view and select **Run** → **Run on server...**.
2. **In the Define a New Server** window, select **Choose an existing server** option and for *Select the server that you want to use*, select **WebSphere Portal V5.1 Test Environment @ localhost**. Also, use default port 9081.

   Note: You can also check the **Set server as project default** checkbox, so you will not be prompted again when you run the portlet.

3. The portlet executes and you will see it in the built-in browser.

9.4.1 Executing the portlet

When the server is started you will see your portlet displayed in a browser as shown in Figure 9-26. Notice that the greeting and the name are not displayed.

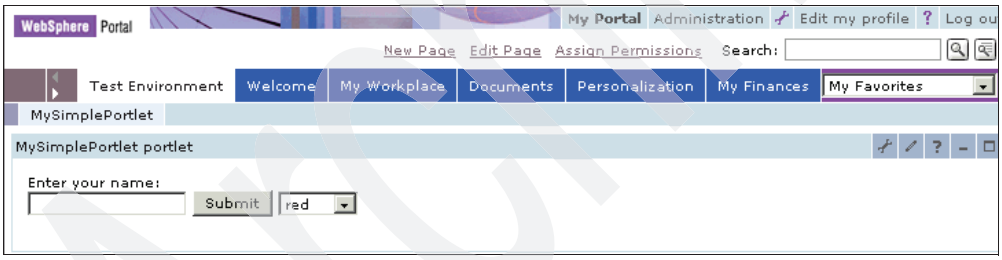


Figure 9-26 MySimplePortlet

1. Enter your name then select **Submit**.
2. You should now see your name displayed in red.
3. Use the drop-down box to select **blue** and then click **Submit**.
4. Your name will now appear in blue. Try again, this time selecting **yellow**.
5. Let's add a greeting now. Select the Config mode icon (the wrench icon in the title bar).
6. Enter **Hello** in the New Greeting box and click **Save.**
7. Select the back button on the title bar.

Archive

8. Your name will now be displayed with the Hello greeting.
9. Let's give your portlet more color. Select the edit icon on the title bar, the edit icon is the pencil.
10. Add as many of the supported colors as you would like by entering the color in the new color text box and then selecting **Save**.
11. If you entered a supported color, you will now be able to see it in the list.



Figure 9-27 Edit mode with colors added

12. Select the back button in the title bar.
13. You can now select your new colors.

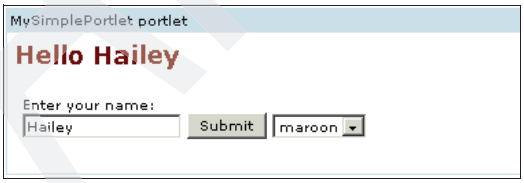


Figure 9-28 Name displayed in new color

Archived

10

Migrating to JSR 168

This chapter will provide an overview of some considerations when migrating from an IBM Portlet to a JSR 168 portlet.

- Modifying the deployment descriptor
- Modifying Java source
- Modifying JSP
- Struts
- JSF
- Portlet services
- Messaging

Cooperative portlets will be covered in Chapter 24, “IBM API declarative cooperative portlets” on page 741, Chapter 25, “IBM API programmatic cooperative portlets” on page 771, Chapter 26, “JSR 168 cooperative portlets” on page 801, and Chapter 27, “Struts cooperative portlets” on page 835. Credential Vault will be covered in Chapter 20, “Credential Vault Service” on page 629.

© Copyright IBM Corp. 2005. All rights reserved.

10.1 Modifying the deployment descriptor

There are major differences between the deployment descriptor for IBM portlets and JSR 168 portlets. As discussed in the overview, IBM portlets are servlets and require the portlet to reference the servlet entry in the web.xml. JSR 168 portlets do not need to reference the web.xml because they are not servlets. Due to the complicated nature of the portlet deployment descriptor, it is recommended that you use Rational Application Developer to modify the portlet.xml. The portlet.xml for JSR 168 portlets do require tags to be placed in a specific order.

10.1.1 doctype

The JSR 168 portlet deployment descriptor uses XML schema for validation and does not require the use of the doctype element. The following line can be removed from the portlet.xml:

```
<!DOCTYPE portlet-app-def PUBLIC "-//IBM//DTD Portlet Application 1.1//EN"
"portlet_1.1.dtd">
```

10.1.2 portlet-app

The root element for the portlet.xml for JSR 168 portlets is the <portlet-app> element. This element has attributes to specify the schema definition, version, and id. Remove the following elements:

```
<portlet-app-def>
.....
</portlet-app-def>
```

The <portlet-app-name> element can also be removed; it is not used in JSR 168 portlets.

The following example shows the portlet-app element for MyJSRPortlet portlet. Update the <portlet-app> element to look like the following. You will want to change the ID for your portlet project.

Example 10-1 portlet-app element

```
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
version="1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd
http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
id="myjsrportlet.MyJSRPortletPortlet.422c1f0220">
```

Archived

10.1.3 concrete-portlet-app

The concept of concrete portlets in JSR 168 is nonexistent. The portlet element in the JSR portlet.xml is used to define the portlet. You will define the portlet title and any initialization parameters there. Save any configuration information along with language definitions and the remove all <concrete-portlet-app> elements.

10.1.4 portlet

JSR also uses the portlet elements like the IBM portlet API. However, there are many different attributes and sub-elements. You can remove the href attribute because, once again JSR portlets do not need to reference the web.xml. JSR portlets also do not contain the version numbers in the portlet.xml. If you need to maintain the version number, it is recommended that you use the manifest file under the meta-inf directory.

Your new <portlet> element should look like the following:

```
<portlet id="myportlet.MyPortletPortlet">
```

The id attribute of the portlet element is optional and can be removed.

10.1.5 portlet-name

The portlet-name element needs to be added to represents the portlet's name:

```
<portlet-name>MyPortlet</portlet-name>
```

10.1.6 web.xml

The IBM portlets required that the servlet be declared in the web.xml. We will not be extending HttpServlet for JSR 168 portlets so the servlet and the servlet-mapping elements can be removed from the web.xml. Copy the servlet class; we will use this for the portlet-class element.

```
<portlet-class>myportlet.MyPortletPortlet</portlet-class>
```

10.1.7 cache

JSR 168 uses the <expiration-cache> for determining the cache. The interpretation of the cache value is the same for both portlet APIs. Copy the value of the portlet cache and then delete the cache elements. JSR 168 does not define shared cache so the expiration-cache element will just accept the cache value as shown in the following line.

```
<expiration-cache>0</expiration-cache>
```

Archived

10.1.8 supports

Instead of supporting markup types, JSR 168 supports MIME types. Both APIs use the supports element, however JSR 168 uses the portlet-mode element to define the supported modes.

Change the portlet.xml from:

Example 10-2 IBM API supports element

```
<supports>
   <markup name="html">
      <view />
      <configure />
      <edit />
      <help />
   </markup>
</supports>
```

to:

Example 10-3 JSR 168 supports element

```
<supports>
   <mime-type>text/html</mime-type>
   <portlet-mode>view</portlet-mode>
   <portlet-mode>edit</portlet-mode>
   <portlet-mode>help</portlet-mode>
   <portlet-mode>config</portlet-mode>
</supports>
```

JSR 168 only defines View, Help, and Edit modes. CONFIG is provided as a custom mode. If using Config mode you will have to define a custom mode. You will also have to override the doDispatch method in the portlet class but we will discuss that later.

The required order of elements under portlet-app is as follows:

1. portlet
2. custom-portlet-mode
3. custom-window-state
4. user-attributes
5. security-constraints

You must create the custom-portlet-mode definition after the last portlet element.

Use the following code to define the CONFIG mode.

Archived

Example 10-4 custom-portlet

```
<custom-portlet-mode>
   <portlet-mode>config</portlet-mode>
</custom-portlet-mode>
```

10.1.9 allows

JSR 168 portlets only support MINIMIZED, MAXIMIZED, and NORMAL window states. By default, all of these modes are supported by the portlet and the <allows> element should be removed form the portlet.xml

10.1.10 config-param

JSR 168 portlets use a PortletPreference object instead of the PortletData and PortletSettings objects. If you have defined any config-param elements in your portlet.xml, you should convert these to portlet-preference elements.

Change:

Example 10-5 config-param in portlet.xml

```
<config-param>
   <param-name>favoriteFood</param-name>
   <param-value>Pizza</param-value>
</config-param>
```

to:

Example 10-6 portlet-preference

```
<portlet-preferences>
   <preference>
      <name>favoriteFood</name>
      <value>Pizza</value>
      <read-only>true</read-only>
   </preference>
</portlet-preferences>
```

The preference element contains a read-only element. If the preference is read-only, only the administrator can update this preference.

10.1.11 Locale settings

JSR 168 portlets define the default locale by using inline text in the portlet.xml.

Archived

default-locale

The default-locale element can be removed, JSR 168 will use the first locale as the default.

supported-locale

All supported locales need to be declared using the supported-locale element.

```
<supported-locale>en</supported-locale>
```

resource-bundles

JSR 168 uses resource bundles to contain the portlet title, short title, and keywords for each supported locale. The locale for each resource should also be added to the supported-locale element.

The resource-bundle element is used to define the resource bundle.

```
<resource-bundle>nls.MyPortlet</resource-bundle>
```

Resource bundles should contain the following three elements

- javax.portlet.title

  This key is used for the portlet title.

- javax.portlet.short-title

  The short-title is used for devices that cannot display the full title.

- javax.portlet.keywords

  Keywords are used when searching portlets.

Note: You do have an option of using the portlet-info tag if you choose not to use the resource bundle.

10.2 Modifying the Java source

Portlets of both APIs follow the MVC design patterns and they use two phase processing. Both have an action and a render phase. the method names are different and all the objects are also different.

10.2.1 Package

The package declarations need to be changed from the IBM portlet API to JSR 168 packages. You can manually change the imports or you can modify the class names and then perform an “Organize Imports” in Rational Application Developer.

Archived

Change:

```
import org.apache.jetspeed.portlet.*;
import org.apache.jetspeed.portlet.event.*;
```

to:

```
import javax.portlet.*;
```

For JSR 168 portlets, you do not have to implement and action listener. There is no equivalent package for org.apache.jetspeed.portlet.event. If you are leveraging any other IBM specific classes, you may need to modify them for JSR 168.

10.2.2 Superclass

The superclass will need to be changed from PortletAdapter to GenericPortlet. Again, if you are implementing IBM portlet API listeners, you can remove the interfaces.

```
public class MyPortletPortlet extends GenericPortlet
```

10.2.3 doXXX methods

All the doXXX methods (doView, doHelp, doEdit, and doConfigure) need to be modified to accept RenderRequest and RenderResponse objects rather than Portletrequest and PortletResponse. The doConfigure will be invoked by overriding the doDispatch method to test for the custom portlet mode CONFIG.

10.2.4 actionPerformed

JSR 168 portlets use the processAction method instead of the actionPerformed method. The processAction method accepts an ActionRequest and ActionResponse object. Change your actionPerformed method to:

```
public void processAction(ActionRequest request, ActionResponse response)
throws PortletException
```

10.2.5 ActionEvent

In the IBM portlet API, the ActionEvent object is used to retrieve an ActionString. The ActionString is used to determine which form made the request. With JSR 168 you should test the request parameters for the Submit button value. Example 10-7 on page 342 Illustrates a JSR form and the processAction method. As you can see, the submit button in the form has the name of the FORM_SUBMIT static variable. Then, in the processAction method, we test if

Archived

this parameter exists. If so, we know that the form associated with this submit button was submitted and we can process accordingly.

Example 10-7 JRS 168 Action Handling

```
JSP Form
<FORM method="POST" action="<portlet:actionURL/>">
<LABEL  for="<%=MyJSRPortletPortlet.FORM_TEXT%>">Enter order id:</LABEL><BR>
<INPUT name="<%=MyJSRPortletPortlet.FORM_TEXT%>" type="text"/>
<INPUT name="<%=MyJSRPortletPortlet.FORM_SUBMIT%>" type="submit"
value="Submit"/>
</FORM>

Portlet Code
public void processAction(ActionRequest request, ActionResponse response)
throws PortletException, java.io.IOException {
   if( request.getParameter(FORM_SUBMIT) != null ) {
      // FORM_SUBMIT form was submitted
      //Do something!
   }
```

Create new unique static variables for each of the submit buttons (or any other field that will result in a request parameter) used to create an action request. Then modify your processAction method to test for these parameters and process the data as need.

10.2.6 Logging

Unlike the IBM portlet API, JSR 168 portlets do not provide a way to determine if logging is enabled (ie. getPortletLog().isErrorEnabled()). You should remove all if statements testing for logging enabled. Logging for JSR 168 portlets should use getPortletContext().log() instead of getPortletLog().error().

10.2.7 JSP includes

In JSR 168 portlets, the MIME type is set in the render (doView, doHelp, doEdit, and doConfigure) method. JSPs and Servlets are included using a PortletRequestDispatcher. Change your JSP includes to look like the following:

Example 10-8 JSP includes

```
response.setContentType("text/html");
PortletContext portletContext = getPortletContext();
PortletRequestDispatcher rd =
portletContext.getRequestDispatcher(VIEW_JSP+getJspExtension(request));
rd.include(request, response);
```

Archived

The getJSPExtension() method in IBM portlets is used to get the file extension. For JSR 168 portlets, you should change this method to just return a String with the value `jsp`.

10.2.8 PortletData and PortletSettings

JSR 168 portlets use a PortletPreferences object in replace of PortletData and Portletsettings. All instances of the PortletData should be changed to use PortletPreferences.

Replace:

Example 10-9 PortletData

```
PortletData portletData = request.getData();
portletData.setAttribute("name", new String("Hailey"));
portletData.store();
```

with:

Example 10-10 PortletPreferences

```
PortletPreferences preferences = request.getPreferences();
preferences.seValue("name", new String("Hailey"));
preferences.store();
```

10.2.9 namespace

The namespace concept for both portlet APIs are used in the same way. They do, however, use different methods.

Change:

```
response.encodeNamespace();
```

to:

```
response.getNamespace();
```

10.2.10 portlet URLs

JSR 168 portlets use PortletURL objects for creating links. There are two types of PortletURLs, ActionURL and RenderURL. Use ActionURLs to create links to invoke and action request and use a RenderURL to invoke a render request. You can use the PortletURL objects to set the portlet mode, window state, and set parameters.

Archived

Use the following to create PortletURLs.

Example 10-11 PortletURL

```
PortletURL myURL = response.createRenderURL();
myURL.setPortletMode(PortletMode.EDIT);
myURL.setWindowState(WindowState.MAXIMIZED);
myURL.setParameter("myParam", "My Parameter");
```

10.3 Modifying the JSP source

The majority of your JSPs will probably not need to be changed. Both the IBM portlet API and the JSP 168 portlet API can leverage the Servlet 2.3 and JSP 1.2 specifications. You will need to change from the IBM JSP custom tags to the JSR custom tags.

10.3.1 taglib

The tag library needs to be changed to point to the JSR 168 tld and not the IBM tld.

For all JSP using portlet tag, change:

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

to:

```
<%@taglib uri="http://java.sun.com/portlet" prefix="portlet" %>
```

You can leave the prefix as `portletAPI` if you choose.

You will then need to define the tld in the web.xml. The tagib element needs to be placed after the error-page element. If you are not using an error-page element, you can place the taglib element after the welcome-file-list. Example 10-12 shows the taglib element in the web.xml.

Example 10-12 taglib element

```
<welcome-file-list>
   .....
   <welcome-file>default.jsp</welcome-file>
</welcome-file-list>
<taglib id="PortletTLD">
   <taglib-uri>http://java.sun.com/portlet</taglib-uri>
   <taglib-location>/WEB-INF/tld/std-portlet.tld</taglib-location>
</taglib>
```

Archived

10.3.2 portletAPI:init

The IBM portlet API uses the init tag to defines the following objects.

- portletRequest of type PortletRequest
- portletResponse of type PortletResponse
- portletConfig of type PortletConfig

JSR 168 uses the portlet:defineObjects tag to generate object for use in the JSP. JSP are called during the render phase. The defineObjects tag defines the following objects.

- renderRequest of type RenderRequest
- renderResponse of type RenderResponse
- portletConfig of type PortletConfig.

Although both tags define a portletConfig variable, they are different types of PortletConfigs.

All JSP scriptlets containing portletRequest and portletResponse should be changed to use renderRequest and renderResponse.

10.3.3 namespace

Both APIs use namespacing to uniquely identify portlets on a page. The portletAPI:encodeNameSpace tag and the portlet:namespace tag have the same function, but they are implemented quite differently. While the IBM API tag take a value attribute used to concant to the end of the namespace, the JSR tag will just return the namespace without accepting an attribute.

Example 10-13 Namespace using tags

```
IBM API namespace
<portletAPI:encodeNamespace value=”name” />

JSR 168 namespace
<portlet:namespace/>
```

You can also use the method the RenderResponse class to get the namespace.

Example 10-14 Namespace using PortletResponse objects.

```
IBM API namespace method
response.encodeNamespace("name");

JSR 168 API
response.getNamespace();
```

Archived

10.3.4 Creating URLs

JSR 168 Portlets define two types of PortletURLs. An ActionURL used to generate an action request and a RenderURL used to generate a render request. An action request will cause the processAction method to be invoked. As discussed in “actionPerformed” on page 341, you should update the name of the submit buttons for your forms. The action value for forms should be an ActionURL. You can either use the JSP or the RenderResponse object to create the URL. RenderURLs will invoke a render request and cause the render method of the portlet to be called. The render method invoke the doDispatch which in turn calls one of the doXXX methods. Through the PortletURLs, you can also change the portlet mode, window state, and add parameters. Example 10-15 demonstrates creating an ActionURL in a form.

Example 10-15 ActionURL in a form

```
Creating an ActionURL using the JSP tag:
<FORM method="POST" action="<portlet:actionURL />" >
    <LABEL class="wpsLabelText"   for="<portlet:namespace
/><%=MyPortletPortlet.TEXT%>">Enter order id:</LABEL><BR>
    <INPUT class="wpsEditField"  name="<portlet:namespace
/><%=MyPortletPortlet.TEXT%>" type="text"/>
    <INPUT class="wpsButtonText" name="<portlet:namespace
/><%=MyPortletPortlet.FORM_SUBMIT%>" value="Submit" type="submit"/>
  </FORM>

Creating an ActionURL using the RenderResponse object:
<FORM method="POST" action="<%= renderResponse.createActionURL() %>" >
    <LABEL class="wpsLabelText"   for="<portlet:namespace
/><%=MyPortletPortlet.TEXT%>">Enter order id:</LABEL><BR>
    <INPUT class="wpsEditField"  name="<portlet:namespace
/><%=MyPortletPortlet.TEXT%>" type="text"/>
    <INPUT class="wpsButtonText" name="<portlet:namespace
/><%=MyPortletPortlet.FORM_SUBMIT%>" value="Submit" type="submit"/>
  </FORM>
```

In Example 10-16 we will create a RenderURL, setting the portlet mode to Edit and the window state to maximized. We will also set some render parameters.

Example 10-16 RenderURL

```
Creating a RenderURL using the JSP tags:
<a href="<portlet:renderURL portletMode="EDIT" windowState="MAXIMIZED">
<portlet:param name="myParam" value="myValue"/>
</portlet:renderURL>">My RenderURL</a>

Creating a RenderURL using the RenderResponse object:
<% PortletURL myRenderURL = renderResponse.createRenderURL();
```

```
   myRenderURL.setPortletMode(PortletMode.EDIT);
   myRenderURL.setWindowState(WindowState.MAXIMIZED);
   myRenderURL.setParameter("myParam", "myValue"); %>
<a href="<%=myRenderURL %>">My RenderURL</a>
```

10.3.5 portletAPI:text

The text tag of the IBM API is deprecated in favor of using JSTL.

10.3.6 encodeURL

The encodeURL method is used in both APIs. However, in JSR 168, the encodeURL must be passed a root relative path. You should use the RenderRequest.getContextPath() method before the path of the included file. In Example 10-17, we will include the myimage.gif image under the images directory.

Example 10-17 encodeURL

```
<%= renderResponse.encodeURL(renderRequest.getContextPath() +
"/images/myimage.gif") %>
```

10.3.7 CSS

We recommend using WSRP styles whenever possible when developing JSR 168 portlets. In WebSphere Portal, all consumed remote portlet are treated as JSR 168 portlets. By using WSRP styles, this help ensure that your portlet will display correctly if provided as a remote portlet on another portal server. If your portlet is consumed on a different vendors portal server, the IBM specific classes specified on your JSP/HTML would not be picked up. This could cause appearance problems in the new environment.

10.4 Struts

A Struts portlet configuration consists of a specific Portlet class to extend from, some init-params to configure it, as well as an specific RequestProcessor class in struts-config.xml. This configuration is slightly different from the IBM Portlet API to the JSR 168 API. In Chapter 15, “Struts portlet development using the JSR 168 API” on page 465 you can find more details that will help you to migrate a Struts Portlet application based on the IBM Portlet API to a Struts Portlet application based on the JSR 168 API.

10.5 JSF

JSF portlet applications have a portlet.xml with a FacesPortlet defined, as well as some init-params to specify the initial page for each portlet mode. The faces-config.xml file for JSF portlet applications must define a FacesContextFactory, as well as some variable and property resolvers. In Chapter 16, “JavaServer Faces portlets” on page 511, you can find more details that will help you to migrate JSF applications to JSF portlet applications that use the JSR 168 portlet API

10.6 Portlet services

JSR 168 portlets can access portlet services in WebSphere Portal version 5.1. Your portlet service will need to be rewritten to work with JSR 168. You can, however, write the portlet so that you can still use it with IBM portlets. The service provider interfaces from org.apache.jetspeed.portlet.service.spi are still supported to provide IBM portlets access to services.

You will have to create a new portlet service interface extending com.ibm.portal.portlet.service.PortletService.

Example 10-18 JSR Portlet Service interface

```
package example.portletservice;

import javax.portlet.RenderRequest;
import javax.portlet.RenderResponse;

import com.ibm.portal.portlet.service.PortletService;

public interface MyJSRPortletService extends PortletService {
   public void HelloService(RenderRequest request, RenderResponse response);

}
```

You can use the same portlet service implementation with a few modifications. You will need to verify that the service implementation implements com.ibm.portal.portlet.service.spi.PortletServiceProvider. This interface provides the methods for the life cycle of the portlet service. You will also need to implement your new JSR portlet service interface. If you still need to use this service for IBM portlets, you can also implement you IBM Portlet API portlet service interface. WebSphere portal provides a class for converting IBM Portlet API object into JSR 168 objects. You can use this class to wrap your JSR 168 objects and pass them into the IBM methods of the portlet service

implementation. This allows only having to maintain code in only one method, be sure that you method behaves the same using this approach.

Example 10-19 JSR Portlet Service implementation

```
package example.portletservice;

import java.io.IOException;

import javax.portlet.RenderRequest;
import javax.portlet.RenderResponse;

import org.apache.jetspeed.portlet.PortletRequest;
import org.apache.jetspeed.portlet.PortletResponse;
import org.apache.jetspeed.portlet.service.PortletServiceUnavailableException;
import org.apache.jetspeed.portlet.service.spi.PortletServiceConfig;
import org.apache.jetspeed.portlet.service.spi.PortletServiceProvider;

import com.ibm.portal.portlet.apiconvert.APIConverterFactory;

public class MyPortletServiceImpl implements MyPortletService,
      PortletServiceProvider, MyJSRPortletService {

   public void HelloService(PortletRequest request, PortletResponse response)
{
      //moved the code from here to HelloService for JSR 168 portlets

      //call the JSR 168 method using the APIConverterFactory class

HelloService(APIConverterFactory.getInstance().getRenderRequest(request),

APIConverterFactory.getInstance().getRenderResponse(response));
   }

   public void HelloService(RenderRequest request, RenderResponse response){
      try {
         response.getWriter().println("Hello World from a Portlet Service!");
      } catch (IOException e) {
         e.printStackTrace();
      }
   }

   public void init(PortletServiceConfig arg0)
         throws PortletServiceUnavailableException {
   }

   public void destroy() {
   }
```

Archived

```
}
```

JSR 168 portlets will register portlet services using JNDI. As with IBM Portlet API portlet services, you should place your JAR file under the <WebSphere Portal>/shared/app directory. You will then need to modify the PortletServiceRegistryService.properties file in <WebSphere Portal>/shared/app/config/services directory. If you are still using the service for IBM portlets you can leave the current entry and create a new entry for use with JSR 168 portlets. The main difference is that when defining the portlet service, you need to prefix the service interface with `jndi:`.

Example 10-20 Registering JSR 168 portlet service

```
#My portlet service
example.portletservice.MyPortletService =
example.portletservice.MyPortletServiceImpl
jndi:example.portletservice.MyJSRPortletService =
example.portletservice.MyPortletServiceImpl
```

Note: In Example 10-20, we use the same implementation class for both portlets. We can do this because we defined methods for both portlet APIs in our implementation class.

You will need to modify your portlet to use a JNDI lookup to get the portlet service. Remove all code from your portlet that retrieves portlet services. The object returned from the JNDI lookup will be of type PortletServiceHome. From the PortletServiceHome object you will be able to access your service. You should perform the lookup in the init method of your portlet. This may take time and only needs to be performed once.

Example 10-21 Retreiving a portlet service from a JSR 168 portlet

```
package myjsrportlet;

import java.io.IOException;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NameNotFoundException;
import javax.naming.NamingException;
import javax.portlet.GenericPortlet;
import javax.portlet.PortletConfig;
import javax.portlet.PortletException;
import javax.portlet.RenderRequest;
import javax.portlet.RenderResponse;

import com.ibm.portal.portlet.service.PortletServiceHome;
```

```
import example.portletservice.MyJSRPortletService;

public class MyJSRPortletPortlet extends GenericPortlet {

   private PortletServiceHome myServiceHome = null;

   public void init(PortletConfig config) throws PortletException {
      super.init(config);
      try{
         Context ctx = new InitialContext();
         Object home =
ctx.lookup("portletservice/example.portletservice.MyJSRPortletService");
         if(home != null){
            myServiceHome = (PortletServiceHome)home;
         }
      } catch(NameNotFoundException nnf){
         config.getPortletContext().log("MyJSRPortletService is not
available");
         nnf.printStackTrace();
      } catch(NamingException ne){
         ne.printStackTrace();
      }
   }

   public void doView(RenderRequest request, RenderResponse response) throws
PortletException, IOException {
       response.setContentType("text/html");
      MyJSRPortletService myService =
(MyJSRPortletService)myServiceHome.getPortletService(MyJSRPortletService.class)
;
       myService.HelloService(request, response);
   }
}
```

10.7 Messaging

Portlets using the Messaging interfaces should change to use a cooperative portlet. The JSR specification does not define a messaging listener and therefore the only option is to use cooperative portlets. See Chapter 26, “JSR 168 cooperative portlets” on page 801 for more information about the cooperative portlet.

Archived

Archived

11

Using JSPs and servlets

In this chapter, we will discuss using JSPs and servlets with JSR 168 portlets. This chapter contains the following sections.

- Overview
- Generating output
- RequestDispatcher
- JSP tags
- Cascading Style Sheets (CSS)

Archived

© Copyright IBM Corp. 2005. All rights reserved.

11.1 Overview

Portlets follow the Model View Controller (MVC) design pattern. In this design pattern, portlets will usually use JSPs to generate the output. A portal page can be composed of many portlets along with the theme and skins.

11.1.1 Generating output

Portlets use the PortletRequestDispatcher.include() method to delegate content generation to JSPs or servlets.

Portlets are displayed on a portal page by using themes and skins. To allow portlets to dynamically match the current theme/skin in which they are contained, portlets should take advantage of cascading style sheets (CSS). When developing JSR 168 portlets, you should make every effort to use a WSRP style if available. You should also avoid using absolute positioning with CSS. The dynamic nature of portal cannot guarantee that your portlet will be displayed in a certain position on the page.

The page that a user sees when accessing a portal page is the result of an aggregation of all the portlets on a page with the themes and skins. Each portlet should contain JSPs that are well formed. If a JSP is not well formed, it could result in other portlets and possible the entire page not displaying correctly. You should use standard HTML tags with no <html>,<head>, or <body> tags. Only elements that can be displayed in the cell of a table can be rendered. Be aware of the size of the portlet when it is rendered, this portlet may share a page with other portlets and may cause problems if the rendered output of the portlet is too big.

Use Java style comments rather than HTML comments, this produces a smaller page to be sent to the client, improving performance. Also the comment will not be visible to the user when viewing the HTML source.

IFRAMEs should be used with caution. While they do provide solutions to some special cases, not all browsers support them. Also the content in an IFRAME may be displayed with scroll bars if the content is larger than the allowed display size. This can cause an unsightly portlet.

Whenever possible, avoid the use of pop-up windows. Pop-ups may cause problems with the current portlets state. Pop-ups are acceptable when the target for the new window is outside portal.

Example 11-1 JavaScript to open a URL in a new window

```
<a href="http://www.ibm.com" target="new">IBM</a>
```

Portlets should be designed to be accessible to users with disabilities. the following coding practices should be applied:

- Do not use only color or sound alone to convey information.
- Use labels with forms to match the description with the information requested.

  Example 11-2 uses labels within a form. The label goes with the “OrderID” form field.

Example 11-2 Using labels with a form

```
<FORM method="POST" action="<portlet:actionURL/>">
   <LABEL  for="OrderID">Enter order id:</LABEL><BR>
   <INPUT name="OrderID" type="text"/>
   <INPUT name="SubmitButton" type="submit" value="Submit"/>
</FORM>
```

- Use the ALT attribute with images to provide a description of the image for those who cannot see it.

JSR 168 provides a set of JSP tags to use while developing pages. These tags along with JSTL will reduce the amount of JSP scripting elements in the page. This also enables developers who are not familiar with JSP scripting elements to still create dynamic JSPs

All JavaScript variable and function names should be namespaced to avoid conflict with other portlets on the same page. The namespace should be added before element name. You can either use the JSP tag <portlet:namespace/> or RenderResponse.getNamesapace() method.

Portlets must use the methods of the portlet API to create links to other resources within the portlet (images, JSPs, applets, etc.). The getEncodeURL method of the RenderResponse object will create the correct link. You must pass a root relative path to this method. To obtain the correct path use the RenderRequest.getContextPath() method to prepend your path.

Example 11-3 Creating a link to an image from a jsp

```
<img src="<%= renderResponse.encodeURL(renderRequest.getContextPath() +
"/images/smile.jpg")%>”alt=”Smile”/>
```

Although your portlets can also generate output for WML and cHTML, we will not discuss these topics here.

Archived

11.2 RequestDispatcher

JSR 168 portlets use the PortletRequestDispatcher to delegate responsibility to servlets and JSPs. The PortletRequestDispatcher can only be used while the portlet is processing a render action. The request dispatcher is obtained from the portlet context by using either the getRequestDispatcher or the getNamedDispatcher method.

11.2.1 PortletContext.getRequestDispatcher

The getRequestDispatcher method takes a String representing the context root relative path of the included file. This path must begin with a '/'. This method will return null if the path is not found.

While using the getRequestDispatcher method, you may include parameters in the path. For example, if your path is /mypath/myjsp.jsp, you can include parameters by adding them to the end of the path /mypath/myjsps.jsp?name=Hailey. Parameters in the URL take precedence over any other render parameters of the same name. These parameters are only in scope for the current request.

Example 11-4 getRequestDispatcher

```
Code in the portlet
String path = "/jsptest/jsp/html/testCase.jsp?name=Hailey";
PortletRequestDispatcher rd = getPortletContext().getRequestDispatcher(path);
rd.include(request,response);

Code in JSP
Hello <%= renderRequest.getParameter("name") %>
```

11.2.2 PortletContext.getNamedDispatcher

The getNamedDispatcher will take the name of a servlet rather than a path. This servlet must be a known servlet and registered in the web deployment descriptor. This method will return null if the servlet is not found.

11.2.3 PortletRequestDispatcher.include

The include() method of the PortletRequestDispatcher class is used to include JSPs and servlets. The RenderRequest and RenderResponse objects passed into the render method must be passed into the include() method.

The include() method can be called at any time during the render method of the portlet.

Note: The included JSPs and Servlets should not use the Servlet RequestDispatcher to do a forward as this may cause unpredictable behavior.

JSPs and Servlets called from the include method will have access to the following request parameters. However, if the RequestDispatcher for a servlet was obtained using the getNamedDispatcher the following attributes will not be available.

All of the following attribute and parameter can be obtained from the requests getAttribute method.

The following five request parameters have corresponding methods in the HttpServletRequest to get the information.

javax.servlet.include.request_uri

The request_uri is the context root relative path of the included JSP or servlet

javax.servlet.include.context_path

The context_path in the context root for the portlet.

javax.servlet.include.servlet_path

The servlet path will be the request_uri minus the context_path

javax.servlet.inclue.path_info

path_info contains information about the path.

javax.servlet.include.query_string

The query_string will contain any parameters that have been added to the path. For example if you had a path of /mypath/myjsps.jsp?name=Hailey, the query_string would contain name=Hailey.

The following three attributes will store portlet objects. When using JSPs it is recommended that you use the defineObjects portlet tags rather than getting the objects from the request.

Example 11-5 Request attributes

```
RenderRequest renderRequest   =
(RenderRequest)request.getAttribute("javax.portlet.request");
RenderResponse renderResponse =
(RenderResponse)request.getAttribute("javax.portlet.response");
PortletConfig portletConfig   =
(PortletConfig)request.getAttribute("javax.portlet.config");
```

Archived

javax.portlet.config

Is an object of type PortletConfig

javax.portlet.request

Is an object of type RenderRequest

javax.portlet.response

Is an object of type RenderResponse

Request and Response objects of included JSPs and Servlets

An included JSP and Servlet does have access to some of the functionality of the request and response objects.

HttpServletRequest

The following methods will perform no actions and or return null

- getProtocol
- getRemoteAddr
- getRemoteHost
- getRealPath
- getRequestURL
- getCharacterEncoding
- setCharacterEncoding
- getContentType
- getInputStream
- getreader
- getContentLength will always return zero

The following methods will be used to return the request parameters listed above

- getPathInfo

  javax.servlet.include.path_info

- getQueryString

  javax.servlet.include.query_string

- getRequestURI

  javax.servlet.include.request_uri

- getServletPath

  javax.servlet.include.servlet_path

The methods of the HttpServletrequest method for handling parameters will be the same as the methods of PortletRequest.

Archived

- getParameter
- getParameterNames
- getParameterValues
- getParameterMap

The properties defined in PortletRequest will be used for the following HttpServletRequest methods.

- getHeader
- getHeaders
- getHeaderNames
- getCookies
- getDateHeader
- getIntHeader

The following list of HttpServletRequest methods will be the as the methods of the PortletRequest with the same or similar name.

- getScheme
- getServerName
- getServerPort
- getAttribute
- getAttributeNames
- setAttribute
- removeAttribute
- getLocale
- getLocales
- isSecure
- getAuthType
- getcontextPath
- getRequestedSessionId
- isRequestedSessionIdValid
- getRequestDispatcher
- getMethod - (this will always return GET)
- getSession
- isRequestedSessionIdFromCookie
- isRequestedSessionIdFromURL
- isRequestedSessionIdFromUrl

HttpServletResponse

The following methods in HttpServletResponse will return null.

- encodeRedirectURL
- encodeRedirectUrl

The following methods in HttpServletResponse will not perform any actions:

Archived

- setContentType
- setContentLength
- setLocale
- addCookie
- sendError
- sendRedirect
- setDataHeader
- addDataHeader
- setHeader
- addHeader
- setIntHeader
- addIntHeader
- setStatus
- containsHeader - (will always return false)

The following list of HttpServletResponse methods will be the as the methods of the PortletResponse with the same or similar name.

- getCharacterEncoding
- setBufferSize
- flushBuffer
- resetBuffer
- reset
- getBufferSize
- isCommited
- getOutputStream
- getWriter
- encodeURL
- endoceUrl

11.3 JSP tags

The JSP tags provided for JSR 168 portlets allow the portlet developers to rely less on using JSP scripting elements. The JSR 168 specification provides a set of custom tags. WebSphere Portal also provides an extended JSP tag for JSR 168 portlets. The Servlet specification also defines the Java Standard tag Library (JSTL). JSTL is a set of standard tags used for logic and presentation in your JSPs.

The following tags are defined by the JSR 168 specification. These tags will allow developers access to the portlet objects such as the renderRequest and the renderResponse objects. They also allow for creating render and action URLs, and encoding namespaces. You must use the following taglib directive in your JSP in order to use these tags.

Archived

```
<@ taglib uri=”http://java.sun.com/portlet” prefix=”portlet” %>
```

11.3.1 defineObjects

The defineObjects tag will create the following objects.

- renderResponse of type javax.portlet.RenderResponse
- renderRequest of type javax.portlet.RenderRequest
- portletConfig of type javax.portlet.PortletConfig

After invoking the defineObjects tag on a page, you can use any of the defined variables in Java scriplets. In Example 11-6, after the defineObjects tag has been called we use the renderResponse object to create a render URL. We also set the portlet mode to EDIT, the window state to maximized, and we added a RenderRequest parameter named “edit_param“.

Example 11-6 Using renderResponse to create a render action

```
<%@ page session="false" contentType="text/html"
   import="java.util.*,javax.portlet.*,jsptest.*" %>
<%@taglib uri="http://java.sun.com/portlet" prefix="portlet" %>
<portlet:defineObjects/>
<% PortletURL url = renderResponse.createRenderURL();
   url.setPortletMode(PortletMode.EDIT);
   url.setParameter("edit_param","This is an edit param!");
   url.setWindowState(WindowState.MAXIMIZED);
%>
<a href="<%= url.toString() %>">Render
URL</a>
```

11.3.2 renderURL

The renderURL tag will generate a URL that triggers a render action pointing to this portlet. You can set the portlet mode, window state and pass parameters using this tag.

The following attributes are optional.

windowState

This attribute will set the window state. For WebSphere portal the only supported states are MINIMIZED, MAXIMIZED, and NORMAL. A JSPException will be thrown if the window state is not valid or the portlet does not specify the window state as supported. If no windowState is specified, the windowState will remain the same as the current request. This attributes does accept a runtime expression for the value (`windowState=”<%= newWindowState %>”`).

portletMode

This indicates the mode of the portlet after executing the render action. WebSphere Portal support VIEW, EDIT, HELP, and the custom mode CONFIG. A JSPException will be thrown if the mode specified is not a mode supported by WebSphere portal or the mode is not supported by the current portlet. If the portlet mode is not specified, the portlet mode will remain the same as the same as the current request. This attributes does accept a runtime expression for the value (`portletMode="<%= newPortletMode %>"`).

var

The var attribute will create a variable with the provide name. This variable will be stored in the page scope and provided the value of the created URL. This attributes does accept a runtime expression for the value (`var="<%= varName %>"`).

Note: When in the source view in Rational Application Developer, when using this variable in a scriplet (<%= myVar %>) you may see a red box on the right side of the screen stating that myVar variable cannot be resolved. This will be resolved during runtime and you should not be concerned about the warning.

secure

The secure attribute will make use of the setSecure method and is currently not supported in WebSphere Portal.

Example 11-7 will create a URL using the renderURL tag. The URL generated is the same as the URL generated in Example 11-6 on page 361.

Example 11-7 Using the renderURL tag to generated a render action

```
<%@ page session="false" contentType="text/html"
import="java.util.*,javax.portlet.*,jsptest.*" %>
<%@taglib uri="http://java.sun.com/portlet" prefix="portlet" %>
<portlet:defineObjects/>
<portlet:renderURL portletMode="EDIT" windowState="maximized" var="myURL">
<portlet:param name="edit_param" value="This is an edit parameter"/>
</portlet:renderURL>
<a href="<%= myURL %>">Render URL</a>
```

11.3.3 actionURL

The actionURL tag will create a URL that triggers the action phase for this portlet. You can set the portlet mode, window state, and pass parameters.

The following attributes are optional.

windowState

This attribute will set the window state. For WebSphere portal the only supported states are MINIMIZED, MAXIMIZED, and NORMAL. A JSPException will be thrown if the window state is not valid or the portlet does not specify the window state as supported. If no windowState is specified, the windowState will remain the same as the current request. This attributes does accept a runtime expression for the value (`windowState=”<%= newWindowState %>”`).

portletMode

This indicates the mode of the portlet after executing the render action. WebSphere Portal support VIEW, EDIT, HELP, and the custom mode CONFIG. A JSPException will be thrown if the mode specified is not a mode supported by WebSphere portal or the mode is not supported by the current portlet. If the portlet mode is not specified, the portlet mode will remain the same as the same as the current request.This attributes does accept a runtime expression for the value (`portletMode =”<%= newPortletMode %>”`).

var

The var attribute will create a variable with the provide name. This variable will be stored in the page scope and provided the value of the create URL. This attributes does accept a runtime expression for the value (`var=”<%= varName %>”`).

Note: When in the source view in Rational Application Developer, when using this variable in a scriplet (`<%= myVar %>`) you may see a red box on the right side of the screen stating that myVar variable cannot be resolved. This will be resolved during runtime and you should not be concerned about the warning

secure

The secure attribute will make use of the setSecure method and is currently not supported in WebSphere Portal.

11.3.4 namespace

The namespace tag should be used for named elements to ensure that they are associated with the current portlet. Typically you would namespace JavaScript functions, variables. Namespacing uniquely identifies that the element belongs to the portlet and avoids conflicts with other portlets on the page. The namespace tag does not contain a body and it does not have any attributes.

Example 11-8 Using namespace with JavaScript elements

```
<script language="JavaScript">
var <portlet:namespace/>userName = "Hailey";
```

```
function <portlet:namespace />helloUser(){
alert(<portlet:namespace/>userName);
}
</script>
<INPUT type="button" value="Click Me"
onclick="<portlet:namespace/>helloUser()"/>
```

11.3.5 param

The param tag defines a renderURL or an actionURL parameter. Example 11-7 on page 362 shows the param tag used in the body of a renderURL tag. The param tag has two required attributes.

The following attributes are required.

name

The name attribute will be used to name the parameter. If this parameter is null or empty, the parameter will not be added. This attribute can be expressed by a runtime value (`name=”<%= myParameterName %>`).

value

The value attribute will represent the String value for the named parameter. If this value is null or empty it will be treated as an empty String. This attribute can also be expressed by a runtime value (`value=”<%= myParameterValue %>`).

11.3.6 IBM tags

WebSphere Portal provides an additional tag for use with JSR 168 portlets. This tag is used for bidirectional languages.

To use this tag, you must include the following tag library.

```
<@ taglib uri=”/WEB-INF/tld/ibm-portlet-ext.tld” prefix=”portletExt” %>
```

bidi

Bidirectional languages are read from right to left. The bidi tag is used to support the bidirectional language.

The following attributes are not required.

dir

The dir attribute specifies the direction for the language. The only two supported values are rtl for right to left and ltr for left to right. The default for this attribute is rtl.

Archived

locale

The locale must be specified in the LocalizerService.properties. This attribute can be expressed by a runtime value (for example locale=”<%= myLocale %>).

11.3.7 JSTL

Java Standard Tag Library (JSTL) is a set of custom tags included in the Servlet specification. These tags provide common functionality used when developing JSPs. While JSTL is not specific to portlets, you can use them while developing the JSPs in your portlet. Although the complete JSTL specification is outside the scope if this book, we will try to give an overview and some examples.

JSTL tags allows portlet developers to use less JSP scripting elements in their JSPs. This is cleaner and less prone to errors. It also enables developers who are not Java programmers to add dynamic content. Although JSTL does provide allot of functionality, there still may be times that scripting cannot be avoided.

The JSTL is composed of four libraries.

core

The core is used for basic functionality such as conditional statements. You can also perform looping and input/output operations from this library. To use the core library you must declare the following taglib directive in your JSP

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
```

xml

The xml library is used for xml processing. You must include the correct taglib directive to use the xml library.

```
<%@ taglib uri="http://java.sun.com/jstl/xml" prefix="x"%>
```

fmt

The format library is used for formatting currency, dates, and other values. This library is also used for internationalization. The following taglib directive is used to declare the format library.

```
<%@ taglib uri="http://java.sun.com/jstl/fmt" prefix="fmt"%>
```

Example 11-9 fmt

```
<%@ taglib uri="http://java.sun.com/jstl/fmt" prefix="fmt" %>
<fmt:setBundle basename="nls.NLSExample"/>
<fmt:message key="message"/>
```

Archive

sql

The sql library is used for queries and database access. The following taglib directive is used for specifying the sgl library.

```
<%@ taglib uri="http://java.sun.com/jstl/sql" prefix="sql"%>
```

Example 11-10 show a JSTL loop. The number is printed along with a line stating if the number is even or odd.

Example 11-10 JSTL example

```
<c:forEach var="i" begin="1" end="10" step="1">
   <c:if test="${ (i % 2) == 0}">
      <c:out value="${i} is an even number!" />
   </c:if>
   <c:if test="${ (i % 2) != 0}">
      <c:out value="${i} is an even number!" />
   </c:if>
<br />
</c:forEach>
```

11.4 Cascading style sheets (CSS)

Cascading style sheets (CSS) allows Web developer to control the look and feel of Web pages. Many pages can be associated with a single CSS file. If changes to a sites appearance are required they only have to updated in one place. WebSphere portlet uses CSS to give a common look for themes and skins. JSR 168 portlets should use the CSS style defined for the WSRP specification whenever possible. This allows the portlets to alter their appearance dynamically depending on the theme/skin they are displayed in. IBM also provide many classes for styles, but once again use the WSRP style whenever possible.

CSS files are located in the <WebSphere Application Server>\installedApps\localhost\wps.ear\wps.war\themes directory. The correct CSS file is used based on the theme name, locale, and client.

We will not explain how CSS works, but we will explain the WSRP styles and where they should be used.

11.4.1 WSRP Styles

WSRP styles are a set of styles defined by the WSRP specification.

Archived

Links

Table 11-1 Anchors

Tag	Style
<a>	There are no styles defined for an anchor.

Fonts

Table 11-2 Fonts

Style	Description
portlet-font	This should be used for normal fonts.
portlet-font-dim	This will be lighter than the portlet-font

Example 11-11 Font styles

```
<div class="portlet-font">
 Hello World!<br />
</DIV>
<div class="portlet-font-dim">
 Hello dimmer World!<br />
</div>
```

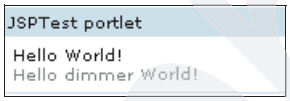


Figure 11-1 Font styles

Messages

Message style are using to modify the appearance of paragraphs.

Table 11-3 Messages

Style	Description
portlet-msg-status	Displaying current status
portlet-msg-info	Additional information
portlet-msg-error	Error messages
portlet-msg-alert	Warning messages
portlet-msg-success	Task completed successfully

Archived

Sections

Section style should be used with division (div) and span HTML tags.

Table 11-4 Sections

Style	Description
portlet-section-header	Section header
portlet-section-body	Normal text
portlet-section-alternate	Text for alternating rows
portlet-section-selected	Selected text
portlet-section-subheader	Subheading text
portlet-section-footer	Section footer
portlet-section-text	All other text that does not fall into one of the previous styles. This could be used for additional information.

Tables

Table styles should be used for HTML table tags.

Table 11-5 Table

Style	Description
portlet-table-header	Table header
portlet-table-body	Text in the cell of a table
portlet-table-alternate	Alternate rows of a table
portlet-table-selected	Selected text in a cell
portlet-table-subheader	Subheading text
portlet-table-footer	Table footer
portlet-table-text	All other text that does not fall into one of the previous styles. This could be used for additional information.

Example 11-12 Table styles

```
<div class="portlet-table-text">
This is my Table.
</div>
<TABLE border="1" width="456">
```

Archivos

```
   <TBODY class="portlet-table-body">
      <TR class="portlet-table-header">
         <TD  width="173">First Name</TD>
         <TD  width="275">Last Name</TD>
      </TR>
      <TR >
         <TD width="173">Bob</TD>
         <TD width="275">Smith</TD>
      </TR>
      <TR class="portlet-table-alternate">
         <TD width="173">Ed</TD>
         <TD width="275">Jones</TD>
      </TR>
      <TR>
         <TD width="173">Sue</TD>
         <TD width="275">Johnson</TD>
      </TR>
   </TBODY>
</TABLE>
```

Forms

Table 11-6 Forms

Style	Description
portlet-form-label	Label for the entire form and not a label specific to a field.
portet-form-input-field	Text of an input field
portlet-form-button	Text of a button
portlet-icon-label	Text surrounding an action icon
portlet-dlg-icon-label	Text surrounding a standard button such as OK or cancel.
portlet-form-field-label	Text for a field label
portlet-form-field	Text for a field other than an input field such as a checkbox or radio button

Example 11-13 WSRP form styles

```
<DIV style="margin: 6px">
<p class="portlet-form-label">Please fill out your information:</p>
<form name="<portlet:namespace />" action="<portlet:actionURL/>">
Please enter your gender: <br />
<label class="portlet-form-field-label" for="<%= WSRPTestPortlet.GENDER
%>">Male</label>
```

Archived

```
<input class="portlet-form-field" type="radio" name="<%= WSRPTestPortlet.GENDER
%>" value="Male"><br />
<label class="portlet-form-field-label" for="<%= WSRPTestPortlet.GENDER
%>">Female</label>
<input class="portlet-form-field" type="radio" name="<%= WSRPTestPortlet.GENDER
%>" value="Female"><br />

<label class="portlet-form-field-label" for="<%= WSRPTestPortlet.FIRST_NAME
%>">First name:</label>
<input class="portlet-form-input-field" type="text" name="<%=
WSRPTestPortlet.FIRST_NAME %>" /><br />
<label class="portlet-form-field-label" for="<%= WSRPTestPortlet.LAST_NAME
%>">Last name:</label>
<input class="portlet-form-input-field" type="text" name="<%=
WSRPTestPortlet.LAST_NAME %>" /><br />
<input class="portlet-form-button" type="submit" name="<%=
WSRPTestPortlet.SUBMIT %>" value="Enter" />
</form>
</DIV>
```

Menus

Table 11-7 Menus

Style	Description
portlet-menu	Menu settings
portlet-menu-item	Deselected menu item
portlet-menu-item-selected	Selected menu item
portlet-menu-item-hover	Mouse over on an deselected menu item
portlet-menu-item-hover-selected	Mouse over on a selected menu item
portlet-menu-cascade-item	Deselected menu item with submenus
portlet-menu-cascade-item-selected	Selected submenu item
portlet-menu-description	Description of the menu
portlet-menu-cation	Menu caption

11.4.2 IBM styles

WebSphere portal provided a large list of classes for use within portlets, too numerous to list here. Once again, it is recommended that you use the WSRP style when available.

Archived

The Style.css will be located in was_root/installedApps/hostname/wps.ear/wps.war/themes/html. In this Style.css you will be able to find both IBM and WSRP styles and a description about the style. WebSphere portlet also provides a HelpStyle.css to provide a constant look and feel to the Help mode of portlets.

During aggregation, the most specific directory is searched first. If a match is not found there then the search gets more general until a match is found. This allows the creation of style for different scenarios. Example 11-14 shows the order of the search.

Example 11-14 Order of precedence during aggregation

1. locale_region
2. locale
3. client
4. theme_name
5. markup

For example, if you use a style that behaves differently in Internet Explorer, you could change that Style under the ie directory while leaving the style in the directory below untouched. Now when an Internet Explorer client access the page they will see the correct style.

Let's say we are using the Science theme and need to create a new class and a special value for clients using Internet Explorer. We could update the class under the ie directory.

WebSphere portlat will search the following directories for the styles. You would need to update the Style.css in the most specific directory under ie in order to see the changes for Internet Explorer. All other browser will pick up the Style.css from the most specific directory not containing ie.

- themes/html/Science/ie/en/en_US/Style.css
- themes/html/Science/ie/en/Style.css
- themes/html/Science/ie/Style.css
- themes/html/Science/Style.css
- themes/html/ie/en/en_US/Style.css
- themes/html/ie/en/Style.css
- themes/html/ie/Style.css
- themes/html/en/en_US/Style.css
- themes/html/en/Style.css
- themes/html/Style.css
- themes/Style.css

Archived

Archived

12

Internationalization

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. WebSphere Portal has been translated into the languages listed on Table 12-1 on page 375.

The WebSphere Portal architecture makes it easy and efficient to provide internationalization support to portals. 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 enable internationalization:

- Using resource bundles
- Translating whole resources
- JSR 168 API considerations
- Dynamically changing the language during the session
- Administration
- Working with characters
- Best practices

© Copyright IBM Corp. 2005. All rights reserved.

12.1 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 12-1 demonstrates a base resource bundle. Example 12-2 demonstrates the resource bundle translated for Spanish. Notice the key names do not change, only the value is translated.

Example 12-1 NLSLab.properties resource bundle

```
welcome = hello
goodbye = goodbye
message = This is the NLSExample portlet
```

Example 12-2 NLSLab_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 12-1.

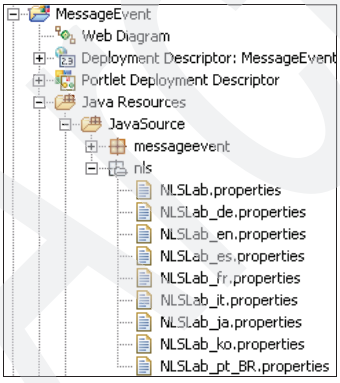


Figure 12-1 Translated resource bundles

The name is important because the Portal will locate the appropriate bundle for you based on the locale you provide. The naming convention for resource bundles is [bundle]_[language]_[country]_[variant].properties. The ISO standard ISO-639 is used for the language codes of most languages.For Hebrew the old

Archived

language code iw is used. The ISO standard ISO-3166 is used for the country/region codes. WebSphere portal supports the use of [variant], although resource bundles supplied with the portal do not use it.

When you use properties files in your code you need only provide the base name of the bundle and it will append the appropriate locale. The locales are listed in Table 12-1.

Table 12-1 Languages supported by WebSphere Portal

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

Archived

Locale Code	Language
pt	Portuguese
pt_BR	Brazilian Portuguese
ro	Romanian
th	Thai
uk	Ukrainian

12.1.1 Creating resource bundles in Rational Application Developer

The resource bundles need to be created in the JavaSource directory as illustrated in Figure 12-1 on page 374. 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 RAD, open the project explorer view in the Web perspective. Locate the JavaSource directory of the portlet you are enabling and right-click. From the context menu, select **New** → **Other**.

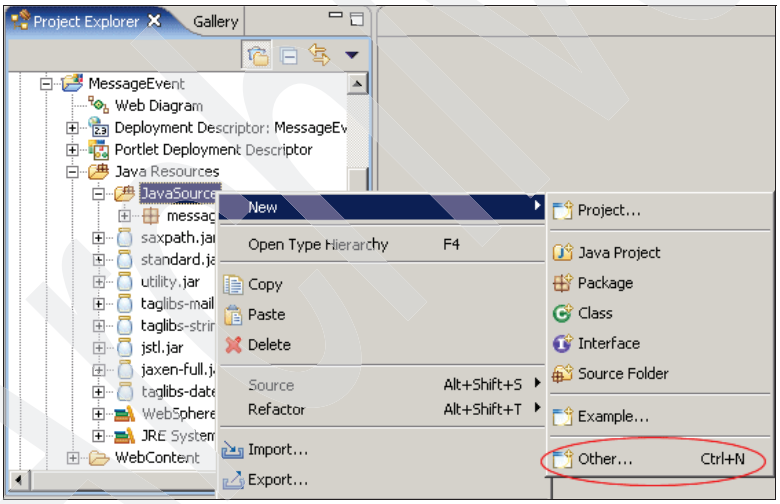


Figure 12-2 Creating a new folder

Select **Simple** → **Folder** and click **Next**.

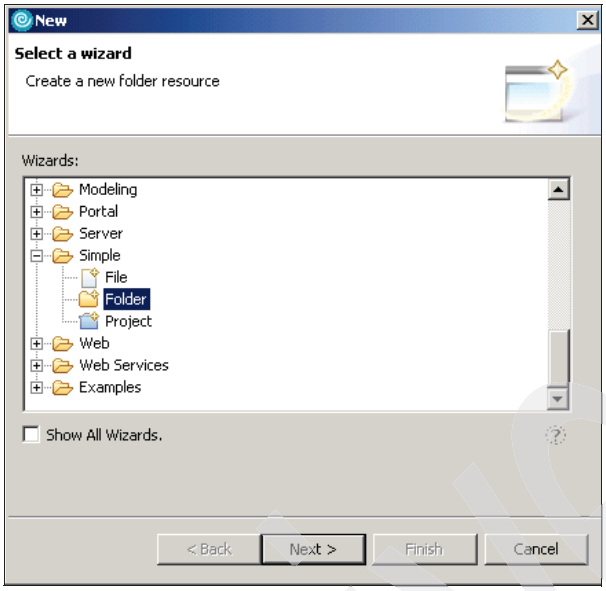


Figure 12-3 Creating an nls folder

Enter the name of the new folder, typically `nls` as shown in Figure 12-4 on page 378.

Archived

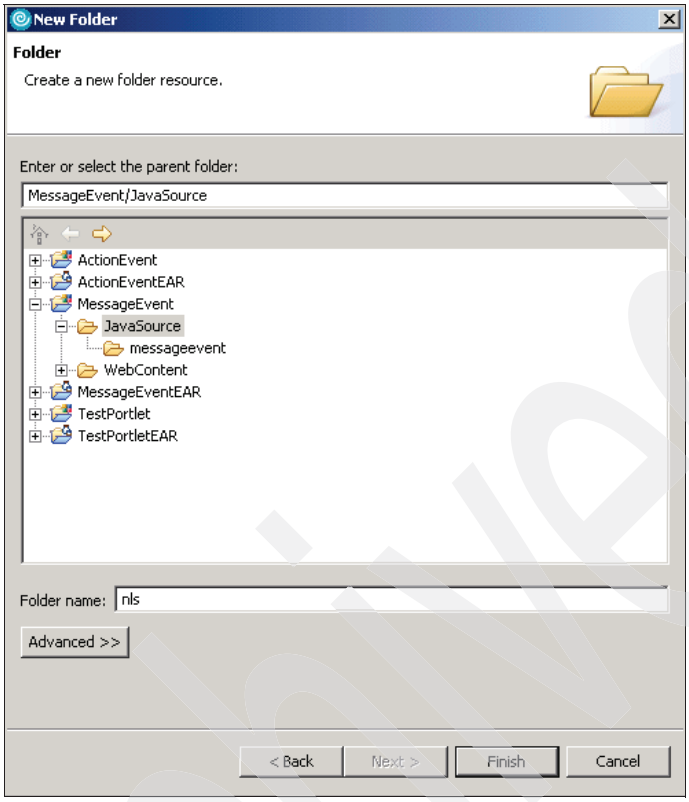


Figure 12-4 Creating the nls folder in RAD

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 12-5 on page 379. Do not include any language codes in the name, or include any spaces in the name of the resource bundle.

Archived

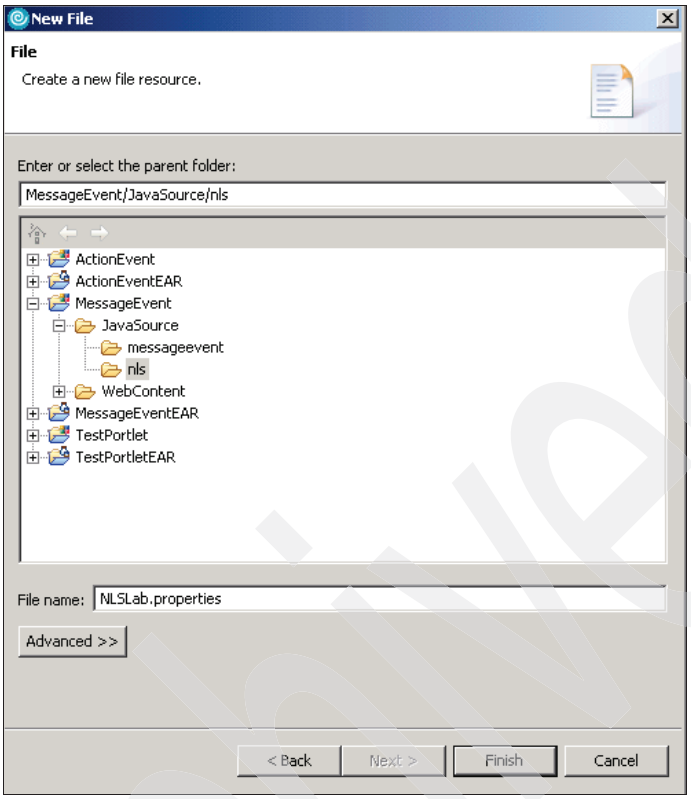


Figure 12-5 Creating the default properties bundle

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 Example 12-1 on page 374. Use **CTRL-S** to save the file.

12.1.2 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 RAD. Another way is 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 12-6 on page 380.

Archived

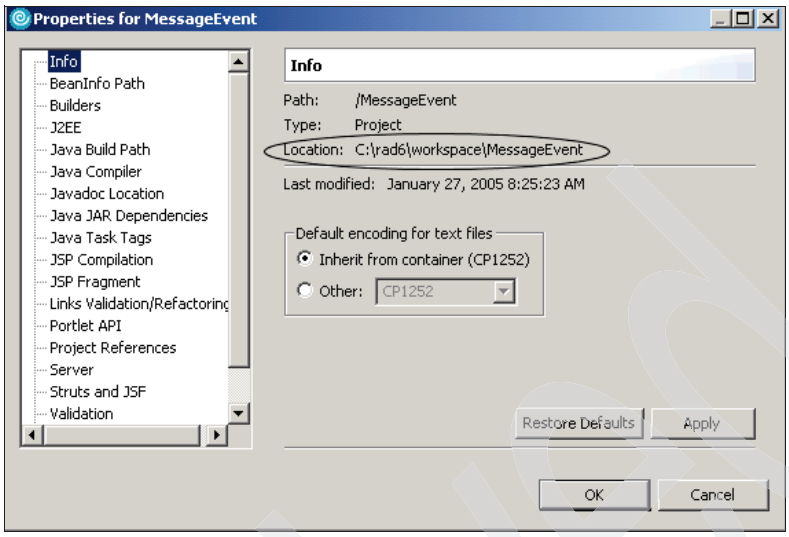


Figure 12-6 Locating the application on the file system

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 RAD environment. To do this, simply select the portlet project, right-click and select **Refresh** as shown in Figure 12-7 on page 381.

Archive

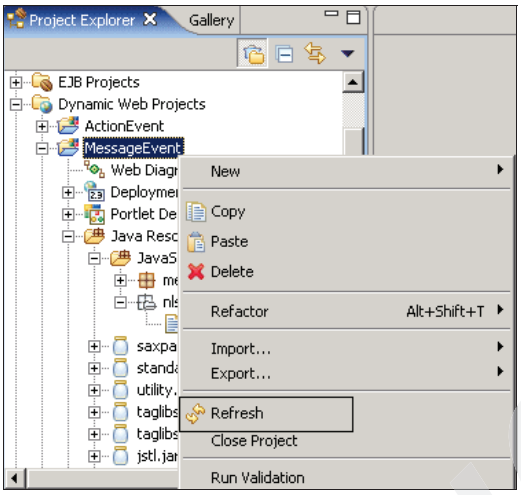


Figure 12-7 Loading resource bundles into RAD

When you are done, the folder should appear as in Figure 12-1 on page 374, depending of course on the number of languages you choose to support.

12.1.3 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 12-3.

Example 12-3 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 12-4 on page 382. 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.
- `Key`: this parameter maps to a key value in the properties file. If the key is not found, a PortletException is thrown.

Archived

- `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 12-4 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 12-4 Accessing resource bundles via the API

```
getPortletConfig().getContext().getText("nls.NLSExample", "welcome",
request.getLocale());
```

12.1.4 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 12.2, "Translating whole resources" on page 383 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. You can include it while you create the portlet application project, in the **Features** window check **JSP Tag Libraries** option. You can also add this feature to an existing portlet application: right-click in your portlet application, select **Properties** and then **Web Project Features**. In Available Web project features, check the **JSP Tag Libraries** and click **OK**.

Your JSP files can access resource bundles in two ways, as shown in the following examples.

Example 12-5 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 12-6 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.

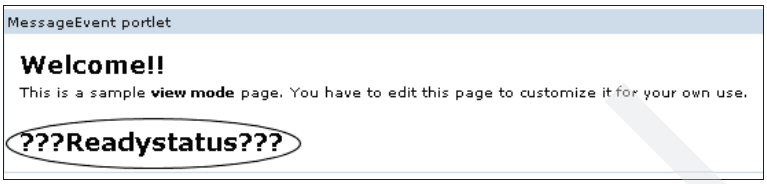


Figure 12-8 Key not found in a properties file

12.2 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 12-9.

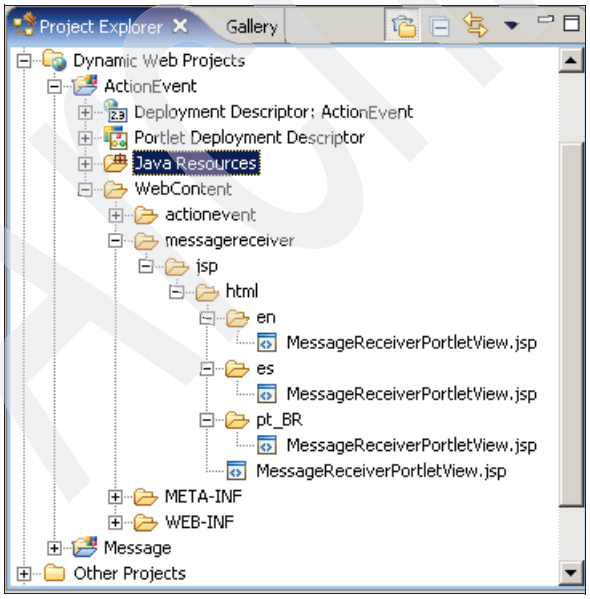


Figure 12-9 NLS directory structure

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 12-7. 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 12-7 Including translated JSP files

```
getPortletConfig().getContext().include("/messagereceiver/jsp/MessageReceiverVi
ew.jsp", request, response);
```

12.3 JSR 168 API considerations

In this section you will see how to access resource bundles in Portlets JSR 168, from JSPs and from portlets.

If you want to access resource bundles in JSPs, you have to use the JSP tag library as we explained in “Accessing resource bundles in JSPs” on page 382.

To access resource bundles in Portlets JSR 168 use getResourceBundle(java.util.Locale) method of javax.portlet.GenericPortlet class as displayed in Example 12-8

Example 12-8 Accessing resource bundles in Portlets JSR 168

```
ResourceBundle resource =
      getPortletConfig().getResourceBundle(request.getLocale());
String valor = resource.getString("welcome");
```

The name of the resource bundles where the information is stored is provided by the portlet deployment descriptor, as shown in Figure 12-10 on page 385.

Archived

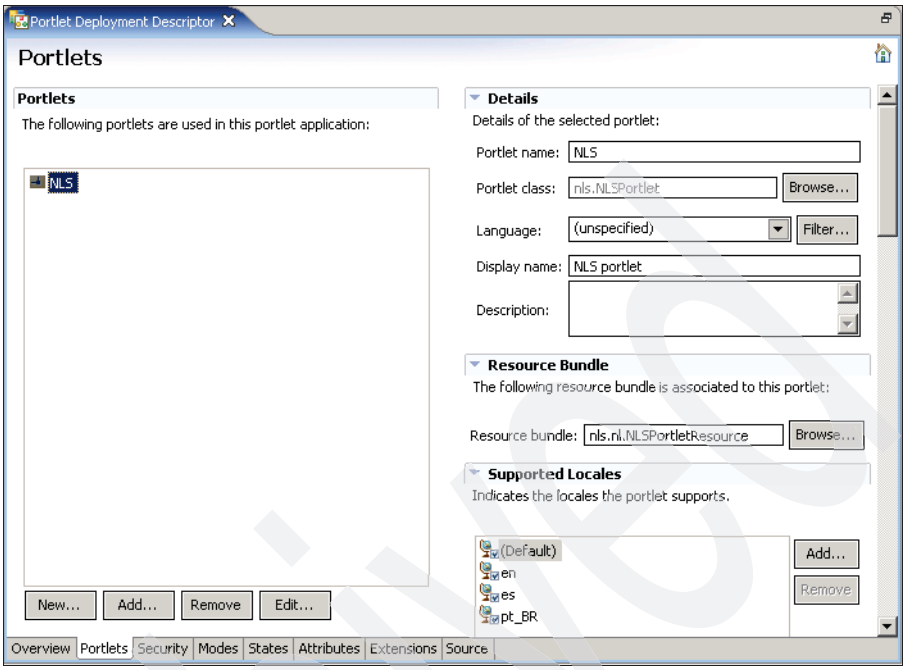


Figure 12-10 Portlet.xml

Unlike IBM Portlets API, in Portlets JSR 168 you can set the portlet title using resource bundles. If you want to set the portlet title using resource bundles, create a resource bundles for each language you want to be supported. Translate the javax.portlet.title key.

Example 12-9 Translate portlet title using resource bundles

```
# Portlet Info resource bundle example
javax.portlet.title=portlet con soporte NLS
javax.portlet.short-title=Prueba NLS
javax.portlet.keywords=
```

You also have to add the supported locales in portlet.xml file. In Figure 12-10 English, Spanish and Brazilian Portuguese are the supported locales for that portlet.

12.4 Dynamically changing the language

If you want users to be able to dynamically change the language during the session, you can use the following command provided by WebSphere Portal:

Example 12-10 Changing the language during the session

```
<wps:url command="ChangeLanguage">
   <wps:urlParam name="locale" value="language"/>
</wps:url>
```

where `language` is the code representation of the language that you want to be available in your portal. The 12.10, “Dynamically changing the language” on page 411 shows how to add links to the portal theme to change to different languages in one click.

12.5 NLS administration

Certain aspects of NLS enablement can be configured via the Administration window in 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.

12.5.1 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 12-11 on page 387.

Archive

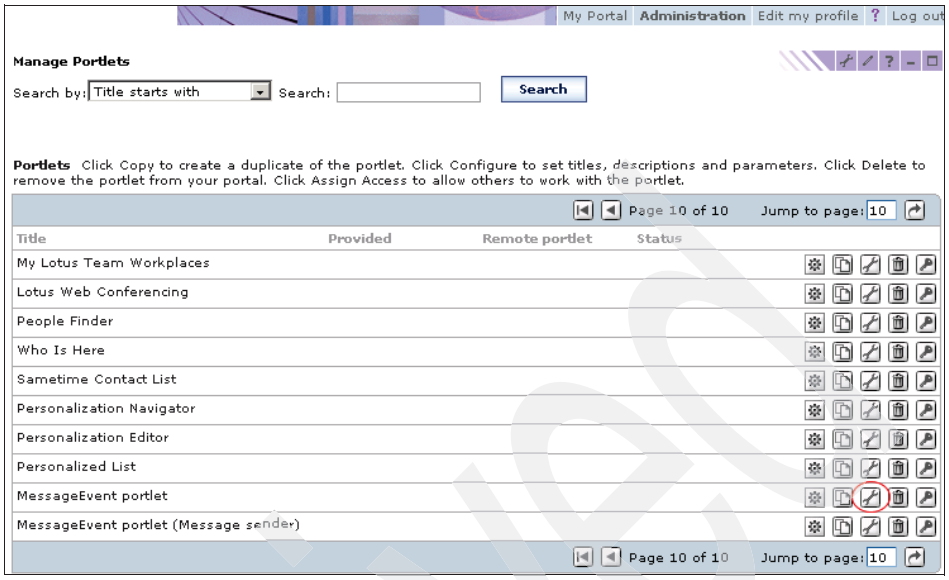


Figure 12-11 Working with portlets

Locate the portlet you want to work with and click the **Configure portlets** icon. At the bottom of the next window, select **I want to set titles and descriptions**. By default, the resulting window, shown in Figure 12-12 on page 388, will only display the languages listed in Table 12-1 on page 375 indicated for support by the portlet.xml.

Archive

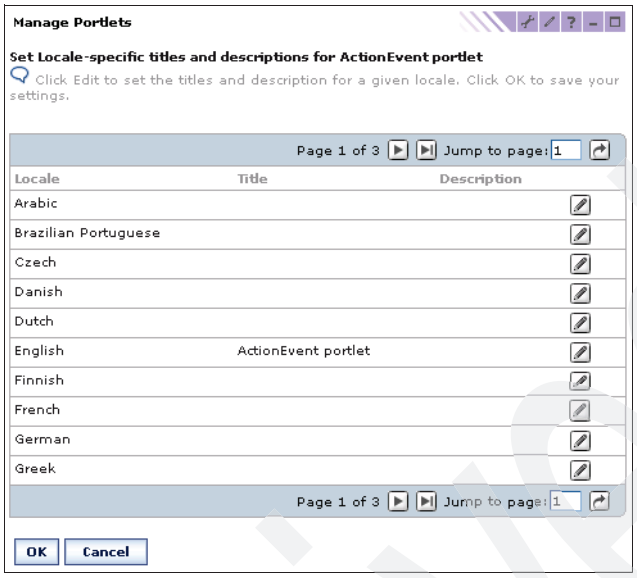


Figure 12-12 Supported languages of the selected portlet

To change the title and/or description, select the edit icon of the language for which you want to set a title and/or description. Enter the new title in the resulting window shown in Figure 12-13.

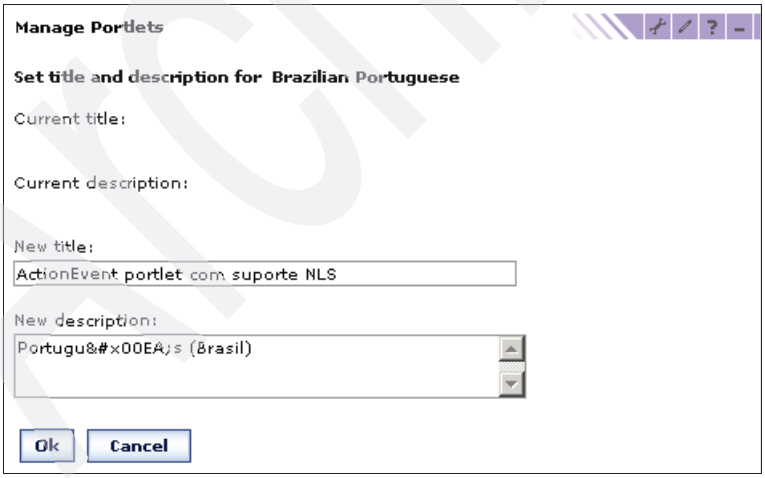


Figure 12-13 Setting the locale-specific title

You may have noticed that only the title and description can be adjusted via the Administration window.

To add support for a locale in the portlet.xml in RAD, open the portlet.xml editor and select the concrete portlet you wish to work with. This is illustrated in Figure 12-14.

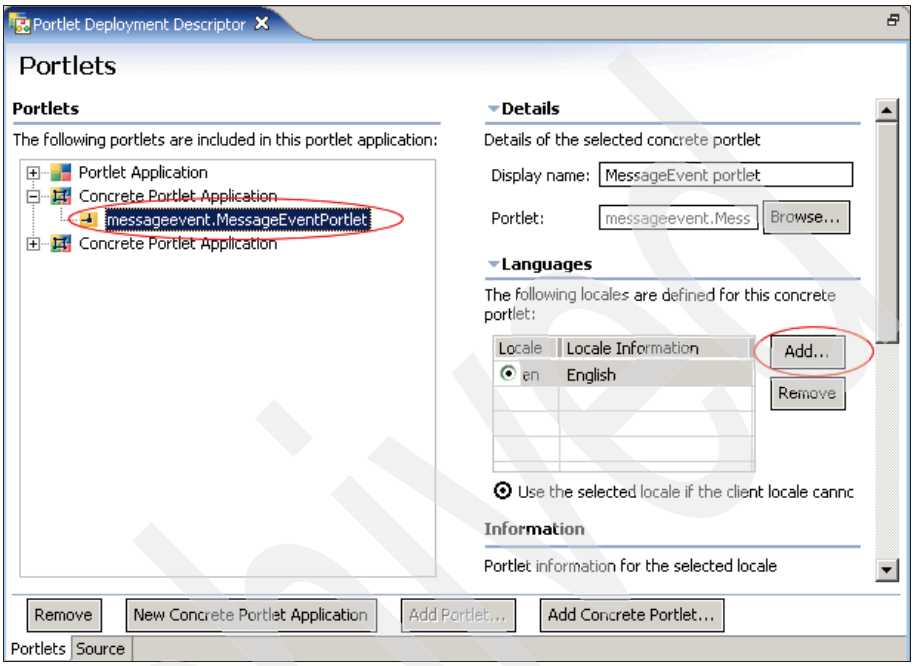


Figure 12-14 Adding locale support to a concrete portlet

To add a new locale, select the **Add** button in the locale section as shown in Figure 12-14. 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 12-15. If the locale you choose is already defined in the portlet.xml, you will be prevented from adding it again.

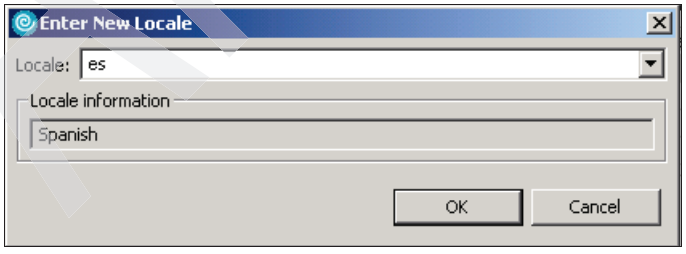


Figure 12-15 Specifying a new locale

All portlets must have a default language specified in the deployment description, otherwise the portlet cannot be installed.

12.5.2 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.

12.5.3 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. In the next window, Edit Layout page, select **Edit properties** button. A window as in Example 12-16 appears.

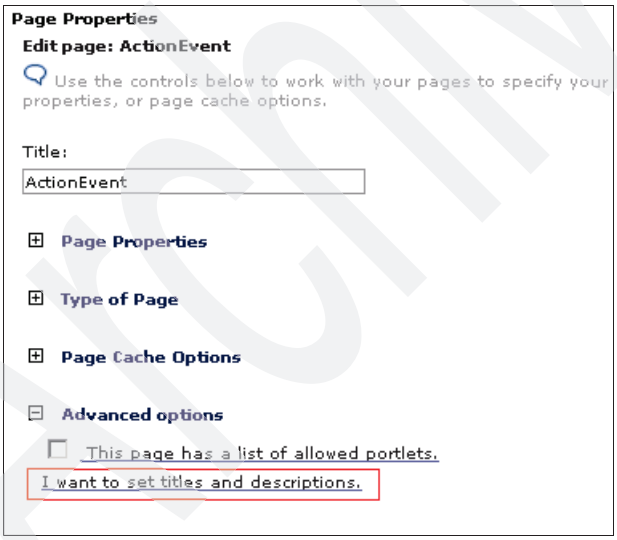


Figure 12-16 Editing page properties

Display the advanced options and select **I want to set titles and descriptions**. 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**.

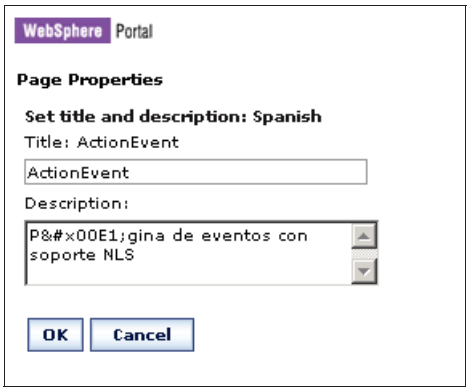


Figure 12-17 Using unicode values

Now you will see the new title and description in the list.

12.5.4 Supporting a new language

To support new languages you need to add that language to the file language.properties located in /wp_root/shared/app/config directory. Then you have to insert resource bundles, with an appropriate name, in the directory located at /wp_root/shared/app/nls. The directory where you have to store the JSP will depend on how the portal locates your JSP for rendering its content.

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 the JAR file wp.ui.jar under the location /wp_root/shared/app/wp.ui.jar/nls.

- button.properties
- commonAdmin.properties
- problem.properties
- field.properties
- engine.properties
- titlebar.properties
- registration.properties
- LocaleNames.properties
- pbruntime.properties
- virtual_principals.properties

Archived

Note: If your portal configuration includes Lotus Collaborative Services, add a new CSRes_language.properties file for the new language to the /wp_root/shared/app/nls directory

12.6 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, if you need to enter characters that 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 12-17 on page 391. The four character unicode values can be found at `http://www.unicode.org/charts`. The entity reference syntax is `&#x0000;` 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.

12.7 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 space. A number of API facilities are available for you to determine the current locale.

Archive

- 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.

12.8 Sample scenario: NLS bundles

In this sample scenario, NLS bundles will be created to support multiple languages. Once you have done this using RAD, the JSP that delivers markup content in View mode for portlet MessageReceiverPortlet 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 12-18.

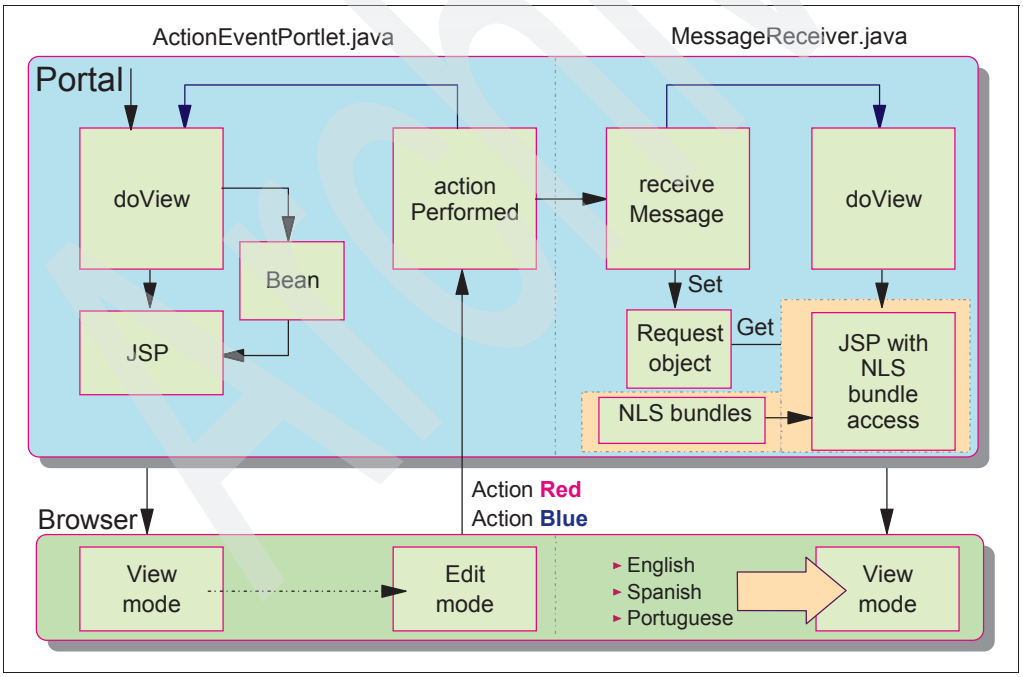


Figure 12-18 National Language Support (NLS) scenario

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.

12.8.1 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 RAD. Click **Start** → **Programs** → **IBM Rational** → **IBM Rational Application Developer V6.0** → **Rational Application Developer**.
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 Resources/JavaSource** folder.

Archived

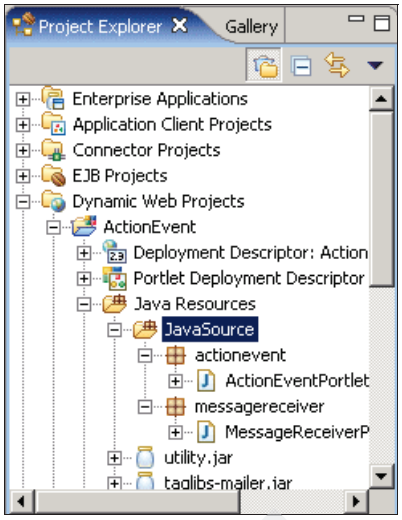


Figure 12-19 Select Java source to create the nls folder

4. Right-click it and select **New** → **Other**.
5. Select **Simple** → **Folder**, and click **Next.**

Archived

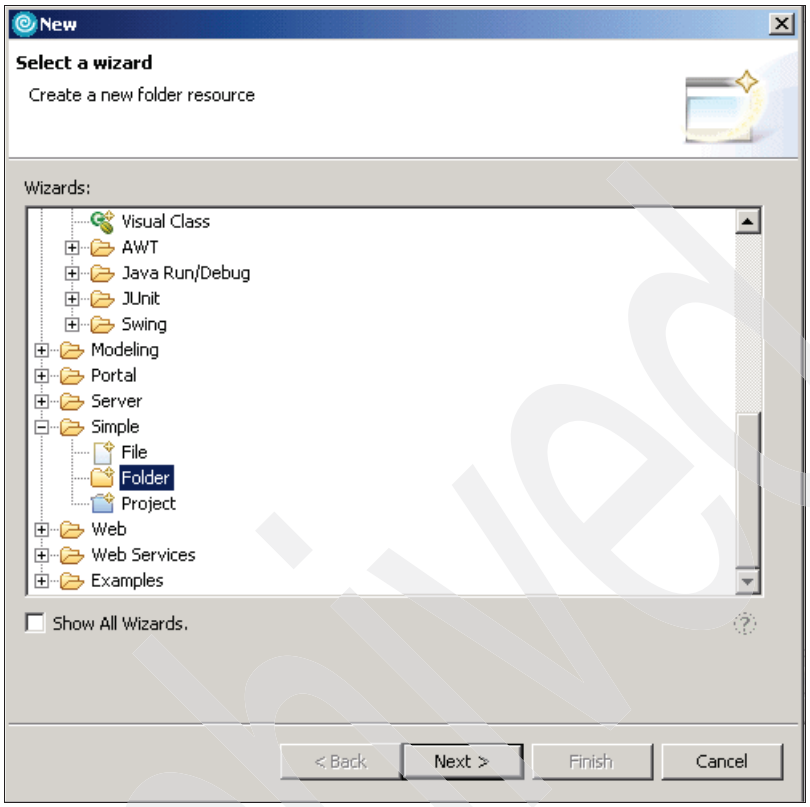


Figure 12-20 Create a new folder

6. In the next window enter the following values:
   - Parent folder: `ActionEvent/JavaSource`
   - Folder name: `nls`
   - Click **Finish**.
7. Your directory structure should now look as shown in Figure 12-21 on page 397.

Archived

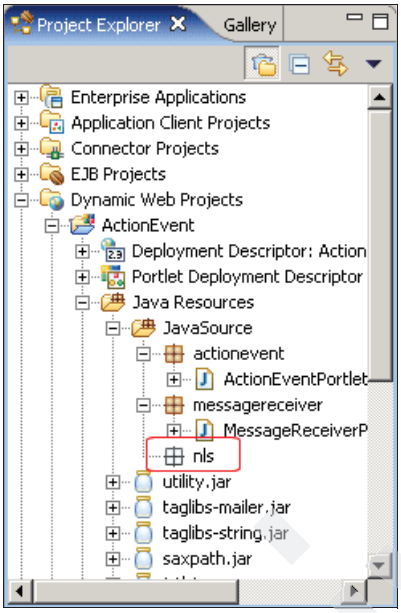


Figure 12-21 A new nls folder

8. Select the new **ActionEvent/Java Resources/JavaSource/nls** folder.
9. Click **File** → **Import**.
10. 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 A, "Additional material" on page 1003.
11. Select all four properties files and click **Finish** to import.

Archived

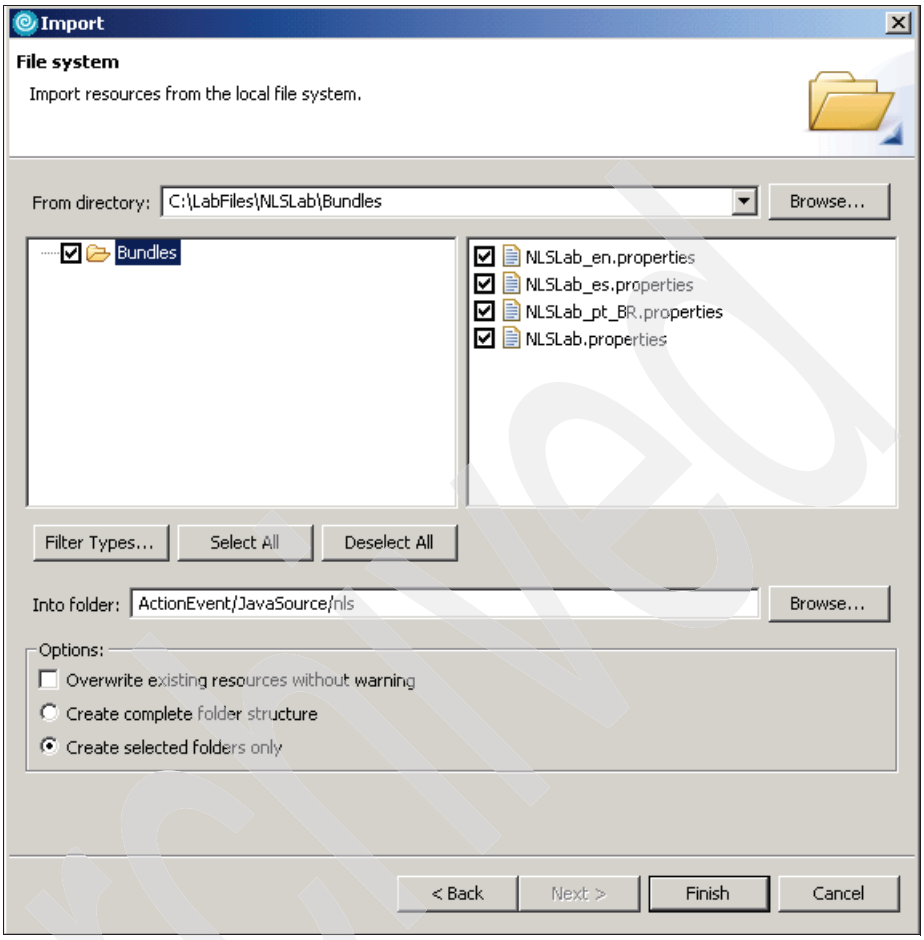


Figure 12-22 Importing the bundles

12. View the files in the nls folder by double-clicking them. Notice how they are structured.

Example 12-11 NLSLab.properties (default)

```
readystatus=Ready to receive message
receivedstatus=Received a message
viewmode=Operating in View mode
redColor=RED
blueColor=BLUE
```

Example 12-12 NLSLab_en.properties (English)

```
readystatus=Ready to receive message
```

```
receivedstatus=Received a message
viewmode=Operating in View mode
redColor=RED
blueColor=BLUE
```

Example 12-13 NLSLab_es.properties (Spanish)

```
readystatus=Listo para recibir mensaje
receivedstatus=Mensaje recibido
viewmode=Operando en modo de visualizaci\u00F3n
redColor=ROJO
blueColor=AZUL
```

Example 12-14 NLSLab_pt_BR.properties (Brazilian Portuguese)

```
readystatus=Pronto para receber mensagem
receivedstatus=Mensagem recebida
viewmode=Operando em modo de Visualização
redColor=VERMELHA
blueColor=AZUL
```

12.8.2 Accessing NLS bundles from JSPs

In this section, you will update the MessageReceiverPortletView.jsp file 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. When you created the ActionEvent portlet you select JSP Tag Libraries in Features window. To be sure these libraries are included follow these steps:
   a. Right-click the **ActionEvent** project.
   b. Select **Properties**.
   c. Select **Web Project Features** from the list.
   d. Check the **JSP Standard Tag library** box if it was not checked.
   e. Click **OK**.
2. Open the MessageReceiverPortletView.jsp file. This file is located in the /WebContent/messagereceiver/jsp/html/ directory.
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 12-15 on page 400.

Archived

Example 12-15 MessageReceiverPortletView.jsp supporting NLS with bundles

```
<%@ page session="false" 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/MessageReceiverPortletView.jsp".

<fmt:setBundle basename="nls.NLSLab"/>

<br>
<% 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.

12.8.3 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** → **Run on Server**.
3. Select **WebSphere Portal V5.1 Test Environment** server and click **Next**.
4. Confirm that the ActionEventEAR project is the only one configured to run in the test environment. Click **Finish** to run your project.
5. The message receiver portlet will now display its markup using NLS.

Archived

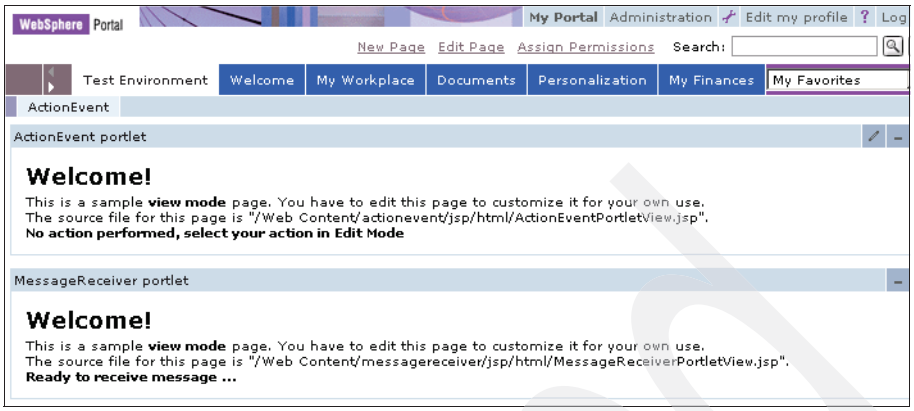


Figure 12-23 ActionEvent portlet with no preferred language selected

6. Select a new locale value by clicking the **Edit my profile** icon to select a preferred language, as shown in Figure 12-24.

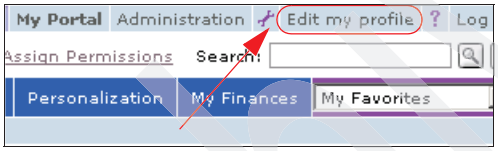


Figure 12-24 Edit my profile icon

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.

Archived

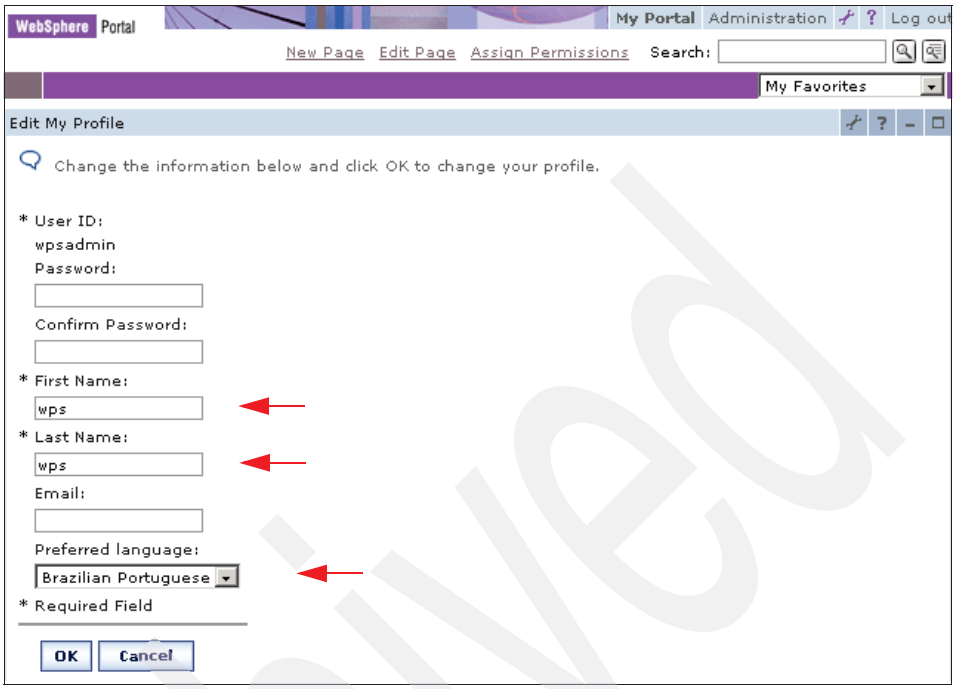


Figure 12-25 User profile

7. Click **OK**. Now you will 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

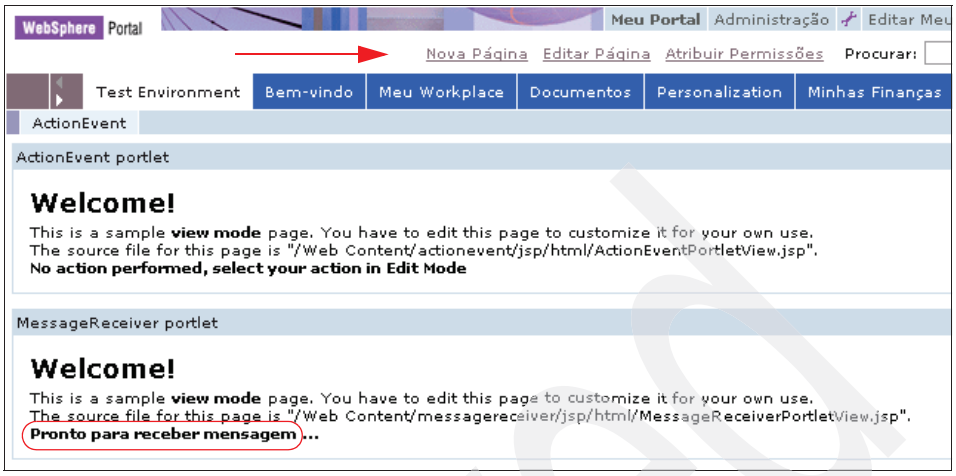


Figure 12-26 ActionEvent Portlet with Brazilian Portuguese as preferred language

8. Edit the user profile again (Editar meu perfil in Brazilian Portuguese) and try **Spanish** (Espanhol) as the new locale.

   The content you specified will display in Spanish.

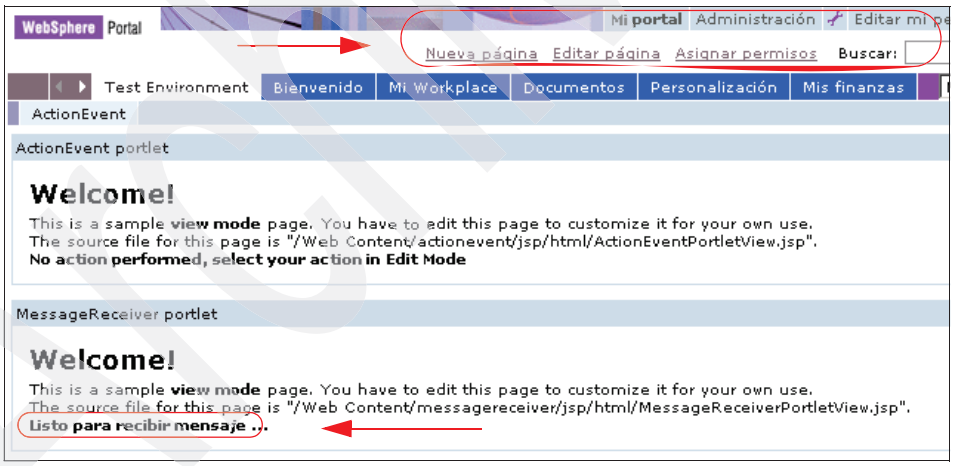


Figure 12-27 ActionEvent portlet with Spanish as preferred language

9. Edit the user profile again and try **French** (**Frances** in Spanish) as the new locale.
10. 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.

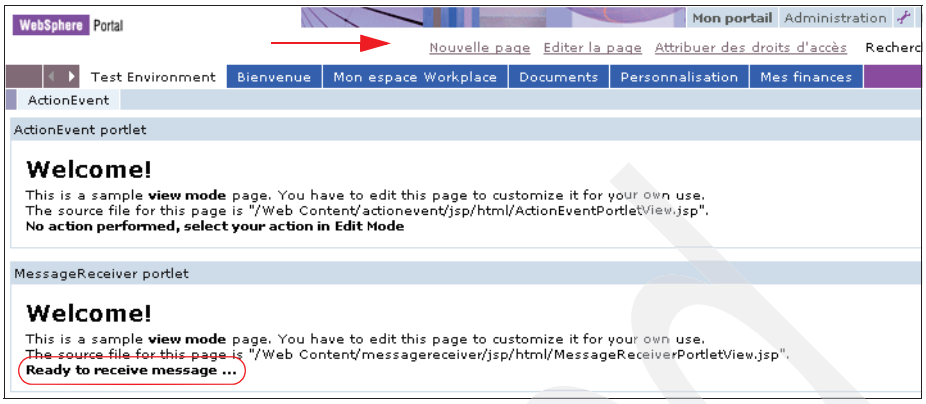


Figure 12-28 ActionEvent portlet with French as preferred language

11. To change your language back to English before you exit, click **Edit my profile** (Editer mon profil) and select **English** (**Anglais** in French) as your language.

12.8.4 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 12-16 ActionEventPortlet.java

```
......
import java.io.IOException;
import java.util.Locale;

......

   public void actionPerformed(ActionEvent event) throws PortletException {
```

```
      if( getPortletLog().isDebugEnabled() )
         getPortletLog().debug("ActionListener - actionPerformed called");

      //get the preferred locale for the user
      Locale loc = event.getRequest().getLocale();

      // ActionEvent handler
      String actionString = event.getActionString();
      PortletRequest request = event.getRequest();

      // 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 an instance of portlet data to store values
         PortletData portData = request.getData();
         ......................

      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 an instance of portlet data to store values
         PortletData portData = request.getData();
.......
```

3. Open the resource bundles located in the ActionEvent/Java Resources/JavaSource/nls folder and make sure the following key-value pairs required for this scenario have been included in the properties files.

Example 12-17 NLSLab.properties (default)

```
redColor=RED
blueColor=BLUE
```

Archived

Example 12-18 NLSLab_en.properties (English)

```
redColor=RED
blueColor=BLUE
```

Example 12-19 NLSLab_es.properties (Spanish)

```
redColor=ROJO
blueColor=AZUL
```

Example 12-20 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.

12.9 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 responsibility for locating the correct file at runtime, depending on the preferred language selected by the user.

Archived

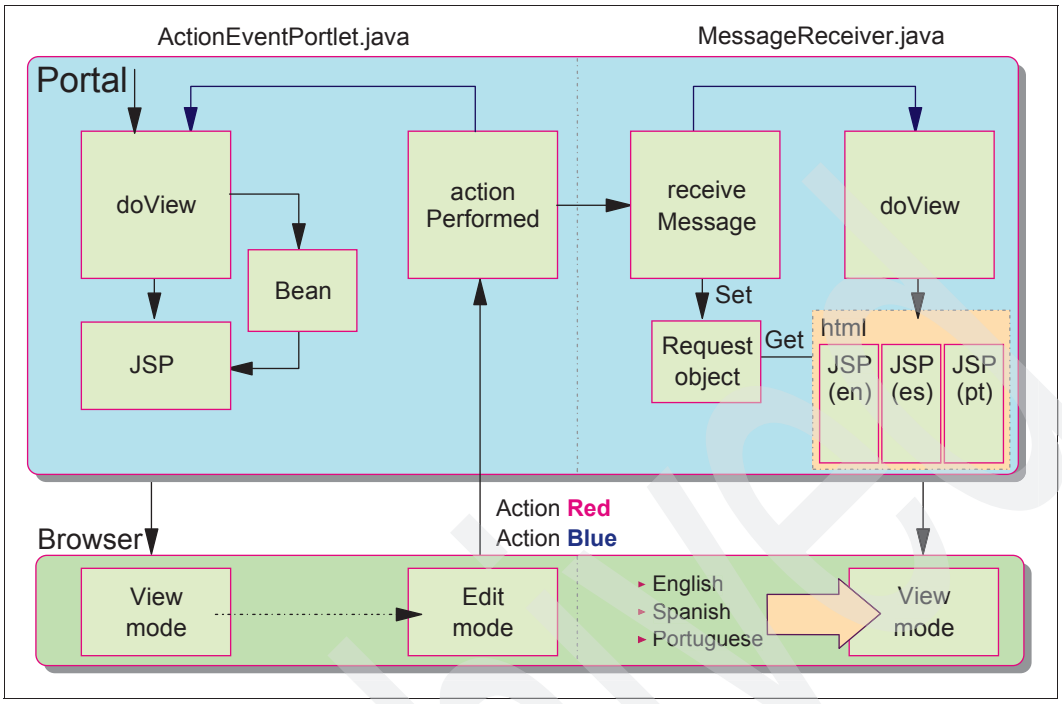


Figure 12-29 Sample scenario

1. Open the WebContent/messagereceiver/jsp/html/MessageReceiverPortletView.jsp page to delete fmt tags and return to static information. Your code should look as shown in Example 12-21.

Example 12-21 MessageReceiverPortletView.jsp

```
<%@ page session="false" 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/MessageReceiverPortletView.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**.
3. Type **en** for the Folder name field and click **Finish**.
4. Right-click **WebContent/messagereceiver/jsp/html/MessageReceiverView.jsp** and select **Copy**.

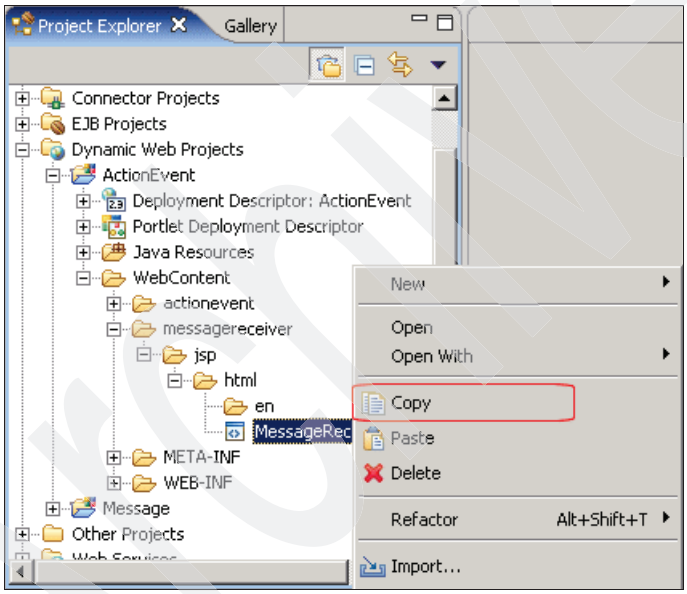


Figure 12-30 Copy a JSP page

5. Now select the **WebContent/messagereceiver/jsp/html/en** folder, right-click it and select **Paste**.
6. Open the page WebContent/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 12-22 MessageReceiverPortletView.jsp (English)

```
<%@ page session="false" 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/MessageReceiverPortletView.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>
```

7. 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 12-31 on page 410.

Archived

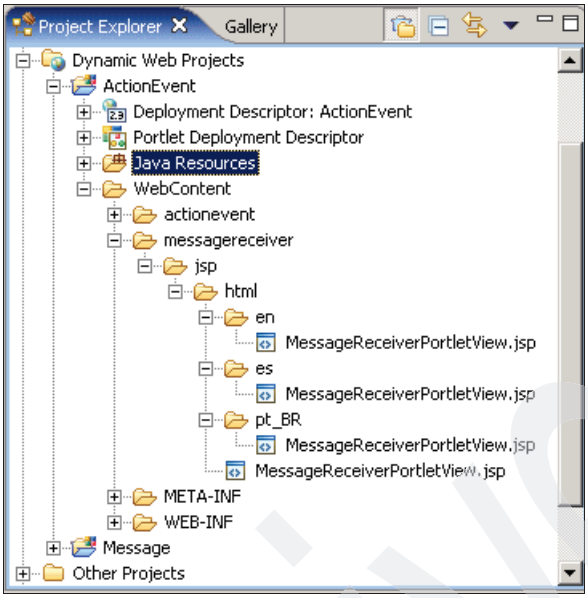


Figure 12-31 Directory structure

8. 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 12-23 MessageReceiverPortletView.jsp (Spanish)

```
<%@ page session="false" 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/MessageReceiverPortletView.jsp".

<br>
<% if (request.getAttribute("MyMessage") == null) { %>
   <B>Listo para recibir mensaje ... </B>
<% } else { %>
   <B>Mensaje recibido:</B>
   <B><%= request.getAttribute("MyMessage") %></B>
```

Archived

```
<% } %>

</DIV>
```

Example 12-24 MessageReceiverPortletView.jsp (Br Portuguese)

```
<%@ page session="false" 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/MessageReceiverPortletView.jsp".

<br>
<% if (request.getAttribute("MyMessage") == null) { %>
   <B>Pronto para receber mensagem ... </B>
<% } else { %>
   <B>Mensagem recebida:</B>
   <B><%= request.getAttribute("MyMessage") %></B>
<% } %>

</DIV>
```

9. Run the scenario (click **Run** → **Run on Server**) and verify your results in multiple languages.
10. Change the locale in the user profile as before and try other supported languages.

12.10 Dynamically changing the language

In the above sample scenarios when you want to see the application in other language you need to changes the user profiles. Here you learn how to use the **ChangeLanguage** command described in 12.4, “Dynamically changing the language” on page 385.

Note that this command only change the language during the current session. Preferred language selected in the user’s profile does not change.

1. Open the Default.jsp file located in was_root/installedApps/node_name/wps.ear/wps.war/themes/html/ directory.

Archived

That is where the default files for WebSphere theme are, if you are using another theme select its corresponding directory.

2. To create links for English, Spanish and Brazilian Portuguese add the following code

Example 12-25 Adding links to the portal theme

```
   .............
            <%-- Admin link bar - New Page, Edit Page, Assign Permissions
                        Don't show these links in solo state --%>
                  <wps:if portletSolo="no">
                    <%@ include file="./AdminLinkBarInclude.jsp" %>
                  </wps:if>
               </td>
            </tr>
   <%-- supported languages --%>
   <tr>
   <td colspan="2">
   <table border="0" cellspacing="5" cellpadding"0">
   <tr>
      <td>
      <a href="<wps:url command='ChangeLanguage'><wps:urlParam name='locale'
value='en'/></wps:url>"><wps:text key="lang_en" bundle="nls.Text"/></a>
      </td>
      <td>
      <a href="<wps:url command='ChangeLanguage'><wps:urlParam name='locale'
value='es'/></wps:url>"><wps:text key="lang_es" bundle="nls.Text"/></a>
      </td>
      <td>
      <a href="<wps:url command='ChangeLanguage'><wps:urlParam name='locale'
value='pt_BR'/></wps:url>"><wps:text key="lang_pt_BR" bundle="nls.Text"/></a>
      </td>
   </tr>
   </table>
   </td>
   </tr>

         <tr>
         <td colspan="2">
         <%-- Show navigation bars for first two levels of page navigation
                        Don't show navigation in solo state --%>
            <wps:if portletSolo="no">
               <%@ include file="./PlaceBarInclude.jsp" %>
               <%@ include file="./PageBarInclude.jsp" %>
            </wps:if>
         </td>
         </tr>
.............
```

Archived

3. In Example 12-25 on page 412 we also translate the links for the different language using resource bundles. Create four new files in wp_root/shared/app/nls/ directory and call them: Text_en.properties, Text_es.properties, Text_pt_BR.properties and Text.properties.

Example 12-26 Text.properties (default)

```
lang_en=English
lang_es=Spanish
lang_pt_BR=Brazilian Portuguese
```

Example 12-27 Text_en.properties (English)

```
lang_en=English
lang_es=Spanish
lang_pt_BR=Brazilian Portuguese
```

Example 12-28 Text_es.properties (Spanish)

```
lang_en=Ingl\u00E9s
lang_es=Espa\u00F1ol
lang_pt_BR=Portugu\u00E9s de Brasil
```

Example 12-29 Text_pt_BR.properties (Brazilian Portuguese)

```
lang_en=Ingl\u00EAs
lang_es=Espanhol
lang_pt_BR=Portugu\u00EAs (Brasil)
```

4. Re-start the server and run the application. You will see a link for each language supported as show in Figure 12-32. Click them to test the scenario.

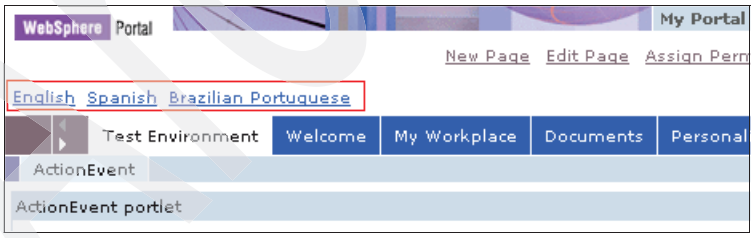


Figure 12-32 Portal theme

Note: Since the preferred language selected by the user does not change, no text translated using the preferred locale for the user will display in the proper language.

Archived

Archived

13

Struts portlets

Struts is a very popular open source framework that embodies the Model-View-Controller (MVC) pattern. The Struts framework is an open source project of The Apache Software Foundation and can be used to effectively design Web applications as well as portlet applications.

The Struts portlet framework was implemented by IBM so that applications written based on the Struts framework could run in WebSphere Portal environment, as well as to enable the development of portlets using the Struts framework.

In this chapter we provide an overview of the Struts framework and discuss the main differences that exist when using the Struts portlet framework.

© Copyright IBM Corp. 2005. All rights reserved.

13.1 Overview

The MVC pattern that is embodied by Struts provides an standard way to separate logic, presentation and data in modular applications, among the three application tiers, as shown in Figure 13-1:

- The model is the set of data and business rules for the application. This is commonly called the application's business logic. This business logic in most cases involves access to data stores, legacy systems and external applications.
- The view is the application's user interface. This tier is also known as the presentation layer.
- The controller defines the way that an application interacts with user input and the model, that is, the overall flow of events. This is called the application logic.

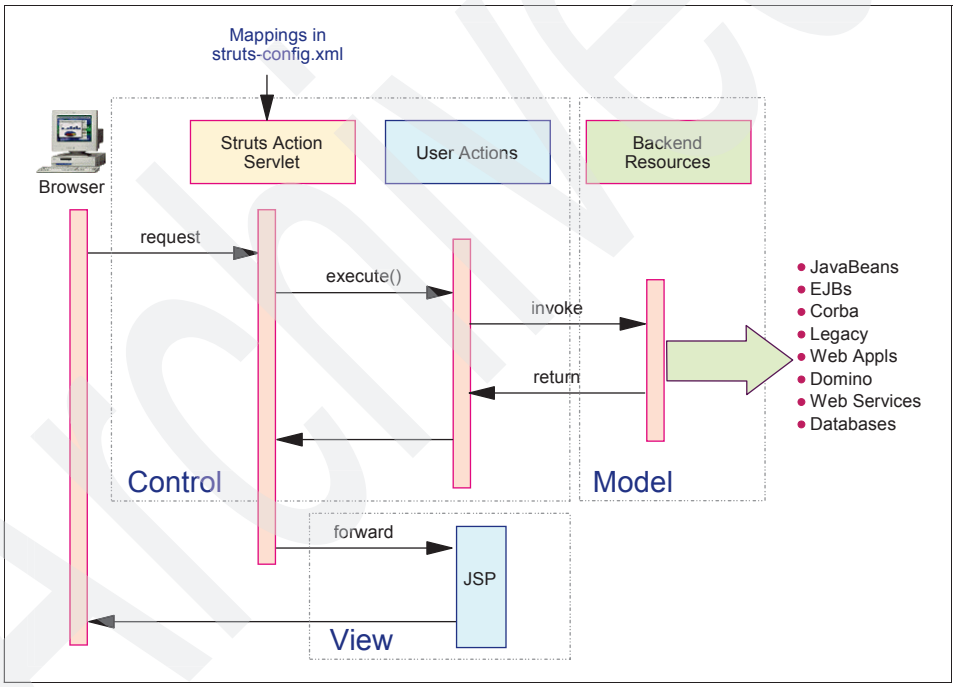


Figure 13-1 Struts and the Model-View-Controller pattern

The following considerations apply to Struts applications:

- All browser requests are submitted to the Struts `ActionServlet`.
- When the HTML form is submitted, the `ActionForm` subclass is automatically populated with the form data.

Archived

- The Struts `ActionServlet` determines which `Action` subclass route to using the mapping that is pre-configured in struts-config.xml.
- The `ActionServlet` passes the control to the `Action` subclass.
- The `Action` subclass can access the form data that is stored in the `ActionForm` subclass and pass it to the back-end business logic for further actions.
- The `Action` subclass invokes the back-end business logic, that can be implemented with any technology.
- The `Action` subclass may update the any necessary data in the `ActionForm` subclass.
- The `Action` subclass passes the control to the JSP that will render the view to be returned to the browser.
- The JSP that will render the view can access the form data that is stored in the `ActionForm`. This data is updated with all changes made by the `Action`.
- The view is returned to the browser.

Note: The core of the Struts framework is to provide a flexible controller layer, interacting with both the view and the model layers.

Struts components

The main Struts components are illustrated in Figure 13-2 on page 418 and they are:

- Actions and ActionServlet
  The struts `ActionServlet` (`org.apache.struts.action.ActionServlet`) handles all browser requests. According to the rules configured in a Struts configuration file (struts-config.xml), the `ActionServlet` invokes an `Action` subclass (subclass of `org.apache.struts.action.Action`). The `ActionServlet` is said to "perform actions", which means that the servlet invokes the perform method of each of the instantiated action classes. Each browser request is mapped to an `Action` subclass in the struts-config.xml file. The `ActionServlet` loads the mapping during initialization. To configure the Web project to pass all browser requests to the `ActionServlet`, map all URIs that end with `.do` (for example, *.do) to the `ActionServlet` in the Web deployment descriptor. You can then provide the actual `Action` subclass mappings in the Struts configuration file for individual request URI, such as /order.do.
- ActionForm
  Struts provides the 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.

- Tag libraries
  Struts provides a variety of custom JSP tags which are simple to use but 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. The custom tags support:
  - Pre-population of a HTML from with values taken from an `ActionForm` subclass.
  - Internationalization, such as providing text that is determined by the user's locale.
  - Logic, such as showing a different title for a page based on how it is used.

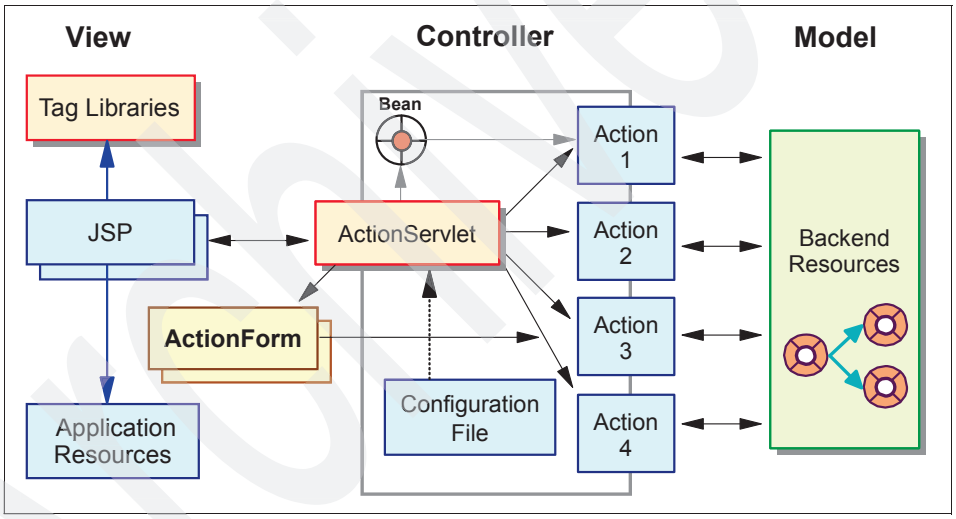


Figure 13-2 Struts components

Struts portlet life cycle

The Struts portlet life cycle is illustrated in Figure 13-3 on page 419 using a typical business transaction called `Deposit` as follows:

1. The index.jsp has a form for data entry (Account number, amount)
2. The ActionServlet fills the DepositForm with the data of the index.jsp, the DepositForm validates the input data, and in case of errors the index.jsp is redisplayed with error messages

3. The ActionServlet calls the DepositAction sub-class and passes the DepositForm
4. DepositAction class invokes Deposit class to perform the business logic
5. The Deposit class performs the database retrieval and update
6. The result is the Account bean returned to the DepositAction and placed into session (or request) data
7. The DepositAction class returns either a "success" or "fail" ActionForward result
8. The ActionServlet calls either the result.jsp (for "success") or the error.jsp (for "fail")
9. The result.jsp displays the Account data.

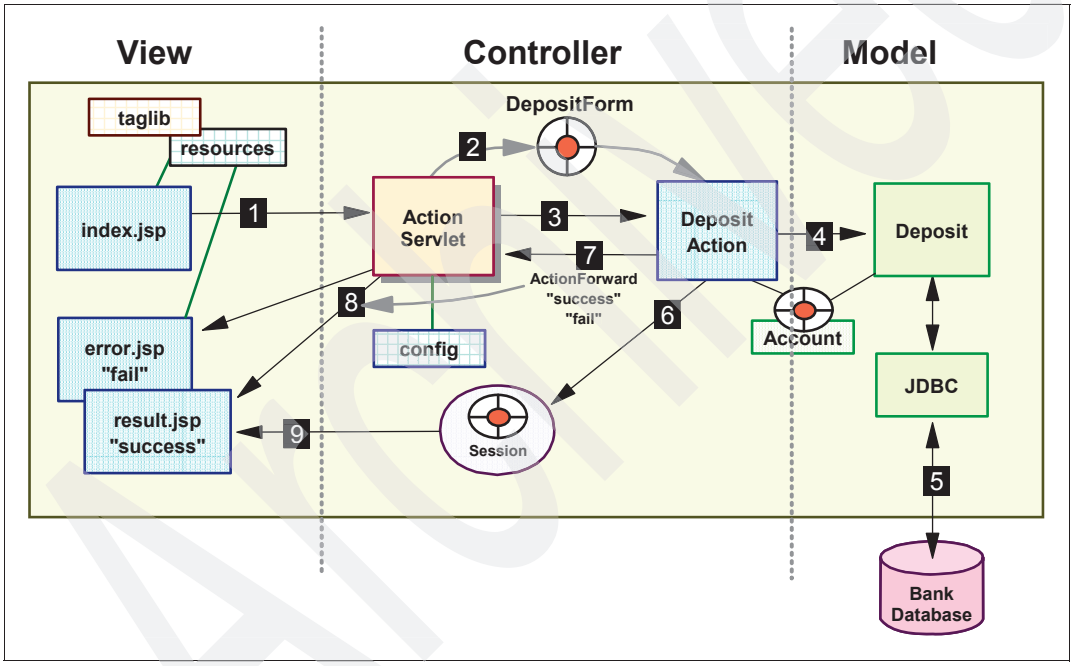


Figure 13-3 Struts portlet life cycle

13.2 The Struts portlet framework

Struts-based application development can be applied to portlets, similar to the way that Struts development is implemented in Web applications. Because of the differences in the servlet and portlet technologies, the Struts portlet framework was developed to merge these two technologies.

Archived

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, that might need to be addressed by the Struts application. The Struts portlet framework also supports Struts portlets developed using either the IBM Portlet API or the JSR the JSR 168 API.

For example, the processing of a Struts portlet using the IBM Portlet API consists of three steps:

1. Initialization. This process is called the first time that the portlet is invoked. In this step the Portal initializes the `WpsStrutsPortlet`. One of the tasks accomplished by the `WpsStrutsPortlet.init` method is initializing the `ActionServlet`. At the end of this step, the `ActionServlet` has already loaded all the configurations stored in the struts-config.xml file.
2. Action phase. This process is called during the portlet action phase (for example, when the user clicks a submit button). In this step, the `WpsStrutsPortlet.actionPerformed` method is invoked. This method will call the `WpsRequestProcessor.process` method, that owns the responsibility to invoke the Struts Action that was requested.

   The `WpsRequestProcessor` also creates an `IViewCommand` (in this case, a subclass named `WpsViewCommand`). The `IViewCommand` encapsulates the information so that the command object can be rendered at a later time, during the rendering phase.
3. Render phase. This process is called during the portlet render phase (that is, after action processing or as the result of refreshing the page). In this step, the `WpsStrutsPortlet.service` method is invoked. This method essentially invokes the `WpsViewCommand.execute` method.

   The execute method receives the request and response objects as parameters. As part of the processing, the previously saved attributes are populated into the request object that is passed in on the execute method. Also, during the render phase, an execution context object (`ViewCommandExecutionContext`) is passed in on the execute call, which provides additional objects to help with the actual execution. At the end of this phase, control is passed to some jsp page.

13.2.1 Struts applications

This section describes the main differences between a servlet-based Struts application and a Struts application created for the portal environment.

Comparison of servlets and portlets

There are two main differences between the portlet and servlet environment that affects a Struts application in WebSphere Portal.

Action processing and rendering

All servlet processing occurs during the `service()` method. The Struts rendering of the page is usually immediately preceded by action processing and essentially part of one step. The request and response object are passed to the service() method, which writes the resulting output to the response object. Portlet processing, however, is implemented in two phases, an event phase and a render phase.

Action processing is performed prior to rendering the display view. Only the request object is passed to the portlet during the event phase. When a Struts application is migrated to the portlet environment, some of the information that was available during the event phase, namely the request parameters, is no longer available during the render 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. See the Struts Portlet Framework chapter of the WebSphere Portal Infocenter

Note: The Struts action is not invoked when changing modes (view, edit, and configure). Only the service method is called (`doView`, `doEdit`) when switching modes. Therefore, you will need to place a call to that action in these methods to invoke Struts actions.

URI construction

URIs are constructed differently for portlets than for servlets. The portlet creates the URI programmatically using the `PortletResponse.createURI` method. The Struts portlet framework has modified the tags in Struts so that they create portal links. The Struts link tags behave the same as they do in the servlet environment, but the URL is a portlet URL and the Struts URL is passed as a parameter on the URL.

Archive

Response object

During portlet action processing the response object is not available. The methods called during Struts processing expect both a request and response. To address the need for a response object, the Struts Portlet framework creates a temporary one. Other than for calling `sendError`, you should not write to the response object. Instead your action should return an `ActionForward` object and let the Struts `RequestProcessor` complete the `doForward` set.

sendError() processing

The typical Struts application has no need to access the response object during the action processing. However, when an application checks and finds some state information invalid during action processing, it is not unusual for a Struts application to report the error using the `response.sendError` method. The Struts portlet framework intercepts the calls to `sendError` and saves the error information in the session. During the later view render phase, this error information is found, and the error information is displayed in lieu of displaying other content for the portlet.

Forwards and redirects

Although you can write portlets using Struts, there are some aspects of the servlet environment that you cannot do in a Struts portlet. For example, Struts provides support for redirects and forwards to other servlets. These are provided because they are functions generally available to servlets. However, these capabilities are not available in portlets because WebSphere Portal does not support forwards or redirects from Struts actions.

The following example demonstrates how to implement a forward to a Struts Action from an Action or a custom tag.

Example 13-1 How to implement a forward from an Action or a custom tag

```
PortletApiUtils portletUtils = PortletApiUtils.getInstance();
if (portletUtils != null) {
   portletUtils.forward( page, request );
}else {
   pageContext.forward( page, request );
}
```

The `PortletApiUtils.getInstance` method is called to see if a `PortletApiUtils` instance is available. If a non null value is returned, then the execution is inside of WebSphere Portal. In this case, the `portletApiUtils.foward` method is used instead of the `PageContext.forward`.

If a forward is required in a JSP, then the logic forward tag is the suggested solution. The forward tag has been modified to use the `PortletApiUtils` forward

implementation, and is the preferred method for a forward from a JSP. The `PortletApiUtils` can be obtained in a JSP through java code, but obtaining the Struts `ModuleConfig` to prefix the path is problematic. The logic forward tag handles these issues.

StrutsAction

The `StrutsAction` class provides an execute method that is passed portlet objects instead of servlet objects so a cast is not required to access portal-specific objects.

13.2.2 Changes to Struts JSPs

The JSPs for Struts applications in the portal environment have to be modified to adapt to the way the portal server expects portlet URIs to be created. There are some changes to the tag library for HTML markup and additional tag libraries have been added to support cHTML and WML markup.

Creating portlet URIs

The Struts application paths, both to actions and to pages, must be sent and retrieved using portlet URIs. Portlet URIs have a specific format. A special API is used to generate the URI and add the desired information to be passed to the portlet. If portlet URIs were not used, control would not get passed to the correct portlet. Thus, the portlet URIs must be used to get control passed to the correct portlet, with the additional path information needed by the Struts application made available. The Struts tags have been modified to automatically provide this needed functionality.

Struts Action mappings are defined in terms of paths. The name and location of page objects (for example, JSPs) are also defined using paths. Thus, although portlets have their own form of URI, it is still necessary to associate the Struts path with an action sent to a portlet and to retrieve that Struts path when the portlet action is handled. Typically a Struts application passes parameters on such a path using the query string on the HTTP URL. Often the actions containing these paths are generated from tags provided by Struts. The most obvious examples of these are the tags for the HTML elements <link> and <form>.

Stylesheets

Many existing Struts applications use the rewrite tag to create a link element for a cascading style sheet. This is not the exact intention of the rewrite tag, which is supposed to create the same path as the link tag without the <a> element. Since the Struts portlet framework had to modify how links are created, the rewrite tag required some customizations to be used to create link elements for style sheets.

Archived

The rewrite tag will create the same path as the link tag, except when the page or forward reference is to a CSS file. In the case where a CSS file is referenced, the rewrite tag will use the Jakarta Struts implementation, which results in a path to the CSS file. Here are examples of how to create link elements for stylesheets using the Struts portlet framework:

- Using a forward

```
<link rel="stylesheet" type="text/css" href="<html:rewrite
forward='baseStyle'/>">
```

- Using a page

```
<link rel="stylesheet" type="text/css" href="<html:rewrite
page='/basestyle.css'/>">
```

- Using the portlet tags

```
<%@ taglib uri="/WEB-INF/tld/portlet.tld" prefix="api" %>
<api:init />
<link rel="stylesheet" type="text/css"
      href="<%= portletResponse.encodeURL( basestyle.css ) %>">
```

Markup support

The Struts tag library has been modified to support the additional markup languages supported by WebSphere Portal. For HTML, the tags that create links have been modified to support portlet URIs. See “Creating portlet URIs” on page 423 for details.

There might be cases where the JSPs for a Struts application need to run in both the servlet and portlet environment. For this reason, page level tags are implemented in tag libraries. The Struts application can use them in its JSPs, but the tags will not generate markup when executed within WebSphere Portal.

You should also refrain from setting color, and fonts. The portal server supports skins and themes that give the page a consistent look and feel. The JSP should be authored so it adheres to the conventions of the theme by using the appropriate style sheet.

Using the cHTML tags

The use of the cHTML tags is similar to the use of the HTML tags. The name of the cHTML tag library file is struts-chtml.tld. The following is an example of the cHTML taglib definition.

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

Archived

Using the WML tags

The WML tags are a new addition for creating a user experience in WAP devices. The use of the WML tags is similar to the use of the Struts HTML tags. Use the following directive to make these tags available to a JSP.

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

The use of the WML tags provided with this distribution is similar to the use of Struts HTML tags. The name of the WML tag library file is struts-wml.tld. The following is an example of the WML version of an index.jsp.

Example 13-2 Using the wml tag in struts portlets

```
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<%@ taglib uri="/WEB-INF/struts-wml.tld" prefix="wml" %>
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>
<p>
<wml:link page="/editRegistration.do?action=Create">
   <bean:message key="index.registration"/>
</wml:link>
<br/>
<wml:link page="/logon.jsp">
   <bean:message key="index.logon"/>
</wml:link>
<br/>
<wml:link page="/overview.do">
   <bean:message key="index.overview"/>
</wml:link>
</p>
```

WML does not have a <form> element like HTML. Struts, however, uses the <form> tag for the scoping of parameters and supporting form beans. For that reason, the WML implementation also includes a <form> tag to support some of the Struts features in WML. The <form> tag in the WML taglib takes an action as an attribute.

The following is an example of a form in WML.

Example 13-3 A form in WML

```
<wml:form action="/logon">

<do type="options" label="send">

  <wml:go method="post">
    <postfield name="username" value="$username"/>
    <postfield name="password" value="$password"/>
  </wml:go>
</do>
```

Archived

```
<bean:message key="prompt.username"/><wml:text property="username"/>
<bean:message key="prompt.password"/><wml:password property="password"
size="16" maxlength="16"/>

</p>

</wml:form>
```

13.2.3 Configuration files

Struts portlet framework specific init parameters have been added for the configuration files that allow customizing the Struts application for the portal environment. The portlet and Web deployment descriptors require configurations specific to the Struts portlet framework. In addition, the Struts configuration file must be changed to specify a portal specific request processor as the controller. These configurations will be covered in the next chapters.

13.2.4 Creating link tags in Struts

Tags that create links need to create links that are serviced by the portlet. The URL that is created because of the Struts processing needs to be passed as a parameter back to the portlet. There should be a common tag for both a servlet and a portlet. The code fragment below demonstrates how to write a tag that can be used by a JSP in both a servlet and a portlet.

Example 13-4 Code to create a link that works in both a servlet and a portlet environment

```
PortletApiUtils portletUtils = PortletApiUtils.getInstance();
if (portletUtils != null) {
   Object pResponse = portletUtils.getPortletResponse((HttpServletRequest)
                                                  pageContext.getRequest());
   Object portletURI =
      portletUtils.createPortletURIWithStrutsURL(pResponse,
                  calculateURL());

   results.append(portletURI.toString() );
} else {
   // servlet environment
   results.append(calculateURL());
}
```

The following are additional considerations:

- The else clause was the original statement for the servlet-only environment

Archived

- An instance of `PortletAPIUtils` is initially obtained. In a non-portlet environment this call would return null.
- The initialization of the Struts portlet framework support would set the instance of the `PortletAPIUtils` implementation.
- A `PortletResponse` object is obtained as an Object
- The `PortletURI` is then created with a `DefaultPortletAction` and a parameter that contains the Struts path through the call to `createPortletURIWithStrutsURL`
- More about links can be found in the chapter on Struts portlet framework in WebSphere Portal Infocenter.

Archived

Archived

14

Creating Struts portlets with the IBM Portlet API

In this chapter, we will cover the development of Struts portlets in Rational Application Developer using the IBM Portlet API.

Note: The portlet application described in this chapter has the following characteristics:

- Portlet API: IBM Portlet API
- Application type: Struts

© Copyright IBM Corp. 2005. All rights reserved.

14.1 Overview

Struts portlet projects share common characteristics with standard portlets and Struts projects, although there are some differences of which you should be aware. The Struts portlet project structure and related resources are dictated by the Struts portlet framework support provided by WebSphere Portal and include in Rational Application Developer.

Struts portlet projects are created using the New Portlet Project wizard. A default Struts-type portlet and, optionally, a Web diagram file will be added in the process of creating the project. The wizard automatically generates Struts portlet configuration files and the necessary updates to the web.xml file, and adds all of the Struts portlet framework tag libraries and JAR files to the project in the directory structure that is required.

We will also add support to Edit mode to show how to update portlet objects (namely the PortletData) in order to store user preferences.

14.2 Creating Struts applications with IBM portlet API

The following sections show how a Struts portlet application is created in Rational Application Developer, configured to work with the IBM portlet API.

14.2.1 Creating a Portlet project

To create a Portlet project, follow these steps:

1. In the menu, select **File** → **New** → **Project**.
2. In the New Project window, select the wizard for **Portlet Project**.
3. Click **Next >**.
4. If the Confirm Enablement window appears, asking if you want to enable the portal development role, click **OK**.

Archive

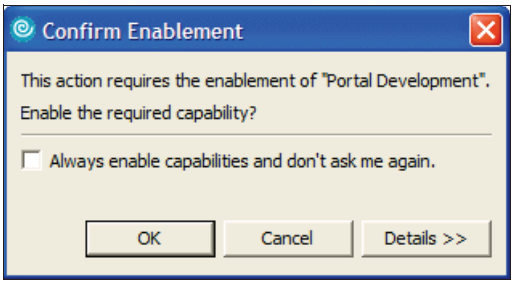


Figure 14-1 Enable the Portal development role

5. In the Portlet project page, type the name of the project: `MyFirstStruts`.
6. Check the **Create a portlet** check box, if it is not already checked. Typically, you do not need to create a portlet when you import a portlet WAR file into the project.
7. Click **Show Advanced >** and select **WebSphere Portal V5.1 stub** for the target server.

Archived

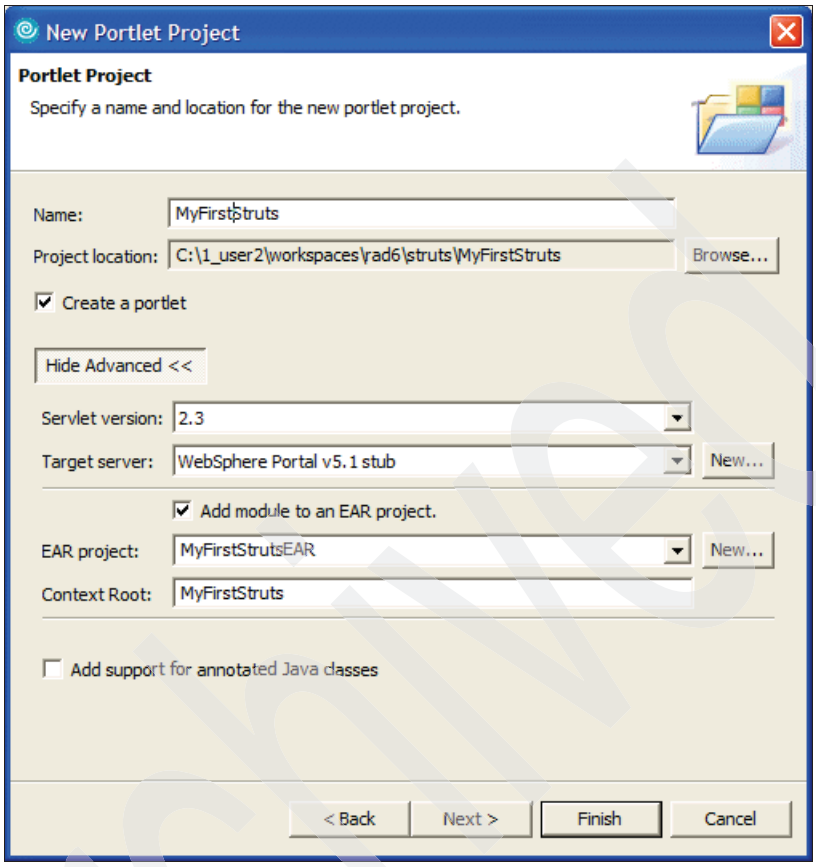


Figure 14-2 The target server you need is WebSphere Portal V5.1 stub

8. Click **Next >**.
9. In the Portlet type page, select **Struts portlet**. Click **Next >**.
10. In the Features page, leave the **Web Diagram** option selected. Click **Next >**.
11. In the Portlet settings page, review the options provided.
    The settings provided in this page are the same when you select a basic portlet.
12. Leave all settings unchanged. Click **Next >**.
13. In the Struts portlet settings page, you can specify the name of the resource bundle that will be used for this Struts portlet application, as well as the Java package where it will be stored. Do not make any changes to these fields.
14. Click **Next >**.

15. In the Miscellaneous page, you can select additional markups and modes that will be supported in this portlet application. Do not check any of the options at this time; we will cover them later in the chapter.
16. Click **Finish**.
17. If the Confirm Perspective Switch window appears, click **YES**.

14.2.2 Inspecting the Struts portlet project

Now that the Struts portlet has been created, let's take a look at what Rational Application Developer have done automatically for us:

- All of the necessary *.tld files were included in the WEB-INF directory.
- All of the necessary *.jar files were included in the WEB-INF/lib directory.
- The struts-config file was created, and the <controller> element that defines the RequestProcessor subclass was automatically generated in the Struts configuration file.

```
<controller
processorClass="com.ibm.wps.portlets.struts.WpsRequestProcessor">
</controller>
```

- The web.xml deployment descriptor was created.
  - A servlet was created and the WpsStrutsPortlet class was specified as the servlet class, which takes an init-param to specify the struts-config file to be used and another to specify the struts mapping. Other init-params were also included but omitted in the sample below to simplify understanding. For more information about all the elements added, see the WebSphere Portal Infocenter.

```
<servlet id="Servlet_1106865613683">
   <servlet-name>myfirststruts.MyFirstStrutsPortlet</servlet-name>
   <display-name>myfirststruts.MyFirstStrutsPortlet</display-name>
   <servlet-class>
   com.ibm.wps.portlets.struts.WpsStrutsPortlet</servlet-class>
   <init-param>
      <param-name>config</param-name>
      <param-value>/WEB-INF/struts-config.xml</param-value>
   </init-param>
   <init-param>
      <param-name>struts-servlet-mapping</param-name>
      <param-value>*.do</param-value>
   </init-param>
</servlet>
```

Archived

Note: The definition of a `config` parameter above is the standard way to define a Struts configuration file and modules. In the Struts portlet framework, each markup and mode is mapped to a specific Struts configuration file and module. If you want to add support for Edit mode in the WML markup language, for example, you should create a `config/wml/edit` parameter pointing to the Struts configuration file for this mode/markup. For more information, refer to 14.2.6, “Adding support to Edit mode” on page 446.

  - Struts portlet tag library definitions were included.
  - Welcome-file list support for Struts modules was supplied.

```
<welcome-file-list>
   <welcome-file>index.html</welcome-file>
   <welcome-file>index.htm</welcome-file>
   <welcome-file>index.jsp</welcome-file>
   <welcome-file>default.html</welcome-file>
   <welcome-file>default.htm</welcome-file>
   <welcome-file>default.jsp</welcome-file>
</welcome-file-list>
```

- The portlet.xml deployment descriptor was created.

14.2.3 Designing the application

The easier way to design our new Struts portlet application is to use the Web diagram editor, and realize the Web page nodes you create in the diagram to launch the proper Rational Application Developer wizards. You can also use the **File** → **New** command on the top menu, or the options in the context menus, but the Web diagram gives a good overview of how the different pages and actions interact in the Struts portlet application.

To design the MyFirstStruts application, follow these steps:

1. Make sure you have the Web perspective active.
2. In the Project Explorer view, double-click the **Web Diagram**.
3. The Web diagram opens.
   You can easily add nodes to the Web diagram using the Palette view. Note that you have some Struts parts that you can pick up from the palette and insert in the Web diagram.

Archived

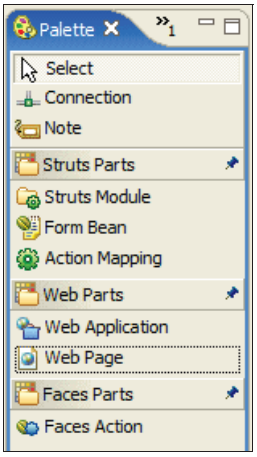


Figure 14-3 The palette view

4. Click **Web Page** on the palette view, then click anywhere in the Web diagram to place it. Name the Web page `index.jsp`.
   You should use this name because this will be the first page in our application. The welcome-file list in the web.xml deployment descriptor is already configured to find this page automatically.
5. Repeat these steps to place two other pages in the Web diagram. Name them `configured.jsp`. and `notConfigured.jsp`.
6. Click **Action Mapping**, click again anywhere in the Web page diagram and name the action mapping `welcome`.
7. Click **Connection**. Then connect index.jsp to the `welcome` action.
8. Click **Connection**. Then connect the `welcome` action to `configured.jsp`.
   d. In the Choose a connection window that appears, expand **Local Forward**, then select **<new>** and click **OK**.
   e. Name the connection `configured`.
9. Click **Connection** again. Then connect `welcome` to `notConfigured.jsp`.
   a. In the Choose a connection window that appears, expand **Local Forward**, then select **<new>** and click **OK**.
   b. Name the connection `notConfigured`.

At this time, your Web diagram should look like Figure 14-4 on page 436.

Archived

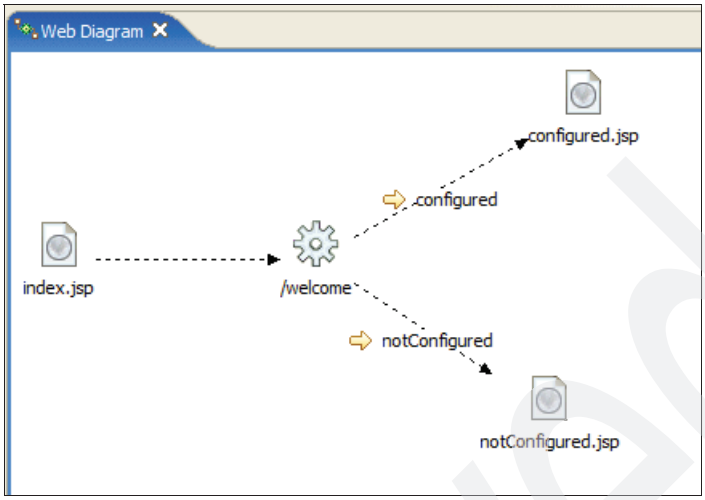


Figure 14-4 The Web diagram

10. Save the Web diagram.

14.2.4 Realizing the application components

Now that all components were placed at the Web diagram, it is time to realize them in real components.

1. Create a global forward in struts-config.xml that will be used by index.jsp.
   a. In the Project explorer view, expand **Dynamic Web Projects** → **MyFirstStruts** → **Struts** → **<default module>**.
   b. Double-click **struts-config.xml**, as shown in Figure 14-5 on page 437.

Archived

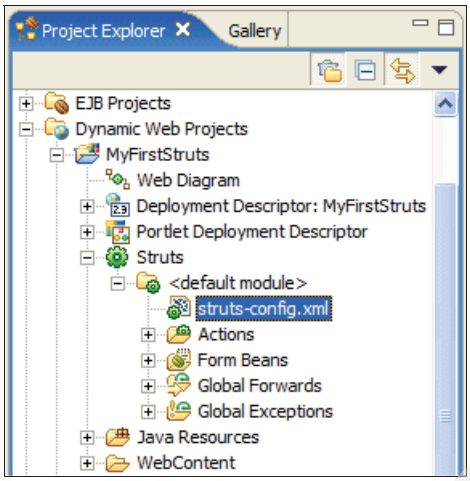


Figure 14-5 Locating the struts-config.xml file

c. In the Struts configuration file editor, select the **Global Forwards** tab, at the bottom.

d. Click **Add**. Name the global forward `welcome`.

e. Under the forward attributes section, enter `/welcome.do` as the Path. Figure 14-6 on page 438 shows the global forward page filled.

Archived

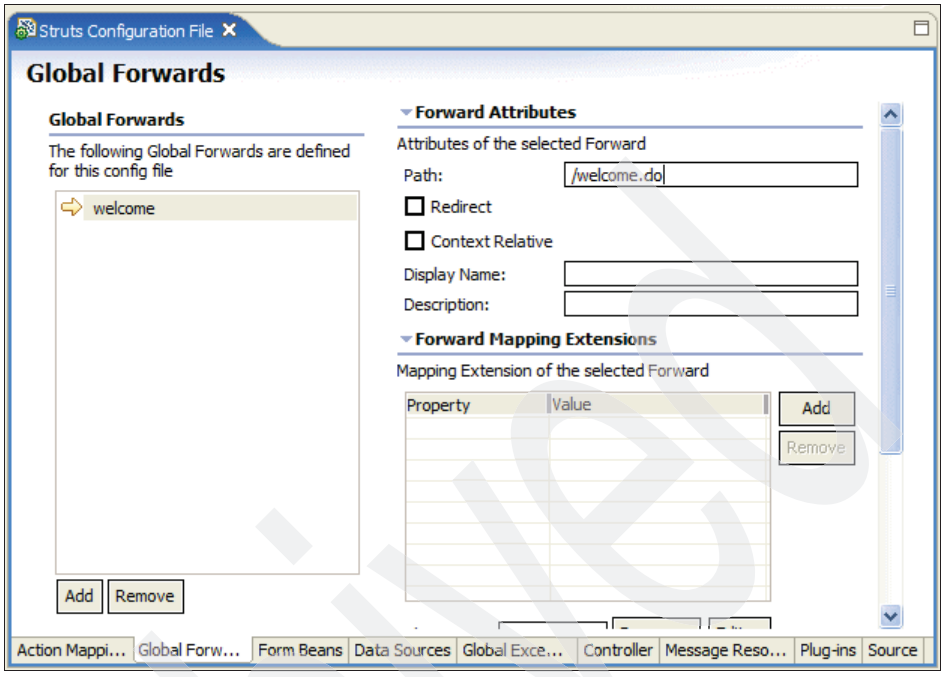


Figure 14-6 The global forward page

f. Save and close the editor.

2. The first component you will realize is the index.jsp file.

   a. In the Web diagram, double-click **index.jsp**.

   b. The New JSP file wizard appears. All fields should have been properly filled by the wizard. Refer to Figure 14-7 on page 439 to make sure of this.

Archive

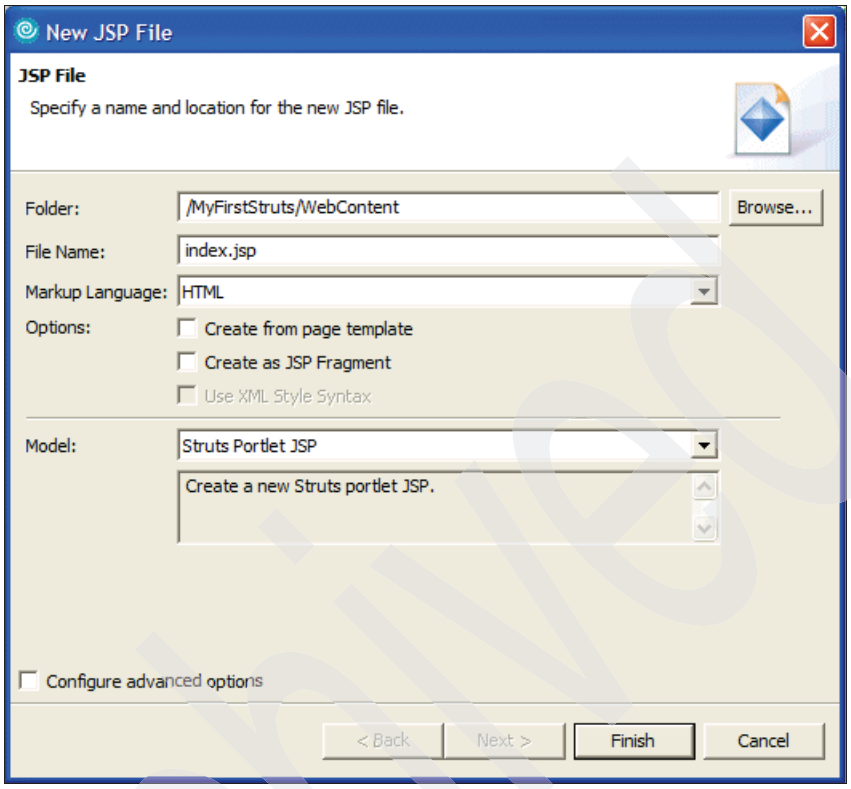


Figure 14-7 New JSP File window

c. Check **Configure advanced options** and click **Next >**.

d. In the Tag libraries page, remove all tag libraries by selecting them and clicking **Remove**.

e. Click **Add**.

f. In the Select a tag library page, check the **/WEB-INF/struts-logic.tld** tag library.

g. Click **OK**.

h. Click **Next**.

i. Click **Next** twice more.

j. In the Form field selection page, uncheck **Generate fields in a form**.

k. Click **Finish**.

l. In the JSP editor, paste the following code at the end of the file:

```
<logic:forward name="welcome"/>
```

This will automatically forward to the global forward `welcome` mapping, that is, the welcome.do action mapping you will build.

m. Save the file and close the JSP editor.

3. The next component you will realize is the notConfigured.jsp file.
   a. In the Web diagram, double-click **notConfigured.jsp**.
   b. The New JSP file wizard appears. Uncheck the **Advanced Options** check box.
   c. Click **Finish**.
   d. In the JSP editor, erase all code that was automatically generated by the wizard.
   e. Paste the code from Example 14-1.

Example 14-1 notConfigured.jsp code

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

<h3><bean:message key="myfirststruts.title"/></h3>
<em><bean:message key="message.mustConfigureFirst"/></em>
```

We will create the messages in the resources file later on.

   f. Save the file and close the JSP editor.

4. Now, realize the configured.jsp file.
   a. In the Web diagram, double-click **configured.jsp**.
   b. The New JSP file wizard appears.
   c. Click **Finish**.
   d. Erase all code that was automatically generated by the wizard and paste the code from Example 14-2:

Example 14-2 configured.jsp code

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

<h3><bean:message key="myfirststruts.title"/></h3>
<em><bean:message key="message.configured"/></em>
```

5. Edit the resources file.
   a. In the Project Explorer view, expand **Java Resources** → **JavaSource** → **myfirststruts.resources**.
   b. Double-click **ApplicationResources.properties**.

Archived

c. Insert the lines shown in Example 14-3.

Example 14-3 ApplicationResources.properties

```
myfirststruts.title=Struts portlet with IBM Portlet API

message.mustConfigureFirst=You must configure this portlet through edit mode.
message.configured=This portlet has been configured.
```

d. Save the file and close the editor.

6. Now you will create a utility class to store the application constants.
   a. In the Project Explorer view, expand **Java Resources**.
   b. Right-click **JavaSource** and select **New** → **Class**.
   c. Enter `myfirststruts` for the package and `Constants` for the class name. Click **Finish**.
   d. Paste the following code in the class:

```
public static final String USERNAME = "username";
public static final String PASSWORD = "password";
```

   e. Save the file and close it.
7. Now you will realize the `welcome` action mapping.
   a. In the Web diagram, double-click **welcome**.

      The New Action Mapping wizard appears, as shown in Figure 14-8 on page 442.

      Note that the wizard has filled many data in this window. These data came from the Web diagram you designed.

      - The name of the Action mapping became the *Action mapping path*.
      - The connections you created as Local forwards were automatically inserted in the Forwards list.
      - The **Create an Action class** radio button was selected, and the Model chosen was the **Struts Portlet Framework Action Mapping**. This is the mapping provided by Struts portlet framework.

   b. Leave all data unchanged and click **Next >**.
   c. The Create an Action class for your mapping shows that our mapping will inherit from com.ibm.wps.struts.action.StrutsAction.
   d. Click **Finish**.

Archived

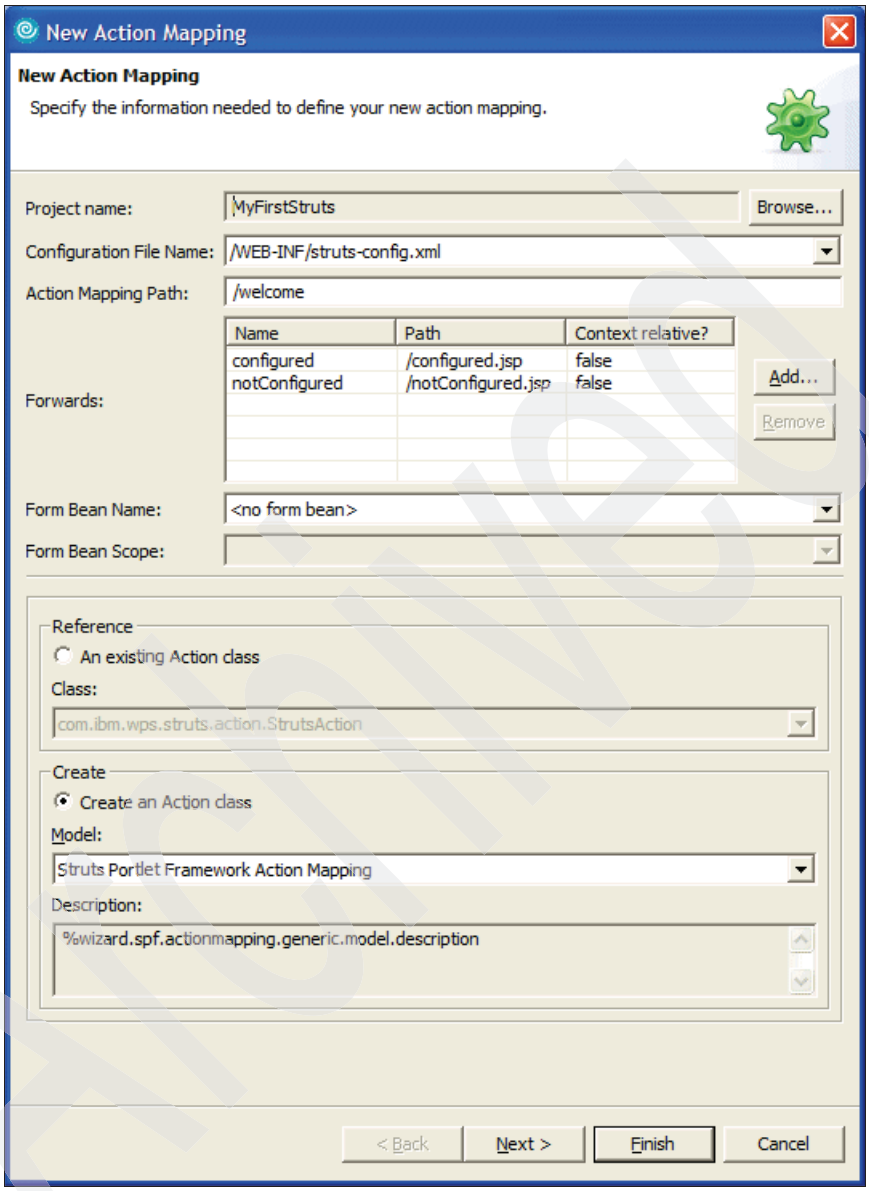


Figure 14-8 The New Action Mapping wizard

e. Replace the execute method with the code from Example 14-4 on page 443:

Example 14-4 The execute method of the welcome Action Mapping

```
ActionForward forward = mapping.findForward("notConfigured");
try {
   PortletApiUtils portletUtils = PortletApiUtils.getUtilsInstance();

   if (portletUtils != null) {
      PortletRequest portletRequest = (PortletRequest)
      portletUtils.getPortletRequest( request );
      PortletData portletData = portletRequest.getData();
      if ( portletData.getAttribute( Constants.USERNAME ) != null ) {
         forward = mapping.findForward("configured");
      }
   }
} catch ( Exception ex ) {
}
return forward;
```

Note that the `PortletApiUtils` is used in order to retrieve the `PortletData` object. You can test to see if the data that the application expects (here, the username) is already stored at the `PortletData` and forward to different pages en each case.

Note also that you have silenced the catch statement. This is not a good practice, but we will show how to enable logging in the application in 14.2.5, “Adding logging support” on page 444.

f. Right-click anywhere in the code and select **Source** → **Organize imports**.

g. Select **com.ibm.portal.struts.common.PortletApiUtils** for the `PortletApiUtils` class and **myfirstStruts.Constants** for the `Constants` class.

h. Click **Finish**.

i. Save the file and close the editor.

8. Run the code with what you have done so far.

   a. Right-click the **MyFirstStruts** project.

   b. Select **Run** → **Run on Server**.

   c. In the Define a new server window, select **Manually define a server**.

   d. Select the server type **WebSphere Portal V5.1 Test Environment**.

   e. Check the **Set server as project default** check box.

   f. Click **Finish**.

   g. After a few minutes, you will see a page similar to Figure 14-9 on page 444.

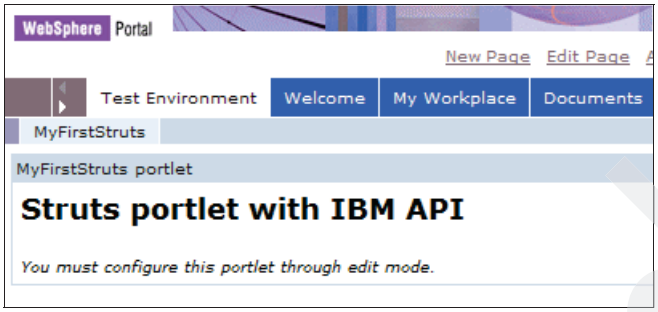


Figure 14-9 MyFirstStruts in the browser

Of course, you are not able to configure the portlet in Edit mode yet; our portlet is not configured to support that. This will be done in 14.2.6, “Adding support to Edit mode” on page 446.

14.2.5 Adding logging support

The Struts Portlet Framework uses the Commons-Logging interface for a logging facility. The Struts Portlet Framework has supplied an implementation of the Commons-Logging Log interface that can be used to map the trace messages to the logging facility used by the portal server. This file is normally found in the *wp_root*/log directory. Trace logging is enabled by setting properties in the *wp_root*/shared/app/config/log.properties file. By default, tracing is disabled. The trace string for tracing a Struts portlet application can be configured in log.properties. The trace string can be modified to add or remove classes.

To start logging our application, proceed as follows:

1. First, enable tracing of struts portlets; proceed as follows:
   a. Open the *wp_root*/shared/app/config/log.properties, where *wp_root* is the folder where the Portal test environment was installed.
   b. Go to the end of the file and add the following traceString.

```
traceString=org.apache.struts.*=all=enabled:
   com.ibm.wps.portlets.struts.*=all=enabled:
   com.ibm.wps.struts.common.*=all=enabled:
   com.ibm.wps.struts.base.*=all=enabled:
   com.ibm.wps.portlets.struts.logging.WpsStrutsTraceLogger=all=enabled:
   myfirststruts.*=all=enabled
```

Important: You can have only one traceString in the log.properties file, and the entire traceString should be specified in only one line.

c. Save the file and close it.

2. Additionally, the Struts Portlet Framework specifies the common logging Log Factory in the META-INF/services directory. The log factory class name is dependent on the WebSphere Portal container. To specify the correct commons `LogFactory` for our application, proceed as follows:

   a. Expand **Dynamic Web Projects** → **MyFirstStruts** → **WebContent**.

   b. Right-click the **META-INF** folder. Select **New** → **Folder**.

   c. Enter `services` as the folder name.

   d. Click **Finish**.

   e. Right-click the **services** folder. Select **New** → **Other**.

   f. In the Select a wizard window, expand **Simple**. Select **File**.

   g. Click **Next >**.

   h. Enter `org.apache.commons.logging.LogFactory` as the file name.

   i. Click **Finish**.

   j. Paste the following line in the file:

   ```
   com.ibm.wps.portlets.struts.logging.StrutsLogFactory
   ```

   k. Save the file and close it.

3. Now let's add some code in the welcome action mapping.

   a. Edit myfirststruts.actions.WelcomeAction.

   b. Add the following instance variable:

   ```
   private Log log = LogFactory.getLog(this.getClass());
   ```

   c. Right-click anywhere in the code and select **Source** → **Organize Imports**.

   d. Select **org.apache.commons.logging.Log** for the `Log` class and click **Next >**.

   e. Select **org.apache.commons.logging.LogFactory** for the `LogFactory` class and click **Finish**.

   f. Add the following code at the beginning of the execute method:

   ```
   if (log.isTraceEnabled()) {
      log.trace("###Welcome###");
   }
   ```

   g. Finally, add the following code in the catch statement:

   ```
   if (log.isDebugEnabled()) {
      log.debug("WelcomeAction: Error determining if user is configured");
   }
   ```

Archived

h. Save the file and close the editor.

4. Now you can run the application again, open the log file at *wp_root*/log folder and look for the Welcome message. You should see a message similar to the following:

```
2005.02.01 17:25:10.461 l myfirststruts.actions.WelcomeAction trace
Servlet.Engine.Transports : 1
  ###Welcome###
```

Note 1: You have to restart the server if it is still running, since you made changes to the configuration files.

14.2.6 Adding support to Edit mode

To add support to Edit mode in our portlet, proceed as follows:

1. Edit the web.xml deployment descriptor.
   a. Expand **Dynamic Web Projects** → **MyFirstStruts.**
   b. Double-click **Deployment Descriptor: MyFirstStruts**.
   c. In the editor, go to the servlets page.
   d. Select **myFirstStruts.MyFirstStrutsPortlet**.
   e. In the Initialization section, click **Add**.
   f. Enter `config/html/edit` as the parameter name, and `/WEB-INF/html/edit/struts-config.xml` as the parameter value.
   g. Click **Finish**.
   h. Go to the **Pages** tab.
   i. Under the Welcome pages section, click **Add**.
   j. Enter `html/edit/welcome.jsp` as the new welcome page name. This will be the welcome page in Edit mode, for a html markup.
   k. Save the file and close the editor.
2. Edit the portlet deployment descriptor.
   a. Double-click **Portlet Deployment Descriptor**.
   b. Expand **Portlet Application**.
   c. Select **myfirststruts.MyFirstStrutsPortlet**.
   d. In the Markups section, click under **Edit** and select **Fragment**. (Figure 14-10 on page 447).

Archived

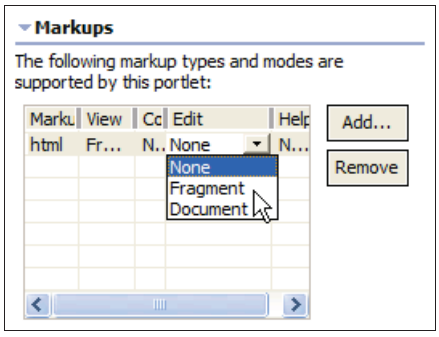


Figure 14-10 Adding support to Edit mode

e. Save the file and close the editor.

3. Add a new struts module to support the Edit mode.

   a. In the Project Explorer view, right-click the **Struts** icon.

   b. Select **New** → **Module**.

   c. Fill in all the fields according to Figure 14-11 on page 448.

      Note that you pointed to the same resources file that you had already created.

Archived

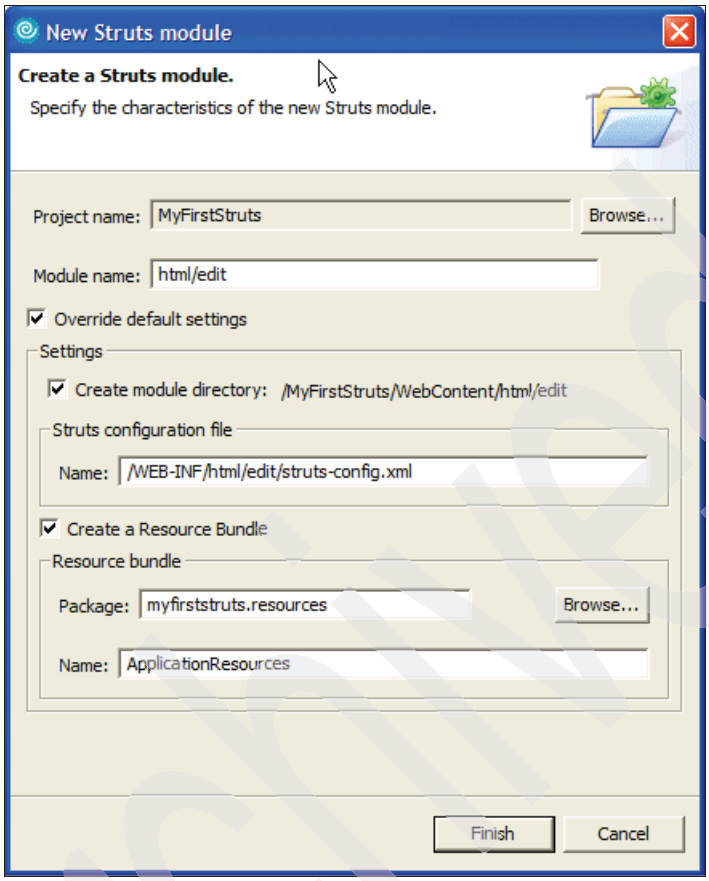


Figure 14-11 New Struts module wizard

d. Click **Finish**.

At this time, a new Struts module should appear under the Struts icon, as shown in Figure 14-12.

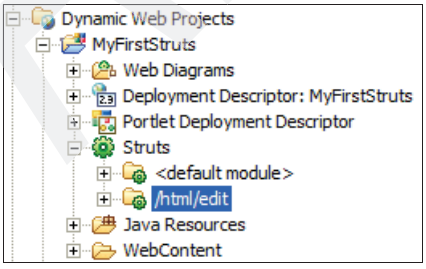


Figure 14-12 The new Struts module

4. Create a new Web diagram for the Edit mode.
   a. Right-click **Web Diagram**.
   b. Select **New** → **Web Diagram**.
   c. Enter `editMode` as the Web diagram file name.
   d. Place four Web pages in the diagram. Name them `html/edit/welcome.jsp`, `html/edit/index.jsp`, `configured.jsp` and `notConfigured.jsp`.
   e. Place two action mappings in the diagram. Name them `saveConfiguration` and `editConfiguration`.
   f. Place a form bean in the diagram. In the Form bean attributes window that appears, enter `userBean` as its name and `request` as the scope.
   g. Create the connections between the components; these are:
      i. welcome.jsp to editConfiguration.
      ii. editConfiguration to userBean.
      iii. saveConfiguration to userBean.
      iv. Local forward named `success` from editConfiguration to index.jsp.
      v. index.jsp to saveConfiguration.
      vi. Local forward named `success` from saveConfiguration to editConfiguration.
      vii. Local forward named `user_not_set` from saveConfiguration to notConfigured.jsp.
      viii. Local forward named `user_set` from saveConfiguration to configured.jsp.
   h. Right-click anywhere in the Web diagram. Select **Change the Struts module association**.
   i. Select **/html/edit** in the list. Click **OK**.
   j. Your diagram should look like Figure 14-13 on page 450. Save it.

Archived

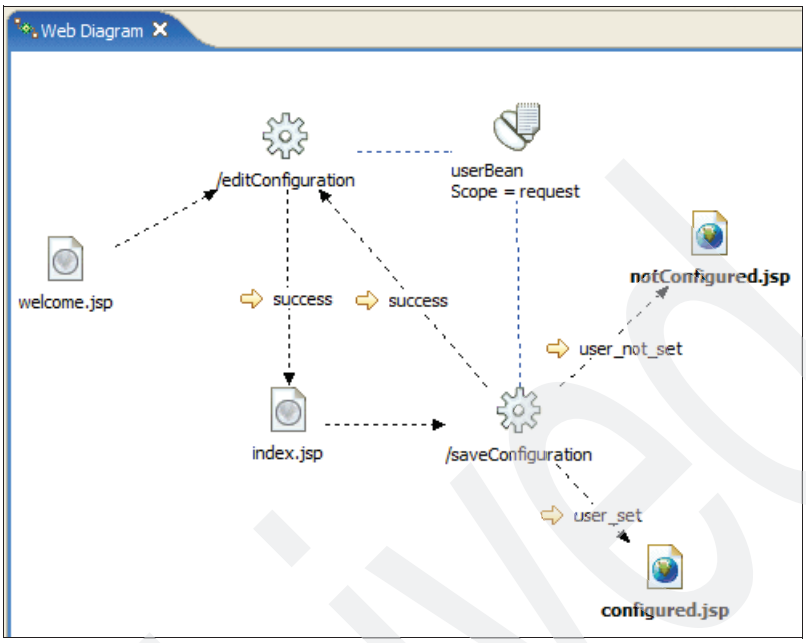


Figure 14-13 The Web diagram for Edit mode

14.2.7 Realizing the new application components

Now that it is time to realize all the components that will support Edit mode in our portlet.

1. Create a global forward in struts-config.xml, that will be used by index.jsp.
   a. In the Project Explorer view, expand **Dynamic Web Projects** → **MyFirstStruts** → **Struts** → **/html/edit**.
   b. Double-click **struts-config.xml**.
   c. Under the Global forwards page, click **Add**.
   d. Name the global forward `Welcome` and enter `/editConfiguration.do` as the path.
   e. Save the file and close editor.
2. Realize welcome.jsp.
   a. In the Web diagram for /html/edit module, double-click **welcome.jsp**.

      The New JSP file wizard appears. Note that the folder is correctly pointing to the html/edit folder under the Web application, since you inserted this information when you created the Web page in the Web diagram.

Archived

b. Click **Finish**.

c. Replace all code in the new jsp with the following code:

```
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic"%>
<logic:forward name="Welcome"/>
```

d. Save the file and close the editor.

3. Next, you will realize the `userBean`.

   a. Double-click **userBean** form bean. This will bring the New Form-Bean wizard.

   b. Leave the form bean name unchanged. Make sure that the **Create an ActionForm class** radio button is selected, and that **Generic Form-bean Mapping** is selected.

   c. Click **Next >.**

   d. Click **Next >** again.

   e. In the Create new fields for your ActionForm class page, you are going to add a couple of fields.

      ix. Click **Add...**.

      x. Enter the field's name `username`. Leave the field type as `String`.

      xi. Click **Add...** again.

      xii. Enter the field's name `password` and leave the field type as `String`.

      xiii. The window will look like in Figure 14-14 on page 452.

   f. Click **Next >.**

Archived

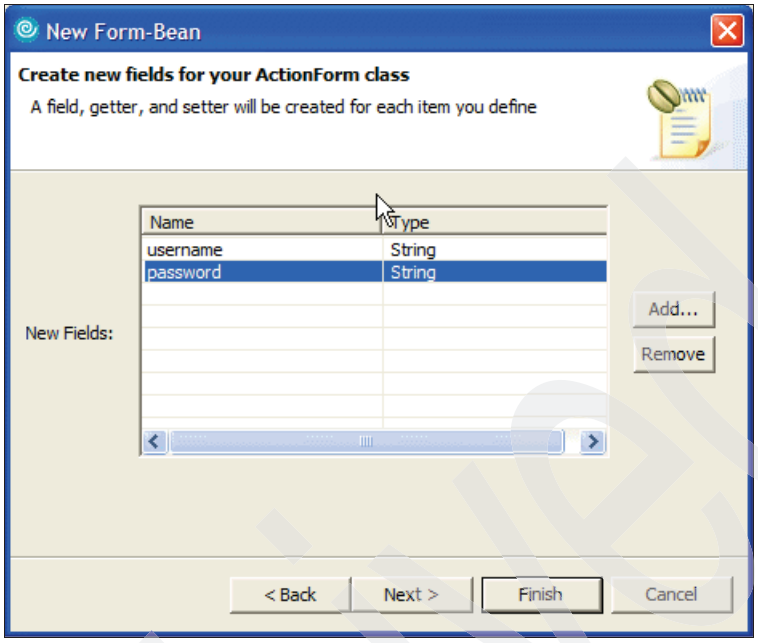


Figure 14-14 Creating new fields for the ActionForm Bean

g. In the Creating a mapping for your ActionForm class, check only **reset** (which takes an HttpServletRequest as parameter and **inherited abstract methods**. Leave all other method stubs unchecked.

h. Click **Finish**.

Rational Developer will generate the ActionForm bean, that extends from rg.apache.struts.action.ActionForm, along with its proper accessor methods. It will also include the form-bean tag in struts-config.xml file.

i. Replace the **reset** method with the following code:

```
this.username = "";
this.password = "";
```

j. Save the file and close it.

4. Next, you will realize the `editConfiguration` action mapping.

a. In the Web diagram, double-click **/editConfiguration**.

b. The New action mapping wizard appears. As every fields are already properly set, click **Finish**.

c. Replace the **execute** method with the following code:

Example 14-5 The execute method from editConfiguration action mapping.

```
saveToken(request);

form.reset(mapping, request);
return (mapping.findForward("success"));
```

The code above initially calls the `saveToken` method, that Struts provides to avoid multiple submissions from one form. Next, the form bean is reset. This forces that the user always enter new values. Finally, the action forwards to `success`, that points to `index.jsp`.

d. Save the file and close it.

5. The next component you will realize is `index.jsp`.

   a. In the Web diagram, double-click **index.jsp**.

   b. Click **Finish**.

   c. Replace the jsp contents with the code from Example 14-6:

Example 14-6 index.jsp for Edit mode

```
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>

<h3><bean:message key="editmode.heading"/></h3>

<p><bean:message key="editmode.instructions"/></p>

<ul>
    <li><bean:message key="editmode.instructions.1"/></li>
    <li><bean:message key="editmode.instructions.2"/></li>
    <li><bean:message key="editmode.instructions.3"/></li>
    <%--<li><bean:message key="editmode.instructions.4"/></li>--%>
    <li><bean:message key="editmode.instructions.5"/></li>
</ul>

<ul>
  <html:messages message="true" id="error">
    <li><font color="red"><bean:write name="error"/></font></li>
  </html:messages>
</ul>

<html:form action="/saveConfiguration.do" urlType="return"
validate="false">

<table>

  <tr>
```


Archived

```
      <th align="right">
        <bean:message key="prompt.username"/>: 
      </th>
      <td align="left">
        <html:text property="username" size="25"/>
      </td>
    </tr>

    <tr>
      <th align="right">
        <bean:message key="prompt.password"/>: 
      </th>
      <td align="left">
        <html:password property="password" size="25"/>
      </td>
    </tr>

    <tr><td><br></td></tr>

    <tr>
      <td align="right">
        <html:submit value="Submit"/>
      </td>
      <td align="left">
        <html:reset/>
      </td>
    </tr>

  </table>

  </html:form>
```

Note that you added the html:messages tag, to support that messages from the action can be processed and displayed correctly.

Important: If you want execute an `Action` that will change the portlet mode to its previous mode, you must use the **urlType** attribute of the html:form tag with the value of `return`. This attribute was added by Struts portlet framework to the tags html:form, html:link and html:rewrite.

6. Finally, let's realize the `saveConfiguration` action.
   a. In the Web diagram, double-click **saveConfiguration** action.
   b. All values should be correct, as the Web diagram had all information the wizard needs to create the action.
   c. Click **Finish**.

Archived

d. This action is the one with the most logic. Replace its contents with the code in Example 14-7 and save the file:

Example 14-7 Listing of SaveConfigurationAction.java

```
package myfirststruts.actions;
import myfirststruts.Constants;
import myfirststruts.forms.UserBean;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.apache.jetspeed.portlet.PortletData;
import org.apache.jetspeed.portlet.PortletRequest;
import org.apache.jetspeed.portlet.Portlet.Mode;
import org.apache.jetspeed.portlet.Portlet.ModeModifier;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionMessage;
import org.apache.struts.action.ActionMessages;
import com.ibm.portal.struts.common.PortletApiUtils;
import com.ibm.wps.struts.action.StrutsAction;

public class SaveConfigurationAction extends StrutsAction {
   private Log log = LogFactory.getLog (this.getClass ());

   public ActionForward execute(ActionMapping mapping, ActionForm form,
         PortletRequest request) throws Exception {

      ActionForward forward = mapping.findForward("success");
      ActionMessages messages = new ActionMessages();

      if (form instanceof UserBean) {
         UserBean userBean = (UserBean) form;
         try {
            PortletApiUtils portletUtils =
                     PortletApiUtils.getUtilsInstance();
            if (portletUtils != null) {
               PortletRequest portletRequest =
                     (PortletRequest) portletUtils
                     .getPortletRequest(request);
               PortletData portletData = portletRequest.getData();

               if (portletRequest.getMode() == Mode.EDIT) {
                  processEditData(request, messages, userBean,
                        portletData);
               } else {
                  log.debug("this command is only valid in edit mode");

                  forward = null;
```

Archived

```
                }
                if (!messages.isEmpty()) {
                   saveMessages(request, messages);

                   portletRequest.setModeModifier(ModeModifier.CURRENT);

                   forward = mapping.getInputForward();
                } else {
                   if (portletData.getAttribute(Constants.USERNAME)
                         != null) {
                      log.debug("Set view mode to user_set");
                      ActionForward actionForward =
                            mapping.findForward("user_set");
                      if (actionForward != null) {
                         portletUtils.createCommand(actionForward
                               .getPath(), request, Mode.VIEW.toString());
                      } else {
                         log.debug("Could not find action forward for view
mode");
                      }
                   } else {
                      log.debug("Set view mode to user_not_set");
                      ActionForward actionForward =
                            mapping.findForward("user_not_set");
                      if (actionForward != null) {
                         portletUtils.createCommand(actionForward.
                               getPath(), request, Mode.VIEW.toString());
                      } else {
                         log.debug("Could not find action forward for view
mode");
                      }
                   }
                }
             }

          } catch (Exception ex) {
             log.debug("Could not save configuration " + ex.toString());
          }

       }
       return forward;
    }

   private void processEditData(PortletRequest request, ActionMessages
messages, UserBean userBean, PortletData portletData) throws Exception {
       if (!isTokenValid(request)) {
          messages.add(ActionMessages.GLOBAL_MESSAGE,
                new ActionMessage("error.transaction.token"));
          log.debug("token is not valid");
```


Archived

```
        } else {
           log.debug("token is valid");
           // token was valid.
           String username = userBean.getUsername();

           if (username != null && !username.equals("")) {
              if (username.equals("error")) {
                 messages.add(ActionMessages.GLOBAL_MESSAGE,
                        new ActionMessage("error.invalid.username"));

              } else {
                 portletData.setAttribute(Constants.USERNAME,
                        userBean.getUsername());
              }
           } else {
              portletData.removeAttribute(Constants.USERNAME);
           }

           if (userBean.getPassword() != null &&
                 !userBean.getPassword().equals("")) {
              portletData.setAttribute(Constants.PASSWORD,
                    userBean.getPassword());
           } else {
              portletData.removeAttribute(Constants.PASSWORD);
           }
           portletData.store();
        }
        resetToken(request);
     }
}
```

Let's take a look at the major points of this code:

- First of all, you had to retrieve the `PortletApiUtils` instance. It is the PortletApiUtils that gives us access to the PortletRequest object.
- Following, you check if the portlet is in Edit mode, with the following code:

  ```
  if (portletRequest.getMode() == Mode.EDIT)
  ```

  This way, you guarantee that the action will work only and if only the portlet mode is Edit.
- Next, the `processEditData` method is executed. First of all, it checks if the token is valid, with

  ```
  if (!isTokenValid(request))
  ```

  This is to avoid multiple submits of the form.

Archived

If the username and password attributes from `UserBean` are null, the code removes these attributes from `PortletData`. If not, the code stores them in `PortletData`, using the `setAttribute` method. The only exception is when the enter `error` as the username. In this case, a message is generated, stating that this is an invalid user name. The `processEditData` also stores the `PortletData` object with its new attribute values, and reset the token.

- Back to the execute method, the code checks if there is any message to be shown. If there is any message, then it obligates the portlet to stay at the same mode, with

  ```
  portletRequest.setModeModifier(ModeModifier.CURRENT);
  ```

  and forwards to the page that invoked this action, with

  ```
  forward = mapping.getInputForward();
  ```

- Finally, the code tests if the username attribute was stored at `PortletData`, if yes, than it creates a command that makes the View mode page change to the one specified by the `user_set` local forward.

Example 14-8 The username attribute was found. Forwarding to user_set

```
ActionForward actionForward = mapping.findForward("user_set");
if (actionForward != null) {
   portletUtils.createCommand(actionForward.getPath(), request,
         Mode.VIEW.toString());
} else {
   log.debug("Could not find action forward for view mode");
}
```

If the username attribute is not stored at `PortletData`, the View mode forward is to the `user_not_set` local forward:

Example 14-9 The username attribute was not found. Forwarding to user_not_set

```
ActionForward actionForward = mapping.findForward("user_not_set");
if (actionForward != null) {
   portletUtils.createCommand(actionForward.getPath(), request,
         Mode.VIEW.toString());
} else {
   log.debug("Could not find action forward for view mode");
}
```

7. Add new keys to resources file:
   a. Edit **myfirststruts.resources.ApplicationResources.properties**.
   b. Add the following keys at the end of the file:

Example 14-10 The new messages to be included in the resources file

```
editmode.heading=Edit Mode
editmode.title=Struts Edit Mode Example

editmode.instructions=Try the options below to demonstrate various Struts
features. When a valid username and password are given, an action forwards to a
JSP in view mode. Valid usernames and passwords are between 3 and 16 characters
long.
editmode.instructions.1=Click the <b>Submit</b> button without a valid username
and/or password to see dynamic validation
editmode.instructions.2=Enter a username of "error" and a valid password to see
ActionErrors supported
editmode.instructions.3=Enter a username of "delete" and a valid password to
remove the configuration
editmode.instructions.4=Refresh the page to demonstrate transaction support
editmode.instructions.5=Enter a valid username and password to complete the
configuration and change the mode to view mode

prompt.username=Username
prompt.password=Password

button.back=back
button.send=Send
button.save=Save
button.cancel=Cancel
button.reset=Reset

error.transaction.token=Transaction token is not valid
error.invalid.username=The username is not valid
```

c. Save the file and close editor.

8. Run the code:

   a. Right-click the `MyFirstStruts` project.

   b. Select **Run** → **Run on Server**.

   c. After the welcome page displays, click the **edit** icon, go to Edit mode.

   d. Try the code. You can choose any of the following:

   - If you leave the username blank, or if you enter `delete`, the username and password attributes will be removed from PortletData, and you will be forwarded to notConfigured.jsp.
   - If you enter `error` as the username, the application shows an error message and continues in Edit mode.
   - If you enter any other value to username and password, the application stores these values at PortletData and forwards to configured.jsp.

Archived

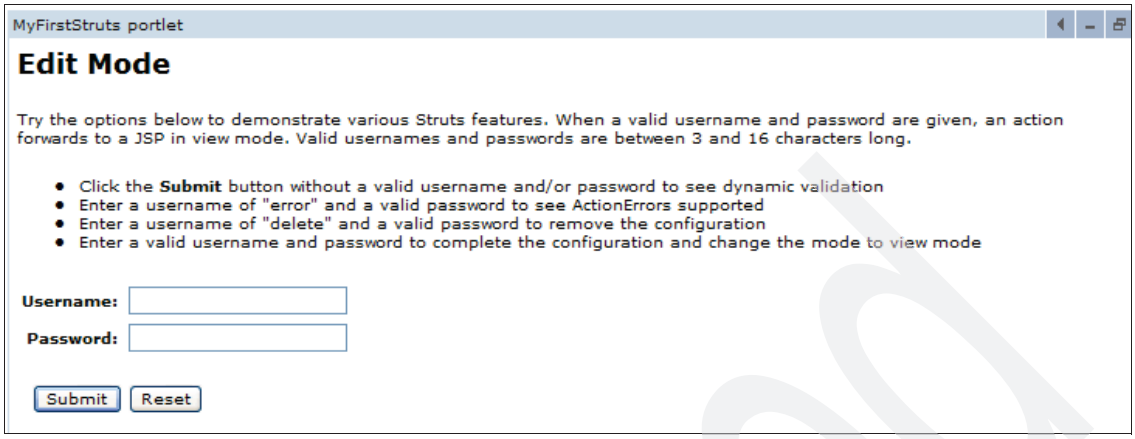


Figure 14-15 The portlet in Edit mode

14.2.8 Adding internationalization support

As you are already using the Struts tags to show messages, adding support to other languages is a straightforward task. Proceed as follows to add Brazilian Portuguese support to the application:

1. Edit the Portlet deployment descriptor.
   a. In Project Explorer view, expand **Dynamic Web Projects** → **MyFirstStruts**.
   b. Double-click **Portlet Deployment Descriptor**.
   c. In the Portlet deployment descriptor window, expand **Concrete Portlet Application**.
   d. Click **myfirststruts.MyFirstStrutsPortlet**.
   e. Under the languages section, click **Add**. See Figure 14-16.

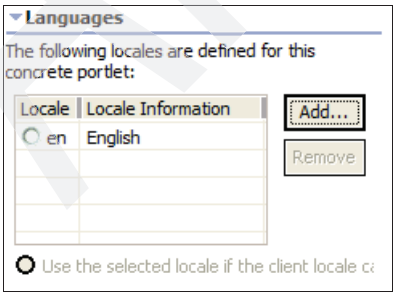


Figure 14-16 The languages section

f. Select **pt_BR** in the Locale list.

g. Click **OK**.

h. Enter `MyFirstStruts portlet` as the Title.

i. Save the file and close the editor.

2. Next, create a resources file for the new locale.

a. Expand **Dynamic Web Projects** → **MyFirstStruts** → **Java Resources** → **JavaSource** → **myfirststruts.resources**.

b. Right-click **ApplicationResources.properties**.

c. Select **Copy**.

d. Right-click the **myfirststruts.resources** package.

e. Select **Paste**.

f. Enter `ApplicationResources_pt_BR.properties` as the new name for the file.

g. Click **OK**.

h. Double-click **ApplicationResources_pt_BR.properties** to edit it.

i. Replace all messages with brazilian portuguese messages. Use the messages from Example 14-11.

Example 14-11 Resources file for brazilian portuguese Locale

```
myfirststruts.title=Portlet baseado em Struts utilizando a API IBM

message.mustConfigureFirst=Você deve primeiramente configurar este portlet em
modo de Edição.
message.configured=Este portlet foi configurado.

editmode.heading=Modo de Edição
editmode.title=Exemplo de Modo de Edição com Struts

editmode.instructions=Tente as opções abaixo para demonstrar várias capacidades
do Struts. Quando um usuário e senha válidos são inseridos, uma action
encaminha para um JSP em modo view.
editmode.instructions.1=Pressione o botão <b>Submit</b> com o usuário em branco
para remover a configuração
editmode.instructions.2=Digite "error" como nome de usuário para ver o suporte
a ActionErrors
editmode.instructions.3=Digite "delete" como nome de usuário para remover a
configuração
editmode.instructions.4=Atualize a página para demonstrar o suporte
transacional
editmode.instructions.5=Digite um nome de usuário e uma senha válidos para
completar a configuração e mudar para o modo View
```

Archived

```
prompt.username=Nome
prompt.password=Senha

button.back=voltar
button.send=Enviar
button.save=Salvar
button.cancel=Cancelar
button.reset=Limpar

error.transaction.token=Token de transação inválido
error.invalid.username=O nome de usuário não é válido
```

j. Save the file and close the editor.

3. Test the new Locale.
   a. Run the application.
   b. In the top menu, click **Edit My Profile**.
   c. Under Preferred language, select **Brazilian Portuguese**.
   d. Click **OK**.
   e. The application now should look like Figure 14-17.

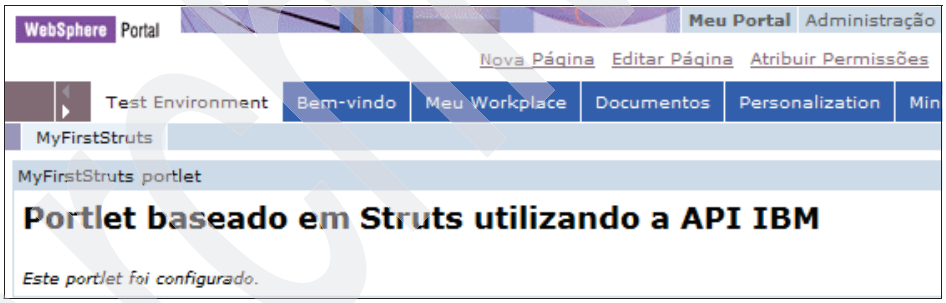


Figure 14-17 Application with new Locale set

14.3 Messaging

The recommended way to exchanges messages involving Struts portlets is through the use of the Property Broker. See Chapter 24, “IBM API declarative cooperative portlets” on page 741 for details about how to implement it. The good thing is that the Struts portlet framework does the job of sending and retrieving the messages, once that there is a match between the properties that are through the wires and the form bean’s properties.

Archived

14.4 Migration

The file structure of the WAR file for a Struts Portlet Framework is the same as that used for a Struts servlet. The Struts application in the portal server is a WAR file just like it is in the servlet environment. The portlet WAR file has some additional JARS and other requirements, but the basis of the implementation is similar to the servlet application.

Existing Struts applications can be migrated using the Struts Portlet Framework so the application can be deployed in WebSphere Portal. Since Struts is a framework there are many variations to how the application can be built with Struts. There are also numerous other frameworks that may also be incorporated into the Struts application. The steps in this section can be used as a starting point for the migration effort, but may not cover all of the issues that can be encountered.

1. Make sure the existing Struts application has been built as a Struts 1.1 application.
2. Check Struts actions to see if the action writes directly to the response object. If it does then, the action must be modified to either return an `ActionForward` instead, or check to see if the `IStrutsPrepareRender` interface should be used.
3. The Web deployment descriptor must be updated to use the `WpsStrutsPortlet` as the servlet class instead of the `ActionServlet`. (See 14.2.2, “Inspecting the Struts portlet project” on page 433)
4. The servlet mapping for Struts actions must be specified as the struts-servlet-mapping init parameter. (See 14.2.2, “Inspecting the Struts portlet project” on page 433.)
5. Create a portlet.xml.
6. Modify the struts-config.xml to specify the `WpsRequestProcessor` as the controller. (See 14.2.2, “Inspecting the Struts portlet project” on page 433)
7. Modify tags that use `pageContext.forward` to use the `PortletApiUtils` forward.
8. Modify JSPs that use a forward to use the logic forward tag.
9. Modify JSPs as necessary to use the Struts tags for creating URLs, like the Struts Link and Form tags.
10. The JAR files from the WEB_INF/lib directory of the SPFLegacyBlank.war must be used. These files are the Struts JARs and the required Struts Portlet Framework JARs. You can find the SPFLegacyBlank.war at *wp_root*/installableApps.

Archive

11. The TLD files must be updated from the WEB_INF/lib directory of SPFLegacyBlank.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. You can find the SPFLegacyBlank.war at *wp_root*/installableApps.
12. 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.

Archived

15

Struts portlet development using the JSR 168 API

In this chapter, we cover the development of Struts portlet applications in Rational Application Developer using the JSR 168 API.

The following topics are covered in this chapter:

- Generating a Struts portlet project
- Designing the application (View mode and Edit mode)
- Realizing the mode application components
- Adding internationalization support
- Adding logging support
- Running and testing the Struts portlet

Note: The portlet application described in this chapter has the following characteristics:

- Portlet API: JSR 168
- Application type: Struts

© Copyright IBM Corp. 2005. All rights reserved.

15.1 Overview

Struts portlet projects share common characteristics with standard portlets and Struts projects, although there are some differences that you should be aware of. The Struts portlet project structure and related resources are dictated by the Struts portlet framework support provided by WebSphere Portal and included in Rational Application Developer.

Struts portlet projects are created using the New Portlet Project wizard. A default Struts-type portlet and, optionally, a Web diagram file, will be added in the process of creating the project. The wizard automatically generates Struts portlet configuration files and the necessary updates to the web.xml file, and adds all of the Struts portlet framework tag libraries and JAR files to the project in the directory structure that is required.

Edit mode support will also be added to show how you can update portlet objects (portlet preferences) in order to store user preferences. The sections in this chapter show how a Struts portlet application is created in Rational Application Developer, and configured to work with the JSR 168 Portlet API.

Note: The sample scenario described in this chapter requires WebSphere Portal Test Environment version 5.1.0.1.

The sample scenario described in this chapter is illustrated in Figure 15-1.

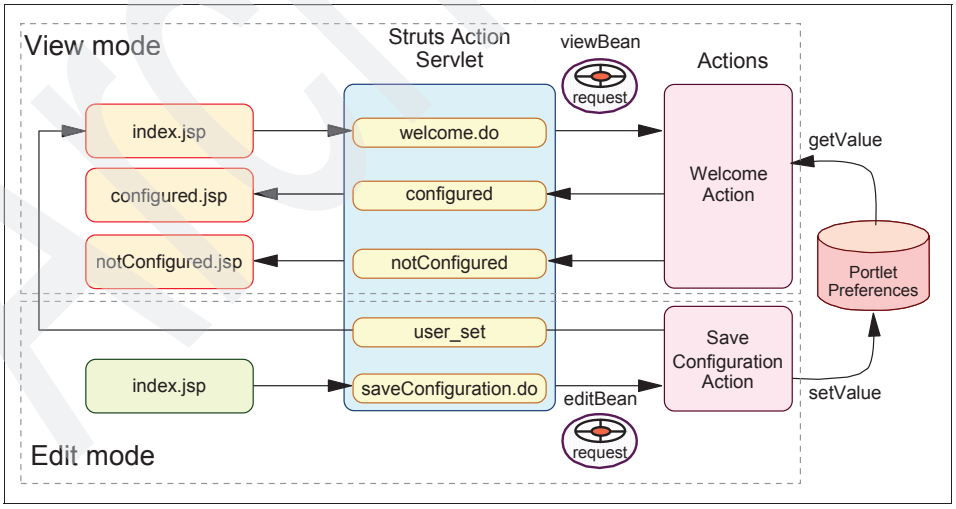


Figure 15-1 JSR 168 Struts portlet

15.2 Message flow

As an exercise, follow the message flow of this scenario which is explained here step by step. The operation flow is as follows (see Figure 15-1 on page 466).

1. At initialization time, Portal invokes the initial View mode page configured in the portlet descriptor (portlet.xml):

```
<portlet-preferences>
         <preference>
            <name>com.ibm.struts.portal.page.view.html</name>
            <value>index.jsp</value>
         </preference>
         ..................
</portlet-preferences>
```

2. The index.jsp page sends a forward with the name welcome:

```
<logic:forward name="welcome"/>
```

3. The configuration file (struts-config.xml) has a global forward indicating that the action welcome should be invoked:

```
<global-forwards>
      <forward name="welcome" path="/welcome.do">
      </forward>
</global-forwards>
```

4. The configuration file (struts-config.xml) states that, for action welcome, the execute method in WelcomeAction will be called and the Action Form viewBean will be passed in the request object:

```
<action-mappings>
      <action name="viewBean" path="/welcome" scope="request"
            type="myfirstjsr168struts.actions.WelcomeAction">
         ..................
      </action>
</action-mappings>
```

5. The configuration file (struts-config.xml) states that the viewBean is the ViewBean class:

```
<form-beans>
      <form-bean name="viewBean"
               type="myfirstjsr168struts.forms.ViewBean">
      </form-bean>
  </form-beans>
```

6. The execute method in the welcome action reads the portlet preferences and if a username field is not found, the forward notConfigured is returned:

```
forward = mapping.findForward("notConfigured");
...................
return forward;
```

Archived

7. The configuration file (struts-config.xml) shows that forward `notConfigured` indicates that notConfigured.jsp should be called.

```
<action-mappings>
      <action name="viewBean" path="/welcome" scope="request"
             type="myfirstjsr168struts.actions.WelcomeAction">
         ..................
         <forward name="notConfigured" path="/notConfigured.jsp">
         </forward>
      </action>
</action-mappings>
```

8. The notConfigured.jsp displays a message indicating to the user that he or she needs to switch to Edit mode and enter a user name and password.
9. When entering Edit mode, Portal invokes the initial Edit mode page configured in the portlet descriptor (portlet.xml):

```
<portlet-preferences>
          <preference>
          ..................
             <name>com.ibm.struts.portal.page.edit.html</name>
             <value>html/edit/index.jsp</value>
          </preference>
</portlet-preferences>
```

10. The Edit mode index.jsp page executes and displays a form for the user to enter a user name and a password.
11. The user enters a user name and a password. The form Submit button sends an action with name `saveConfiguration.do`:

```
<html:form action="/saveConfiguration.do" ..............
```

Note: The `do` option in the action parameter indicates that the forward is an action and not a JSP.

12. The Struts Action Servlet populates (setter methods) the associated form bean (editBean) with the user name and password. The Edit mode configuration file (struts-html-edit.xml) shows that the execute method in `SaveConfigurationAction` should be called and the form bean `EditBean` should be passed in the request object.

```
<form-beans>
      <form-bean name="editBean"
type="myfirstjsr168struts.html.edit.forms.EditBean">
      </form-bean>
   </form-beans>
<!-- Action Mappings -->
   <action-mappings>
      <action name="editBean" path="/saveConfiguration" scope="request"
type="myfirstjsr168struts.html.edit.actions.SaveConfigurationAction">
          ..................
```


Archived

```
            </forward>
        </action>
    </action-mappings>
```

13. The execute method in SaveConfigurationAction executes and gets the user name and password from the bean (getter methods). These values are stored as portlet preferences.

14. The execute method (SaveConfigurationAction) returns the forward `user_set`:

```
forward = mapping.findForward("user_set");
   return forward;
```

15. The Edit mode configuration file (struts-html-edit.xml) showsthat the View mode index.jsp should now be called.

```
   <action-mappings>
      <action name="editBean" path="/saveConfiguration" scope="request"
type="myfirstjsr168struts.html.edit.actions.SaveConfigurationAction">
         <forward contextRelative="true" name="user_set"
                              path="/index.jsp">
         </forward>
      </action>
   </action-mappings>
```

Note: `contextRelative="true"` indicates that the page is found in the context root and not in `/html/edit/`.

16. This closes the loop. View mode index.jsp sends the forward to execute its associated action. The action in View mode reads the portlet preferences, stores the values in the viewBean (setters) and this time returns the forward `configured`.

17. View mode configuration file (struts-config.xml) states that configured.jsp should now be invoked:

```
<action-mappings>
      <action name="viewBean" path="/welcome" scope="request"
           type="myfirstjsr168struts.actions.WelcomeAction">
         <forward name="configured" path="/configured.jsp">
         </forward>
         ..........
      </action>
   </action-mappings>
```

18. Page configured.jsp executes, reads the user name from the viewBean and displays it.

15.3 Creating a Portlet project

Follow these steps to create the Portlet project.

Archived

1. Start Rational Application Developer V6 if not already running.
2. If prompted, select the default workspace or any other workspace of your preference for this session.
3. In the menu, select **File** → **New** → **Project**.
4. In the New Project window, select the wizard for **Portlet Project (JSR 168)**.

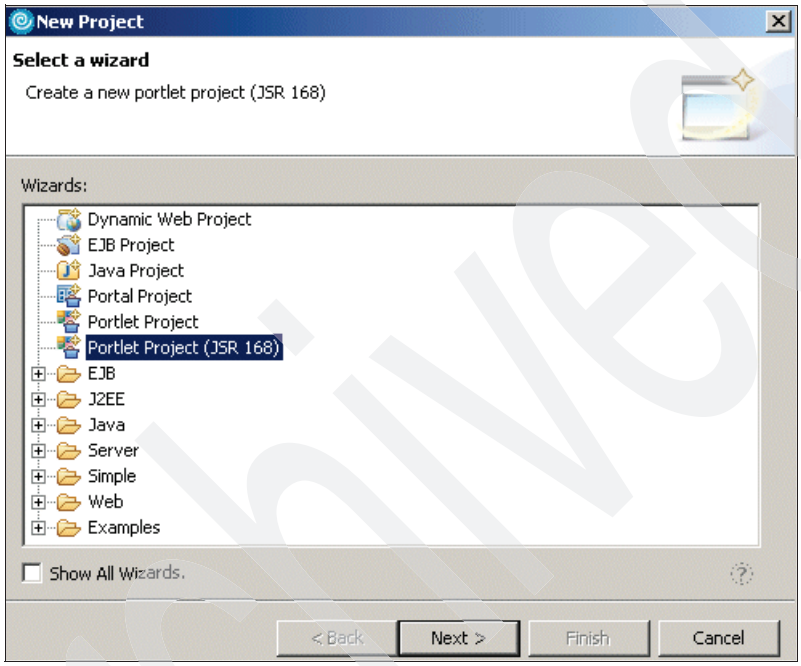


Figure 15-2 JSR 168 Portlet Project

5. Click **Next**.
6. If the Confirm Enablement window appears, asking you if you want to enable the portal development role, click **OK**.

Archived

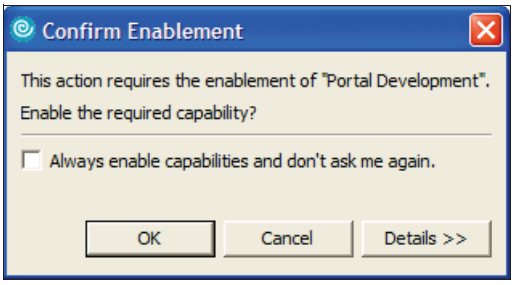


Figure 15-3 Enable the Portal development role

7. In the Portlet Project (JSR 168) page, enter the project name as `MyFirstJSR168Struts`.
8. Check the **Create a portlet** check box, if it is not already checked.

   Note: Typically, you will not need to create a portlet when importing a portlet WAR file into your project.
9. Click **Show Advanced** and select **WebSphere Portal V5.1 stub** for the target server. Click **Next**.
10. In the WebSphere Portal version box, select **5.1**.

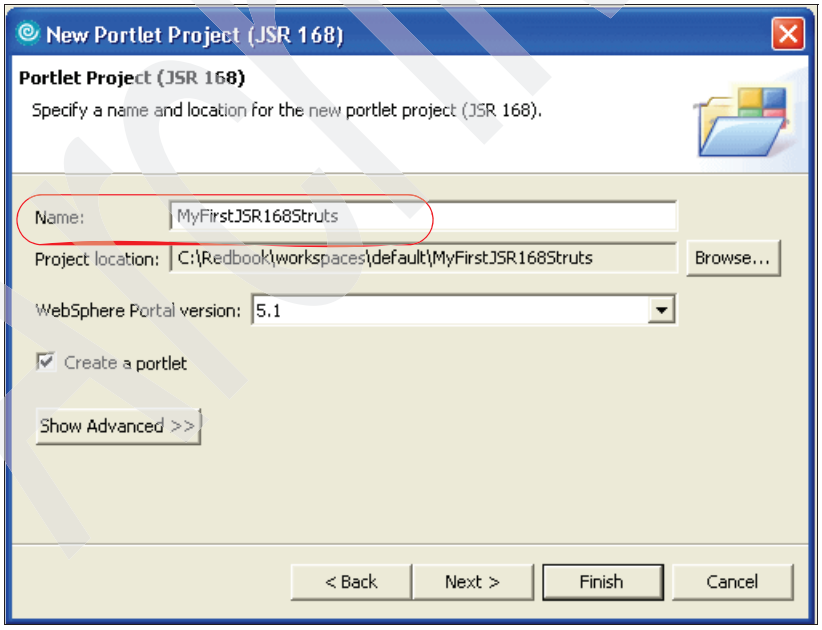


Figure 15-4 Select WebSphere Portal Version 5.1

Archived

11. Click **Next**.

12. In the Portlet type page, select **Struts portlet (JSR 168)**.

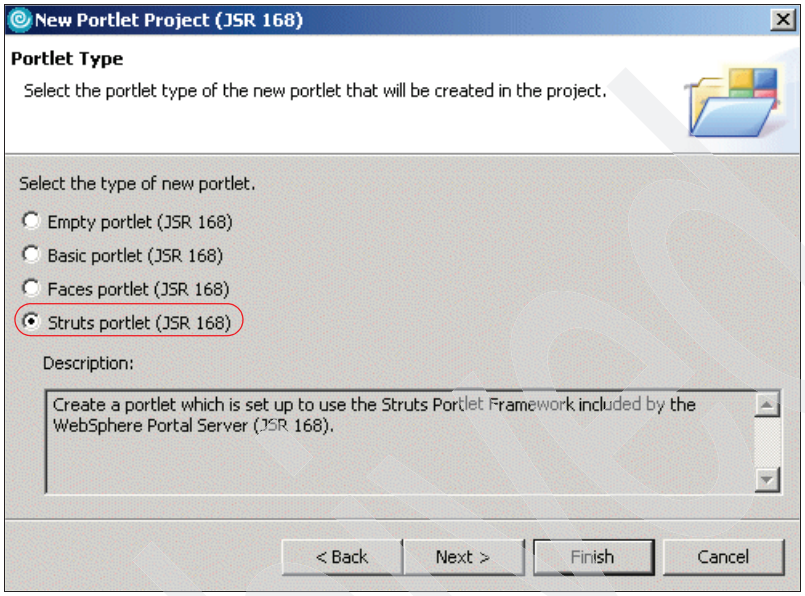


Figure 15-5 Struts portlet (JSR 168) selection

13. Click **Next**.

14. In the Features page, leave the Web Diagram option selected.

Archived

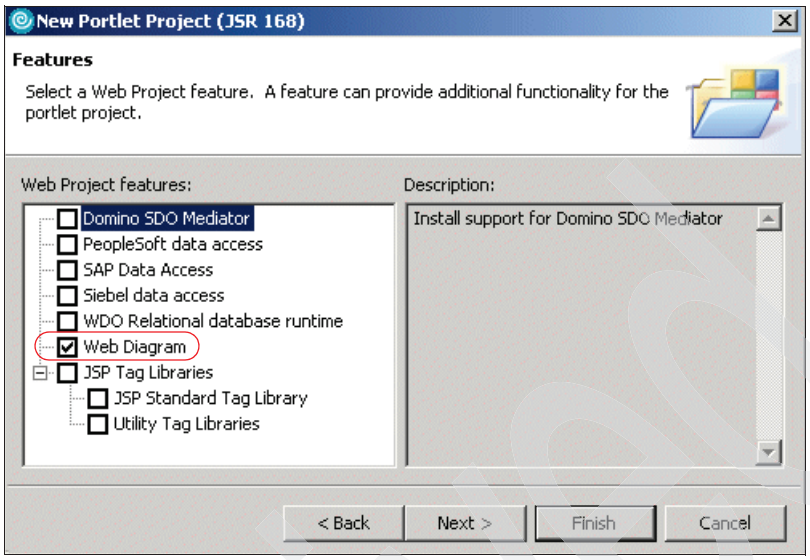


Figure 15-6 Project features

15. Click **Next**.
16. In the Portlet settings page, review the options provided.

    Note: The settings provided in this page are the same when you select a basic portlet.
17. Leave all settings unchanged. Click **Next**.
18. In the Struts portlet settings page, you can specify the name of the resource bundle that will be used for this Struts portlet application, as well as the Java package where it will be stored. Take default values and do not make any changes to these fields.

Archived

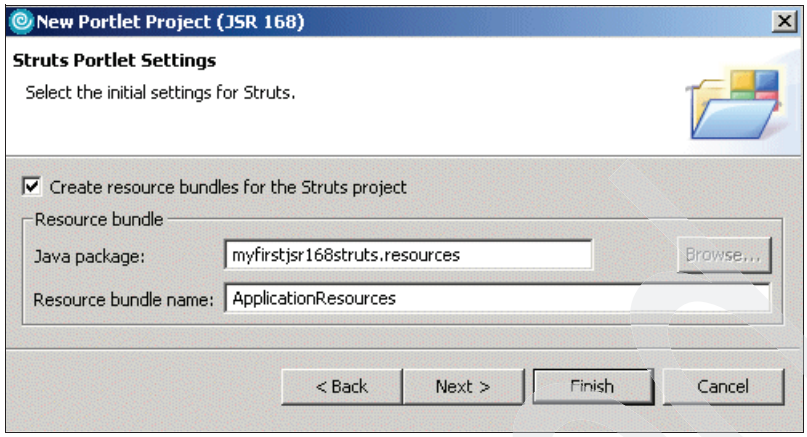


Figure 15-7 Struts Portlet Settings

19. Click **Next**.

20. In the Miscellaneous page, you can select additional markups and modes that will be supported in this portlet application. Check **Add Edit mode** as illustrated in Figure 15-8.

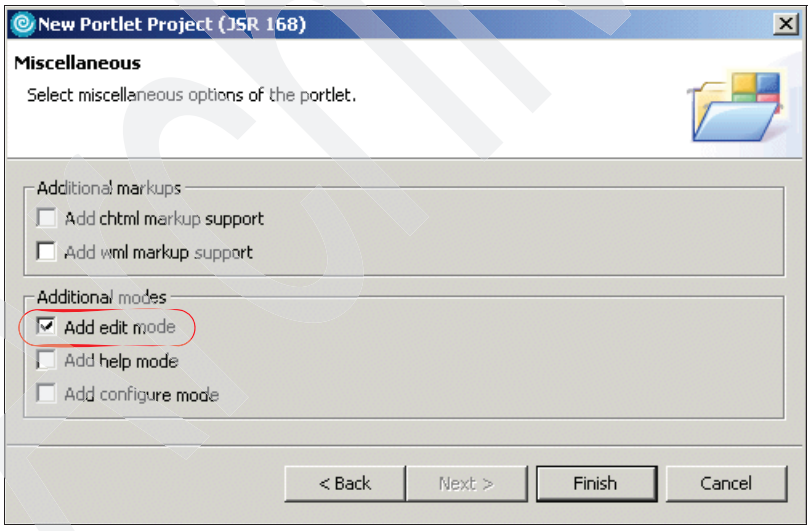


Figure 15-8 Adding Edit mode support

21. Click **Finish** to generate the project.

22. If the Confirm Perspective Switch window appears, click **Yes**.

15.3.1 Inspecting the Struts portlet project

Now that the Struts portlet has been created, let's take a look at what Rational Application Developer has generated for you.

- All of the necessary *.tld files are included in the WEB-INF directory.
- All of the necessary *.jar files are included in the WEB-INF/lib directory.
- The struts-config file was created, and the <controller> element that defines the RequestProcessor subclass was automatically generated in the Struts configuration file. The struts-html-edit configuration file was also created.

```
<controller
processorClass="com.ibm.portal.struts.portlet.WpRequestProcessor">
</controller>
```

- The web.xml deployment descriptor was created.
  - Struts portlet tag library definitions were included.
  - Welcome-file list was supplied.

```
<welcome-file-list>
   <welcome-file>index.html</welcome-file>
   <welcome-file>index.htm</welcome-file>
   <welcome-file>index.jsp</welcome-file>
   <welcome-file>default.html</welcome-file>
   <welcome-file>default.htm</welcome-file>
   <welcome-file>default.jsp</welcome-file>
</welcome-file-list>
```

  Note: This list is not really necessary; as we will see, the Struts Portlet framework provides a mechanism to identify which page must be displayed first for each different mode and markup language.

- The portlet.xml deployment descriptor was created.
  - A portlet was created and the `StrutsPortlet` class was specified as the servlet class. The `config` init-param was created to specify the struts-config file to be used and the `struts-servlet-mapping` init-param was created to specify the struts mapping. Other init-params were also included but omitted in the sample below to simplify understanding.
  - A preference pointing to the initial page to be used in View mode as well as a preference pointing to the initial page for Edit mode were also created by the wizard as shown in Figure 15-1 on page 476.

    Note: For more information about other added elements, see the WebSphere Portal Infocenter.

Archived

Example 15-1 Generated portlet preferences in portlet descriptor (portlet.xml)

```
<portlet-preferences>
   <preference>
      <name>com.ibm.struts.portal.page.view.html</name>
      <value>index.jsp</value>
   </preference>
   <preference>
      <name>com.ibm.struts.portal.page.edit.html</name>
      <value>html/edit/index.jsp</value>
   </preference>
</portlet-preferences>
```

- The definition of a `config` parameter and a `com.ibm.struts.portal.page.<mode>.<markup>` preference above is the standard way to define a Struts configuration file and modules and the first page to be displayed for each specific mode and markup language.
- In the Struts portlet framework, each markup and mode is mapped to a specific Struts configuration file and module. If you want to add support for Edit mode in the wml markup language, for example, you should create a `config/wml/edit` parameter pointing to the Struts configuration file and a `com.ibm.struts.portal.page.edit.wml` preference pointing to the initial page for this mode/markup.
- Figure 15-9 illustrates the generated modules for View mode (default module) and Edit mode (/html/edit).

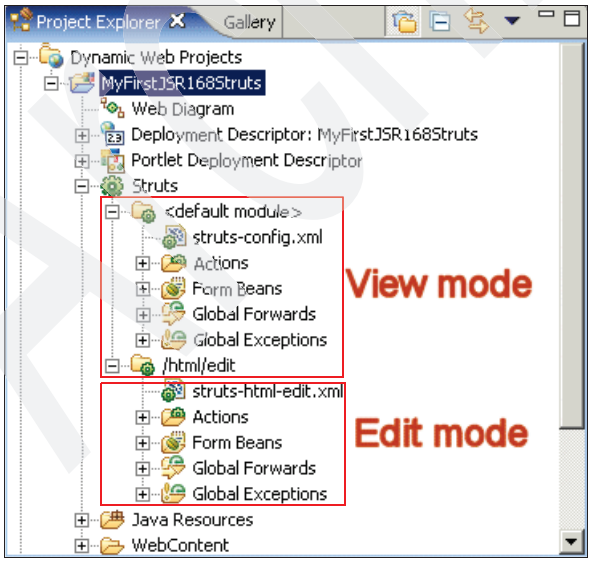


Figure 15-9 Generated Struts portlet

15.4 Designing the application (View mode)

The easiest way to design a Struts portlet application is by using the Web diagram editor. You will then realize the Web page nodes that you created in the diagram by launching the proper Rational Application Developer wizards.

You can also use the **File** → **New** command on the top menu, or the options in the context menus, but the Web diagram gives you a good overview of how the different pages and actions interact in the Struts portlet application.

Follow these steps to design the MyFirstStruts portlet application:

1. Make sure you have the Web perspective as active.
2. In the Project Explorer view, double-click the **Web Diagram**.
3. This Web diagram opens. Read about how to add and connect nodes.

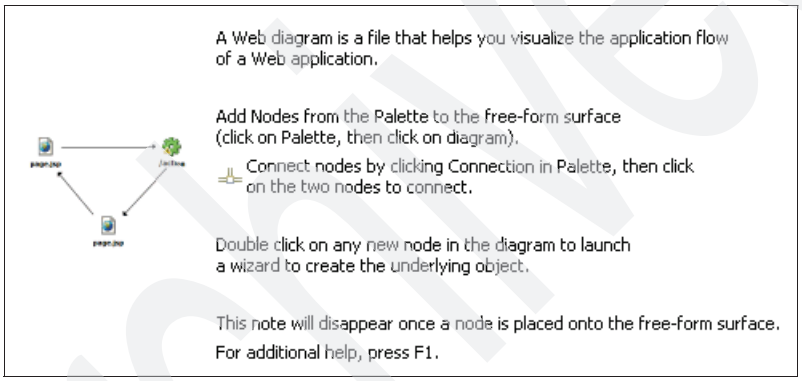


Figure 15-10 Initial Web diagram (empty)

4. You will add nodes to the Web diagram using the Palette view. Notice that you have some Struts parts that you can pick up from the palette and then insert into the Web diagram.

Archived

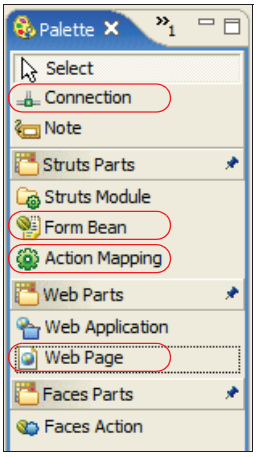


Figure 15-11 The palette view

5. Click **Web Page** on the palette view, then click anywhere in the Web diagram to place it. Name the Web page `index.jsp`. See Figure 15-14 on page 480 for a suggested location of index.jsp.

   Note: Unless you change the default name in the portlet descriptor, you should use this name because this will be the first page in your application, and the portlet.xml deployment descriptor is already configured to find this page automatically:

```
<portlet-preferences>
   <preference>
      <name>com.ibm.struts.portal.page.view.html</name>
      <value>index.jsp</value>
   </preference>
   ...............
</portlet-preferences>
```

6. Repeat these steps and place two other pages in the Web diagram. Name them `configured.jsp` and `notConfigured.jsp`.
7. Click **Action Mapping** on the palette view. Click again anywhere in the Web page diagram and name the action mapping `welcome`.
8. Click **Form Bean** on the palette view and place it in the Web diagram. Name it `viewBean` and scope should be `request`.

Archived

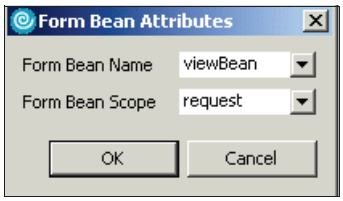


Figure 15-12 Action Form bean for View mode welcome action

9. Create four connections as follows:

   a. Click **Connection** on the palette view. Then connect `index.jsp` to `welcome` action.

   b. Click **Connection** on the palette view. Then connect `welcome` action to `configured.jsp`.

      i. In the Choose a connection window that appears, expand **Local Forward**, then select **<new>** and click **OK**.

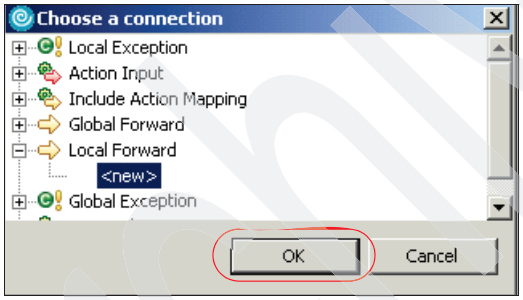


Figure 15-13 Choose a connection

      ii. Name the connection `configured` as shown in Figure 15-14 on page 480.

   c. Click **Connection** again. Then connect `welcome` to the `notConfigured.jsp`.

      i. In the Choose a connection window that appears, expand **Local Forward**, then select **<new>** and click **OK**.

      ii. Name the connection `notConfigured`.

   d. Click **Connection**. Then connect welcome action to viewBean.

   At this time, your portlet application Web diagram should look as illustrated in Figure 15-14 on page 480.

Archived

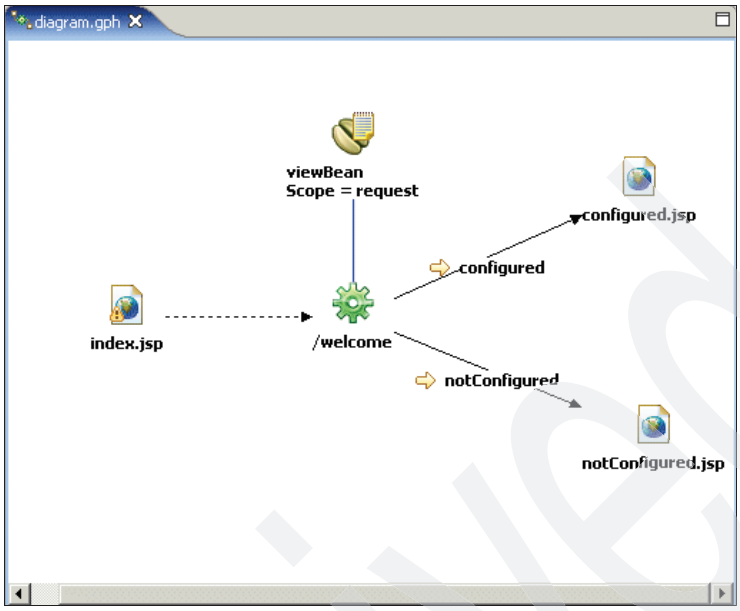


Figure 15-14 Web diagram (View mode)

10. Save the Web diagram.

15.4.1 Realizing the application components

Now that all components are in the Web diagram, it is time to realize them into real components. Before you do this you will need to create a global forward in the struts-config.xml, this will be used by the initial page (`index.jsp`).

Follow these steps:

1. In Project explorer view, expand **Dynamic Web Projects** → **MyFirstJSR168Struts** → **Struts** → **<default module>**.
2. Double-click **struts-config.xml**, shown in Figure 15-15 on page 481.

Archived

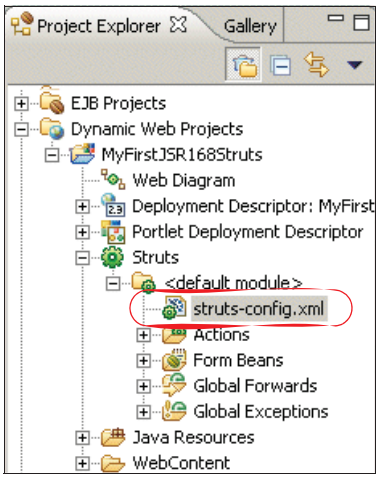


Figure 15-15 Locating the struts-config.xml file

3. In the Struts configuration file editor, select the **Global Forwards** tab, at the bottom. Click **Add** and name the global forward `welcome`.
4. Under the forward attributes section, enter `/welcome.do` as the Path. Figure 15-16 on page 482 shows the Global Forwards page.

Archived

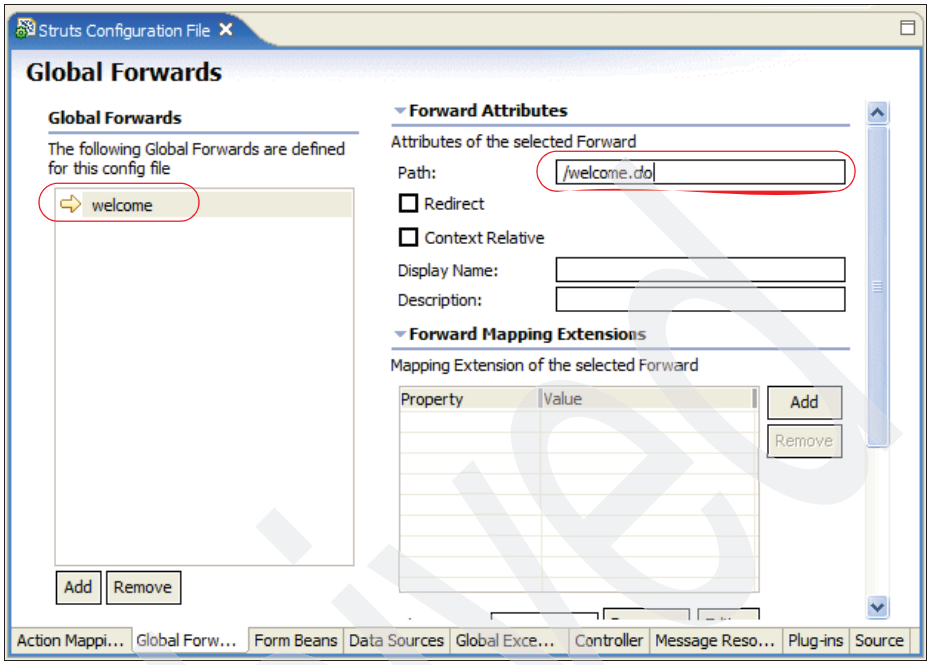


Figure 15-16 The global forward page

5. The resulting global forward will be like this:

```
<global-forwards>
      <forward name="welcome" path="/welcome.do">
      </forward>
</global-forwards>
```

Note: The global forward will indicate a broken link since the action has not been defined yet.

6. Save and close the file.

15.4.2 Realizing the index.jsp page

The first component you will realize is the initial index.jsp page as follows:

1. In the Web diagram, double-click **index.jsp**.
2. The `New JSP File` wizard appears. All fields should have been properly added by the wizard. See Figure 15-17 on page 483 to verify that this is the case.

Archive

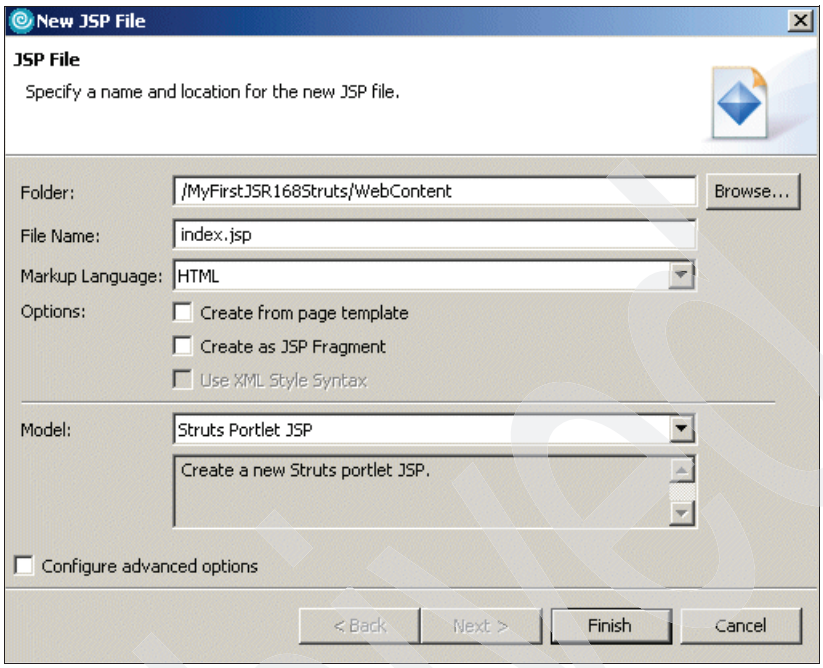


Figure 15-17 New JSP File window

3. Check **Configure advanced options** and click **Next >**.
4. In the Tag libraries page, remove all tag libraries by selecting them and clicking **Remove**.
5. Click **Add**.
6. In the Select a tag library page, check the **/WEB-INF/struts-logic.tld** tag library. Click **OK**.

Archived

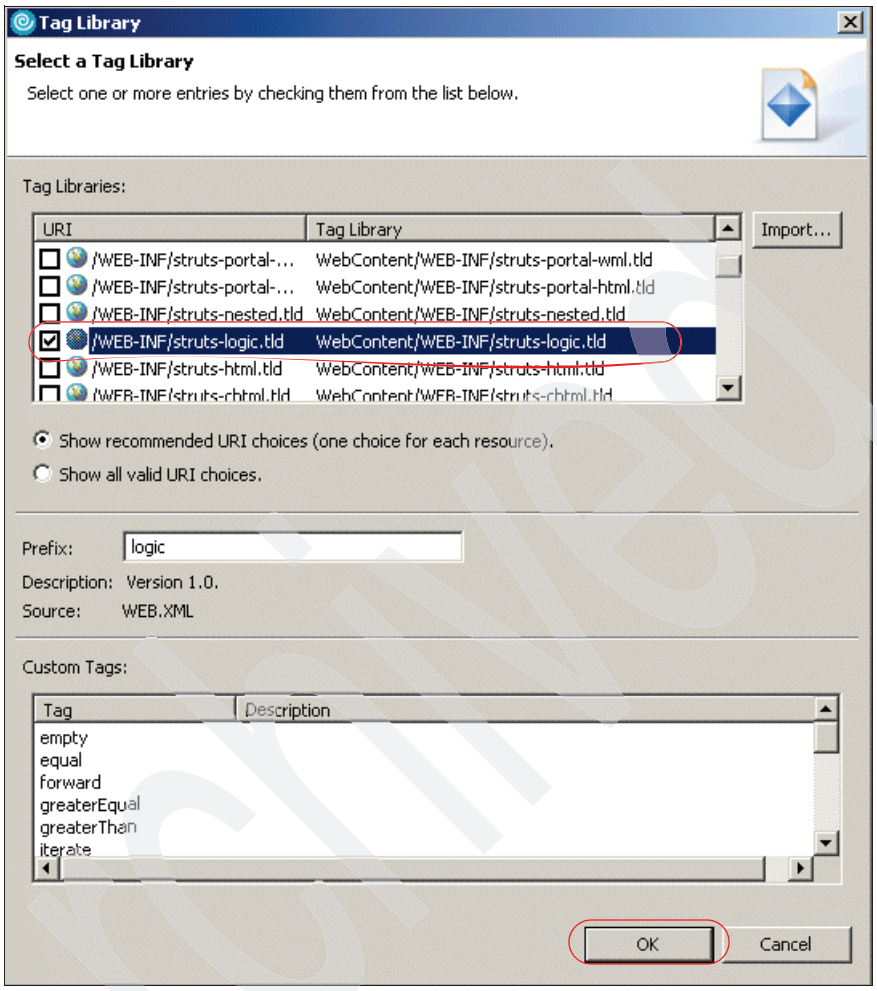


Figure 15-18 Selecting a tag library

7. Click **Next.**

Archive

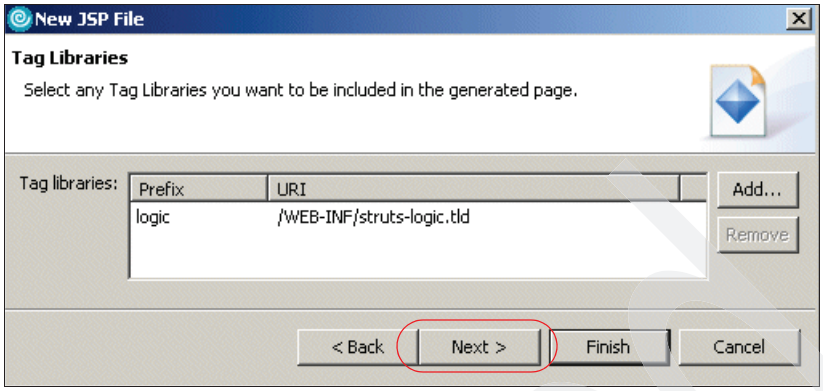


Figure 15-19 Selected tag library

8. JSP file options. Take default values for encoding, content type and other values. Click **Next.**
9. JSP file method stubs. Take default values for method stub creation. Click **Next**.
10. In the Form field selection page, uncheck **Generate fields in a form.** Click **Finish** to generate the JSP.

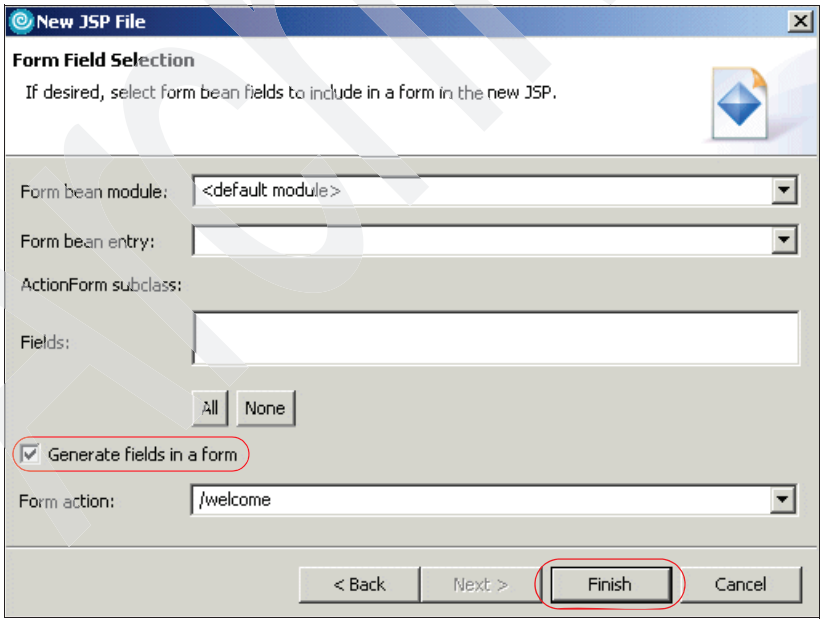


Figure 15-20 Form Field Selection

11. In the JSP editor, select the source tab and replace the page with the following code shown in Example 15-2.

Example 15-2 index.jsp page

```
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic"%>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
<logic:forward name="welcome"/>
```

Note: This tag indicates that it will forward to the global forward `welcome and the action` mappings should resolve where to go. In this scenario welcome.do should point to the action code. This will be done later in 15.4.7, "Realizing the welcome action mapping" on page 490.

12. Save the file and close the JSP editor.

15.4.3 Realizing the notConfigured.jsp

The next component you will realize is the notConfigured.jsp file. Follow these steps:

1. In the Web diagram, double-click **notConfigured.jsp**.
2. The New JSP file wizard appears. Uncheck the **Configure Advanced Options** check box.
3. Click **Finish** to generate the JSP**.**
4. In the JSP editor, replace all the code that was generated by the wizard with the code illustrated in Example 15-3.

Example 15-3 notConfigured.jsp page

```
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<h3><bean:message key="myfirstjsr168struts.title"/></h3>
<em><bean:message key="message.mustConfigureFirst"/></em>
```

Note: The messages will be created later in the resources file.

5. Save the file and close the JSP editor.

15.4.4 Realizing the Form Bean

The next component you will realize is the ViewBean.java file. Follow these steps:

1. In the Web diagram, double-click **viewBean.**
2. Examine and take default values in the `New Form-Bean` window.

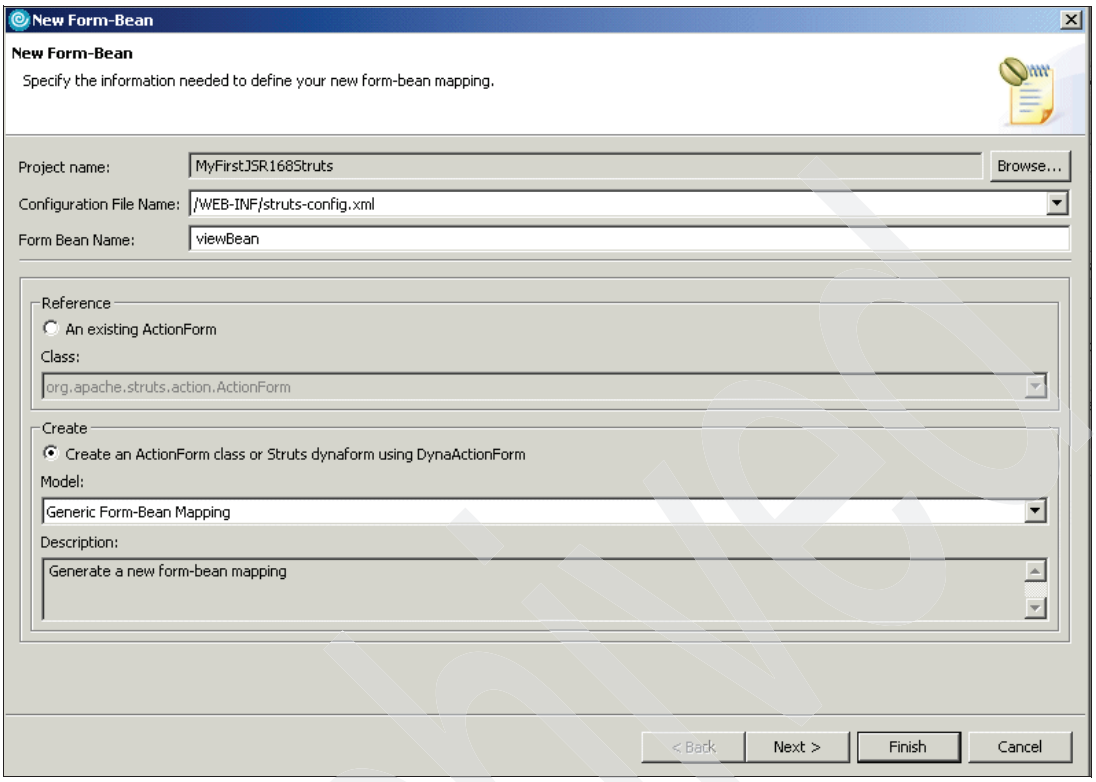


Figure 15-21 New Form Bean window

3. Click **Next**.
4. In the `Choose new fields for your ActionForm class`, do not select any items. New fields will be added manually to other items (for example configured.jsp). Click **Next**.
5. In the `Create new fields for your ActionForm class` window:
   a. Click **Add** to add a field
   b. Enter userid for the name and select type String.

   Note: A setter and a getter methods will be generated for this property (userid).

Archive

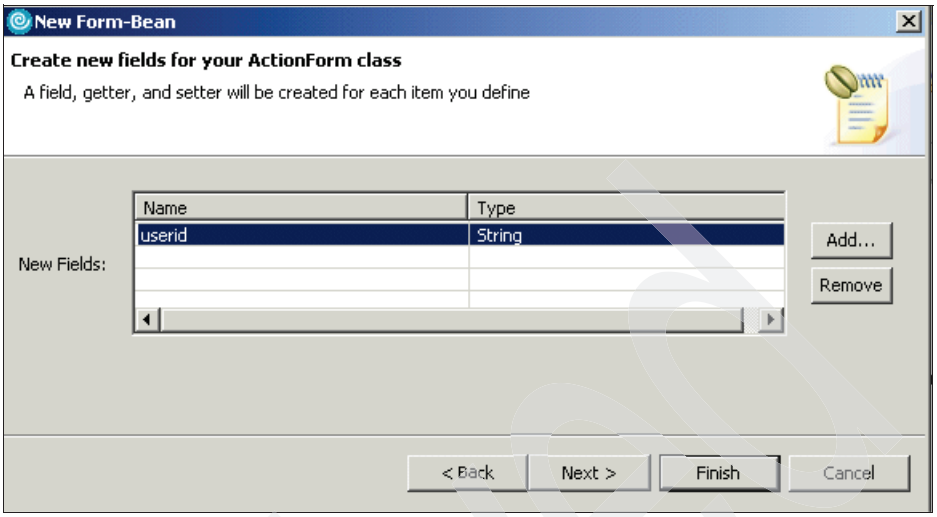


Figure 15-22 Defining viewBean (ActionForm class)

6. In the Create a mapping window, uncheck all method stubs and click **Finish** to generate the bean.

Archive

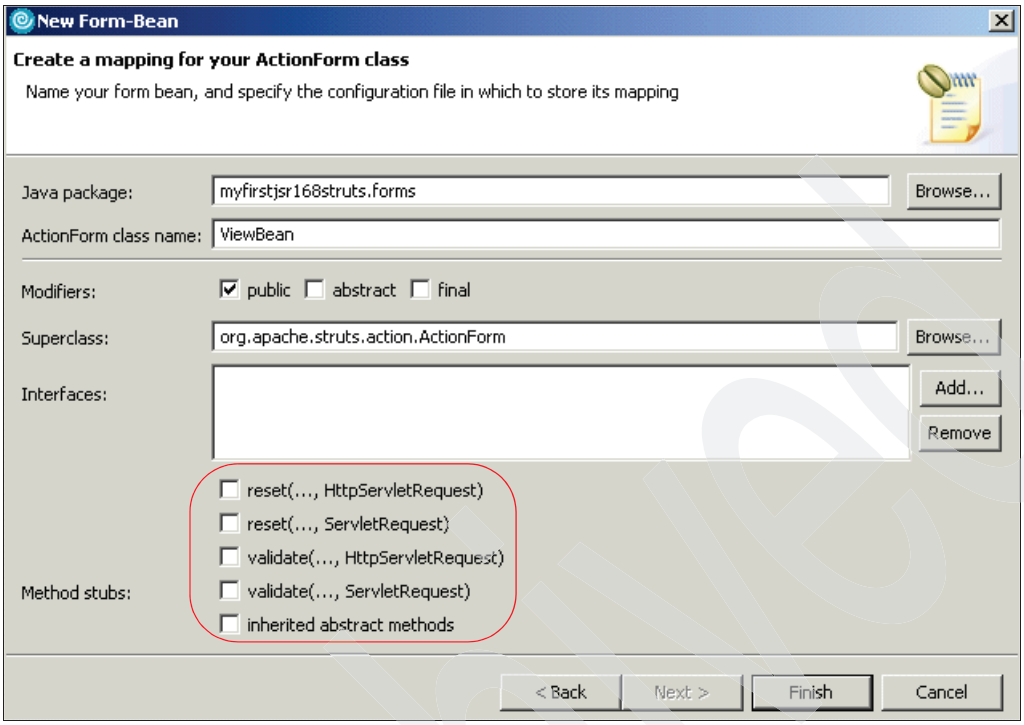


Figure 15-23 Create a mapping

7. Inspect, save and close the bean (ViewBean class).

15.4.5 Realizing configured.jsp

The next component you will realize is the configured.jsp file. Follow these steps:

1. In the Web diagram, double-click **configured.jsp**.
2. The New JSP file wizard appears. Click **Finish** to generate the JSP.
3. Erase all the code that was automatically generated by the wizard and paste the code shown in Example 15-4.

Example 15-4 configured.jsp page

```
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<h3><bean:message key="myfirstjsr168struts.title"/></h3>
<em><bean:message key="message.configured"/></em>
<BR>
<bean:write name="viewBean" property="userid"/>
```

15.4.6 Editing the resources file

In this section you will provide the messages, required by the portlet application, in the resources file as follows:

1. In Project Explorer view, expand **Java Resources** → **JavaSource** → **myfirstjsr168struts.resources**.
2. Double-click **ApplicationResources.properties**.
3. Insert the lines below:

Example 15-5 ApplicationResources.properties

```
myfirstjsr168struts.title=Struts portlet with JSR 168 Portlet API

message.mustConfigureFirst=You must configure this portlet through edit mode.
message.configured=This portlet has been configured for user:
```

4. Save the file and close the editor.

15.4.7 Realizing the welcome action mapping

The next component you will realize is the welcome action mapping. Follow these steps:

1. In the Web diagram, double-click **welcome**. The New Action Mapping wizard appears, as shown in Figure 15-24 on page 491.

Archived

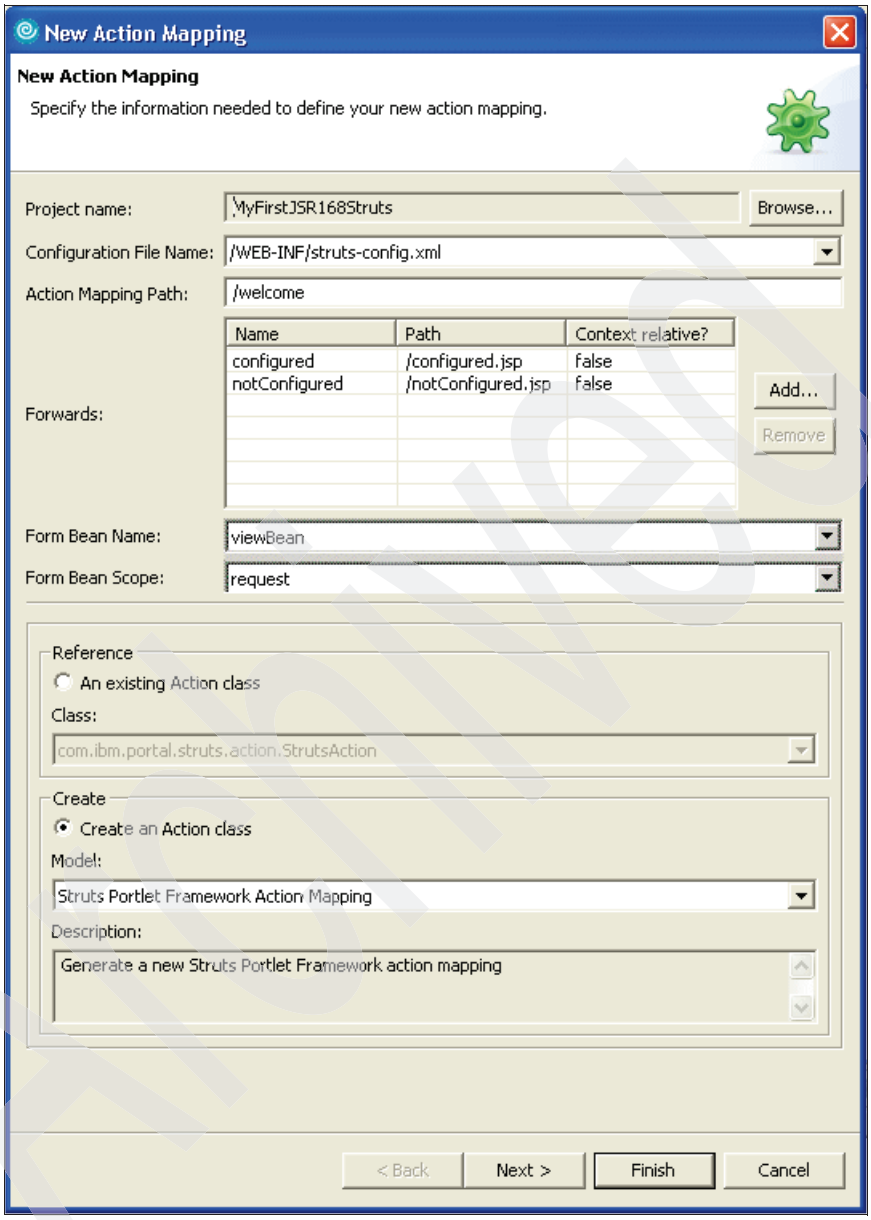


Figure 15-24 The New Action Mapping wizard

2. Notice that the wizard has filled in most of the data in this window. This data came from the Web diagram you previously designed. For example:

   a. Action mapping name was entered as the **Action Mapping Path**.

b. The connections you created as Local forwards were automatically inserted into the **Forwards** list.

c. The **Create an Action class** radio button was selected, and the **Model** chosen was the **Struts Portlet Framework Action Mapping**. This is the mapping provided by Struts portlet framework.

3. No changes are required at this time so leave all data unchanged and click **Next**.

4. The **Create an Action class for your mapping** window shows that the mapping will inherit from com.ibm.wps.struts.action.StrutsAction. Click **Finish** to generate the Action class for your mapping.

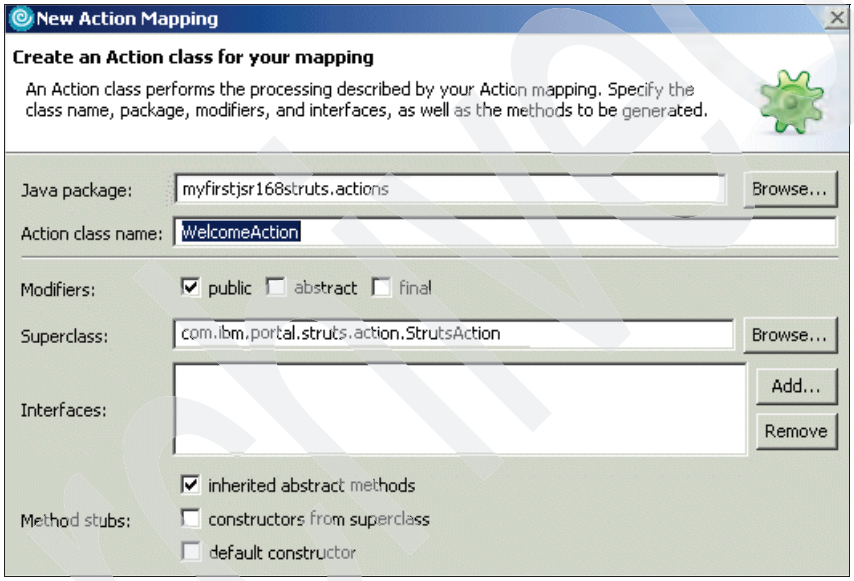


Figure 15-25 Create an Action class for your mapping

5. Replace the generated execute method with the code shown in Example 15-6.

Example 15-6 Execute() method in the Welcome Action class

```
    public ActionForward execute(ActionMapping mapping, ActionForm form,
            PortletRequest request, PortletResponse response) throws Exception
{

    System.out.println("===>View: Entering welcome action...");
    ActionForward forward = mapping.findForward("configured");
    ViewBean viewBean = (ViewBean) form;
    try {
```

```
        PortletPreferences portletPreferences = request.getPreferences();
        String username = portletPreferences.getValue("username", null);
        System.out.println("===>View: username = *" + username + "*");
        viewBean.setUserid(username);

        if (username == null || username.equals("")) {
          System.out.println("===>View: portlet not configured...");
            forward = mapping.findForward("notConfigured");
        }

      } catch (Exception ex) {
      }
      return forward;

    }
```

6. If you get errors, you might need to organize your imports.
7. Save the file and close the editor.

15.4.8 Running the Struts portlet

Execute the following steps to run and test the Struts portlet (View mode only):

1. Right-click `MyFirstJSR168Struts` project and select **Run** → **Run on Server**.
2. In the Define a new server window, select **Manually define a server**.
3. Select the server type **WebSphere Portal V5.1 Test Environment**.
4. Click **Finish**.
5. Wait a few minutes so the Portal Test Environment is started and you will see a page similar to Figure 15-26.



Figure 15-26 MyFirstJSR168Struts in the browser

Archived

Note: The portlet has not been updated to fully support Edit mode yet. Adding the portlet Edit mode code will be done next in 15.5, “Designing the application (Edit mode)” on page 494 in this chapter.

15.5 Designing the application (Edit mode)

Execute the following steps to create a new Web diagram for Edit mode:

1. Right-click **Web Diagram**.
2. Select **New** → **Web Diagram**.
3. Enter `editMode` as the Web diagram file name.
4. Click **Finish** to create the empty Web diagram.
5. Web pages. Refer to Figure 15-28 on page 495 and place two pages in the diagram with the following names:
   a. `html/edit/index.jsp`
   b. `index.jsp` (Note: this is View mode index.jsp)
6. Action mappings. Place an action mapping in the diagram with name save`Configuration.`
7. Form Bean: Place a form bean in the diagram. In the Form Bean Attributes window that appears enter the following:
   – Form Bean Name: edit`Bean`
   – Form Bean Scope: `request`
8. Connections. Create the following connections:
   a. A connection from index.jsp to saveConfiguration action mapping.
   b. A connection from saveConfiguration to View mode index.jsp. Use `user_set` as the local forward name.
   c. A connection from saveConfiguration to editBean.
9. Right-click anywhere inside the Web diagram and select **Change the Struts module association**. From the list select **/html/edit** (Edit mode). Click **OK**.

Archived

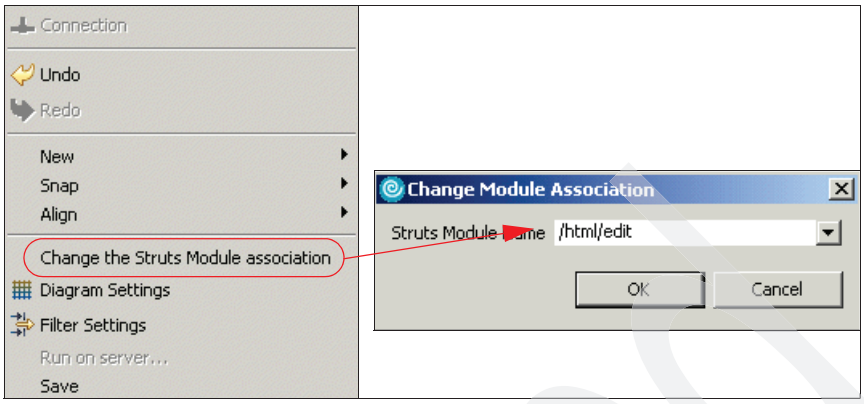


Figure 15-27 Change the Struts module association

10. Verify that your Web diagram looks as illustrated in Figure 15-28.

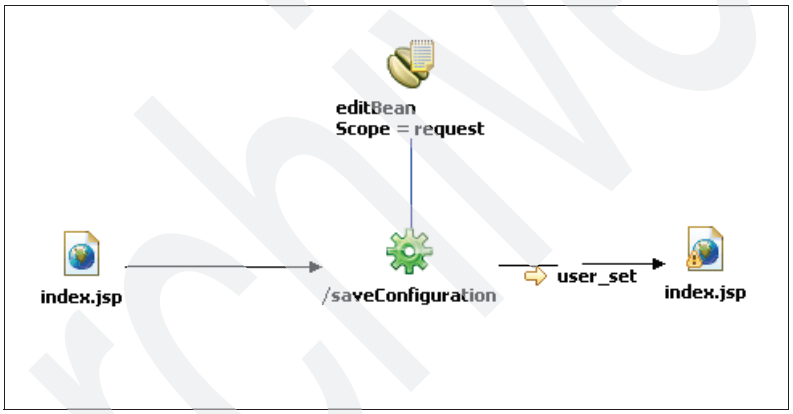


Figure 15-28 Web diagram for Edit mode

11. Save the Web diagram.

15.6 Realizing the Edit mode application components

In this section you will realize the new application components that will support Edit mode in your portlet.

15.6.1 Realizing the editBean

In this section you will realize the `userBean`. Execute the following steps:

1. In the Web diagram, double-click **EditBean** form bean. This will bring up the New Form-Bean wizard.

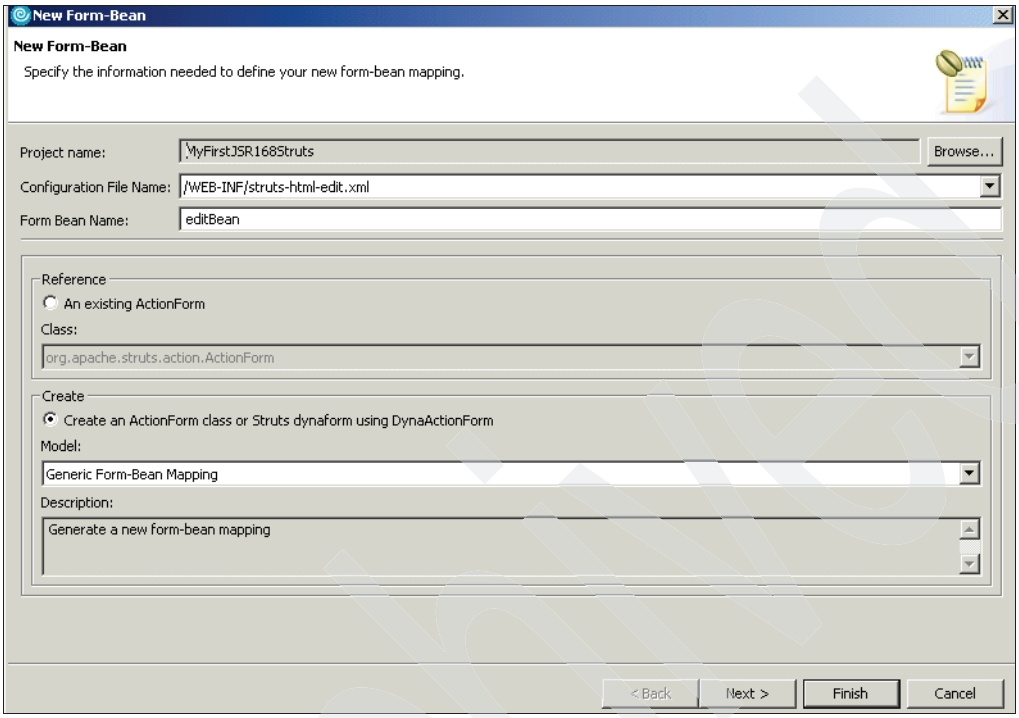


Figure 15-29 New Form-Bean wizard

2. Leave the form bean name unchanged. Make sure that the **Create an ActionForm class** radio button is selected, and that **Generic Form-bean Mapping** is selected.
3. Click **Next.**
4. In the Choose new fields for your ActionForm class window, do not select any fields and click **Next**.
5. In the Create new fields for your ActionForm class window, you will add two fields:
   a. Click **Add.** Enter the field's name `username`. Leave the field type as `String`.
   b. Click **Add** again. Enter the field's name `password` and leave the field type as `String`.
6. The window will look like in Figure 15-30 on page 497. Click **Next.**

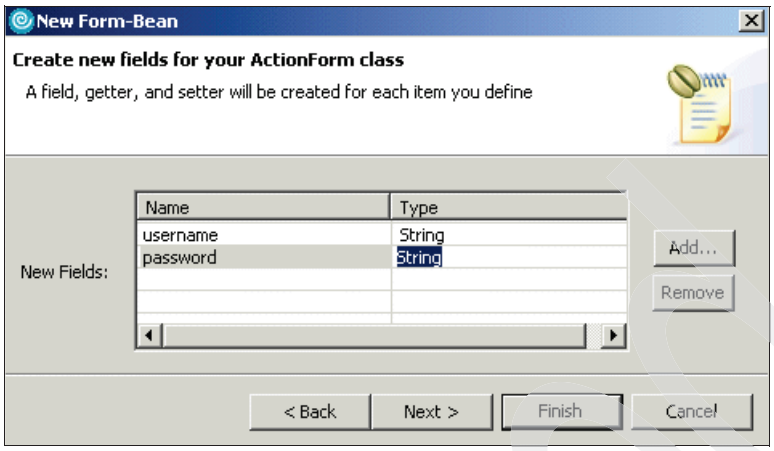


Figure 15-30 Creating new fields for the ActionForm Bean

7. In the Creating a mapping for your ActionForm class window, uncheck all method stubs as shown Figure 15-31. These methods are not needed in this scenario.

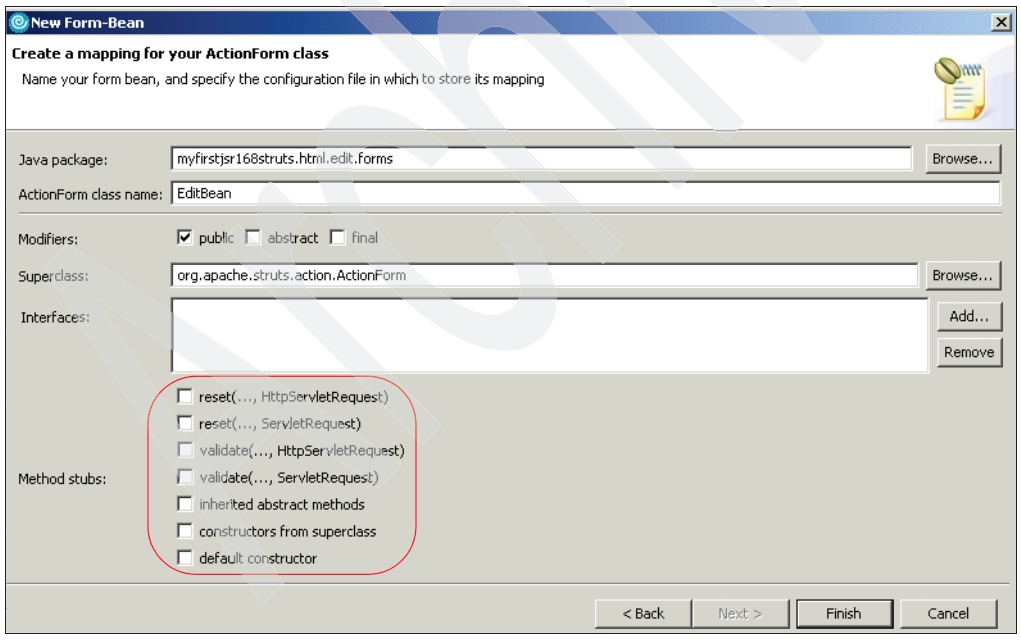


Figure 15-31 Creating a mapping

8. Click **Finish** to create the bean. Rational Application Developer will generate the ActionForm bean, that extends from rg.apache.struts.action.ActionForm, along with its proper methods. In addition, it will also include the required form-bean tag in the struts-config.xml file for Edit mode.
9. Save the file and close it.

15.6.2 Realizing index.jsp

In this section, you will realize the index.jsp. Follow these steps:

1. In the Web diagram, double-click **index.jsp**.
2. Click **Finish** to generate the page.
3. Replace the generated jsp content with the following code:

Example 15-7 index.jsp for Edit mode

```
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>

<h3><bean:message key="editmode.heading"/></h3>

<p><bean:message key="editmode.instructions"/></p>
<ul>
   <li><bean:message key="editmode.instructions.1"/></li>
</ul>

<html:form action="/saveConfiguration.do" portletMode="view"
validate="false">

   <table>
      <tr>
         <th align="right">
            <bean:message key="prompt.username"/>: 
         </th>
         <td align="left">
            <html:text property="username" size="25"/>
         </td>
      </tr>
      <tr>
         <th align="right">
            <bean:message key="prompt.password"/>: 
         </th>
         <td align="left">
            <html:password property="password" size="25"/>
         </td>
      </tr>
```

Archived

```
            <tr><td><br></td></tr>
            <tr>
               <td align="right">
                  <html:submit value="Submit"/>
               </td>
               <td align="left">
                  <html:reset/>
               </td>
            </tr>
         </table>

      </html:form>
```

4. Review the code:
   a. The saveConfiguration action has not been defined yet, so you get a broken link message.
   b. The `portletMode="view"` attribute in the `form` tag instructs that the portlet will return to View mode after submitting the form.
5. Save the file and close it.

15.6.3 Realizing the saveConfiguration action

In this section you will realize the saveConfiguration action. Execute the following steps:

1. In the Web diagram, double-click **saveConfiguration** action.
2. All values should be correct, as the Web diagram had all information the wizard needs to create the action.

   Note: In this case, the Context relative field is set as `true` for the `user_set` forward. It indicates that the index.jsp page associated to this forward is in the context root. That is, it refers to the View mode index.jsp page.
3. Click **Finish**.
4. This action is the one with the most logic. Replace its contents with the following and save the file:

Example 15-8 The listing for saveConfiguration action

```
package myfirstjsr168struts.html.edit.actions;

import javax.portlet.PortletPreferences;
import javax.portlet.PortletRequest;
import javax.portlet.PortletResponse;

import myfirstjsr168struts.html.edit.forms.EditBean;
```


Archived

```
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

import com.ibm.portal.struts.action.StrutsAction;

public class SaveConfigurationAction extends StrutsAction {


public ActionForward execute(ActionMapping mapping, ActionForm form,
       PortletRequest request, PortletResponse response) throws
       Exception {

   System.out.println("===>Edit: Entering saveConfiguration action...");
   // get the ActionForward for the mode that is being set.
   ActionForward forward = mapping.findForward("user_not_set");
   EditBean userBean = (EditBean) form;
   String username = userBean.getUsername();
   String password = userBean.getPassword();
   System.out.println("===>Edit: username = *" + username + "*");

   PortletPreferences portletPreferences = request.getPreferences();
   portletPreferences.setValue("username", username);
   portletPreferences.setValue("password", password);
   portletPreferences.store();

   forward = mapping.findForward("user_set");
   return forward;
}
}
```

5. Review the code. See 15.2, “Message flow” on page 467 for details.
6. Save the file.
7. Close the file.

15.7 Adding new keys to the resources file

Add entries to the Edit mode associated resource file:

1. Edit myfirstjsr168struts.resources.ApplicationResources.properties.
2. Add the key value pairs listed in Example 15-9.

Example 15-9 Edit mode resource file

```
editmode.heading=Edit Mode
```

```
editmode.title=Struts Edit Mode Example

editmode.instructions=Try the options below to demonstrate various
Struts features. When a valid username is given, an action forwards to a
JSP in view mode.
editmode.instructions.1=Enter a valid username to complete the
configuration

prompt.username=Username
prompt.password=Password

button.back=back
button.send=Send
button.save=Save
button.cancel=Cancel
button.reset=Reset

error.transaction.token=Transaction token is not valid
error.invalid.username=The username is not valid
```

c. Save the file and close editor.

15.8 Running the portlet

Run and test the portlet. Execute the following steps:

1. Stop the Portal Test Environment.
2. Right-click the `MyFirstJSR168Struts` project.
3. Select **Run** → **Run on Server**.
4. After the welcome page displays, click the **edit** icon, go to Edit mode.
5. Enter a user name and password; the application stores these values as `PortletPreferences` and forwards to View mode index.jsp.

Archived

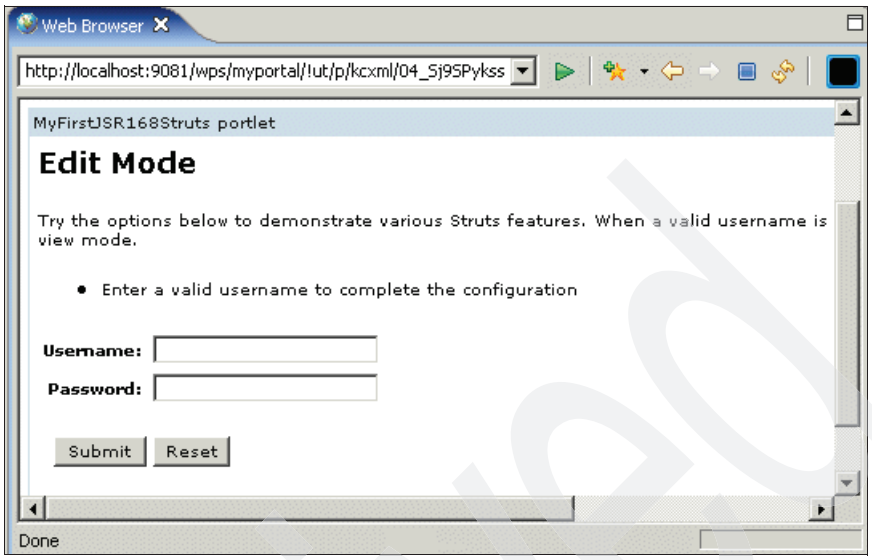


Figure 15-32 The portlet in Edit mode

6. Enter your name and a password.
7. Click **Submit** to invoke the action (Edit Mode).
8. Click **Reset** to clear the fields.

15.9 Adding internationalization support

As you are already using the Struts tags to show messages, adding support to other languages is a straightforward task.

Note: In this scenario, only View mode will be enabled for internationalization. Edit mode can also be implemented in the same manner but we are not doing it here.

Figure 15-33 on page 503 shows the resource files that you need to add and update to support internalization in your portlet. You will add internationalization support for View mode only and therefore for other modes (such as Edit mode) and the Struts framework will use the default values. That is, keys that are not found in the locale properties files will be taken from the file with no suffix. In this scenario, key values in the default files are in English.

Archived

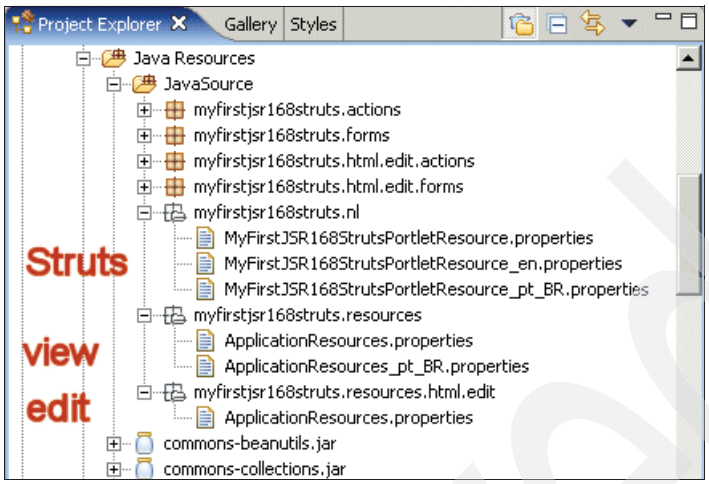


Figure 15-33 Resource files for internationalization (bundles)

15.9.1 Portlet application View mode internationalization

You will create a resource file for the new locale. Proceed as follows to add Brazilian Portuguese support to your portlet application View mode:

1. Expand **Dynamic Web Projects** → **MyFirstJSR168Struts** → **Java Resources** → **JavaSource** → **myfirstjsr168struts.resources**.
2. Right-click **ApplicationResources.properties**.
3. Select **Copy**.
4. Right-click the **myfirstjsr168struts.resources** package.
5. Select **Paste**.
6. Enter the new name for this file as `ApplicationResources_pt_BR.properties`.
7. Click **OK**.
8. Double-click **ApplicationResources_pt_BR.properties** to edit it.
9. Replace all messages with brazilian portuguese messages shown in Example 15-10.

Example 15-10 Resources file for Brazilian portuguese locale

```
myfirstjsr168struts.title=Portlet baseado em Struts utilizando a API JSR 168

message.mustConfigureFirst=Você deve primeiramente configurar este portlet em
modo de Edição.
message.configured=Este portlet foi configurado.
```

Archived

```
editmode.heading=Modo de Edição
editmode.title=Exemplo de Modo de Edição com Struts

editmode.instructions=Tente as opções abaixo para demonstrar várias capacidades
do Struts. Quando um usuário válido é inserido, uma action encaminha para um
JSP em modo view.
editmode.instructions.1=Digite "error" como nome de usuário ou deixe o nome de
usuário em branco para ver o suporte a ActionErrors.
editmode.instructions.2=Digite "delete" como nome de usuário para remover a
configuração.
editmode.instructions.3=Atualize a página para demonstrar o suporte
transacional.
editmode.instructions.4=Digite um nome de usuário válido para completar a
configuração e retornar para view mode.

prompt.username=Nome
prompt.password=Senha

button.back=voltar
button.send=Enviar
button.save=Salvar
button.cancel=Cancelar
button.reset=Limpar

error.transaction.token=Token de transação inválido
error.invalid.username=O nome de usuário não é válido
```

d. Save the file and close the editor.

15.9.2 Struts framework internationalization support

You also need to create a new resources file to be used by the Struts framework for internationalization of your portlet preferences such as the portlet title and so on. Execute the following steps:

1. Expand **Dynamic Web Projects** → **MyFirstJSR168Struts** → **Java Resources** → **JavaSource** → **myfirstjsr168struts.nl**.
2. Right-click **MyFirstJSR168StrutsPortletResource.properties**.
3. Select **Copy**.
4. Right-click the **myfirstjsr168struts.nl** package.
5. Select **Paste**.
6. Enter `MyFirstJSR168StrutsPortletResource_pt_BR.properties` as the new name for the file.
7. Click **OK**.

8. You do not have to add or change anything in this file, unless you want to change the portlet title for example.
9. Save the file and close the editor.

15.9.3 Editing the Portlet deployment descriptor

Update the portlet descriptor to add support for the pt_BR locale. Execute the following steps:

1. In Project Explorer view, expand **Dynamic Web Projects** → **MyFirstJSR168Struts**.
2. Double-click **Portlet Deployment Descriptor**.
3. In the Portlet deployment descriptor window, click **MyFirstJSR168Struts**.
4. Under the supported locales section, click **Add**. See Figure 15-34.

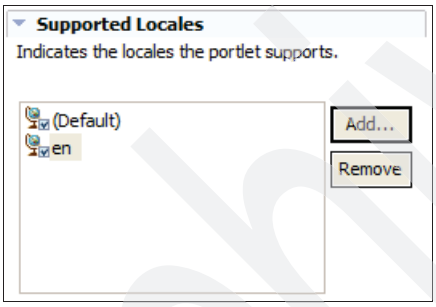


Figure 15-34 Supported locales section

5. Select **pt_BR** in the Locale list.
6. Click **OK**.
7. Save the file and close the editor.

15.9.4 Testing the new locale

1. Stop the Portal Test Environment.
2. Run the portlet application.
3. In the top menu, click **Edit My Profile**. If you need login, enter the required fields. User ID is wpsadmin and password is also wpsadmin.
4. Under Preferred language, select **Brazilian Portuguese**.
5. Click **OK**.
6. The application now should look like Figure 15-35 on page 506.

Archived



Figure 15-35 Portlet display using the new locale pt_BR

7. Select **Editar Meu Perfil** and locale **Ingles** if you want to go back to locale English.

15.10 Adding logging support to the application

The Struts Portlet Framework uses the Commons-Logging interface for a logging facility. The Struts Portlet Framework has supplied an implementation of the Commons-Logging Log interface that can be used to map trace messages to the logging facility used by the portal server. This file is normally found in the *wp_root*/log directory. Trace logging is enabled by setting properties in the *wp_root*/shared/app/config/log.properties file.

Note: By default tracing is disabled.

The trace string for tracing a Struts portlet application can be configured in the log.properties file. The trace string can be modified to add or remove classes.

Proceed as follows to enable logging in the portlet included in this sample scenario:

1. First, enable tracing of struts portlets as follows:
   a. Open the *wp_root*/shared/app/config/log.properties, where *wp_root* is the folder where the Portal test environment was installed. In this scenario, the file is located in the following path:

      c:\Progra~1\Portal51UTE\PortalServer\shared\app\config\log.properties

   b. Go to the end of the file after the following statement:

      ```
      #traceString=*=all=disabled statement
      ```

      and add the following traceString to enable Struts logging:

      ```
      traceString=org.apache.struts.*=all=enabled:
      com.ibm.portal.struts.*=all=enabled:
      ```

```
com.ibm.wps.struts.common.*=all=enabled:
com.ibm.wps.struts.base.*=all=enabled:
com.ibm.wps.standard.struts.logging.StrutsBaseTraceLogger=all=enabled:
myfirstjsr168struts.*=all=enabled
```

Important: Make sure you have only one traceString in the log.properties file. Also, the entire traceString must be specified in only one line. In addition, in this sample scenario, myfirst168struts is the name of the package.

c. Save the file and close it.

2. Additionally, the Struts Portlet Framework specifies the commons logging Log Factory in the META-INF/services directory. The log factory class name is dependent on the WebSphere Portal container. To specify the correct commons LogFactory for our application, proceed as follows:

   a. Expand **Dynamic Web Projects** → **MyFirstJSR168Struts** → **WebContent**.

   b. Right-click the **META-INF** folder. Select **New** → **Folder**.

   c. Enter `services` as the folder name. Click **Finish**.

   d. Right-click the **services** folder. Select **New** → **Other**.

   e. In the Select a wizard window do the following:

      i. Expand **Simple.**

      ii. Select **File**

      iii. Click **Next >**.

Archived

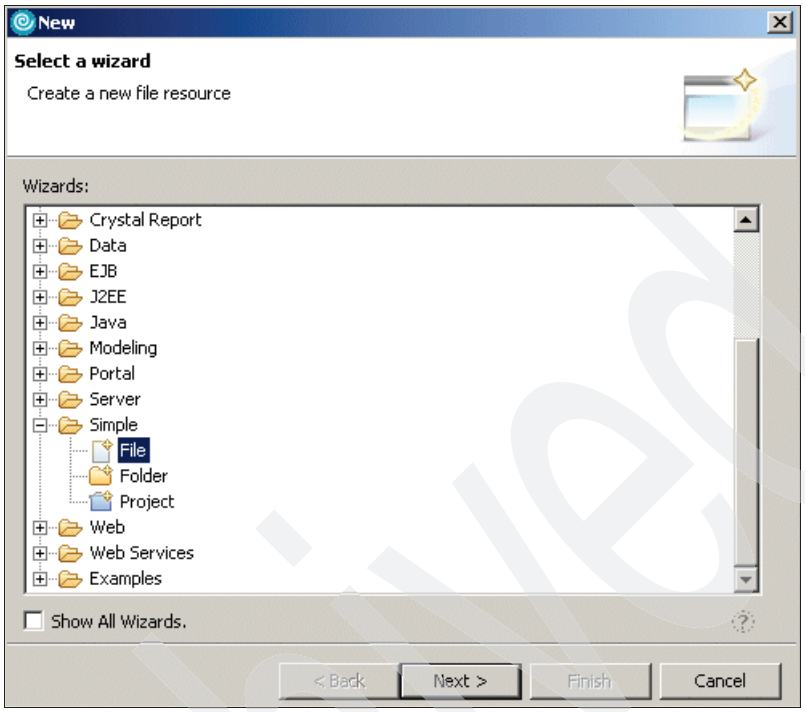


Figure 15-36 Select a wizard

f. Enter `org.apache.commons.logging.LogFactory` as the file name.

g. Click **Finish**.

h. Paste the following line in the file:

```
com.ibm.portal.struts.logging.StrutsLogFactory
```

i. Save the file and close it.

3. Add code in the welcome action mapping:

a. Go to **MyFirstJSR168Struts** → **Java Resources** → **JavaSource** → **myfirstjsr168struts.actions**

b. Edit **WelcomeAction**.

c. Add the following instance variable just before the execute method:

```
private Log log = LogFactory.getLog(this.getClass());
```

d. You will need to organize your imports. Right-click anywhere in the code and select **Source** → **Organize Imports**.

e. Select **org.apache.comons.logging.Log** for the `Log` class and click **Next.**

Archived

f. Select **org.apache.commons.logging.LogFactory** for the `LogFactory` class and click **Finish**.

g. Add the following code at the beginning of the execute method:

```
if (log.isTraceEnabled()) {
   log.trace("Welcome ###");
}
```

Note: Use ### or any other symbols to facilitate the search in the log file.

h. Finally, add the following code in the catch statement:

```
if (log.isDebugEnabled()) {
   log.debug("WelcomeAction: Error determining if user is configured");
}
```

i. Save the file and close editor.

4. Now you can run the application again. For example:

a. Stop the server to pick up the new property changes.

b. Select **MyFirstJSR168Struts** and click **Run** → **Run on Server**.

c. Open the log file at <*wp_root*>/log folder and look for the Welcome message by searching for the ### or any other symbols you used.

d. In this sample scenario the log files are in the following directory:

C:\Program Files\Portal51UTE\PortalServer\log\

e. Figure 15-37 illustrates a sample Struts log entry.

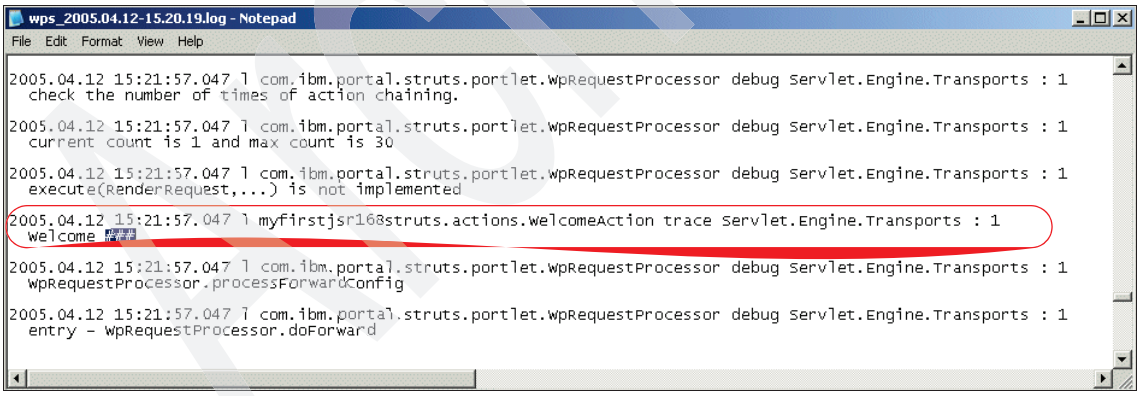


Figure 15-37 Sample Struts log entry

IBM Confidential

Archived

16

JavaServer Faces portlets

JavaServer Faces (JSF) is a technology that helps you build user interfaces for dynamic Web applications that run on the server. The JavaServer Faces framework manages UI states across server requests and offers a simple model for the development of server-side events that are activated by the client. JSF is consistent and easy to use. JSF is a standard defined by JSR 127.

This chapter presents an overview of the JSF framework and how it works. Analyzing a simple JSF application, we will present the main items that compose a JSF application. We will also talk about the specifics of JSF running in WebSphere Portal and how to migrate JSF applications to WebSphere Portal.

Note: For details about JavaServer Faces technology see *IBM WebSphere Studio V5.1.2 JavaServer Faces and Service Data Objects*, SG24-6361.

© Copyright IBM Corp. 2005. All rights reserved.

16.1 Overview

JavaServer Faces is based on the Model-View-Controller (MVC) design pattern and used to develop and build Web and Portal applications. For JSF, this means that the controller component is a servlet, the model component is represented by Java Beans, and the view comprises JSF components with little or no application code.

As illustrated in Figure 16-1, the main components of JSF technology are:

- An API for representing and managing UI components and their state; handling events, input validation at the server-side, and data conversion; defining page navigation; supporting internationalization and accessibility; and providing extensibility for all these features.
- An extensive set of reusable user interface (UI) components.
- JavaServer Pages (JSP) custom tag libraries for expressing a JavaServer Faces interface within a JSP.

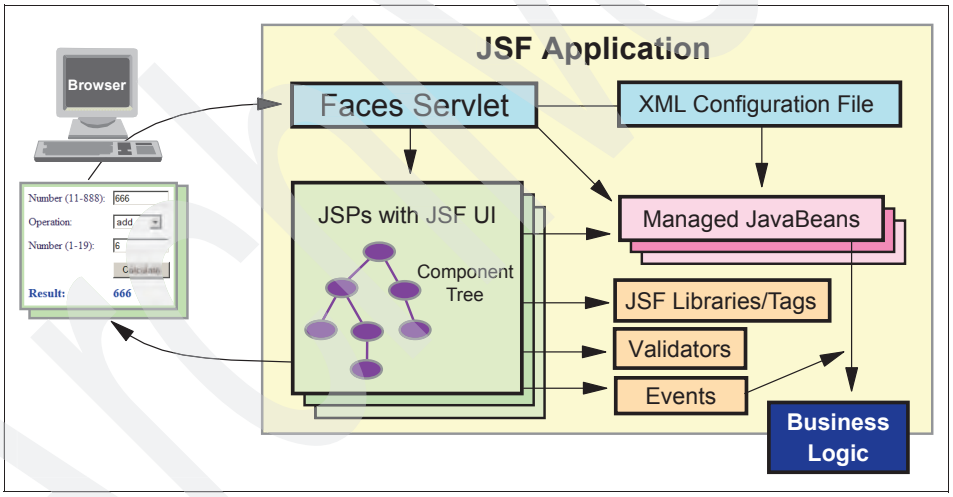


Figure 16-1 JavaServer Faces components

JSF technology provides a clean separation between behavior and presentation, like the one that is traditionally offered by client-side UI architectures. Figure 16-2 on page 513 illustrates the JavaServer Faces components within the MVC design pattern.

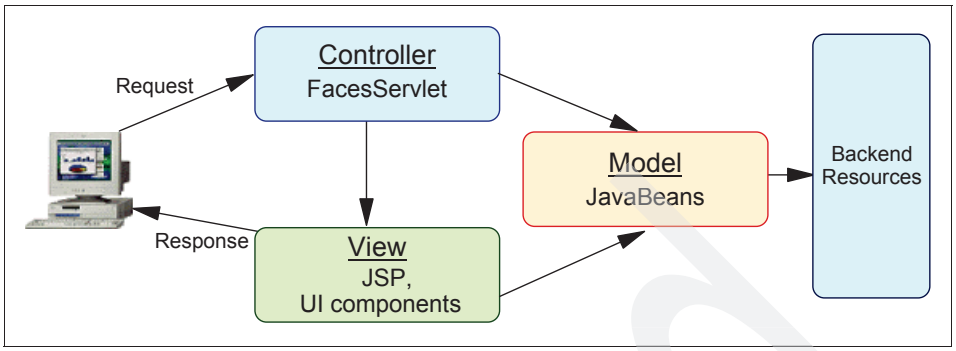


Figure 16-2 JavaServer Faces technology and the MVC design pattern

The actual work of a JSF application is executed by processing events triggered by the JSF components on the pages. These events are fired by user actions. For example, when the user clicks a button, the button triggers an event. The JSF programmer has to decide what the JSF application will do when a particular event happens. This is accomplished by implementing event listeners.

Note: JavaServer Faces technology is event-driven.

When an event is created, an HTTP request is sent to the server. Specifically, the request is sent to the JSF provided servlet called the `FacesServlet`. A JSF application in the Web container has its own `FacesServlet`.

The `FacesServlet` servlet creates an object called `FacesContext`, which holds the information required for request processing. That is, for Web applications, the `ServletContext`, `ServletRequest` and `ServletResponse` objects are passed to the `service` method of `FacesServlet`. For portlet applications, Portal passes the `PortletContext`, `PortletRequest` and `PortletResponse` objects to the `service` method of `FacesServlet`.

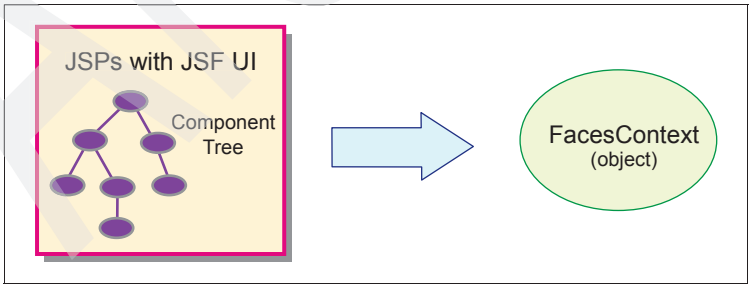


Figure 16-3 The FacesContext object

Once the `FacesContext` object has been created and populated, the `FacesServlet` servlet passes control to the `lifecycle` object. The `lifecycle` object processes the `FacesContext` object in six different phases, as follows:

1. Restore component tree
2. Apply request values
3. Process validators
4. Update model values
5. Invoke application
6. Render response

Also, the JSF application configuration is done through an application configuration file, where you can register Java Beans used in the application, define the program-control flow with page-navigation rules, and perform many other configuration tasks. The application configuration file is an XML file that is normally put in the WEB-INF directory and called faces-config.xml.

16.1.1 Life cycle of a JSF page

The life cycle of JSF pages is similar to that of a JSP page. The client makes an HTTP request for the page, and the server responds with the page translated to HTML. However, due to the extra features offered by JSF technology, there are some additional services provided by the life cycle to process a page.

A JSF page is represented by a tree of UI components, called a *view*. The life cycle begins when a client makes a request for the page. During the life cycle, the JSF implementation must build the view while considering state saved from a previous submission of the page. When the client submits a page, the JSF implementation must accomplish several tasks, such as validating the data input of components in the view and converting input data to types specified on the server side. The JSF implementation performs all these tasks as a series of steps in the life cycle.

Which steps in the life cycle are executed depends on whether or not the request originated from a JSF application and whether or not the response is generated with the rendering phase of the JSF life cycle. We will focus on the most complete scenario, where a JSF component submit a request to a JSF application using the `FacesServlet`.

Figure 16-4 on page 515 illustrates the steps in the JSF request-response life cycle.

Abbrev

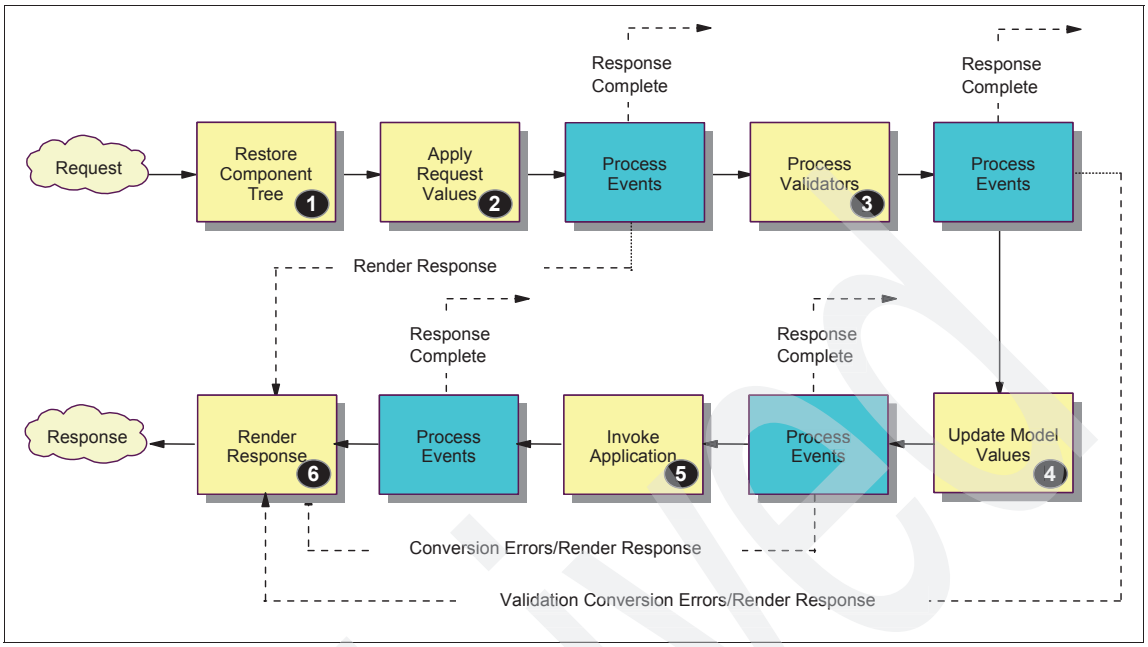


Figure 16-4 The phases of the JSF life cycle

The life cycle handles two kinds of requests: *initial requests* and *postbacks*. When a user makes an initial request for a page, he or she is requesting the page for the first time. When a user executes a postback, he or she submits the form contained on a page that was previously loaded into the browser as a result of executing an initial request. When the life cycle handles an initial request, it only executes the restore view and render response phases because there is no user input or actions to process. Conversely, when the life cycle handles a postback, it executes all of the phases.

Restore component tree

The JSF implementation begins this phase when a request for a JSF page is made. During this phase, the JSF implementation builds the component tree (also known as the view) of the page, wires event handlers and validators to components in the view and saves the view in the `FacesContext` instance.

All the application's component tags, event handlers, converters, and validators have access to the `FacesContext` instance.

If this is an initial request to the page, the JSF implementation builds an empty view during this phase and the life cycle advances to the render response phase. If the request is a postback, a view corresponding to this page already exists, so

the JSF implementation restores the view by using the state information that is saved.

Apply request values

The purpose of this phase is to give each component the opportunity to update its current value using the information included in the current request, such as parameters, headers, and cookies.

If any conversion of values is to be done and fails, an error message associated with the component is generated and queued on `FacesContext`. This message will be displayed during the render response phase, along with any validation errors resulting from the process validations phase.

If events have been queued during this phase, the JSF implementation broadcasts the events to interested listeners.

If the application needs to redirect to a different Web application resource or generate a response that does not contain any JSF components, it can call `FacesContext.responseComplete`.

At the end of this phase, the components are set to their new values, and messages and events have been queued.

Process validations

As part of this creation of the view for this request, zero or more validator instances can be registered for each component. In addition, component classes themselves can implement validation logic in their validate() methods. At the end of this phase, all configured validations are completed.

Validations that fail cause messages to be enqueued via calls to `FacesContext.addMessage`, and the valid property on the corresponding components are set to false. If any of the validate() methods that were invoked called `FacesContext.responseComplete`, the life cycle processing of the current request must be immediately terminated. If any of the validate() methods that were invoked called `FacesContext.renderResponse`, control must be transferred to the render response phase of the request processing life cycle. The same conditions are true for an event listener that processed a queued event. If none of these conditions occurs, control proceeds to the next phase to update model values.

Update model values

If this phase of the request processing life cycle is reached, it means that the incoming request is syntactically and semantically valid according to the validations that were performed. The local value of every component in the

component tree has been updated, and it is now appropriate to update the application's model data in preparation for performing any application events that have been queued.

The JSF implementation will update only the bean properties pointed at by an input component's value attribute. If the local data cannot be converted to the types specified by the bean properties, the life cycle advances directly to the render response phase so that the page is re rendered with errors displayed. This is similar to what happens with validation errors.

If the application needs to redirect to a different Web application resource or generate a response that does not contain any JSF components, it can call `FacesContext.responseComplete`.

If events have been queued during this phase, the JSF implementation broadcasts them to interested listeners.

Invoke application

During this phase, the JSF implementation handles any application-level events, such as submitting a form or linking to another page.

If the application needs to redirect to a different Web application resource or generate a response that does not contain any JSF components, it can call `FacesContext.responseComplete`.

If the view being processed was reconstructed from state information saved by a previous request and if a component has fired an event, these events are broadcast to interested listeners.

Render response

This phase accomplishes two things at the same time: causes the response to be rendered to the client, and causes the state of the response to be saved for processing on subsequent requests. The reason for handling both of these responsibilities in one phase is because the act of rendering the response in JSP applications can cause the view to be built as the page renders. Therefore, the state of the view cannot be saved until after it is rendered to the client.

If the request is a postback and errors were encountered during the apply request values phase, process validations phase, or update model phase, the original page is rendered during this phase. If the page contain `message` or `messages` tags, any queued error messages are displayed on the page.

Archived

Event processing

During several phases of the request processing life cycle, events can be queued, for example, via a call to the queueEvent method on the source UIComponent instance, or a call to the queue() method on the FacesEvent instance. These queued events must now be broadcast to interested event listeners. The broadcast is performed as a side effect of calling the appropriate life cycle management method (processDecodes(), processValidators(), processUpdates(), or processApplication()) on the UIViewRoot instance at the root of the current component tree. For each queued event, the broadcast() method of the source UIComponent will be called to broadcast the event to all event listeners who have registered an interest on this source component for events of the specified type. A boolean flag is returned indicating whether this event has been handled completely and whether the JSF implementation can remove it from the event queue.

It is also possible for event listeners to cause additional events to be queued for processing during the current phase of the request processing life cycle. Such events must be broadcast in the order they were queued after all originally queued events are broadcast, but before the life cycle management method returns.

16.2 A simple JSF application

A JSF application is just like any other Java Web application. It runs in a servlet container, and it typically contains:

- JSP pages
- Event listeners
- Java Beans that hold data and application-specific functionality
- Server-side classes, such as database access beans

In addition to these common items, a JSF application also has:

- A custom tag library for rendering UI components on a page. This is called the component tag library, and it is provided by the JSF implementation. The component tag library eliminates the need to hardcode UI components in any specific markup language, such as HTML. This results in completely reusable UI components.
- A custom tag library for representing event handlers, validators and other actions. This is called the core tag library, and it is provided by the JSF implementation. The core tag library makes it easy to register events, validators and actions on the components.
- UI components represented as stateful objects on the server.

Archive

- *Backing beans*, which define properties and functions for UI components.
- Validators, converters, event listeners and event handlers.
- An application configuration resource file for configuring application resources.

The example in this section is a simplification of the `calculator` application, developed in Chapter 17, "JavaServer Faces portlet development" on page 551. This application asks for two numbers and an operator. The application only accepts sum and subtraction operators. After performing the necessary validations, the application shows the result of the operation and lets the user decide if another operation should be done or if the application should be ended.

16.2.1 Creating the pages

The task of creating JSF pages involves laying out UI components on the pages, mapping these components with beans, and adding other core tags.

Example 16-1 shows the first page of the calculator application.

Example 16-1 welcome.jsp

```
<%@taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<HTML>
   <HEAD>
      <TITLE>Calculator</TITLE>
   </HEAD>
   <BODY>
      <f:view>
         <h:form id="form1">
            Number 1: 
            <h:inputText id="number1" value="#{calculator.number1}"
required="true">
               <f:convertNumber type="number"/>
            </h:inputText>
            Operation: 
            <h:inputText id="operation" value="#{calculator.operation}"
               validator="#{calculator.validateOperator}" required="true">
            </h:inputText>
            <h:message style=”color: red; text-decoration:overline”
               for="operation"/>
            Number 2: 
            h:inputText id="number2" value="#{calculator.number2}"
required="true">
               <f:convertNumber type="number"/>
            </h:inputText>
            <h:commandButton id="submit" value="Submit" action="success" />
```

Archive

```
        </h:form>
      </f:view>
   </BODY>
</HTML>
```

This page demonstrates some important features used in most of the JSF applications you will write.

The taglib references

To use the custom tags that represent JSF components, two taglib directives are specified on top of the page:

```
<%@taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
```

The first tag directive above points to the component tag library. The second one points to the core tag library.

The root tree tag

All JSF components must be enclosed in the `view` tags of the core tag library:

```
<f:view>
<%-- all JSF components come here --%>
</f:view>
```

The form tag

The `form` tag represents an input form that allows the user to input data and submit it to the server. All UI components that represent editable components (such as text fields, text areas and menus) must be nested inside the `form` tag.

The inputText tag

The `inputText` tag represents a text field component. In the `welcome.jsp` example, the first text field has three attributes: `id`, `value` and `required`.

The id attribute corresponds to the ID of the component object represented by this tag. If you do not specify the component's id attribute, the JSF implementation will generate one automatically. We need to specify the component's id when we want to refer to it inside other components. In this example, we can see that the `operation` component is referenced by the `message` component.

The `value` attribute binds the `number1` component with the bean property `calculator.number1`, which holds the data entered into the text field. We can also bind a component instance (instead of a component's property) to a property using the tag's `binding` attribute.

Archived

The `required` attribute tells to JSF implementation that a validation error should be fired if the component's value was not entered. We talk about validation in 16.3.5, "Validation model" on page 536.

The `operation` component also has an `validator` attribute. This attributes tells to JSF implementation to call the `calculator.validateOperator` method, to verify whether the value entered in the component is valid or not.

The convertNumber tag

By nesting the `convertNumber` tag within a component tag, we register a `Converter` onto the component. The converter registered by `convertNumber` will convert the value to and from a valid number style (currency, integer, decimal number or percent). In this case, the valid style is a decimal number (as stated by the `type` attribute of the `convertNumber` tag).

This tag is shown here just to let you know how conversion works. It could be omitted, once that `UIInput` components already perform this kind of conversion automatically between the component model and the domain model for standard types. 16.3.3, "Conversion model" on page 531 provides an overview of JSF conversion model.

The commandButton tag

The `commandButton` tag represents the button used to submit the data entered in the form. The `action` attribute specifies an outcome that helps the navigation mechanism decide which page to open next. The navigation model is explained in 16.2.2, "Defining navigation rules" on page 521.

The message tag

The `message` tag displays an error message if the data entered in the `operation` field does not comply with the rules specified in the `calculator.validateOperation` method. The error message displays wherever you place the `message` tag on the page. The `for` attribute refers to the component whose value failed validation, in this case the `operation` component represented by the `inputText` tag in the `welcome.jsp` page. Note that the tag representing the component whose value is validated must include an `id` attribute so that the `for` attribute of the `message` tag can refer to it.

16.2.2 Defining navigation rules

The JSF implementation has a powerful rule-based system to define which page to go after the user clicks a button or a hyperlink. These rules are defined in the application configuration resource file. Example 16-2 on page 522 shows the navigation rules defined for the calculator application.

Example 16-2 Navigation rules for the calculator application

```
<navigation-rule>
   <from-view-id>/welcome.jsp</from-view-id>
   <navigation-case>
      <from-outcome>success</from-outcome>
      <to-view-id>/result.jsp</to-view-id>
   </navigation-case>
</navigation-rule>
<navigation-rule>
   <from-view-id>/result.jsp</from-view-id>
   <navigation-case>
      <from-outcome>success</from-outcome>
      <to-view-id>/welcome.jsp</to-view-id>
   </navigation-case>
   <navigation-case>
      <from-outcome>end</from-outcome>
      <to-view-id>/end.jsp</to-view-id>
   </navigation-case>
</navigation-rule>
```

Each `navigation-rule` element defines how to get from one page (specified in the `from-view-id`) to other pages in the application. The `navigation-rule` elements can contain any number of `navigation-case` elements, each of which defines the page to open next (defined by `to-view-id`) base on a logical outcome (defined by `from-outcome`.

The outcome can be defined either by the `action` attribute of the `UICommand` component that submits the form, or by the return value of an *action method* in a backing bean.

The `welcome.jsp` page illustrates the first way of defining the outcome:

```
<h:commandButton id="submit" value="Submit" action="success" />
```

Here, the commandButton component specifies the action `success`. If we look at the navigation rules, we will see that this outcome makes the JSF implementation call the `result.jsp` page.

If we wanted to perform some processing to determine the outcome, for example, to check if the password entered by the user is valid or not, and return an outcome of `success` or `failure`, the commandButton `action` attribute would point to a method, instead of specifying directly the outcome. If we have a `login` component that implements this logic in its `verify` method, this would result in a tag similar to the following:

```
<h:commandButton id="submit" value="Submit" action="#{login.verify}" />
```

Archived

16.2.3 Developing the beans

A typical JSF application couples a backing bean with each page in the application. The backing bean defines properties and methods associated with the UI components used on the page. We can bind backing beans properties to either a component instance or a component value.

Backing beans can also define methods that perform functions for the component, such a validation and event handling.

We saw in “The inputText tag” on page 520 that we bind a component’s value to a bean property using the component tag’s `value` attribute to refer to the property. Example 16-3 shows the `CalculatorBean` backing bean property that maps to the data for the `number1` component:

Example 16-3 A property defined in the backing bean CalculatorBean

```
private int number1;
...
public int getNumber1() {
   return number1;
}
public void setNumber1(int number1) {
   this.number1 = number1;
}
```

Note that this bean property is just like any other bean property, that is, it is defined by a set of accessor methods and a private field.

A property can be any of the basic primitive and numeric types or any Java object type for which an appropriate converter is available. JSF technology automatically converts the data to the type specified by the bean property. Table 16-1 shows which types are accepted by which component tags.

Table 16-1 The acceptable types for each component value.

Component	Acceptable types
UIInput, UIOutput, UISelectItem, UISelectOne	Any basic primitive and numeric types, as well as any Java object type for which an appropriate Converter implementation is available.
UIData	array of beans, List of beans, single bean, java.sql.ResultSet, javax.servlet. jsp.jstl.sql.Result, javax.sql.RowSet.
UISelectItems	java.lang.String, Collection, Array, Map.

Archived

Component	Acceptable types
UISelectBoolean	boolean or Boolean.
UISelectMany	array or List. Elements of the array or List can be any of the standard types.

In addition to binding components and their values to backing bean properties, we can refer to a backing bean method from a component tag. See 16.5, “Backing bean management” on page 540 for more information about this topic.

Adding managed bean declarations

Once that the backing beans are developed, we have to configure them in the application configuration resource file so that the JSF implementation can automatically create new instances of it whenever they are needed. Example 16-4 shows the managed bean declaration for `CalculatorBean`.

Example 16-4 Declaration of CalculatorBean at the configuration file

```
<managed-bean>
   <managed-bean-name>calculator</managed-bean-name>
   <managed-bean-class>pagecode.beans.CalculatorBean</managed-bean-class>
   <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

The application configuration file can specify initial values to the bean’s properties. This is done by using the `managed-property` tag in the `managed-bean` definition. If we want to specify an initial value of 30 for `number1`, the `managed-bean` tag would look like the one in Example 16-5. Properties that are not set an initial value in the `managed-bean` declaration will be initialized to whatever the constructor of the bean class has the instance variable set to.

Example 16-5 Declaration of CalculatorBean with an initial value set to number1 property

```
<managed-bean>
   <managed-bean-name>calculator</managed-bean-name>
   <managed-bean-class>pagecode.beans.CalculatorBean</managed-bean-class>
   <managed-bean-scope>session</managed-bean-scope>
   <managed-property>
      <property-name>number1</property-name>
      <property-class>int</property-class>
      <value>30</value>
   </managed-property>
</managed-bean>
```

The JSF implementation processes the configuration file on application startup time. When the `CalculatorBean` is first referenced from a page, the JSF

implementation initializes it and stores it in session scope if no instance exists. The bean is then available for all pages in the application. The scope can also be set as `request` or `application`.

16.3 User interface component model

JSF UI components are configurable, reusable elements that compose the user interfaces of JSF applications. A component can be simple, such as a button, or compound, such as a table, which can be composed of several other components.

The component architecture of JSF technology includes:

- A set of `UIComponent` classes for specifying the state and behavior of UI components.
- A rendering model that defines the several ways of rendering the components.
- An event and listener model that defines how to handle component events.
- A conversion model that defines how data converter are registered onto a component.
- A validation model that defines how validators are registered onto a component.

16.3.1 User interface component classes

The `UIComponent` interface defines the contract between a UI component and the JSF implementation. This interface must be implemented by every UI component, either directly or indirectly.

The JSF implementation provides base implementation class to this interface, called `UIComponentBase`. This class defines the default state and behavior of a UI component. It is recommended that you extend the `UIComponentBase` class, rather than implement the `UIComponent` interface directly, to minimize the impact of any future changes to the `UIComponent` interface to your application.

The UI component classes included with JSF technology are:

- `UIColumn`, which represents a column in a `UIData` component.
- `UICommand`, which represents a control that performs actions when the user activate it.
- `UIData`, which represents a data binding to a collection of data represented by a `DataModel` instance.

Archived

- `UIForm`, which represents an input form.
- `UIGraphic`, which displays an image.
- `UIInput`, which represents a component that gets and displays user input.
- `UIMessage`, which displays a localized message.
- `UIMessages`, which displays a set of localized messages.
- `UIOutput`, which displays data output.
- `UIPanel`, which is a container to child components.
- `UIParameter`, which represents substitution parameters used to configure parent components.
- `UISelectBoolean`, which represents a component that have a boolean value.
- `UISelectItem`, which represents a single select item for select components.
- `UISelectItems`, which represents an entire set of items.
- `UISelectMany`, which represents a component that allows the user to select zero or more values from a set of values.
- `UISelectOne`, which represents a component that allows the user to select zero or one value from a set of values.
- `UIViewRoot`, which represents the root of the component tree.

The component classes also implement one or more *behavioral interfaces*, each of one defines a different behavior for a set of components whose classes implement it. These interfaces are as follows:

- `ActionSource`: By implementing this interface, the component is able to fire an action event.
- `EditableValueHolder`: This interface extends `ValueHolder` and specifies additional features for editable components, such as validation and emitting value-change events.
- `NamingContainer`: Each component rooted at the component that implements this interface must have a unique ID.
- `StateHolder`: Indicates that the component has state that must be saved between requests.
- `ValueHolder`: Denotes that the component maintains a local value as well as the option of accessing data in the model tier.

You only need to use the component classes and behavioral interfaces if you are a component writer. Otherwise, you only need to know the appropriate tags that will instruct the JSF implementation on how to render a specific component. These tags are covered in the next section.

Archived

16.3.2 Component rendering model

The JSF component architecture separates the functionality of the components from its rendering. In this way, the behavior of a component can be defined once, despite of the multiple renderers each of which defines a different way to render the component.

For example, a `UISelectOne` component has three different renderers. One of them renders the component as a set of radio buttons. Another renders it as a combo box. And the third one renders it as a list box.

The component tag library provides a set of tags, each one composed of the component functionality (defined in the `UIComponent` class) and the rendering attributes (defined by the `Renderer` class). This tag library supports all the component tags listed in Table 16-2.

Table 16-2 The UI component tags

Tag	Description	Rendered As	Appearance
`<h:column>`	Represents a column of data in a `UIData` component	A column in an HTML table	A column in a table
`<h:commandButton>`	Submits a form	An HTML `<input type=`*`type`*`>` element, where the *`type`* value can be `submit`, `reset`, or `image`	A button
`<h:commandLink>`	Links to another page or location on a page	An HTML `<a href>` tag	A hyperlink
`<h:dataTable>`	Represents a data wrapper.	An HTML `<table>` tag	A table that can be updated dynamically
`<h:form>`	Represents an input form.	An HTML `<form>` tag	Invisible
`<h:graphicImage>`	Displays an image	An HTML `<img>` tag	An image
`<h:inputHidden>`	A hidden variable in a page	An HTML `<input type=hidden>` tag	Invisible
`<h:inputSecret>`	Allows a user to input a string without the actual string appearing in the field	An HTML `<input type=password>` tag	A text field, which displays a row of characters instead of the actual string

Redbooks

Tag	Description	Rendered As	Appearance
<h:inputText>	Allows the user to input a string	An HTML <input type=text> tag	A text field
<h:inputTextarea>	Allows the user to input a multiline string	An HTML <input type=textarea> tag	A multirow text field
<h:message>	Displays a localized message	An HTML tag if styles are used	A text string
<h:messages>	Displays localized messages	An HTML tag if styles are used	A text string
<h:outputLabel>	Displays a nested component as a label for a specified input field	An HTML <label> tag	Plain text
<h:outputLink>	Links to another page or location on a page without generating an action event	An HTML <a> tag	A hyperlink
<h:outputFormat>	Displays a localized message	An HTML <p> tag	Plain text
<h:outputText>	Displays a line of text	An HTML <p> tag	Plain text
<h:panelGrid>	Displays a table	An HTML <table> tag with <tr> and <td> tags	A table
<h:panelGroup>	Groups a set of components under one parent	An HTML <tr> tag with <td> tags	A row in a table
<h:selectBooleanCheckbox>	Allows a user to change the value of a Boolean choice	An HTML <input type=checkbox> tag	A checkbox
<h:selectItem>	Represents one item in a list of items in a UISelectOne component	An HTML <option> tag	Invisible
<h:selectItems>	Represents a list of items in a UISelectOne component	A list of HTML <option> tags	Invisible
<h:selectManyCheckbox>	Displays a set of checkboxes from which the user can select multiple values	A set of HTML <input type=checkbox> tags	A set of checkboxes
<h:selectManyListbox>	Allows a user to select multiple items from a set of items, all displayed at once	An HTML <select> element	A list box

Archive

Tag	Description	Rendered As	Appearance
<h:selectManyMenu>	Allows a user to select multiple items from a set of items	An HTML <select> element	A scrollable combo box
<h:selectOneListbox>	Allows a user to select one item from a set of items, all displayed at once	An HTML <select> element	A list box
<h:selectOneMenu>	Allows a user to select one item from a set of items	An HTML <select> element	A scrollable combo box
<h:selectOneRadio>	Allows a user to select one item from a set of items	An HTML <input type=radio> element	A set of radio buttons

IBM extensions to the component model

One of the major benefits of JSF is its ability to be extended with useful components to support application requirements. The JSF specification has included a base set of components that map to HTML controls. Rational Application Developer now includes a powerful set of extension components that improve productivity and add richness to the existing base components. The JSF community will eventually include a large set of open source and product supplied components that customers from small businesses to enterprises can use in application development. Table 16-3 lists some of the extension components that are provided by IBM in Rational Application Developer. For a complete reference of extended JSF components provided by Rational Application Developer, refer to Rational Application Developer online help.

Table 16-3 Some of IBM extension components

Faces component	Description	Tag	Type
Command-Button	Creates a push button that can have text or an image. The button may run an action.	<hx:commandExButton>	UICommand
Link	Creates a hyperlink to the URL you specify.	<hx:outputLinkEx>	UICommand
Image	Displays an image on the page. Can display an image from a data source as well.	<hx:graphicImageEx>	UIGraphic
File Upload	Displays an input field and associated Browse button for uploading a file.	<hx:fileUpload>	
Horizontal Rule	Creates a horizontal line to visually separate information on the page.	<hx:outputSeparator>	UIOutput

Faces component	Description	Tag	Type
Panel Group Box - Snap to border	Creates a container to group other components, organizing them along the sides of the panel.	`<hx:panelLayout>`	`UIPanel`
Panel Group Box - List	Creates a container to group other components, organizing them as a vertical or horizontal list.	`<hx:panelBox>`	`UIPanel`
Panel Menu Bar	Inserts a panel that places commands into a menu bar. You can drag and drop buttons, hyperlinks and horizontal rules to the panel or add them from within the Properties views of the components. You can also add a sub-menu bar within a menu bar.	`<hx:panelActionbar>`	`UIPanel`
Panels - Tabbed	Creates a set of overlapping tabbed pages. Each page can contain a set of components. The user clicks a tab to show the contents of that page.	`<odc:tabbedPanel>`	
Rich Text Area	Inserts a rich text editor component. Unlike the `<h:inputTextarea>`, this component can contain different fonts and sizes, tables, links, and numbered and bulleted lists.	`<r:inputRichtext>`	
Generic A/V Player	Creates a media player on the Web page and plays a file.	`<hx:playerGenericPlayer>`	
Macromedia Flash Player	Creates an instance of the Macromedia Flash Player in order to play Flash files on the Web page.	`<hx:playerFlash>`	
Macromedia Shockwave Player	Plays Macromedia Shockwave Player files on the Web page.	`<hx:playerShockwave>`	
RealOne Player	Plays Real Network RealOne Player files on the Web page.	`<hx:playerRealPlayer>`	
Windows Media Player	Creates an instance of the Windows Media Player in order to play media files on the Web page.	`<hx:playerMediaPlayer>`	

Redbooks

16.3.3 Conversion model

JSF components can be associated with server-side object data, represented by a Java Beans component. When this occurs, the application has two views of the component's data:

- The model view, in which data is represented as data types, such as `int` or `long`.
- The presentation view, in which data is represented in a manner that can be read or updated by the user. For example, a `java.util.Date` might be represented as a text string in the format mm/dd/yyyy or a set of tree text strings.

The JSF implementation automatically converts component data between these two views when the bean property associated with the component is of one of the types presented in Table 16-1 on page 523. For example, if a `UIInput` component is associated with bean property of type `int`, the JSF implementation will automatically convert the components data from `String` to `int`.

You might want to convert a component's data to a type other than a standard type. To allow this, JSF technology provides a way for you to register a `Converter` implementation on `UIOutput` components and components whose classes subclass `UIOutput`.

You can either use the standard converters that come with JSF implementation or create your own custom converter. In order to create and use a custom converter, three things must happen:

- The `Converter` interface must be implemented.
- The newly created `Converter` must be registered with the application, in the application configuration resource file. This is done in the `Converter` tag.
- The `Converter` tag must be referenced from the tag of the component whose data must be converted.

The following sections explain how to make each of these three things happen.

Implementing the Converter interface

The `Converter` interface defines the following two methods:

```
Object getAsObject(FacesContext context,UIComponent component,String newValue)
String getAsString(FacesContext context,UIComponent component,Object value)
```

The first method converts a string into an object of the desired type, throwing a `ConverterException` if the conversion cannot be carried out. This method is called when a string is submitted from the client, typically in a text field. The

second method converts an object into a string representation to be displayed in the client interface.

Imagine that we had to develop a custom converter for credit card numbers. This converter would allow users to enter a credit card number with or without spaces. That is, the user could type the card number in any of the following forms:

```
1234567890123456
1234 5678 9012 3456
```

Example 16-6 shows the code for the custom converter. Notice that the `getAsString` method tries to format the credit card number in a way that is pleasing to the user. The digits are separated with white spaces, depending on the credit card type. Table 16-4 shows the most common credit card formats.

Table 16-4 Most common credit card formats

Card Type	Digits	Format
MasterCard	16	5xxx xxxx xxxx xxxx
Visa	16	4xxx xxxx xxxx xxxx
Visa	13	4xxx xxx xxx xxx
Discover	16	6xxx xxxx xxxx xxxx
American Express	15	37xx xxxxxx xxxxx
American Express	22	3xxxxx xxxxxxxx xxxxxxxx
Diners Club, Carte Blanche	14	3xxxx xxxx xxxxx

Example 16-6 CreditCardConverter.java

```
package com.yourco.converters;

import javax.faces.application.FacesMessage;
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.convert.Converter;
import javax.faces.convert.ConverterException;

import com.yourco.beans.CreditCard;

public class CreditCardConverter implements Converter {

   public Object getAsObject(FacesContext context, UIComponent component,
         String newValue) throws ConverterException {
      boolean foundIllegalChar = false;
      char illegalChar = '\0';
```

```
    StringBuffer sb = new StringBuffer(newValue);
    int i = 0;
    while (i < sb.length() && !foundIllegalChar) {
       char c = sb.charAt(i);
       if (Character.isDigit(c)) {
          i++;
       } else if (Character.isWhitespace(c)) {
          sb.deleteCharAt(i);
       } else {
          foundIllegalChar = true;
          illegalChar = c;
       }
    }
    if (foundIllegalChar) {
       FacesMessage errmsg = new FacesMessage(FacesMessage.SEVERITY_ERROR,
             "An invalid character was found.",
             "The character " + illegalChar + "is not valid.");
       throw new ConverterException(errmsg);
    }
    return new CreditCard(sb.toString());
}

public String getAsString(FacesContext context, UIComponent component,
      Object value) throws ConverterException {
    int[] boundaries = null;
    String v = value.toString();
    int length = v.length();
    switch (length) {
    case 13:
       boundaries = new int[]{4, 7, 10};
       break;
    case 14:
       boundaries = new int[]{5, 9};
       break;
    case 15:
       boundaries = new int[]{4, 10};
       break;
    case 16:
       boundaries = new int[]{4, 8, 12};
       break;
    case 22:
       boundaries = new int[]{6, 14};
       break;
    default:
       return v;
    }
    StringBuffer sb = new StringBuffer();
    int start = 0;
    for (int i = 0; i < boundaries.length; i++) {
```

Archived

```
            int end = boundaries[i];
            sb.append(v.substring(start, end));
            sb.append(" ");
            start = end;
        }
        sb.append(v.substring(start));
        return sb.toString();
    }
}
```

The `CreditCard` class referenced in the code is very simple. It contains just the credit card number. See Example 16-6 on page 532:

Example 16-7 CreditCard.java

```
package com.yourco.beans;
public class CreditCard {
    private String creditCardNumber;
    public CreditCard(String creditCardNumber) {
        this.creditCardNumber = creditCardNumber;
    }
    public String toString(){ return creditCardNumber; }
}
```

Registering the converter

To register the converter, we have to associate a symbolic ID with the `Converter` class. This is done with the following entry in `faces-config.xml`:

Example 16-8 Registering the converter in faces-config.xml

```
<converter>
   <converter-id>CreditCard</converter-id>
   <converter-class>
      com.yourco.converters.CreditCardConverter
   </converter-class>
</converter>
```

In Example 16-8, we chose theID `CreditCard` for our credit card converter.

Referencing the converter

To reference the converter, we can use the `f:converter` tag in our `UIInput` component, specifying the converter ID:

```
<h:inputText value="#{someBean.card}">
   <f:converter converterId="CreditCard"/>
</h:inputText>
```

Archived

We could also be more succint and use the `converter` attribute of the `UIInput` component:

```
<h:inputText value="#{someBean.card}" converter="CreditCard"/>
```

16.3.4 Event and listener model

JSF applications are event-driven. Following the Java 2 event model, any object in a JSF application can be designed to generate or receive events. For example, a `UICommand` component can generate an event when the user clicks it.

There are three participants involved in the event and listener model:

- The event source, which is the object whose state changes.
- The event object, which encapsulates the state changes in the event source.
- The event listener, which is the object that wants to be notified of the state changes of the event source.

In summary, when an event occurs, the event source generates an `Event` object and sends it to the event `Listener`. The `Event` object stores information about the event. To be notified of an event, an application must provide an implementation of the `Listener` class and must register it on the component that generates the event.

JSF technology supports three kinds of events: value-change events, action events, and phase events.

An *action event* is fired by command components, for example, `h:commandButton` and `h:commandLink`, when the button or link is activated.

A *value-change event* occurs when the user modifies the value of a component represented by `UIInput` or one of its subclasses and the enclosing form is submitted. Value-change events are fired only if no validation errors were detected.

A *phase event* is fired by the JSF life cycle.

The application can react to action events or value-change events in either of two ways:

- Implement an event listener class to handle the event and register the listener on the component by nesting either a `valueChangeListener` tag or an `actionListener` tag inside the component tag.
- Implement a method of a backing-bean to handle the event and refer to the method with a method-binding expression from the appropriate attribute of the component's tag.

Archived

In Example 16-9, you can see how to attach a value change listener to a menu. The example also uses the `onchange` attribute to force a form submit after the menu's value is changed. This example was extracted from the calculator application that is built in Chapter 17, "JavaServer Faces portlet development" on page 551.

Example 16-9 Attaching a valueChangeListener to a component

```
<h:selectOneMenu styleClass="selectOneMenu" id="operation"
valueChangeListener="#{pc_Calculate.handleOperationValueChange}"
onchange="sumit()">
   ..........
</h:selectOneMenu>
```

Example 16-10 handles the `ValueChangeEvent`. This example was also extracted from the calculator application that is built in Chapter 17, "JavaServer Faces portlet development" on page 551.

Example 16-10 Handling a ValueChangeEvent

```
public void handleOperationValueChange(ValueChangeEvent valueChangedEvent) {
   log("OperationValueChangeEvent start");
   HtmlSelectOneMenu operation =
         (HtmlSelectOneMenu)valueChangedEvent.getSource();
   String opnew = (String)valueChangedEvent.getNewValue();
   String opold = (String)valueChangedEvent.getOldValue();
   log("operation old="+opold+" new="+opnew);
   getCalculator().setOperation(opnew); // is also done later by binding
   getOpchanged().setValue("changed");  // change the output text field
   getCalc().setValue(opnew);           // change the Calculate button
   log("OperationValueChangeEvent end");
}
```

16.3.5 Validation model

JSF technology provides a set of classes for validating input values entered into input components, such as text fields. These classes are called *validators*.

JSF offers some standard validators that you can use in your applications. Alternatively, you can write your own validator if none of the standard validators suits your needs.

Basically, a validator is an implementation class that checks an input value and sends an error message if the input is not valid. You use a validator by nesting it inside an input component whose input needs to be validated. If the validator decides that the user's input is invalid, the JSF `FacesServlet` redisplays the JSP

page from which the form was submitted, without copying the local value to the Java Bean instance bound to the input component.

The JSF core tag library defines a set of tags that correspond to a set of standard classes for performing common data validation checks. Table 16-5 shows all the standard validation classes and corresponding tags.

Table 16-5 The validator classes

Validator Class	Tag	Function
DoubleRangeValidator	validateDoubleRange	Checks whether the local value of a component is within a certain range. The value must be a floating-point or convertible to floating-point
LengthValidator	validateLength	Checks whether the length of a component's local value is within a certain range. The value must be a `java.lang.String`
LongRangeValidator	validateLongRange	Checks whether the local value of a component is within a certain range. The value must be any numeric type or String that can be converted to a long

To specify the range that the validators shown in Table 16-5 must check, you use the `maximum` and `minimum` attributes inside each of the validator's tag. These values can be hardcoded in the tag or refer to a backing-bean property.

You can also specify that an input component has a `required` attribute. By doing this, the JSF implementation checks whether the value of the component is null or is an empty String.

The validation model also allows you to create your own custom validator and corresponding tag to perform custom validation. There are two ways to implement custom validators:

- Implement a `Validator` interface that performs the validation (see Example 16-11 on page 538). By doing this, you must also:
  - Register the `Validator` implementation with the application (See Example 16-12 on page 538).
  - Create a custom tag or use a `validator` tag to register the validator on the component (See Example 16-13 on page 539).

Archive

- Implement a backing-bean method that performs the validation. By doing this, you must also reference the validator from the component tag's `validator` attribute.

Example 16-11 Implementing a validator

```
package itso.jsf.calculator;

import javax.faces.application.FacesMessage;
import javax.faces.component.UIComponent;
import javax.faces.component.UIInput;
import javax.faces.context.FacesContext;
import javax.faces.validator.Validator;
import javax.faces.validator.ValidatorException;

public class OddValidator implements Validator {

   public void validate(FacesContext arg0, UIComponent arg1, Object arg2)
         throws ValidatorException {
      System.out.println("OddValidator start");
      UIInput field = (UIInput)arg1;
      int value = ((Long)arg2).intValue();
      System.out.println("Field="+field.getId()+" Value="+value);
      if (value%2 == 1) {
         field.setValid(true);
         System.out.println("OddValidator end: valid");
      } else {
         System.out.println("OddValidator end: invalid");
         FacesMessage errmsg = new FacesMessage
            (FacesMessage.SEVERITY_ERROR,
               "2nd number not odd.",
               "Second number must be odd.");
         throw new ValidatorException(errmsg);
      }
   }
}
```

Example 16-12 Registering the validator

```
<validator>
   <description>Registers the OddValidator</description>
   <validator-id>oddValidator</validator-id>
   <validator-class>itso.jsf.calculator.OddValidator</validator-class>
</validator>
```

Archived

Example 16-13 Invoking the validator with the validator tag

```
h:inputText styleClass="inputText" id="number2" required="true" maxlength="2"
   size="12" value="#{pc_Calculate.calculator.number2}">
   <f:validator validatorId="oddValidator"></f:validator>
</h:inputText>
```

16.4 Navigation model

Page navigation is a crucial aspect of all Web applications. The JSF navigation model makes it easy to define page navigation and to handle any additional processing needed to choose the sequence in which pages are loaded.

In the JSF technology, *navigation* is a set of rules for choosing the next page to be displayed after a `UICommand` is clicked, that is, a button or hyperlink. These rules are defined in the application configuration resource file.

To handle the simplest navigations, you have to:

- Define the rules in the application configuration resource file (see Example 16-16 on page 540).
- Refer to an outcome `String` from the button or hyperlink component's `action` attribute. This outcome `String` is used by the JSF implementation to select the navigation rule to be used. Here is an example of this. Notice that the `action` attribute is defining the outcome of `back`:

  ```
  <hx:commandExButton type="submit" value="Back" action="back">
  </hx:commandExButton>
  ```

In more complicated applications, you also must provide one or more action methods, which perform some processing to determine which page should be displayed next. For example, a `login` action can be triggered when the user submits a form by clicking a button. This action can determine whether the login data entered is valid or not, and return a logical outcome `String (in this case either` "success" or "failure"). The `NavigationHandler` receives this outcome and determines which page to display next by matching the outcome or the action method reference against the navigation rules in the application configuration resource file. The Example 16-14 shows how to reference a method that implements this kind of dynamic navigation. Example 16-15 on page 540 shows how this method can be implemented. This method has no parameters and a return type `String`.

Example 16-14 Invoking an action method to determine the correct outcome

```
<h:commandButton label="login" action="#{loginController.veryfyUser}"/>
```

Example 16-15 Deciding the correct outcome

```
String veryfyUser() {
   if (...)
      return "success";
   else
      return "failure";
}
```

Each navigation rule defines how to navigate from one particular page to any number of other pages in the application. Each navigation case within a navigation rule defines a target page and either a logical outcome, a reference to an action method, or both. Example 16-16 shows an example navigation rule from the calculator application described in 16.2.2, “Defining navigation rules” on page 521.

Example 16-16 An example navigation rule

```
<navigation-rule>
   <from-view-id>/welcome.jsp</from-view-id>
   <navigation-case>
      <from-outcome>success</from-outcome>
      <to-view-id>/result.jsp</to-view-id>
   </navigation-case>
</navigation-rule>
```

This rule states that when a button or hyperlink component on `welcome.jsp` is activated, the application will navigate from the `welcome.jsp` page to the `result.jsp` page if the outcome referenced by the button or hyperlink component’s tag is `success`.

16.5 Backing bean management

Typical JSF applications include one or more backing beans. These are JavaBean components associated with UI components used in a page. A backing bean holds UI component properties, each of which is bound to either a component’s value or a component’s instance. Backing beans can also define methods that perform functions associated with a component, including validation, event handling, and navigation processing.

The binding of component values and instances to backing beans is done through the use of JSF expression language (EL) syntax. This syntax uses the delimiters # { }. A JSF expression can be a value-binding expression or a method binding expression. It can also accept mixed literals and the evaluation syntax and operators of the JSP 2.0 expression language.

To illustrate value-binding expressions and method-binding expressions, take a look at Example 16-17, which defines the `operation` component of the calculator application.

Example 16-17 An UIInput component using both value-binding and method-binding expressions

```
<h:inputText id="operation" value="#{calculator.operation}"
   validator="#{calculator.validateOperator}" required="true">
</h:inputText>
```

This tag binds the `operation` component's value to the `calculator.operation` backing bean property. it also refers to the `calculator.validateOperator` method, which performs validation of the component's local value, which is whatever the user enters into the field corresponding to this tag.

A tag representing a component that implements `ActionSource` can refer to backing bean methods using `actionListener` and `action` attributes. The `actionListener` attribute refers to a method that handles an action event. The `action` attribute refers to a method that performs some processing associated with navigation and returns a logical outcome `String`, which the navigation system uses to determine which page to display next.

A tag can also bind a component instance to a backing bean property, instead of a component's value. This is done by referencing the property from the `binding` attribute:

```
<inputText binding="#{calculator.operationComponent}"
```

The property referenced from the `binding` attribute must accept and return the same component type as the component instance to which it is bound. Example 16-18 shows an example property the can be bound to the component represented by the preceding `inputText` tag.

Example 16-18 Binding a component instance to a backing bean property

```
UIInput operationComponent = null;
...
public void setOperationComponent(UIInput operationComponent) {
   this.operationComponent = operationComponent;
}
public UIInput getOperationComponent() {
   return operationComponent;
}
```

These are the advantages of binding a component instance to a bean property:

- The backing bean can programmatically modify component attributes, like the `rendered` attribute, that specify whether the component will be rendered or not.
- The backing bean can instantiate components rather than let the page author do so.

These are the advantages of binding a component's value to a bean property:

- The page author has more control over the component attributes.
- The backing bean has no dependencies on the JSF API, allowing a greater separation between the presentation layer and the model layer.
- The JSF implementation can perform conversions on the data based on the type of the bean property without the developer needing to apply a converter.

In most situations, you will bind a component's value rather than its instance to a bean property.

Backing beans are created and stored with the application using the managed bean creation facility, which is configured in the application configuration resource file, as shown in 16.2.3, "Developing the beans" on page 523. When the application starts up, it processes this file, making the beans available to the application and instantiating them when they are first referenced by the component tags.

16.6 JSF in portlets

JSF applications behavior is much similar when inside portlets. In this chapter, we will see how JSF applications work as a portlet, how does the JSF life cycle maps to the portlet life cycle and some JSF portlet programming guidelines.

16.6.1 JSF portlet runtime

The JSF portlet runtime is the component that makes possible to run JSF applications as portlets in WebSphere Portal. The JSF portlet runtime is found in a different jar file for each portlet API:

- For the JSR 168 API, the jar file is jsf-portlet.jar.
- For the IBM Portlet API, the jar file is jsf-wp.jar.

Figure 16-5 on page 543 shows how the JSF portlet runtime and the JSF servlet runtime interact with the common JSF runtime.

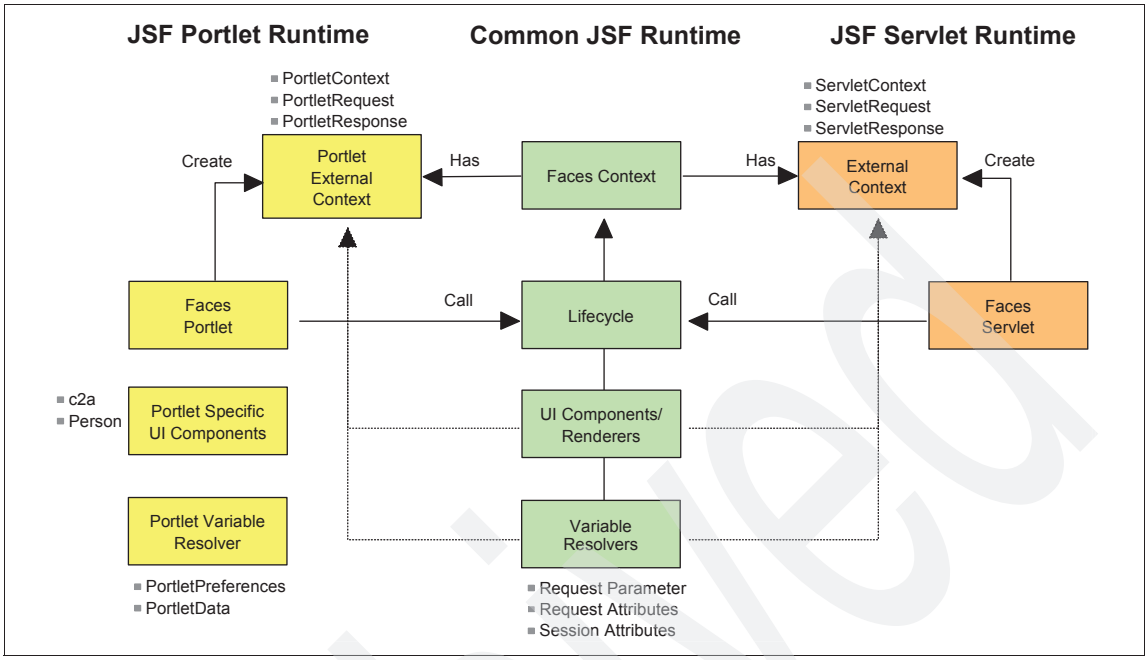


Figure 16-5 How the JSF portlet runtime interacts with the common JSF runtime

The common JSF runtime defines the elements that were covered in the previous sections. The only new element here is the *variable resolver*. The variable resolver determines the value of the first variable in an expression. For example, the standard variable resolver looks up managed beans and handles the predefined variables such as `cookie`, `view` and `session`.

The JSF servlet runtime is based on the `FacesServlet`, and is in charge of calling the JSF life cycle and creating an `ExternalContext`. Through the `ExternalContext`, the JSF application can access the servlet objects, such as `ServletContext`, `ServletRequest` and `ServletResponse`.

The JSF portlet runtime is based on the `FacesPortlet`, that accomplishes essentially the same tasks as the `FacesServlet` does. That is, it is in charge of calling the JSF life cycle and creating a `PortletExternalContext`. Through the `PortletExternalContext`, the JSF portlet can access the portlet objects, such as `PortletContext`, `PortletRequest` and `PortletResponse`.

The JSF portlet runtime also defines some portlet specific UI components, such as `c2a` for cooperation and `person` for people awareness.

Finally, the JSF portlet runtime defines a variable resolver that permits value-binding of portlet objects, such as `PortletPreferences` and `PortletData`.

Archived

16.6.2 Mapping between portlet phases and JSF phases

Figure 16-6 shows the mapping between the portlet an JSF phases. This mapping is showing the portlets methods that are invoked in JSR 168 portlets, but the mapping is analogous in IBM API portlets.

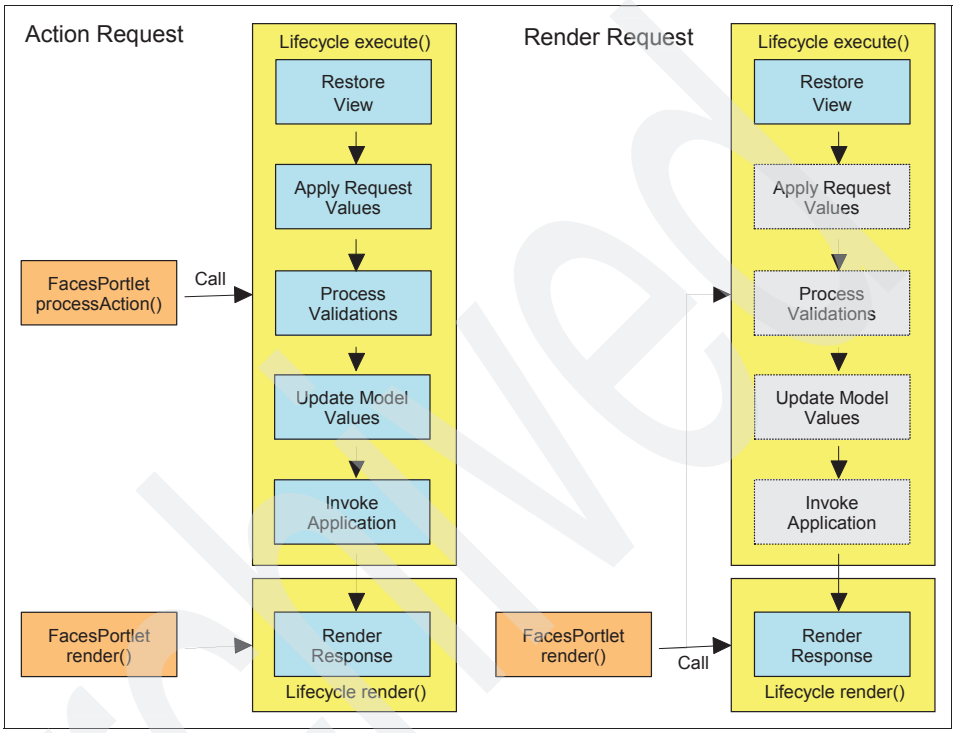


Figure 16-6 Mapping between portlet phases and JSF phases

As we know, portlets have two processing phases: the action phase and the render phase.

If the portlet fired an action, it will send an action request to the server. An example of firing an action is submitting a form or clicking a link. In this case, the WebSphere Portal runtime will call both the action phase and the render phase of the portlet. In the action phase, the `FacesPortlet.processAction` method is invoked and it calls the five first phases of the JSF life cycle. In the render phase, the `FacesPortlet.render` method is invoked and it calls the remaining phase of the JSF life cycle.

There are times when the server receives only a render request, instead of an action request. An example is when you have portlets A and B in the same page and the user interacts with portlet A. While portlet A will process the action and

render again, portlet B will only have to be rendered, because there are no actions associated with it. In this case, the WebSphere Portal runtime will call only the `FacesPortlet.render` method of portlet B. The `FacesPortlet.render` method will call all six JSF life cycle phases, but since there are no parameters associated with the request, most of the life cycle phases will be skipped.

16.6.3 Welcome page and navigation in JSF portlets

The welcome-file list in web.xml is not used by JSF portlets to determine which is the initial page to be displayed. This is done through a parameter, named `com.ibm.faces.portlet.page.`*`mode`*, where *`mode`* is the mode you want to set the initial page to. This means that you will have a different parameter to set the initial page to each mode that the portlet supports.

- In IBM Portlet API, this parameter is a config-param in the concrete portlet, in portlet.xml. Example 16-19 specifies the initial pages to both View and Edit modes of a portlet that uses the IBM Portlet API.

Example 16-19 Configuration of JSF initial pages in IBM Portlet API

```
<config-param>
   <param-name>com.ibm.faces.portlet.page.view</param-name>
   <param-value>/index.jsp</param-value>
</config-param>
<config-param>
   <param-name>com.ibm.faces.portlet.page.edit</param-name>
   <param-value>/html/edit/index.jsp</param-value>
</config-param>
```

- In JSR 168 API, this parameter is a init-param in portlet.xml. Example 16-20 specifies the initial pages to both View and Edit modes of a portlet that uses the JSR 168 API.

Example 16-20 Configuration of JSF initial pages in JSR 168 API

```
<init-param>
   <name>com.ibm.faces.portlet.page.view</name>
   <value>/index.jsp</value>
</init-param>
<init-param>
   <name>com.ibm.faces.portlet.page.edit</name>
   <value>/html/edit/index.jsp</value>
</init-param>
```

Once the initial page is displayed, the navigation between pages follows the page navigation rules defined in faces-config.xml. See 16.2.2, “Defining navigation rules” on page 521 for more details.

Tip: Navigation is defined within a portlet mode. If you want to switch portlet mode, refer to “Changing portlet modes” on page 547. Take care when switching modes during navigation, to avoid the portlet being in a inconsistent state.

16.6.4 Programming guidelines

The following topics provide some programming guidelines when developing JSF portlets.

Accessing the portlet API in JSF actions

You can obtain `PortletRequest`, `PortletResponse` and `PortletContext` objects through the `ExternalContent` object. You have to make the appropriate casting when retrieving these objects, as shown in Example 16-21.

Example 16-21 Obtaining portlet API objects through the ExternalContext

```
PortletContext context =
      (PortletContext)facesContext.getExternalContext().getContext();
```

Table 16-6 shows the methods available at `ExternalContext` and the objects returned for each API.

Table 16-6 Objects returned by each ExternalContext method

ExternalContext method	JSR 168 API	IBM portlet API
getRequest()	PortletRequest (ActionRequest or RenderRequest)	PortletRequest
getResponse()	PortletResponse (ActionResponse or RenderResponse	PortletResponse
getContext()	PortletContext	PortletContext

Value binding for the portlet API

You can use value binding expressions to bind attributes of portlet API objects to UI components.

Suppose you have the following parameter in portlet.xml of a portlet using IBM portlet API:

```
<config-param>
   <param-name>myName</param-name>
   <param-value>John Doe</param-value>
```

```
</config-param>
```

You can bind this value to a `UIInput` component's value, with the following code:

```
<h:inputText id="name" value="#{portletSettings.myName}"/>
```

You can also access the value from JSF code:

```
ValueBinding binding =
facesContext.getApplication().createValueBinding("#{portletSettings.myName}");
System.out.println("Value Binding: myName=" + binding.getValue(facesContext));
```

The value binding of portlet API attributes can be done because when you create a JSF portlet, a portlet variable resolver is defined, according to what we have seen in 16.6.1, "JSF portlet runtime" on page 542. Table 16-7 shows the configuration objects that Portlet variable resolvers enable the use of value binding to, and the corresponding expressions that bind the values of these objects.

Table 16-7 Configuration objects that can be obtained through value binding

Portlet API	Expression	Configuration Object
JSR 168 API	#{portletPreferences.*name*}	PortletPreferences
IBM portlet API	#{portletApplicationSettings.*name*}	PortletApplicationSettings
	#{portletSettings.*name*}	PortletSettings
	#{portletData.*name*}	PortletData

Changing portlet modes

It is possible to change the portlet mode in a JSF action. The following example shows how to do this using the JSR 168 API.

Example 16-22 Changing mode in a JSF portlet using JSR 168 API

```
ActionResponse response = (ActionResponse)
      facesContext.getExternalContext().getResponse();
try {
   response.setPortletMode(PortletMode.VIEW);
} catch (PortletModeException e) {
   // Your exception handling code here
}
```

Example 16-23 on page 548 shows how to change mode using the IBM portlet API.

Archived

Example 16-23 Changing mode in a JSF portlet using IBM portlet API

```
PortletRequest request = (PortletRequest)
       facesContext.getExternalContext().getRequest();
try {
       request.setModeModifier(Portlet.ModeModifier.PREVIOUS);
} catch (AccessDeniedException e) {
   // Your exception handling code here
}
```

16.6.5 Limitations in JSF portlets

There are limitations that exist when running JSF portlet applications:

- The file upload component is not supported (`hx:fileUpload`).
- Components that support the download of binary data are not supported.
  - Image (`hx:graphicImageEx`), when bound to data, as in:

    ```
    <hx:graphicImageEx value="#{myBean.photo}"/>
    ```

  - Link (`hx:outputLinkEx`), when bound to data, as in:

    ```
    <hx:outputLinkEx value="#{myBean.resume}"/>
    ```

  - Media Player (`hx:playerGenericPlayer`, `hx:playerFlash`, `hx:playerMediaPlayer`, `hx:playerRealPlayer`, `hx:playerShockwave`), when bound to data, as in:

    ```
    <hx:playerGenericPlayer value="#{myBean.movie}"/>
    ```

- When adding an Image component to a Faces portlet page, you must specify the URL relative to the document WebContent root folder, rather than relative to the project root folder:

  Wrong:

  ```
  <hx:graphicImageEx value="/.YourPortlet/theme/yourimage.gif"/>
  ```

  Correct:

  ```
  <hx:graphicImageEx value="theme/yourimage.gif"/>
  ```

16.7 Migration

Existing JSF applications can be migrated in order to run in WebSphere Portal. Since JSF is a framework there are many variations to how the application can be built with JSF. The steps in this section can be used as a starting point for the migration effort, but may not cover all of the issues that can be encountered.

Archived

1. Check if there are any file upload component in the application, or if the application uses components that support the download of binary data. These features are not supported in JSF portlets and cannot be migrated (see 16.6.5, “Limitations in JSF portlets” on page 548).
2. Decide to which Portlet API you want to migrate your JSF application. The migration can be done either to the JSR 168 API or to the IBM Portlet API.
3. Create a portlet.xml according to the API you decided to migrate to.
   - For the JSR 168 API, refer to 8.9, “Deployment descriptors” on page 287.
   - For the IBM Portlet API, refer to 4.12.2, “portlet.xml” on page 160.
4. Specify the `FacesPortlet` class. This class will perform the tasks that the `FacesServlet` does in a conventional JSF application.
   - In JSR 168 API, the `FacesPortlet` is declared in portlet.xml and points to the class `com.ibm.faces.webapp.FacesGenericPortlet`. The existing `FacesServlet` in web.xml should be deleted.
   - In IBM Portlet API, `FacesPortlet` is declared in web.xml and points to the class `com.ibm.faces.webapp.WPFacesGenericPortlet`. The existing `FacesServlet` in web.xml should be replaced by the `FacesPortlet`.
5. Specify the initial page to be loaded in each portlet mode, according to 16.6.3, “Welcome page and navigation in JSF portlets” on page 545.
6. Import the correct jar file that contains the JSF Portlet Runtime into the WEB-INF/lib directory:
   - In JSR 168 API, this is the `jsf-portlet.jar`.
   - In IBM Portlet API, this is the `jsf-wp.jar`.
7. Include a `faces-context-factory` in `faces-config.xml`:
   - In JSR 168 API, include this `factory` tag:

```
<factory>
   <faces-context-factory>
      com.ibm.faces.context.PortletFacesContextFactoryImpl
   </faces-context-factory>
</factory>
```

   - In IBM Portlet API, include this `factory` tag:

```
<factory>
   <faces-context-factory>
      com.ibm.faces.context.WPPortletFacesContextFactoryImpl
   </faces-context-factory>
</factory>
```

8. However, in `faces-config.xml`, you need to specify variable and property resolvers to correctly use value binds:

Archived

– In JSR 168 API, include this `application` tag. If you already have an `application` tag in your file, include only the inner tags shown below:

```
<application>
   <variable-resolver>
      com.ibm.faces.databind.SelectItemsVarResolver
   </variable-resolver>
   <variable-resolver>
      com.ibm.faces.application.PortletVariableResolver
   </variable-resolver>
   <property-resolver>
      com.ibm.faces.databind.SelectItemsPropResolver
   </property-resolver>
</application>
```

– In IBM Portlet API, include this `application` tag. If you already have an `application` tag in your file, include only the inner tags shown below:

```
<application>
   <variable-resolver>
      com.ibm.faces.databind.SelectItemsVarResolver
   </variable-resolver>
   <variable-resolver>
      com.ibm.faces.application.WPPortletVariableResolver
   </variable-resolver>
   <property-resolver>
      com.ibm.faces.databind.SelectItemsPropResolver
   </property-resolver>
</application>
```

9. Be careful with pages that use JavaScript code. You will have to namespace all JavaScript calls and references to JSF components. Refer to 17.9, “Implementing a value change event” on page 583 for an example of how to do this.
10. If your existing JSF application already supports multiple locales, remember to update `portlet.xml` to include the supported languages. Refer to 17.10, “Implementing internationalization” on page 587 to see how to do this.
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.
12. Refer to Chapter 8, “JSR 168 API” on page 251 and Chapter 4, “IBM Portlet API” on page 115 for details about each API, so that you can make your application take advantage of the features provided by each one.

Archived

17

JavaServer Faces portlet development

This chapter provides a sample scenario related to JavaServer Faces (JSF) portlet development to illustrate the various features available in Rational Application Developer V6 and Portal Tools.

In this chapter, you will develop JSF portlets in order to perform the following tasks:

- Creating the JSF project and page layout
- Implementing component attributes and validation
- Binding the front end to the Managed Bean
- Invoking the business logic
- Page navigation, validators and value change events
- Internationalization

Note: The portlet application described in this chapter has the following characteristics:

- Portlet API: JSR 168
- Application type: JavaServer Faces

© Copyright IBM Corp. 2005. All rights reserved.

17.1 The calculator application

The first application to be created is a simple calculator. A JavaBean named `CalculatorBean` is included to provide the basic mathematical operations on two integer (long) numbers.

This bean has five properties:

- Operand 1
- Operand 2
- Operation
- Result
- Error message

The bean also has associated getter and setter methods for all of its properties. In addition, the `calculate()` method calculates the result of one of the following operations:

- **A**dd
- **S**ubtract
- **M**ultiply
- **D**ivide

The result is stored in the first number (operand 1) for subsequent operations. If a division does not result in an integer, an exception is thrown. Finally, a `toString()` method displays the numbers and the operation.

17.2 Creating the project

You will create the calculator project as a Portlet Project (JSR168) project. Follow these steps to create it:

1. In the menu, select **File** → **New** → **Project**.
2. In the New Project window, select the wizard for Portlet Project (JSR 168).
3. Click **Next >**.
4. If the Confirm Enablement window appears, asking if you want to enable the portal development role, click **OK**.
5. In the Portlet project page, type the name of the project `Calculator`.
6. Check the **Create a portlet** check box, if it is not already checked.
   Note: Typically, you do not need to create a portlet when you import a portlet WAR file into the project.
7. In the WebSphere Portal version box, select **5.1**. Click **Next >**.

Archived

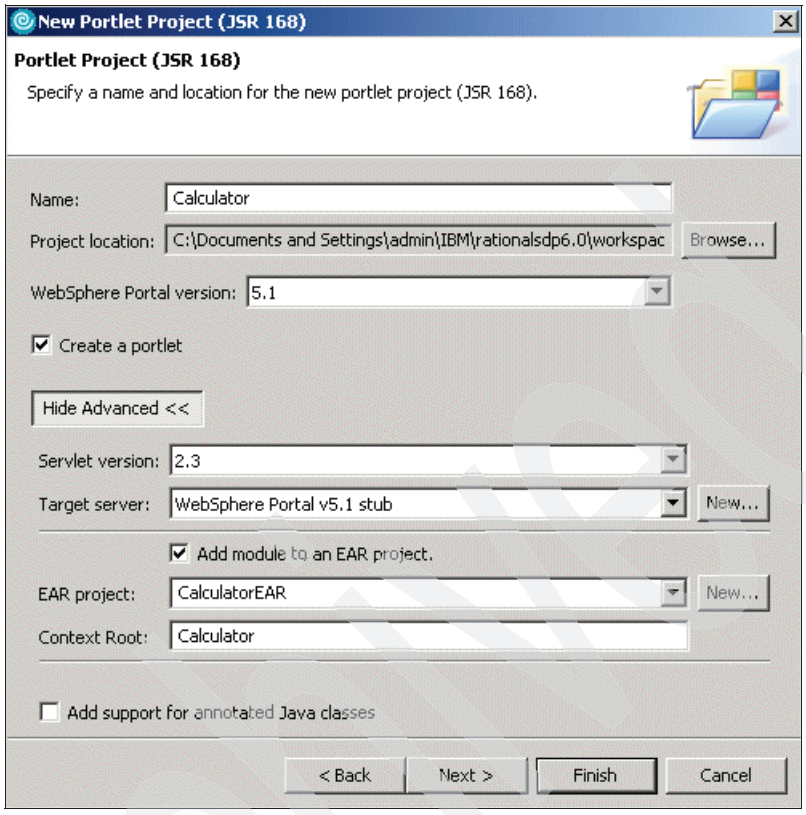


Figure 17-1 Portlet project (JSR 168)

8. In the Portlet type page, select **Faces portlet (JSR 168)**. Click **Next >**.

Archived

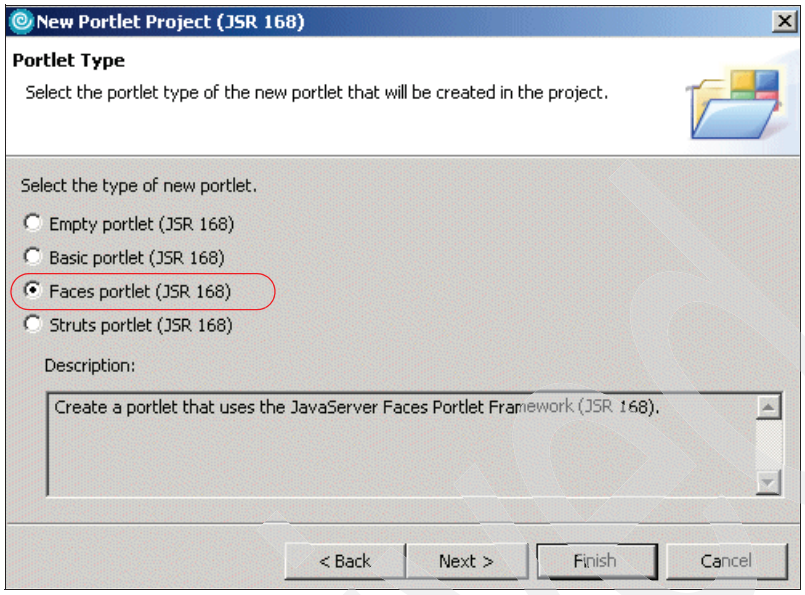


Figure 17-2 Selecting the portlet type

9. Click **Next** > again in the Features page.

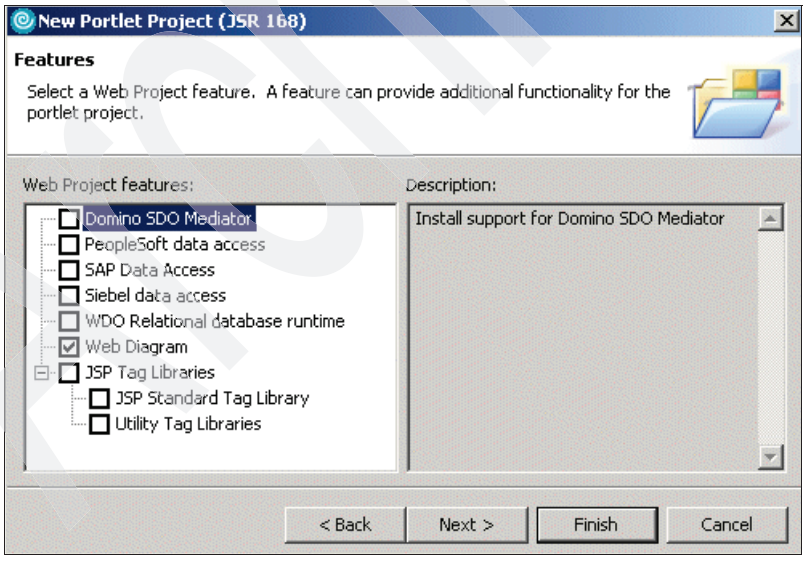


Figure 17-3 Features

10. In the Portlet settings page, review the options provided and leave all settings unchanged. Click **Next >**.
11. In the Miscellaneous page, select only the **View mode** for the portlet. Enter `/calculate.jsp` as the name of the initial page.

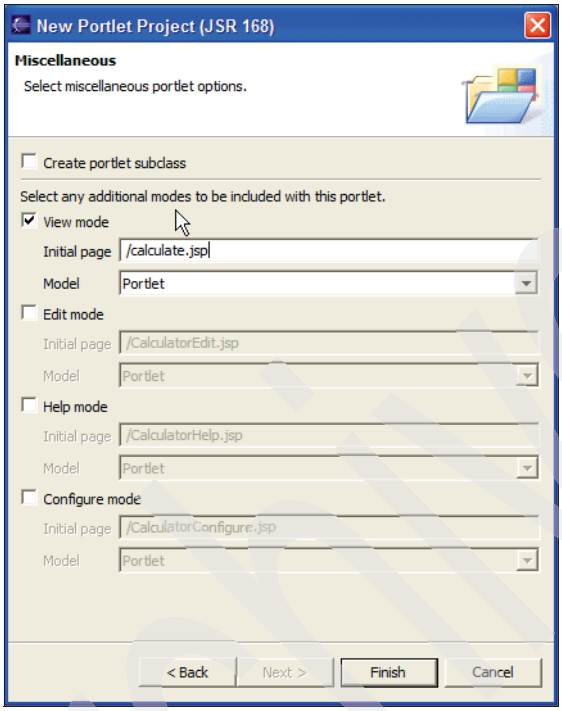


Figure 17-4 Selection of portlet modes

12. Click **Finish** to generate the portlet.
13. If the Confirm Perspective Switch window appears, click **Yes**.

17.2.1 Inspecting the JSF portlet project

Now that the JSF portlet project has been created, let's take a look at what Rational Application Developer has generated:

1. Under Java Resources/JavaSource, you will find a package named calculator.nl with two resource bundles, and a package named pagecode, with two files: Calculate.java and PageCodeBase.java (Figure 17-5 on page 556).

Archived

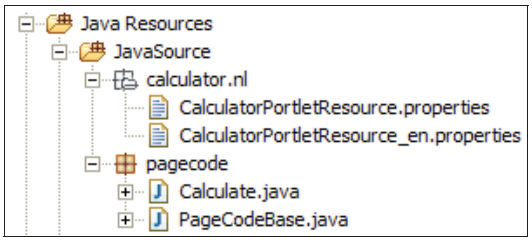


Figure 17-5 Java resources created in the project

a. Open Calculate.java. The class is empty for now. It is a subclass of PageCodeBase.

b. Open PageCodeBase. This class contains shared code for all JSF pages:

- Access to `FacesContext`, `requestScope` and so forth.
- `gotoPage` - switch to another JSF page
- `findComponent` - find any JSF component
- `findComponentInRoot` - find a JSF component under the root component
- `resolveExpression` - resolves a JSF reference: #{xxxx}
- `log` - simplified test output to `System.out.println`

2. There is a new style sheet, stylesheet.css, in the WebContent/theme folder (Figure 17-6 on page 557). Open the file; it has many predefined styles.
3. In the WEB-INF/lib folder you find the JSF runtime JAR files (Figure 17-6 on page 557):
   - jsf-api and jsf-impl are the basic JSF JAR files from Sun.
   - jsf-ibm has the IBM extensions.
   - jsf-portlet has the JSF portlet runtime for JSR 168 API.
   - jstl and jstl_el are JSP standard tag libraries.
   - commons-xxx are common utilities.
   - jaxen-full, saxpath and standard are JAR files needed by the JSP standard tag libraries.

Archived

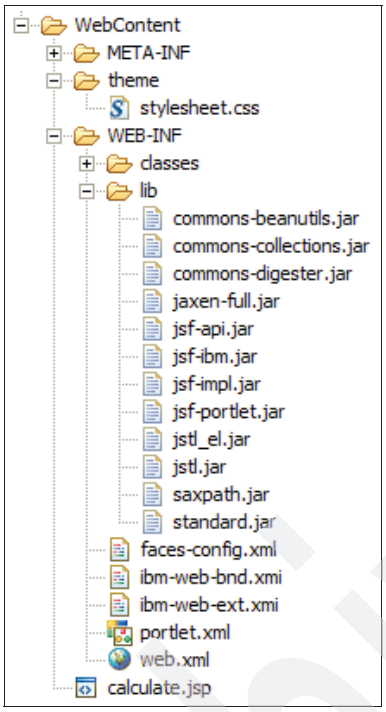


Figure 17-6 Resources under WebContent

4. In the WEB-INF you find the JSF configuration file, faces-config.xml. Open this file:
   - It contains the definition of a `FacesContextFactory` specific to JSR 168: `com.ibm.faces.context.PortletFacesContextFactoryImpl`. In an IBM portlet API, this factory would be:

     ```
     com.ibm.faces.context.WPPortletFacesContextFactoryImpl
     ```

   - It contains a life cycle listener and already one managed bean, `pc_Calculate`, which is the Java code for the JSF page.
   - It contains a `PropertyResolver` and two `VariableResolver`. One of the variable resolvers is specific to JSR 168:

     ```
     com.ibm.faces.application.PortletVariableResolver
     ```

     In IBM portlet API this variable resolver would be:

     ```
     com.ibm.faces.application.WPPortletVariableResolver
     ```

5. Open the deployment descriptor, web.xml. It contains a servlet named `JS Resource Servlet`. This servlet is an IBM extension for generated JavaScript functions.

Archived

6. Open the portlet deployment descriptor, portlet.xml.
   - It contains the `FacesPortlet` definition. The portlet name is `Calculator` and it points to the class `com.ibm.faces.webapp.FacesGenericPortlet`.
   - The portlet has an init parameter pointing to the initial page of View mode:

```
<init-param>
   <name>com.ibm.faces.portlet.page.view</name>
   <value>/calculate.jsp</value>
</init-param>
```

17.3 Creating the page layout

Now you will create the basic JSP layout of the calculate.jsp.

1. In the Design view, select the text **Place content here.** and replace it with `JSF Calculator`.
2. Select the text **JSF Calculator**, and on the top menu select **Insert** → **Paragraph** → **Heading 1**.

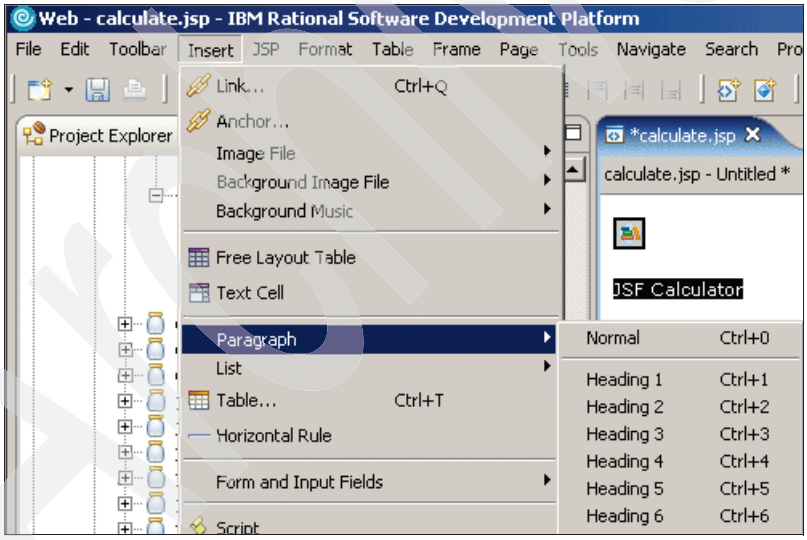


Figure 17-7 Portlet heading

3. Select the **Output** component in the Faces Components palette. Drag the cursor under the heading and drop the component.

Archived

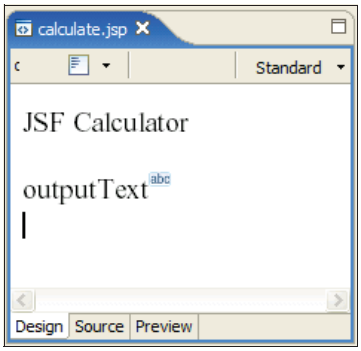


Figure 17-8 Creating the page layout for the calculator

4. Select the **outputText** component in the Design view.
   a. In the Properties view, change the Id to `comment`.
   b. Change the Value to `Created using Application Developer`.
   c. Enter style properties: `font-size: 18; font-weight: bold`.

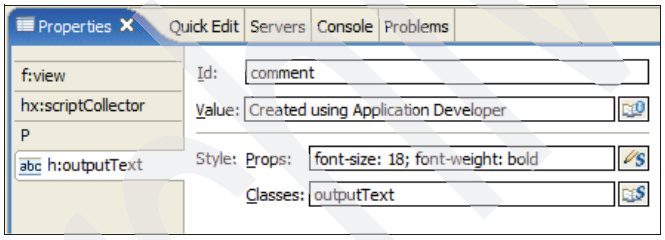


Figure 17-9 The comment component properties

5. Position the cursor behind the output text and press **Enter** twice to add `<BR>`s.
6. From the palette, select the **Display Errors** component and drop it under the breaks. This is for validation error messages.
7. Select the new component and notice in the Properties view the `messages` style class (see Figure 17-10 on page 560).

Archived

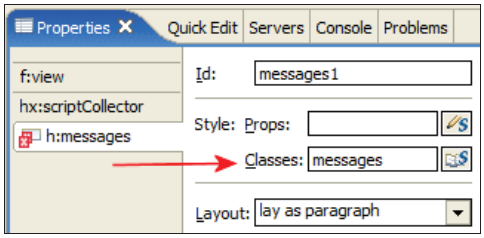


Figure 17-10 The messages component properties

8. Open the file stylesheet.css.

   a. In the Styles view (Figure 17-11), find the .messages class and open it (double-click it).

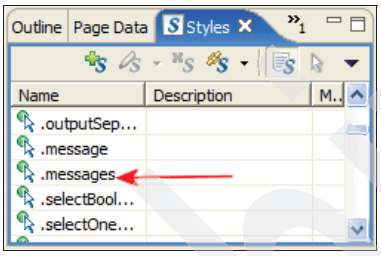


Figure 17-11 The Styles view

   b. In the Add Style --- .messages window, select **Font** and set the color to `Red`. Click **OK** and the code is inserted into the `stylesheet.css` file:

```
.messages {
   color: red
}
```

   c. Change the `.message` entry in the same way. This class will be used for error messages attached to one field. Save and close the stylesheet.css. The `{Error Messages}` component is now red.

9. Insert a table with five rows and two columns under the error messages field. Select **Insert** → **Table**, change Rows to `5`, Border width to `0`, and Padding inside cells to `3`.

Archived

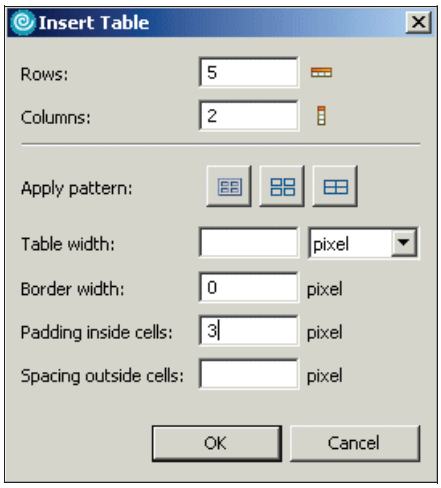


Figure 17-12 Insert a table

10. Click **OK**. Your page now should look similar to Figure 17-13.

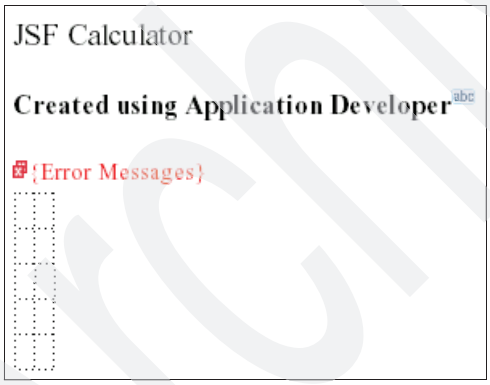


Figure 17-13 The page layout after inserting a table

11. Select **File** → **Save All**.

12. Open the `Calculate.java` file by selecting **Edit Page Code** from the context menu inside the JSP in Page Designer (right-click). You can also open the Java file directly, if you prefer. Notice the code that has been generated, for example:

```
protected HtmlOutputText comment;
protected HtmlOutputText getComment() {
   if (comment == null) {
      comment = (HtmlOutputText) findComponentInRoot("comment");
```

Archived

```
    }
    return comment;
}
```

Notice also that each component is defined with a lazy getter method; therefore you will never use the variable in your own methods, but always in the associated getter method.

13. Fill in the recently created table (Figure 17-14):

    a. Drop Output components into column 1, rows 1, 2, 3, and 5.
    b. Drop Input components into column 2, rows 1 and 3.
    c. Drop a Combo Box component into column 2 row 2.
    d. Drop a Command - Button component into column 2 row 4.
    e. Drop an Output component into column 2 row 5.

    The table should now look as illustrated in Figure 17-14.

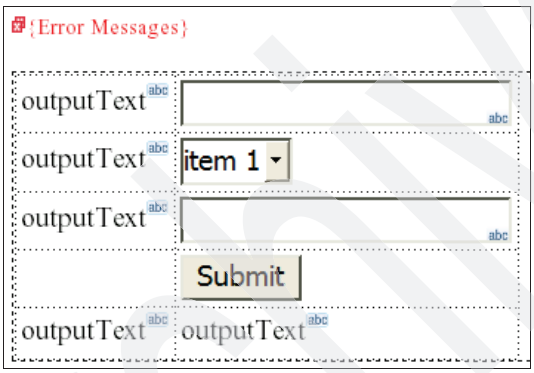


Figure 17-14 Inserting JSF components into the table

14. Change the Output components in column 1. Select each component and in the Properties view change the Value field (Figure 17-15 on page 563):

    a. Column 1 row 1: `Number 1 (11-888):`
    b. Column 1 row 2: `Operation:`
    c. Column 1 row 3: `Number 2 (0-19, odd):`
    d. Column 1 row 5: `Result:`

    The table should now look as illustrated in Figure 17-15 on page 563.

Archived

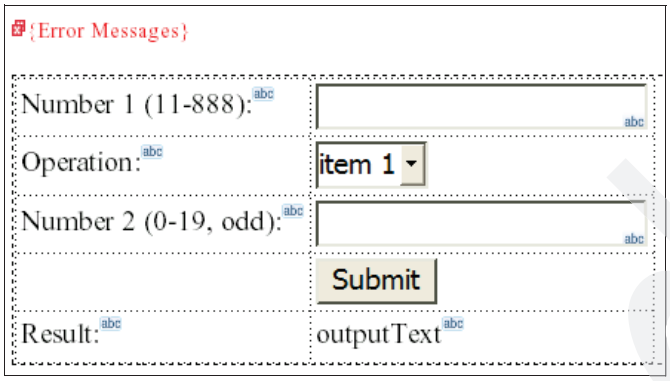


Figure 17-15 Changing the output components value property

15. Select **File** → **Save All**.

17.4 Implementing component attributes and validation

Execute the following steps:

1. Select the first Input component. In the Properties view, h:inputText tab, do the following (Figure 17-16 on page 564):
   a. Change the id to `number1`.
   b. Change the width to 12.
   c. In the Format combo box, select **Number**.
   d. Select **Decimal** as the type.
   e. Check the **Integer only** checkbox.

Archived

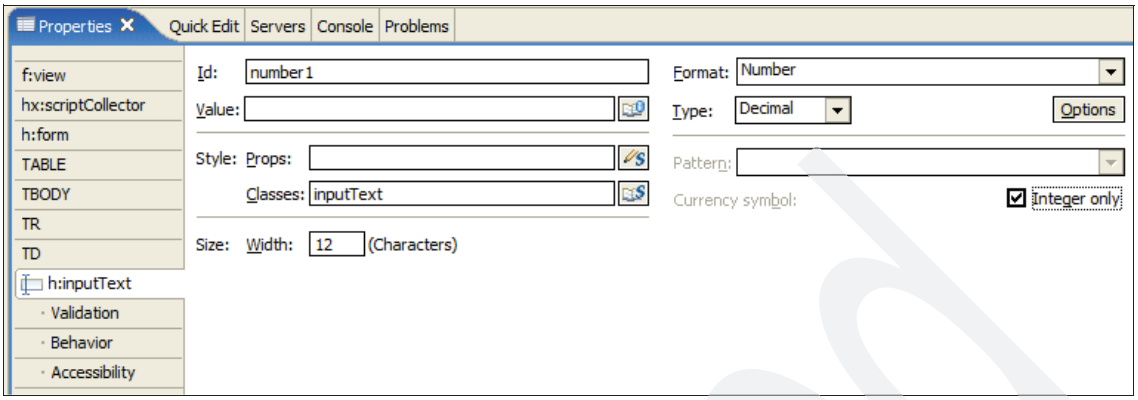


Figure 17-16 Properties for number1 component

2. Change to the Validation tab:
   a. Check the **Value is required** checkbox.
   b. Set Minimum and Maximum to 11 and 888.
   c. Check the **Display validation error messages in a error message control** checkbox.
   d. An error message field is added to the right of the input field. Select it and on the Properties view, if not already there, enter `message` in the Styles: Classes property.

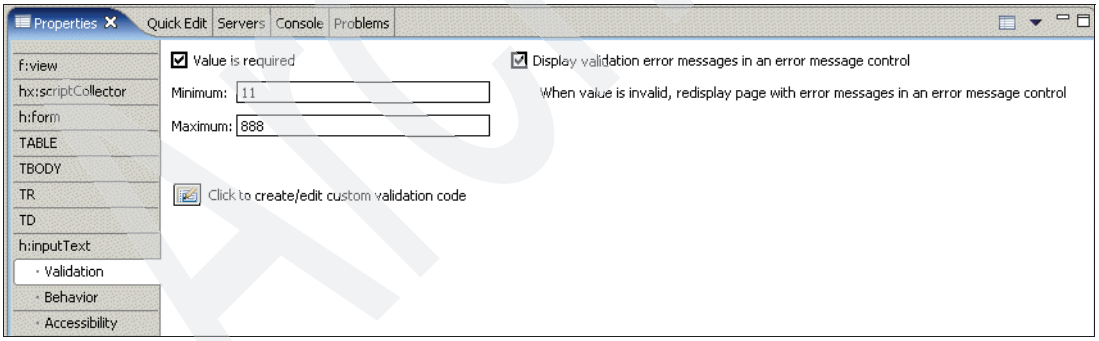


Figure 17-17 Validation properties

3. Select the **Input** component again and change to the Behavior tab:
   a. Check **Auto-advance to next field**.
   b. Enter 3 in the field After user types ... characters.

Archived

c. Click the **All Attributes** icon to see all the values you have entered. You can also enter the values directly there.

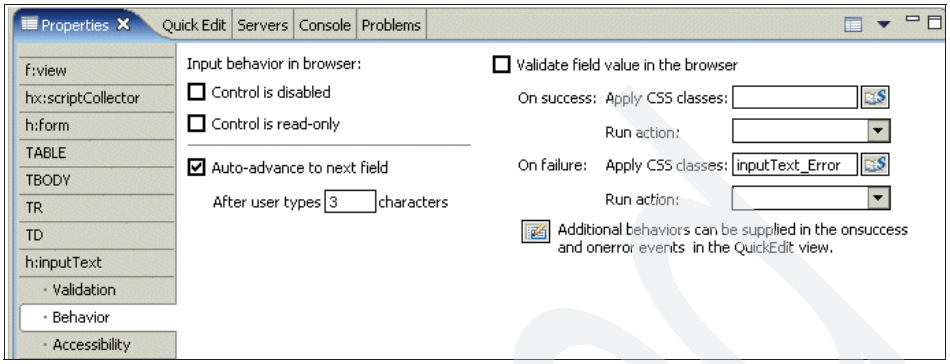


Figure 17-18 Behavior properties

4. Similarly, select the second Input component. In the Properties view, do the following:
   a. h:inputText tab: id=`number2`, width=12, Format=**Number**, Type=**Decimal** and **Integer only** checkbox.
   b. Validation tab: **required**, minimum= 0, maximum=19.
   c. Behavior tab: **Auto-advance to next field** and enter 2 in the field *After user types ... characters*.
5. Select the **Combo Box** component (operation column 2). In the Properties view, h:selectOneMenu tab:
   a. Change id to `operation`.
   b. In the *Add a choice for each item in the combo box dropdown* table (Figure 17-19 on page 566):
      i. Click the **Add Choice** button four times.
      ii. Change the generated names to `add`, `subtract`, `multiply`, and `divide`.
      iii. Change the generated values to `Add`, `Subtract`, `Multiply` and `Divide`.

Archive

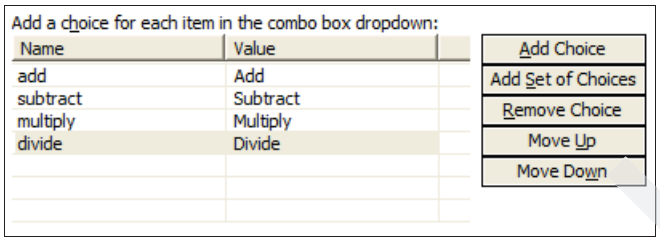


Figure 17-19 Setting combo box choices (operations)

6. Select the **Submit** button. In the Properties view:
   a. hx:commandExButton tab: enter `calc` as the id.
   b. In the Display options tab: Replace `Submit` for `Calculate` as the Button label.
7. Select the **Result:** Output component (column 1). In the Properties view, set the Style: Props to:

   ```
   color: blue; font-size: 18; font-weight: bold
   ```

8. Select the last Output component (column 2). In the Properties view, set the id to `result` and the Style: Props to:

   ```
   color: blue; font-size: 18; font-weight: bold
   ```

9. The page design should be similar to Figure 17-20.

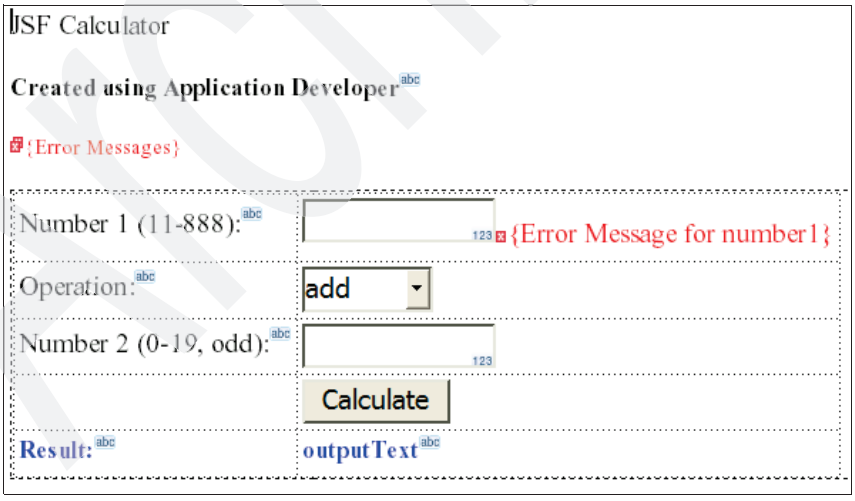


Figure 17-20 Basic design for the calculator

Archived

10. Now, save the JSF page and take a look at some of the tags in the source tab. For example:

- Tag libraries:

```
<%@taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@taglib uri="http://java.sun.com/portlet" prefix="portlet"%>
<%@taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@taglib uri="http://www.ibm.com/jsf/html_extended" prefix="hx"%>
```

- JSF style sheet:

```
<LINK rel="stylesheet" type="text/css"
   href='<%= renderResponse.encodeURL(renderRequest.getContextPath() +
   "/theme/stylesheet.css") %>'
   title="Style">
```

- The whole page is contained in a <f:view> tag. Around the table is a <f:form> tag.

- Input field with error message field:

```
<h:inputText styleClass="inputText" id="number1" size="12"
          required="true" maxlength="3">
   <f:convertNumber integerOnly="true"/>
   <f:validateLongRange minimum="11" maximum="888"></f:validateLongRange>
</h:inputText>
<h:message for="number1"></h:message>
```

- Combo box:

```
<h:selectOneMenu styleClass="selectOneMenu" id="operation">
   <f:selectItem itemValue="Add" itemLabel="add" />
   <f:selectItem itemValue="Subtract" itemLabel="subtract" />
   <f:selectItem itemValue="Multiply" itemLabel="multiply" />
   <f:selectItem itemValue="Divide" itemLabel="divide" />
</h:selectOneMenu>
```

- Submit button (uses an IBM extension tag):

```
<hx:commandExButton type="submit" value="Calculate"
      styleClass="commandExButton" id="calc"></hx:commandExButton>
```

17.4.1 Testing the validation

At this point, you can test your validation definitions:

1. Right-click the **Calculator** project.
2. Select **Run** → **Run on Server**.
3. In the Define a new server window, select **Manually define a server**.
4. Select the server type **WebSphere Portal V5.1 Test Environment**.

Archived

5. Check the **Set server as project default** check box.
6. Click **Finish**.
7. The calculator application appears in the browser. Leave the fields empty and click **Calculate**.
8. You get two error messages in the common error message field, and one message in the error message field of number 1 (Figure 17-21).

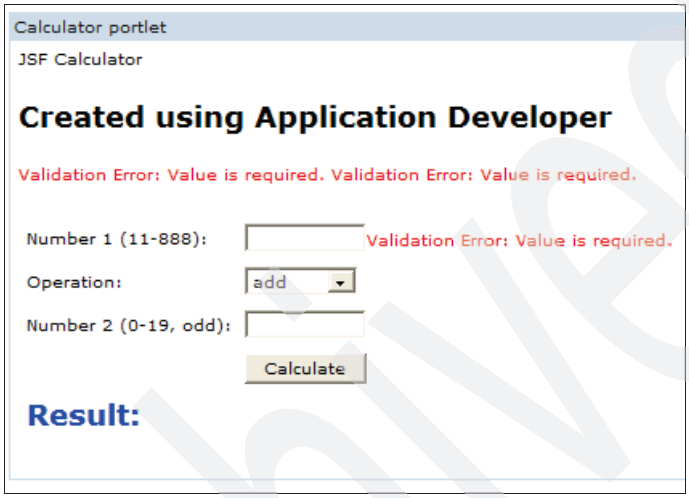


Figure 17-21 Testing the input validation

9. Enter 9 and 2 for the two numbers. You should get a validation error for the first number.
10. Enter an invalid value for the second number and you get an error message as well.
11. Enter valid values and no error messages should be displayed.

Note: The second number is only validated for the range. Standard validation does not check for odd numbers. You will implement this in 17.8, “Implementing a validator” on page 580.

17.5 Binding the front end to the calculator

With the UI design completed, you can now hook it up to the `CalculatorBean` to perform the operations and display the result. Execute the following steps:

1. Import the `CalculatorBean`.

Archived

a. Create a Java package named `itso.jsf.calculator`.

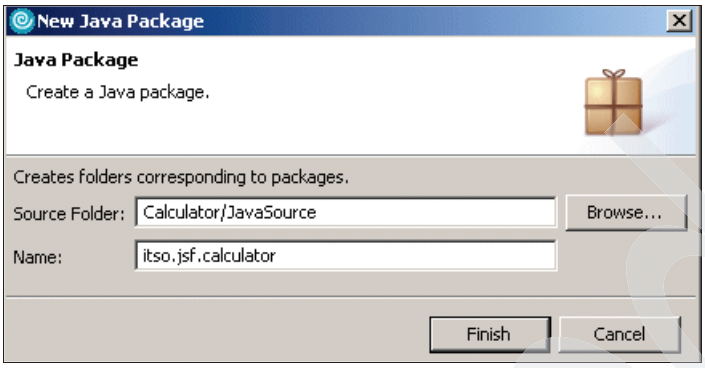


Figure 17-22 Creating a package

b. Select the package and in the context menu, select **Import...**.

c. Select **File System**, locate and select the file **CalculatorBean.java**.

d. Click **Finish**.

2. Open the calculate.jsp page in Page Designer and go to the Page Data view. In the context menu, select **New** → **JavaBean** (Figure 17-23).

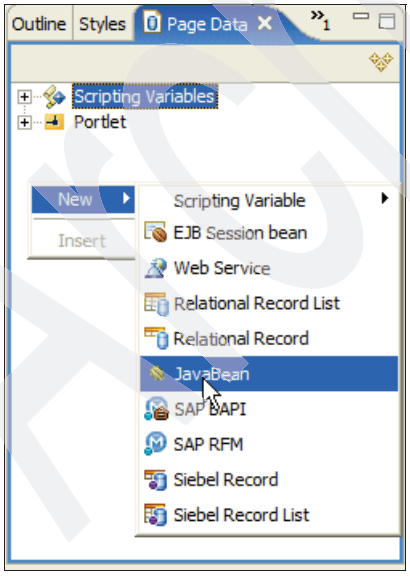


Figure 17-23 Creating a new JavaBean in the page

3. In the Add JavaBean dialog (Figure 17-24 on page 570):

Archived

a. Enter `calculator` as the name.

b. For the class click the **Browse** icon, then enter `Calcu` to locate the `CalculatorBean`. Click **OK**.

c. Select **Make this JavaBean reusable** (this will create a new Managed Bean in faces-config.xml) and select **session** for the scope.

d. Click **Initialize Properties...**.

   i. In the dialog, click **Add**.

   ii. In the name column, select the **operation** property.

   iii. In the value column, enter `Subtract` and press **Enter**.

   iv. Click **OK**. When you run the code, the `subtract` operation will be preselected.

e. Click **Finish** so the bean can be added to the Page Data view.

f. Save the JSP.

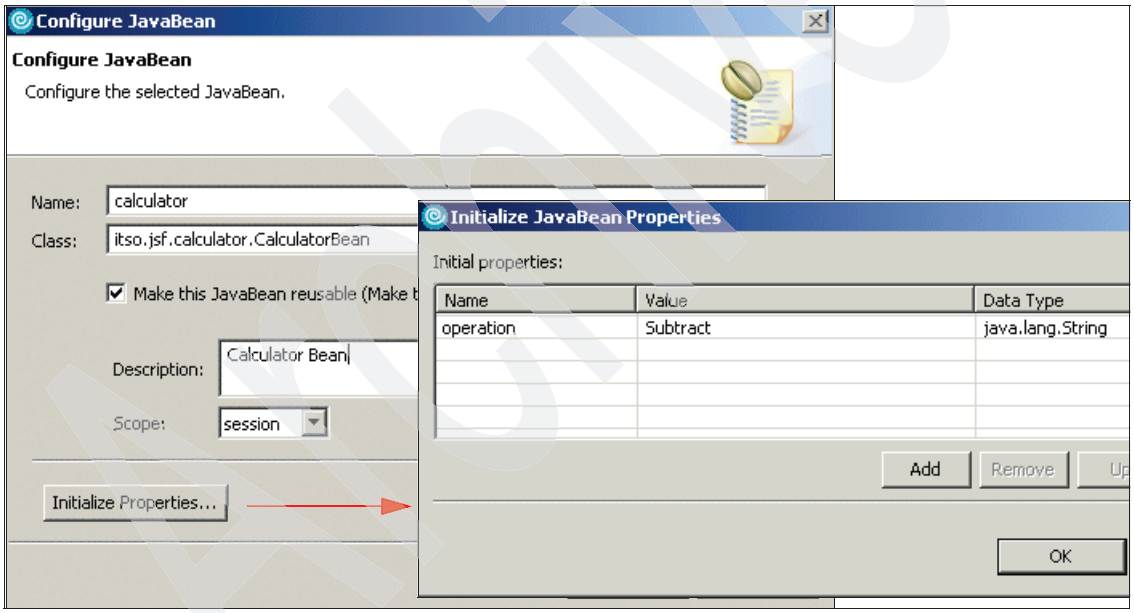


Figure 17-24 Defining the JavaBean

4. Open the faces-config.xml file and you will see the new managed bean.

5. The `calculator` bean is also added to the Calculate.java code with a `getCalculator` method.

6. Select the first input field (**number1**).

a. In the Properties view, for the Value field, click the icon and select the `number1` property of the `calculator` bean.

b. Notice the generated binding: `#{pc_Calculate.calculator.number1}` in the value field.

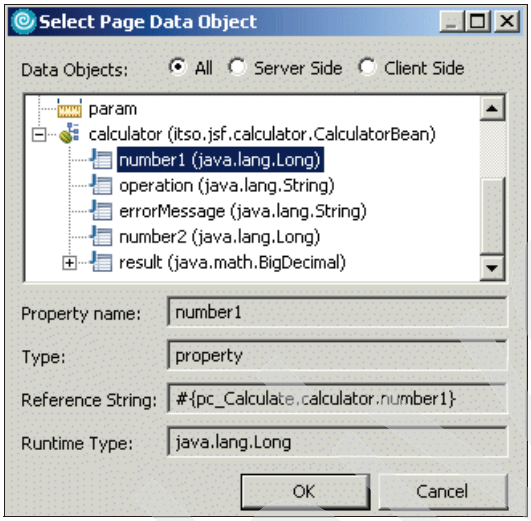


Figure 17-25 Page data objects

7. Repeat this for the number2 input field and bind it to the `number2` property.
8. Expand the `calculator` bean in the Page Data view. Select the **operation** property and drag/drop it onto the combo box. This has the same effect. Select the **result** property and drag/drop it onto the output field in column 2.
9. Save the file.
10. The property names appear in the Design view for the input and output components, as shown in Figure 17-26.

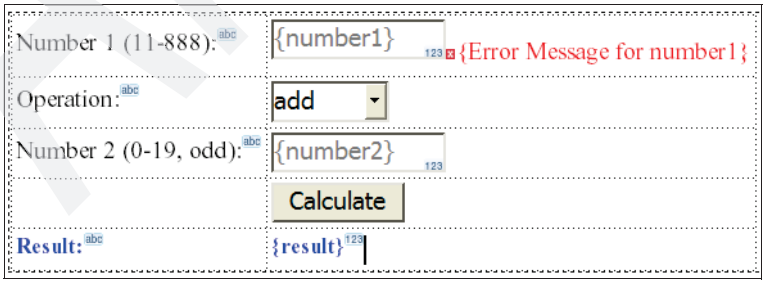


Figure 17-26 Binding properties to the JSF components

Archived

17.5.1 Testing the binding

The front end is now coupled to the `calculator` bean and therefore you can now test the binding. For example:

1. In the Servers view, select the **WebSphere Portal V5.1 Test Environment @ localhost** and from the context menu select **Restart Project** → **CalculatorEAR**.

   Note: You have to do this when the configuration file changes.

2. Run the `Calculator` project.

   a. Notice that no values are displayed in the number fields and `subtract` is displayed for the operation.

   b. These values come from the `calculator` bean. Enter some values and click **Calculate**.

   c. See that the values are assigned to the bean by watching the test output in the Console view.

17.6 Invoking the business logic of the calculator

To actually use the calculator, you have to invoke the `calculate` method of the bean when **Calculate** is clicked.

1. In the Page Designer, select the **Calculate** button. In the Quick Edit view, select **Command** in the left pane. Click in the right pane and this sample code appears:

```
// Type Java code that runs when the component is clicked
// TODO: Return outcome that corresponds to a navigation rule
return "";
```

2. Replace the generated action code with the code shown in Example 17-1.

Example 17-1 Action code to invoke the calculate method

```
log("CalculateAction start");
try {
   getCalculator().calculate();
} catch (Exception e) {
   log("Calculator-Exception: "+e.getMessage());
   if (getCalculator().getErrorMessage() == null)
      getCalculator().setErrorMessage("Exception: "+e.getMessage());
} finally {
   log("CalculateAction end");
}
return null;
```

Archived

3. Save the calculate.jsp. Open the Calculate.java file and notice that the action code has been added in the `doCalcAction` method. You can also make further changes either in the Quick Edit view or in the Java class.
4. Select the **Calculate** button. In the Properties view, click the **Show All Attributes** icon ( ). You can see that the `action` is set to `#{pc_Calculate.doCalcAction}`. This is the link to the Java code.
5. Rerun the project (Figure 17-27). Perform some calculations. For example, try to multiply 123 by 14.

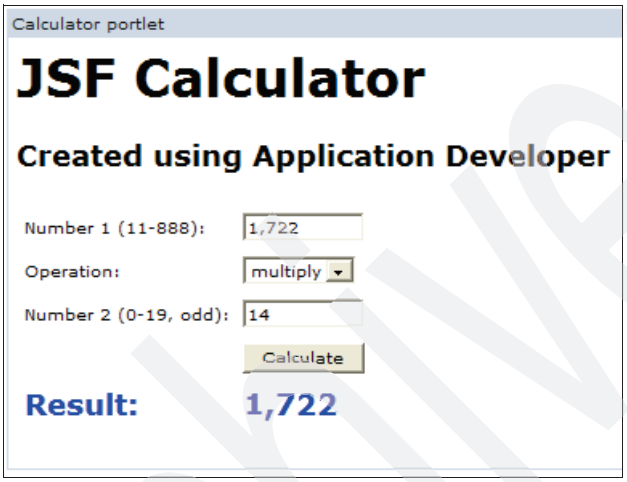


Figure 17-27 The calculator application showing the results

6. Try the divide operation with values that result in a non-integer result. Notice the exception in the Console. The result is still calculated.

   Note: You will implement an error page to display the error message in 17.6.1, “Implementing an error page” on page 573.

17.6.1 Implementing an error page

The exception error can be reported back on the same page. However, to illustrate page switching, in this section you will implement an error page.

1. Under WebContent, right-click and select **New** → **Faces JSP file**. Enter `error.jsp` as the name and click **Finish**.

Archived

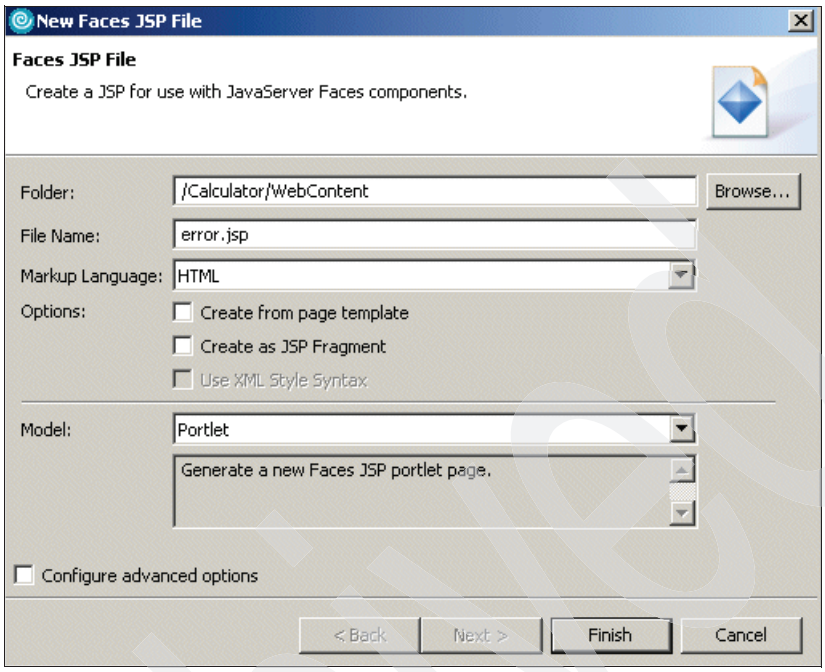


Figure 17-28 Faces JSP file

2. Replace the *Place content here* content with a heading 1, `JSF Calculator Error`. See 17.3, “Creating the page layout” on page 558.
3. In the Page Data view, right-click and select **New** → **JavaBean**.
4. In the dialog, select **Add existing reusable JavaBean** (Figure 17-29 on page 575).
5. Select the **calculator** bean and click **Finish**.

Archive

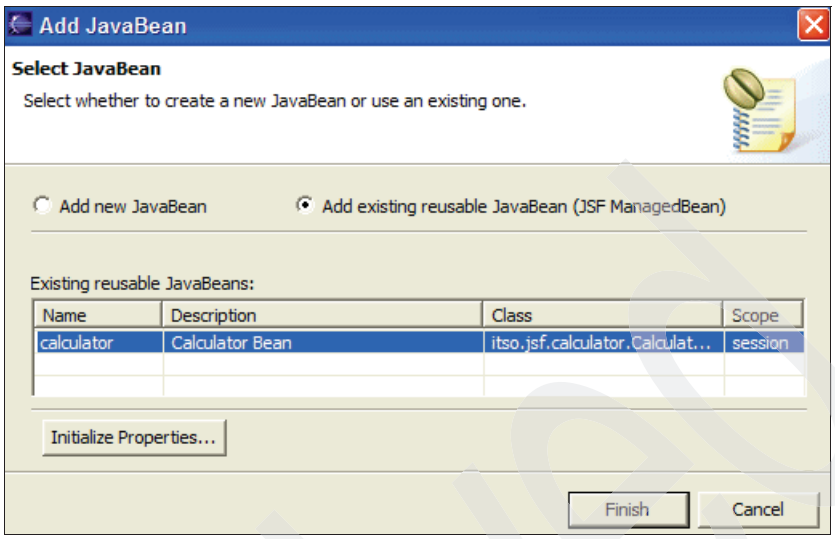


Figure 17-29 Adding a reusable JavaBean

6. Expand the `calculator` bean.
7. Select the **errorMessage** property and drag/drop it under the heading.
8. In the Insert JavaBean dialog (Figure 17-30 on page 576), select **Displaying data (read-only)**.
9. If needed, change the label to `Error message:` and click **Finish**.

   Note: This creates a table with a label (`Error message:`) and an output text bound to the `calculatorBean.errorMessage` property, plus a generic Error Messages component.

Archive

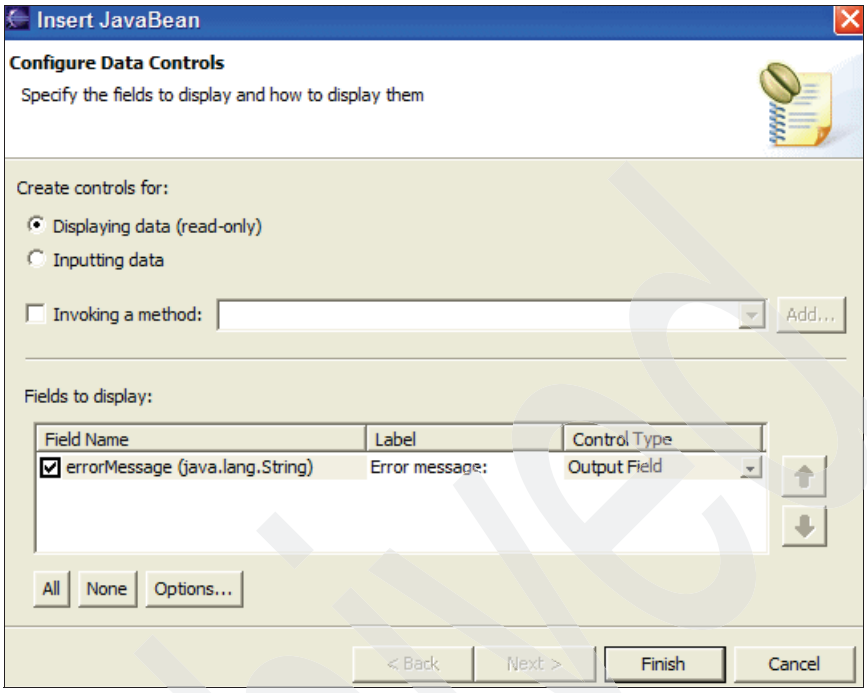


Figure 17-30 Binding the error message from the bean to the error JSP

10. From the palette, add a `Command - Button` at the bottom.
11. Change the label in the Properties view (Display options tab) to `Back`.
12. Save the error.jsp (Figure 17-31).
13. Notice the matching `Error.java` code in the `pagecode` package.

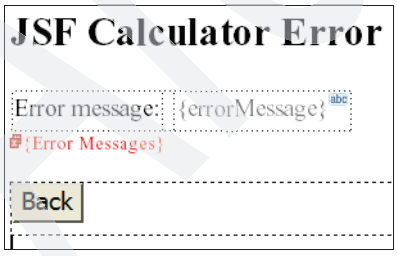


Figure 17-31 The error page

17.7 Implementing page navigation

In this section, you will implement the navigation between the calculate.jsp and the error.jsp. Navigation rules are stored in the configuration file. There are local rules (for one JSF page) and global rules (for all JSF pages).

1. In the error.jsp, select the **Back** button.
2. In the Properties view, hx:commandExButton tab, click **Add Rule** to add a navigation rule.

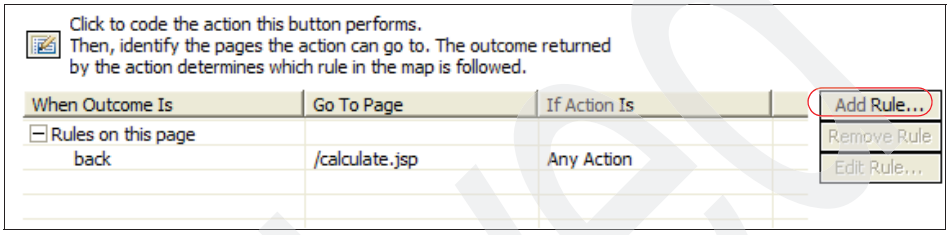


Figure 17-32 Adding a navigation rule

3. Select the **calculate.jsp** page, enter `back` as the outcome and leave other fields unchanged (Figure 17-33).
4. Click **OK** to add the rule.

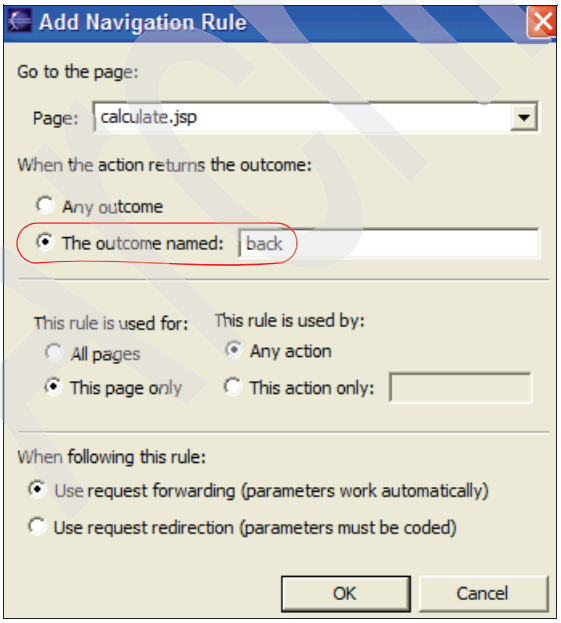


Figure 17-33 Adding a navigation rule

Archived

5. Select the **Back** button and in the Properties view, click the **All Attributes** icon ( ).
6. Enter `back` in the action attribute.

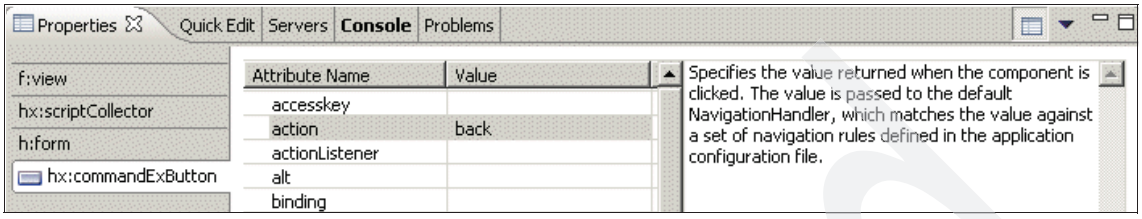


Figure 17-34 Action attribute

Note: The symbolic alias name of the action can be entered directly, or in the action logic. You will enter the action alias in the logic for the calculate.jsp.

7. Save the file.
8. In the calculate.jsp, select the **Calculate** button.
9. In the Properties view, hx:commandExButton tab, add two new rules:
   a. `calculate`, which stays on the page.
   b. `error`, which goes to the error page.

   Note: If needed, click **All Attributes** again to add new rules.
10. Define the `error` rule as a global rule, by selecting the **All pages** option.
11. For this case, a global rule is not really necessary, but it illustrates the concept of using global rules. Figure 17-35 illustrates the entered rules.

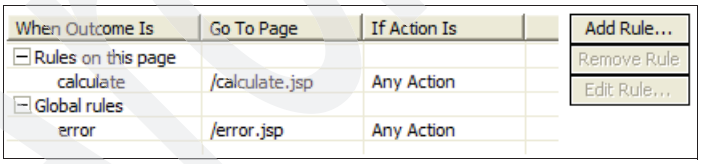


Figure 17-35 Navigation rules defined for the calculate JSP

12. Open the faces-config.xml file and locate the code that was added for the navigation rules.

Example 17-2 Navigation rules in faces-config.xml

```
<navigation-rule>
<from-view-id>/error.jsp</from-view-id>
   <navigation-case>
      <from-outcome>back</from-outcome>
```

```
      <to-view-id>/calculate.jsp</to-view-id>
   </navigation-case>
</navigation-rule>
<navigation-rule>
   <from-view-id>/calculate.jsp</from-view-id>
   <navigation-case>
      <from-outcome>calculate</from-outcome>
      <to-view-id>/calculate.jsp</to-view-id>
   </navigation-case>
</navigation-rule>
<navigation-rule>
   <navigation-case>
      <from-outcome>error</from-outcome>
      <to-view-id>/error.jsp</to-view-id>
   </navigation-case>
</navigation-rule>
```

13. Next, change the action code for the `Calculate` button as follows:

    a. In case of an exception, go to the error page.

    b. Otherwise, stay on the calculate page.

    This process is accomplished by returning one of the two symbolic names, `calculate` or `error` as highlighted in Example 17-3.

    Note: You can make the changes in the Quick Edit view or directly in the Calculate.java file.

Example 17-3 Action code for the Calculate button changed to reflect new rules

```
log("CalculateAction start");
try {
   getCalculator().calculate();
   return "calculate"; // stay on the calculate page
} catch (Exception e) {
   log("Calculator-Exception: "+e.getMessage());
   if (getCalculator().getErrorMessage() == null)
      getCalculator().setErrorMessage("Exception: "+e.getMessage());
   return "error"; // go to the error page
} finally {
   log("CalculateAction end");
}
return null; // remove the existing return
```

> **Note:** Instead of returning the `calculate` String, you can return `null` or an empty string and the `calculate.jsp` is redisplayed.

14. Save all files.

15. Run the code again, then try a division that produces the exception. The error page is displayed (Figure 17-36). Click **Back** to get back to the calculator.

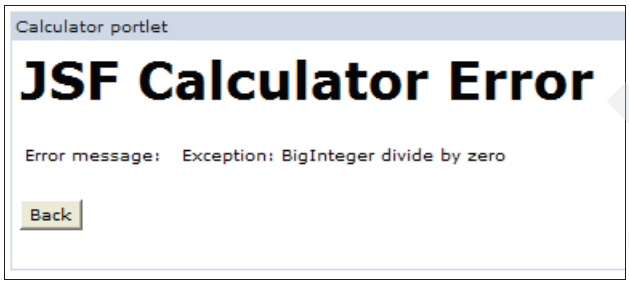


Figure 17-36 Error page showing the exception

17.8 Implementing a validator

We want to restrict the second number to odd values. Since there is no standard validator for this, you will need to write your own validator. In addition, you will also need to do the following:

- Register your validator in the configuration file.
- Link your validator to the `number2` field so it can be checked.

Execute the following steps:

1. Right-click the `itso.jsf.calculator` package
   a. Select **New** → **Class**.
   b. Enter `OddValidator` as name.
   c. Click **Add** for Interfaces.
   d. Locate the `javax.faces.validator.Validator` interface and click **OK**.
   e. Select **Inherited abstract methods**. Click **Finish**.

Archived

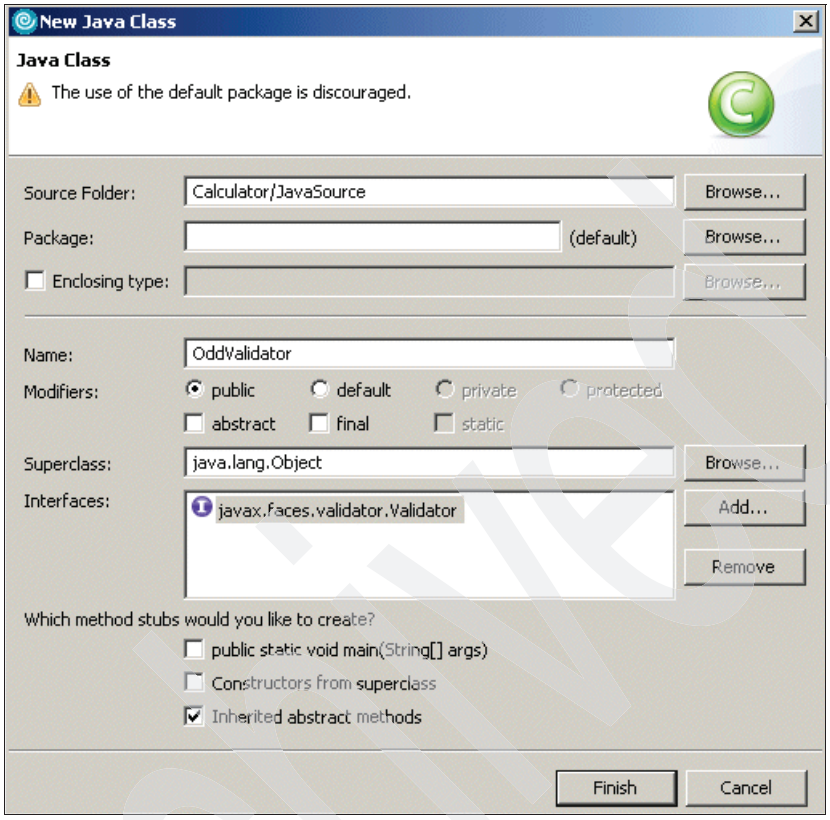


Figure 17-37 New Java class

2. The skeleton class is generated with this method skeleton:

```
public void validate(FacesContext arg0, UIComponent arg1, Object arg2)
      throws ValidatorException {
}
```

   Where:

   – FacesContext arg0: provides access to all components
   – UIComponent arg1: component to be validated
   – Object arg2: value (of the component) to be validated

3. Replace the method body with the code below:

Example 17-4 The validator code

```
System.out.println("OddValidator start");
UIInput field = (UIInput)arg1;
int value = ((Long)arg2).intValue();
```

```
System.out.println("Field="+field.getId()+" Value="+value);
if (value%2 == 1) {
   field.setValid(true);
   System.out.println("OddValidator end: valid");
} else {
   System.out.println("OddValidator end: invalid");
   FacesMessage errmsg = new FacesMessage(FacesMessage.SEVERITY_ERROR,
         "2nd number not odd.", "Second number must be odd.");
   throw new ValidatorException(errmsg);
}
```

4. To resolve the classes, right-click in the code and select **Source** → **Organize imports**.
5. Review the validator code:
   a. Access the `arg1` field to display its name, you can use `UIInput` or `HtmlInputText` classes.
   b. Obtain the value from `arg2`, you know its data type.
   c. Create an error message, a `FacesMessage`, to handle errors:
      i. The first message is the *summary* message (displayed in the global error message field).
      ii. The second message is the *detailed* message (displayed in an error message field attached to the input field). A detailed message has not been defined and therefore it will not be used in this case.
6. Save and close the `OddValidator`.
7. Register the validator. Open the `faces-config.xml` file and at the bottom ad the lines shown in Example 17-5.

Example 17-5 The validation registration in faces-config.xml

```
<validator>
   <description>Registers the OddValidator</description>
   <validator-id>oddValidator</validator-id>
   <validator-class>itso.jsf.calculator.OddValidator</validator-class>
</validator>
```

8. In the `calculate.jsp`, select the `number2` field and go to the Source page in the Design view.
   a. Place the cursor before the end of the `number2` field and just after the </f:validateLongRange> tag:

   ```
   ....</f:validateLongRange .......> PUT CURSOR HERE .......
   ```

   b. In the menu bar, select **JSP** → **Insert Custom**.

Archived

c. In the Insert Custom Tag dialog select the **"f"** tag library (at left) and **validator** (at right) and click **Insert**.

d. Click **Close**. The custom tag is inserted.

e. Enter `oddValidator` for the validatorId attribute, either directly in the Source, or in the Properties view. This generate the following tag:

```
<f:validator validatorId="oddValidator"></f:validator>
```

f. Save the `calculator.jsp`.

g. Restart the enterprise application. Enter an even value and you should see the error message. Notice that the summary message is displayed.

h. Notice also the test output in the Console.

17.9 Implementing a value change event

This event allows you to monitor value changes. This event can be invoked immediately or when the next action (button) is performed. As an example in this sample scenario you will monitor the change of the combo box operation.

1. From the palette, select an `Output` component and drop it to the right of the operation combo box (Figure 17-38). You will place a text there when the operation has changed.



Figure 17-38 Output text to monitor changes in the combo box values

2. In the Properties view, change the Id to `Opchanged`. Make sure the output component is still selected.

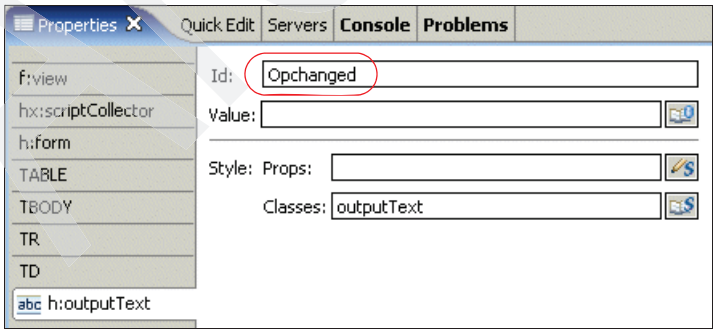


Figure 17-39 Properties view

3. Select the operation combo box. In the Quick Edit view, select **Value Changed** (left). Click in the code pane (right), and enter this code:

Example 17-6 Code to handle the value change event

```
log("OperationValueChangeEvent start");
HtmlSelectOneMenu operation = (HtmlSelectOneMenu)valueChangedEvent.getSource();
String opnew = (String)valueChangedEvent.getNewValue();
String opold = (String)valueChangedEvent.getOldValue();
log("operation old="+opold+" new="+opnew);
getCalculator().setOperation(opnew); // is also done later by binding
getOpchanged().setValue("changed");  // change the output text field
getCalc().setValue(opnew);           // change the Calculate button
log("OperationValueChangeEvent end");
```

4. Save the calculate.jsp.
5. Open the Calculate.java file and you will find the handleOperationValueChange method with your code.
6. Review the generated code:
   a. The method that is generated (Calculate.java) has one parameter (valueChangedEvent).
   b. From this parameter you can access the source component and the old and new values.
   c. In this scenario, you set the Opchanged output field to the value changed.
   d. You use the new value to change the label in the Calculate button.
   e. Notice in the Properties view, with All attributes button selected, the valueChangeListener is set to #{pc_Calculate.handleOperationValueChange1}.

      Note: The last character in the method name can be nothing or a consecutive digit, depending on how many operations you have.
7. Run the code. You can see in the Console that the event is invoked. However, no change is visible in the GUI. This is because the event is not immediate and the refresh of the GUI after the action looses the changes if a navigation rule is involved. If your action logic returned null, the changes would be visible.
8. To make the event immediate, you have to handle the onchange event. But before you do that, you will create a method to facilitate dealing with the portlet namespace:
   a. Open the PageCodeBase.java file. Create the following attribute:

      ```
      protected String namespace;
      ```

   b. Create a method to get the portlet namespace from the RenderResponse:

Example 17-7 getNamespace method

```
public String getNamespace() {
   if (namespace == null) {
      RenderResponse response =
(RenderResponse)facesContext.getExternalContext().getResponse();
      namespace = response.getNamespace();
   }
   return namespace;
}
```

c. You may need to organize your imports.

9. Now you can handle the `onchange` event. Select the `operation` combo box. In the Quick Edit view select `onchange` (left). Click in the code pane (right) and enter this code:

```
document.forms['view<portlet:namespace/>:form1'].submit();
return false;
```

10. This creates a JavaScript function that is invoked from the combo box (Example 17-8).

Example 17-8 JavaScript to handle the onchange event

```
<SCRIPT type="text/javascript">
function func_1(thisObj, thisEvent) {
//use 'thisObj' to refer directly to this component instead of keyword 'this'
//use 'thisEvent' to refer to the event generated instead of keyword 'event'
document.forms['view<portlet:namespace/>:form1'].submit();
return false;
}</SCRIPT>
```

11. Next, you have to change the name of the JavaScript function, to include the namespace, so that it does not collide with the name of other portlets' scripts on the page:

a. Change the name of `func_1` function to `<portlet:namespace/>func_1`. The function signature should now be like this:

```
function <portlet:namespace/>func_1(thisObj, thisEvent) {
```

b. In the Source code of the combo box you will find:

```
<h:selectOneMenu ..... onchange="return func_1(this, event);">
```

You need to change the call to the JavaScript function. Change the onchange attribute value to the following:

```
onchange="return #{pc_Calculate.namespace}func_1(this, event);"
```

Archived

Note 1: "view<namespace>" is a client ID of UIView (<f:view>). In the normal portlet programming pattern, it should be "<namespace>view", so that the portlet container removes "<namespace>" part in the request parameters. However, JSF requires the request parameter names match to the client IDs and if the porlet container removes "<namespace>" part, JSF would fail to apply the request parameters to UI components. That's why JSF portlet uses "view<namespace>" instead, in order to avoid ID collision with other JSF portlets in the same page.

Note 2: The reason to use value bind to retrieve the namespace in the onchange attribute is because JSF components does not accept runtime expressions as values. So you cannot use `portlet:namespace` here. For this reason we created a namespace property in `PageCodeBase` class, with its proper get method. Inside the JavaScript function, you do not have this limitation, so you could use the `portlet:namespace` tag.

12. Save the file and retest. When you change the operation, you see the immediate result (Figure 17-40).
13. Notice that the button changes back to `Calculate` at the next operation because a navigation rule is involved. If you change the action logic to `return null`, the changes to the user interface will be permanent.

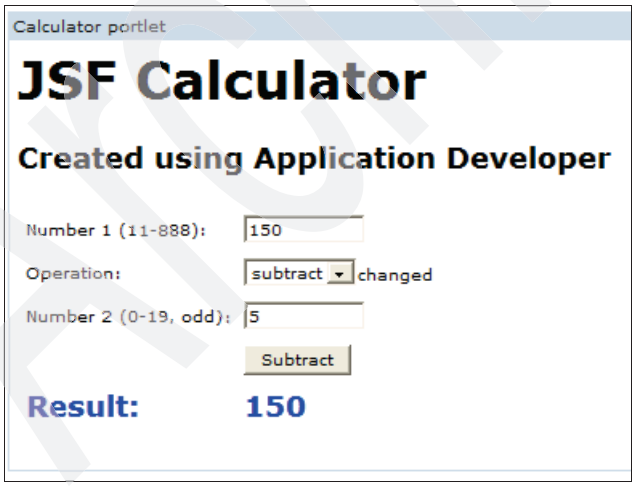


Figure 17-40 Result of value changed event

17.10 Implementing internationalization

In this section, you will use the internationalization feature to create JSF portlets to run in multiple languages. You will include the text constants and error messages into a properties file used by the JSF components and the business logic.

However, in order to simply illustrate the concept, you will only implement a few constants. Since you already have a properties file created by the portlet project wizard, you will use this same file to include the key-value pairs for this internationalization scenario.

Execute the following steps:

1. Open the `calculator.nl.CalculatorPortletResource.properties` file. Include these lines at the end of the file, then save and close it:

   ```
   page_comment=* Created using Application Developer *
   text_operation=Calculator operation:
   text_calculate=Perform Calculation
   ```

   Note: `CalculatorPortletResource.properties` file is the name of the default file (no locale suffix). If a locale is not supported by the portlet or the specific properties file is not found, the user gets values from this properties file. For example, if you want to create specific values for English, you will need to create a file with name `CalculatorPortletResource_en.properties`.

2. Create a properties file for Brazilian Portuguese.
   a. Right-click **CalculatorPortletResource.properties**.
   b. Select **Copy**.
   c. Right-click the **calculator.nl** package.
   d. Select **Paste**.
   e. Enter `CalculatorPortletResource_pt_BR.properties` as the new name for the file. This is the suffix for brazilian Portuguese.
   f. Click **OK**.
   g. Change the three last keys, to use Brazilian Portuguese text:

   ```
   page_comment=* Criado com Application Developer *
   text_operation=Operação:
   text_calculate=Efetuar Cálculo
   ```

   h. Save the file and close the editor.
3. Edit the Portlet deployment descriptor.
   a. Double-click **Portlet Deployment Descriptor**.
   b. In the Portlet deployment descriptor window, click **Portlets** →**Calculator**.

c. Under the supported locales section, click **Add**. See Figure 17-41.

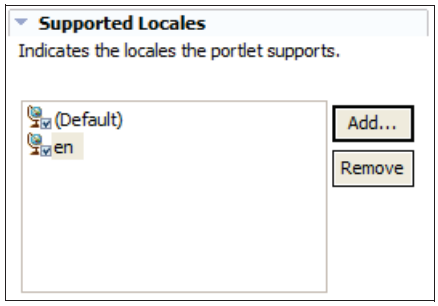


Figure 17-41 The supported locales section

d. Enter `pt_BR` in the Locale list.

e. Click **OK**.

f. Save the file and close the editor.

4. You also need to specify the supported languages in the `faces-config.xml` file.

   a. Open the faces-config.xml file.

   b. Within the `<application>` tag, include the following tags:

```
<locale-config>
   <supported-locale>en</supported-locale>
   <supported-locale>pt_BR</supported-locale>
</locale-config>
```

   c. Save the file and close it.

5. In the `calculate.jsp` Source code, after the `jsf/core` tag library, enter:

```
<f:loadBundle var="constants"
basename="calculator.nl.CalculatorPortletResource"/>
```

   Note: This tag provides access to the text constants for the selected language.

6. In the Design view, select the `comment` field (`Created using Application Developer`). In the Properties view, change the `value` attribute to:

```
#{constants.page_comment}
```

7. Similarly change the `Operation:` text to:

```
#{constants.text_operation}
```

8. Change the `Calculate` button label in the `Display options` tab to:

```
#{constants.text_calculate}
```

Archived

9. The page should look like Figure 17-42.

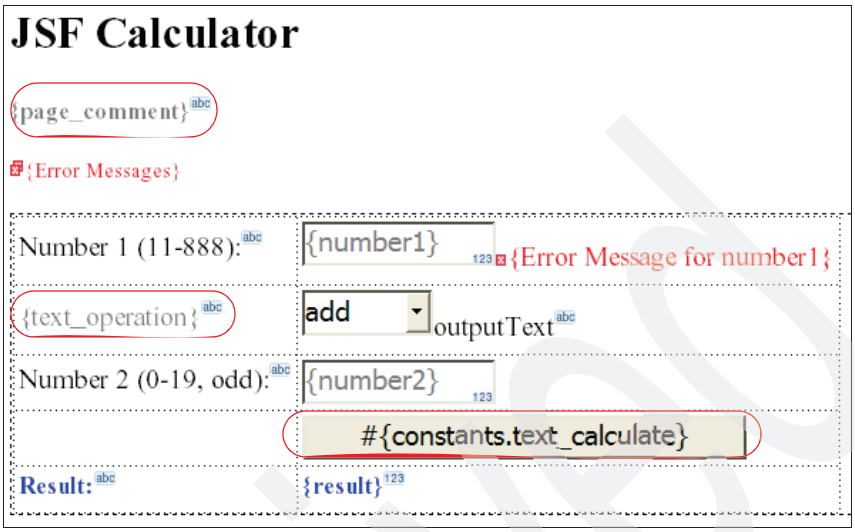


Figure 17-42 Page with components changed to run in multiple languages

10. Run the project again and change your preferred language in the profile to Brazilian Portuguese. Use Edit my profile option and enter the required information (password is wpsadmin).
11. You will have to run the project again so that the JSF framework accepts the change.
12. Notice that the text has changed to the values contained in the resource file.
13. Now try to specify a value for `number1` that is out of the permitted range (11-888). You will notice that the error message is also translated to Brazilian Portuguese, although you did not have to be concerned about (Figure 17-43 on page 590).

Archived



Figure 17-43 The calculator internationalized to Brazilian Portuguese

17.10.1 Internationalization of standard validator messages

The JSF framework is already prepared to translate the standard validator messages to many languages. If your language is not automatically translated or if you want to change the default validator messages, proceed as follows:

1. In the faces-config.xml file, register the resource bundle that will store the validator messages.

   Within the `<application>` tag, include the following tag:

   ```
   <message-bundle>calculator.nl.CalculatorPortletResource</message-bundle>
   ```

   Note: The message-bundle element represents a set of localized messages. This element contains the fully-qualified path to the resource bundle containing the localized messages, in this case, `calculator.nl.CalculatorPortletResource`.

2. Include the following key in `CalculatorPortletResource.properties` (default locale bundle):

   ```
   javax.faces.validator.NOT_IN_RANGE=Valid values must be in the range of {0} to {1}
   ```

3. Include the following key in `CalculatorPortletResource_pt_BR.properties`:

Archived

```
javax.faces.validator.NOT_IN_RANGE=Este campo só aceita valores entre {0} e
{1}
```

4. Run the code again and try to specify a value for `number1` that is out of the permitted range (11-888). You will notice that the error message has changed to the one specified in the properties file.

   For your reference, the Table 17-1 shows the keys for the standard messages associated with standard JSF components. The messages for the IBM extended components can be found at the `jsf-ibm.jar` file.

Table 17-1 Keys for the standard messages

Key	Standard Message
javax.faces.component.UIInput.CONVERSION	Conversion error occurred
javax.faces.component.UIInput.REQUIRED	Value is required
javax.faces.component.UISelectOne.INVALID	Value is not a valid option
javax.faces.component.UISelectMany.INVALID	Value is not a valid option
javax.faces.validator.NOT_IN_RANGE	Specified attribute is not between the expected values of {0}and {1}
javax.faces.validator.DoubleRangeValidator.MAXIMUM	Value is greater than allowable maximum of {0}
javax.faces.validator.DoubleRangeValidator.MINIMUM	Value is less than allowable minimum of {0}
javax.faces.validator.DoubleRangeValidator.TYPE	Value is not of the correct type
javax.faces.validator.LengthValidator.MAXIMUM	Value is greater than allowable maximum of {0}
javax.faces.validator.LengthValidator.MINIMUM	Value is less than allowable minimum of {0}
javax.faces.validator.LongRangeValidator.MAXIMUM	Value is greater than allowable maximum of {0}
javax.faces.validator.LongRangeValidator.MINIMUM	Value is less than allowable minimum of {0}
javax.faces.validator.LongRangeValidator.TYPE	Value is not of the correct type

Arrium

Archived

18

Additional Faces portlet sample scenarios

This chapter provides sample scenarios of JSF portlet development, to illustrate additional features available in Rational Application Developer and the Portal Tools.

Note: The portlet applications described in this chapter have the following characteristics:

- Portlet API: IBM Portlet API (scenario 1) and JSR 168 (scenario 2)
- Application type: JavaServer Faces

© Copyright IBM Corp. 2005. All rights reserved.

18.1 The call center application

The application you will build is a call center application. The call center application consists of a page that contain two portlets to list trouble tickets for a customer and view the details of a trouble ticket.

Both portlets are created using Faces (JSF) portlet with prebuilt JavaBean that represent the customers and trouble tickets.

Note: To make the example simple, static data is used instead of accessing a database.

In this example, you will add support for Edit mode to select a customer ID. You will also include a Click-to-Action tag in the list portlet to send a ticket ID to the detail portlet.

18.1.1 Creating the project

We will create the project as a Portlet Project project. Follow these steps to create it:

1. In the menu, select **File** → **New** → **Project**.
2. Select Portlet Project and click **Next**.
3. If the Confirm Enablement window appears, asking if you want to enable the portal development role, click **OK**.
4. In the Portlet project page, type the name of the project `Ticket`.
5. Check the **Create a portlet** check box, if it is not already checked.
6. In the WebSphere Portal version box, select **5.1**.
7. Click **Next >**.
8. In the Portlet type page, select **Faces portlet**.
9. Click **Next >**.
10. Click **Next >** again in the Features page.
11. In the Portlet settings page, review the options provided.
    Leave all settings unchanged. Click **Next >**.
12. Check **Edit mode** and click **Finish**.
13. Click **Yes** in the Confirm Perspective Switch dialog.

    A JSP file is opened with Page Designer.
14. Select **Java Resources/JavaSource** in Project Explorer and select **Import** from the context menu.

Archived

15. Select **Zip file** and click **Next**.
16. Click **Browse** and select the **com.ibm.faces.portlet.example.zip** file.
17. Click **Finish**.

    The prebuilt Java Bean package (`com.ibm.faces.portlet.examples`) is shown under JavaSource.

18.1.2 Creating the page layout

We will now create a data table in TicketView.jsp to show the list of trouble tickets. Proceed as follows:

1. In the Design view, delete the text `Place content here.`
2. In the Page Data view, select **New** → **JavaBean** from the context menu.
3. Enter `ticketList` for the Name and `com.ibm.faces.portlet.examples.TicketList` for the class. If you wish, you can use the icon next to Class to browse and type `TicketList` to find the class.
4. Click **Finish**.

   `ticketList` is shown in Page Data view (Figure 18-1).

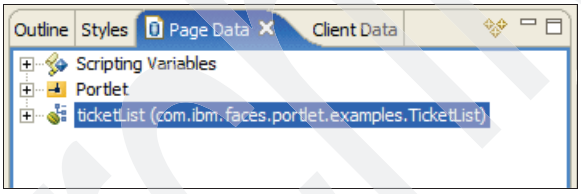


Figure 18-1 The Page Data view showing the ticketList Java Bean

5. Expand **ticketList** and drag and drop **ticketList/tickets** from Page Data view onto Page Designer.
6. Type `com.ibm.faces.portlet.examples.Ticket` in Type and click **OK** in the Object Type dialog (Figure 18-2 on page 596).

Archived

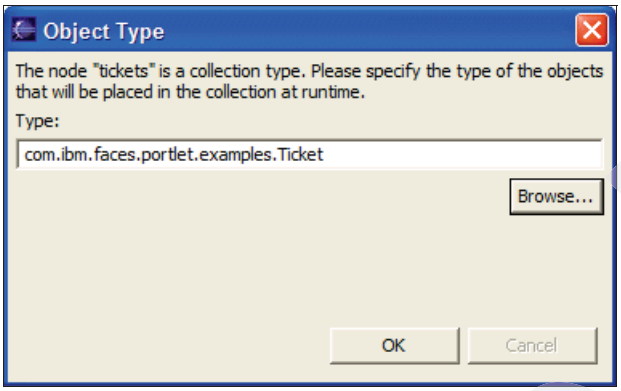


Figure 18-2 The object type dialog

7. In the Insert JavaBean dialog (Figure 18-3 on page 597), click **None** and select the fields **ticketId**, **title** and **status**. Sort the fields in this order clicking Up/Down (icons). Enter `Ticket ID` as the label of the ticketId field.

Archived

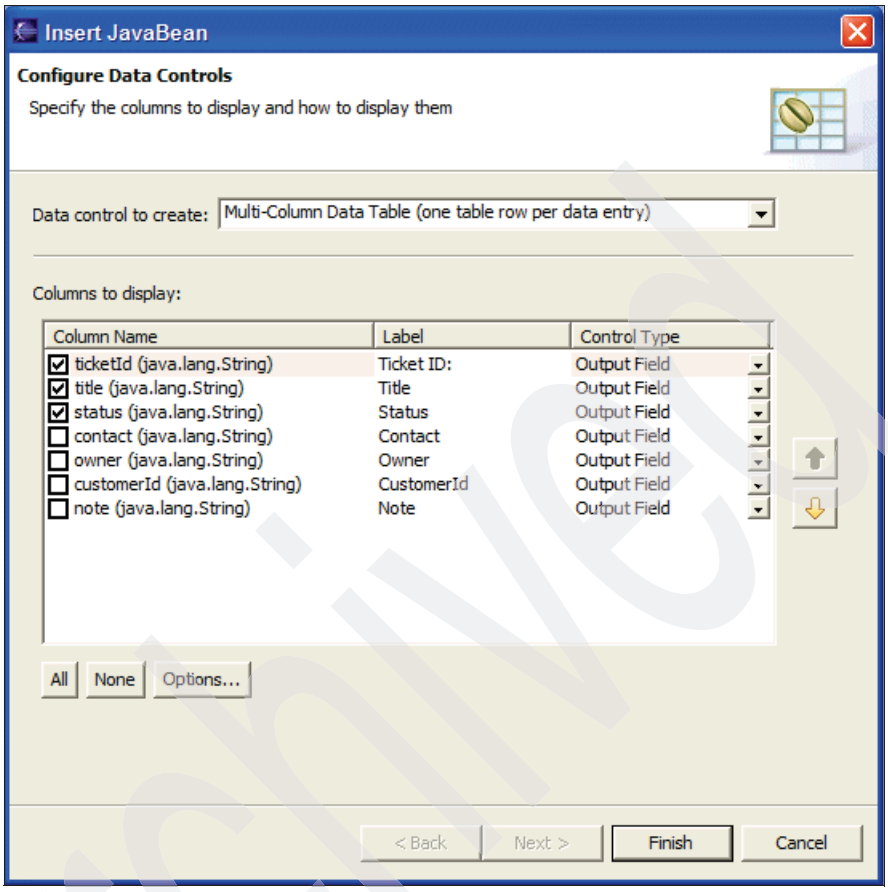


Figure 18-3 Inserting the ticketList Java Bean in the page

8. Click **Finish**.

   A data table with outputs is inserted.

9. Select one of the cells and change the focused node by clicking **h:dataTable** on the Change focused node button at the top of Design view (Figure 18-4 on page 598).

Archive

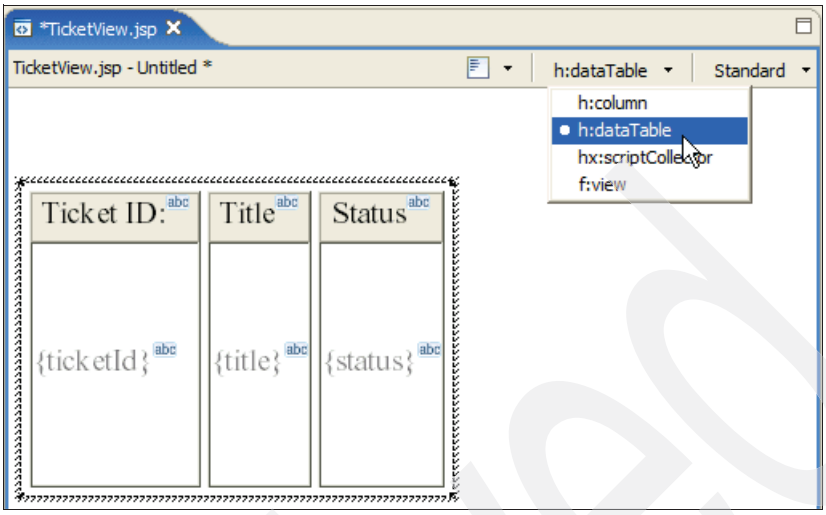


Figure 18-4 Changing the focused node

10. In the Display options tab, Enter 5 in Rows per page and click the **Add a deluxe pager** button (Figure 18-5)

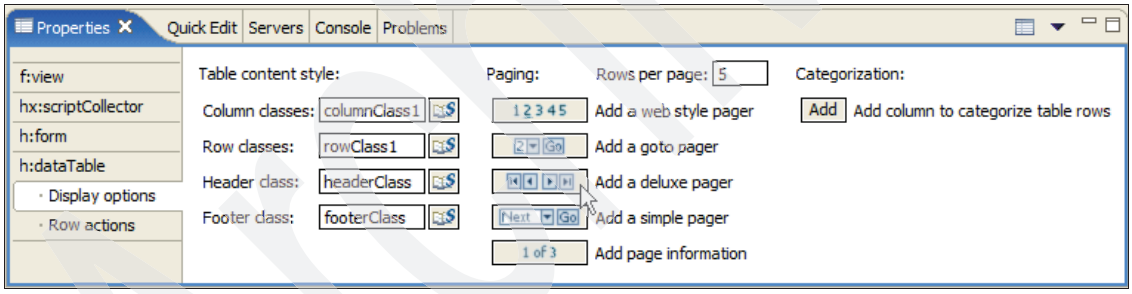


Figure 18-5 Setting the display options for the dataTable

Your page should look similar to Figure 18-6 on page 599.

Archived

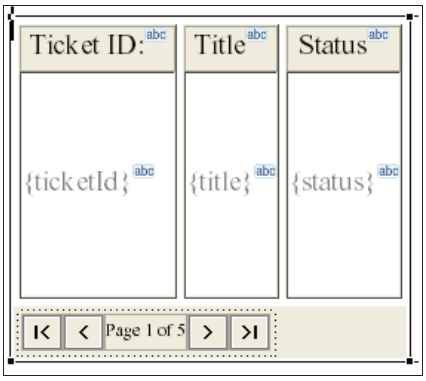


Figure 18-6 Layout of the dataTable after setting the display options

11. Save the file.

18.1.3 Defining a parameter for the list

Now you will define a parameter to filter the list.

1. Expand **Portlet** in Page Data view.
2. Select **PortletData** and select **Add Attribute** from the context menu (Figure 18-7).

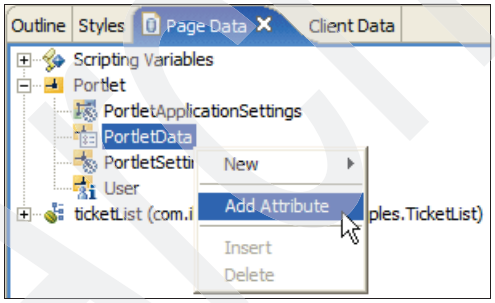


Figure 18-7 New PortletData attribute

3. Type `customerId` as the Attribute name and click **OK**.
4. Select **ticketList** and select **Configure** from the context menu.
5. Click **Initialize Properties**.
6. Click **Add**.
7. Select **customerId** from Name.

Arrived

8. Select **PortletData/customerId** from Value (click the icon to view the dialog shown in Figure 18-8).

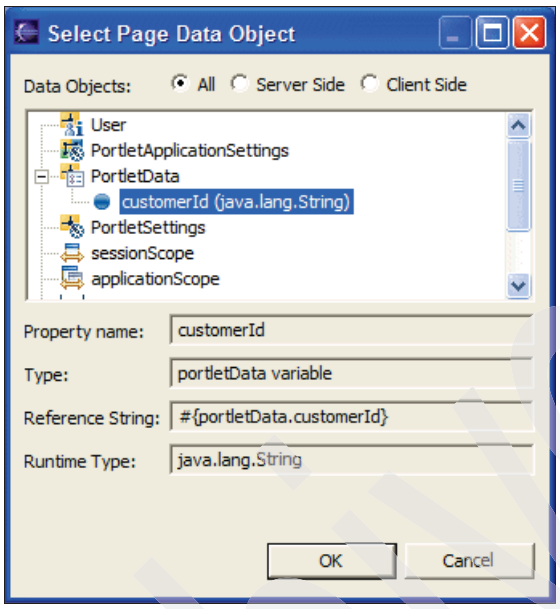


Figure 18-8 Selecting an Object from Page Data

9. Click **OK**.
10. Click **OK** again.
11. Click **Finish**.
12. Select **File** → **Save** from the menu bar.
13. Open WebContent/TicketEdit.jsp with Page Designer.
14. Replace the text `Place content here.` with `Customer ID:`.
15. Drag and drop Combo Box from Faces Components of Palette view, next to the text `Customer ID:`.
16. Drag and drop Command - Button from the Faces Components of the Palette view, next to the combo box. Your page should look like Figure 18-9.

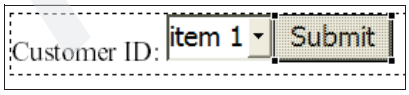


Figure 18-9 Layout of TicketEdit.jsp

17. Select **New** → **JavaBean** from the context menu of Page Data view.

Archived

18. Enter `customerList` for the Name and `com.ibm.faces.portlet.examples.CustomerList` for the class.
19. Click **Finish**.

    `customerList` is shown in Page Data view.
20. Expand customerList/customers/Contained Type.
21. Enter `com.ibm.faces.portlet.examples.Customer` in Type and click **OK** in the Object Type dialog.
22. Drag and drop **customerList/customers/Contained Type/customerId** on the combo box in Page Designer.

    The combo box gets wider to show the choice specification, but it is normal behavior.
23. Expand **Portlet** in Page Data view.
24. Select **PortletData** and select **Add Attribute** from the context menu.
25. Enter `customerId` in Attribute name and click **OK**.
26. Select the combo box and in the Properties view, h:selectOneMenu tab, click the  icon next to Value.
27. Select **PortletData/customerId** and click **OK** in Page Data Object dialog.
28. Select **File** → **Save** from the menu bar.

18.1.4 Creating a detail portlet

Now you will create a new portlet in the same Portlet project to show the trouble tickets detail.

1. Select **File** → **New** → **Portlet** from the menu bar.
2. Select **Ticket** for the Project.
3. Select **Faces portlet** and click **Finish**.
4. A JSP file named `Ticket2View.jsp` is opened with Page Designer.
5. Select **New** → **JavaBean** from the context menu of Page Data view.
6. Enter `ticketDetail` for the Name and `com.ibm.faces.portlet.examples.TicketDetail` for the class.
7. Click **Finish**.

   `ticketDetail` is shown in Page Data view.
8. Replace the text `Place content here.` with `Ticket ID:`.
9. Drag and drop Input from Faces Components of Palette view, next to the text, `Ticket ID:`.

Archive

10. Drag and drop Command - Button from Faces Components of Palette view, next to the input.
11. Drag and drop **ticketDetail/ticketId** from Page Data view on the input.
12. Drag and drop **ticketDetail/ticket** from Page Data view onto Page Designer, below the `Ticket ID` text.
13. Select the following fields:
    - title
    - status
    - owner
    - contact
    - note

    and sort in the above order by clicking Up/Down (icons).
14. Click **Finish**.

    A table with outputs is inserted (Figure 18-10).

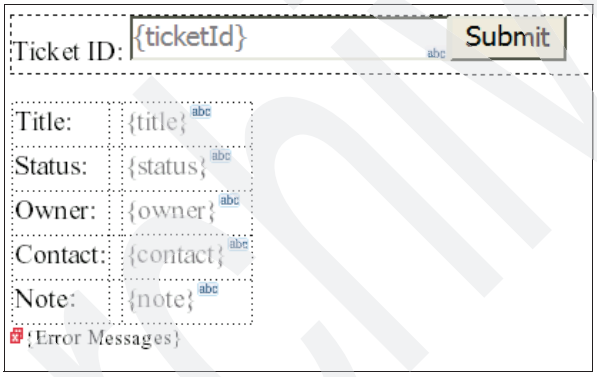


Figure 18-10 Layout of the Details portlet

15. Select **File** → **Save** from the menu bar.

18.1.5 Linking the portlets

Now you will link the two portlets:

1. Switch to TicketView.jsp.
2. Drag and drop Click-to-Action Output Property from Portlet of Palette view on the output labeled {ticketId}.

   Note: You need to drop Click-to-Action Output Property **ON** the output component to bind value automatically. To ensure that you are dropping on

Archived

the right place, be sure that during the drop the enclosing black rectangle is shown around the output component.

The Insert Click-to-Action Output Property dialog is shown (Figure 18-11).

3. Type `ticketId` in Data type.
4. Select **Ticket portlet** from Source portlet and click **OK**.

   The wizard creates the Ticketportlet.wsdl file and import the pbportlet JAR file in the project.

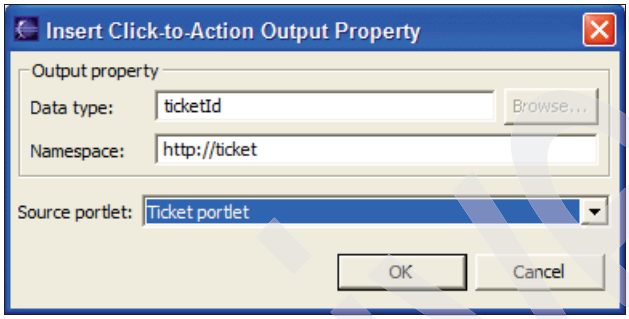


Figure 18-11 Inserting a Click-to-Action output property

5. Select **File** → **Save** from the menu bar.
6. Select **Portlet Deployment Descriptor/Ticket2 portlet** in Project Explorer view and select **Cooperative** > **Enable Target** from the context menu.
7. Click the first **Browse** button to select the Data type for the Input property.
8. Select **ticketId** (Figure 18-12) and click **OK**.

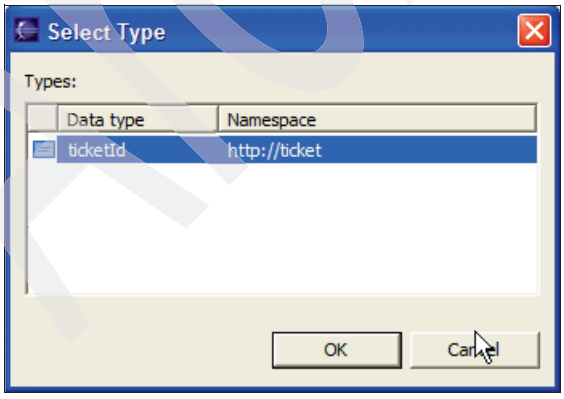


Figure 18-12 Select Data type for the Input property

9. Click the second **Browse** to select the target action.
10. Select **/Ticket2View.jsp/form1/button1** (Figure 18-13) and click **OK**.

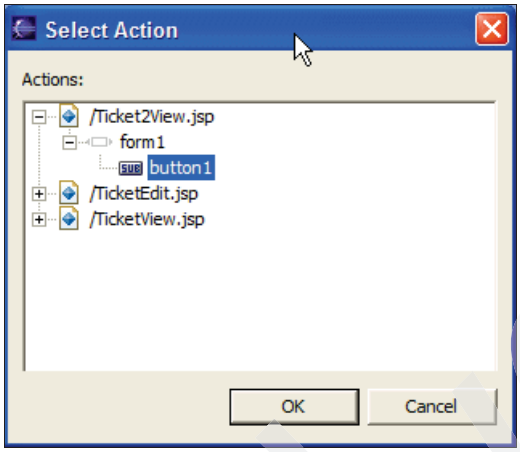


Figure 18-13 Select the target action

11. Type `Show Detail` in Label. Your dialog should look like Figure 18-14 on page 605.
12. Click **OK**.

    The wizard creates the Ticket2portlet.wsdl file.

Archived

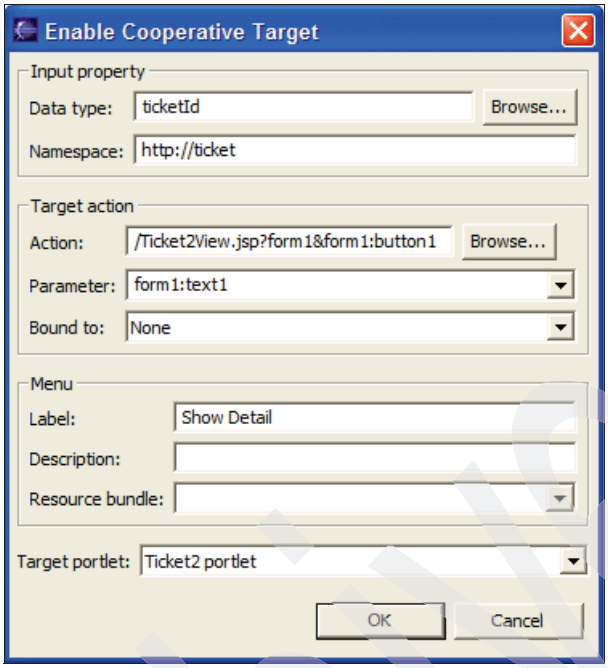


Figure 18-14 Enable Cooperative Target dialog

18.1.6 Testing the application

Execute the following steps:

1. Select the **Ticket** project in the Project Explorer view and select **Run** → **Run on Server** from the context menu.
2. Select **IBM/WebSphere Portal V5.1** and click **Finish**. The browser opens and shows a page with the portlets.
3. Click the Edit mode icon on the portlet window title of the left portlet.
4. Select **C0002** from the combo box and click **Submit**.
5. Click the back icon on the portlet window title. Trouble tickets are shown in the data table.
6. Click the Click-to-Action icon in the data table and select **Show Detail**.
7. The detail of the trouble ticket is shown in a table in the Detail portlet .

Archived

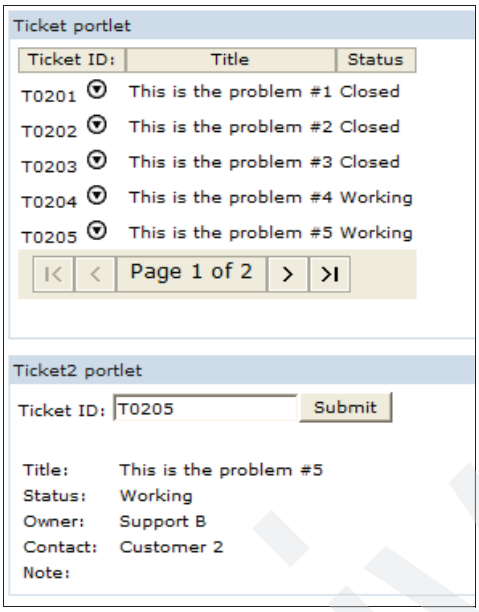


Figure 18-15 Running the application

18.2 The Web service client portlet

Portals generally serve as clients to back end services, so being able to quickly build portlets that can call Web services is essential. In this application, you will see how easily you can search for a known Web service published on the Internet and create a portlet client using JSF components to access it. You will build a portlet that calls a Traffic Condition Web service.

Note: A network connection to the Internet will be needed to run this example.

18.2.1 Creating the project

We will create the Traffic Condition project as a Portlet Project (JSR168) project. Follow these steps to create it:

1. In the menu, select **File** → **New** → **Project.**
2. In the New Project window, select the wizard for **Portlet Project (JSR 168).**
3. Click **Next >**.

Archived

4. If the Confirm Enablement window appears, asking if you want to enable the portal development role, click **OK**.
5. In the Portlet project page, type the name of the project `TrafficCondition`.
6. Check the **Create a portlet** check box, if it is not already checked. Typically, you do not need to create a portlet when you import a portlet WAR file into the project.
7. In the WebSphere Portal version box, select **5.1**.
8. Click **Next >**.
9. In the Portlet type page, select **Faces portlet (JSR 168)**.
10. Click **Finish**.
11. If the Confirm Perspective Switch window appears, click **YES**.

18.2.2 Creating a new Web service client

Now you will locate an existing Web Service and add it to the Page Data view.

1. With the **TrafficConditionView.jsp** file selected, right-click in the Page Data view and select **New** → **Web Service** (Figure 18-16).

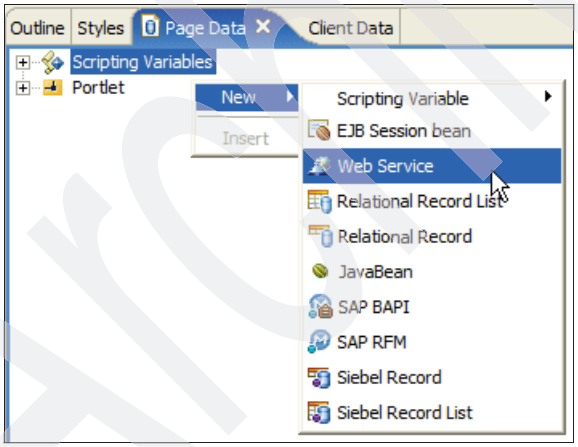


Figure 18-16 Creating a new Web Service in the Page Data

2. In the Web Service Discovery Dialog (Figure 18-17 on page 608), select **Web Services from a known URL**.

Archived

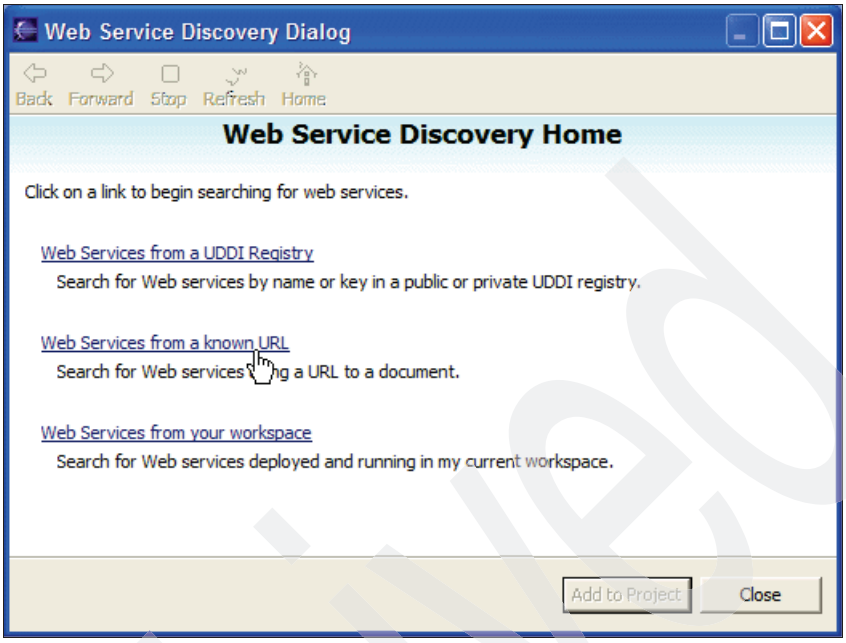


Figure 18-17 Web Service Discovery dialog

3. On the next page (Figure 18-18 on page 609), enter `http://www.xmethods.net/sd/2001/CATrafficService.wsdl` for the URL field. Then click **Go**.

Archived

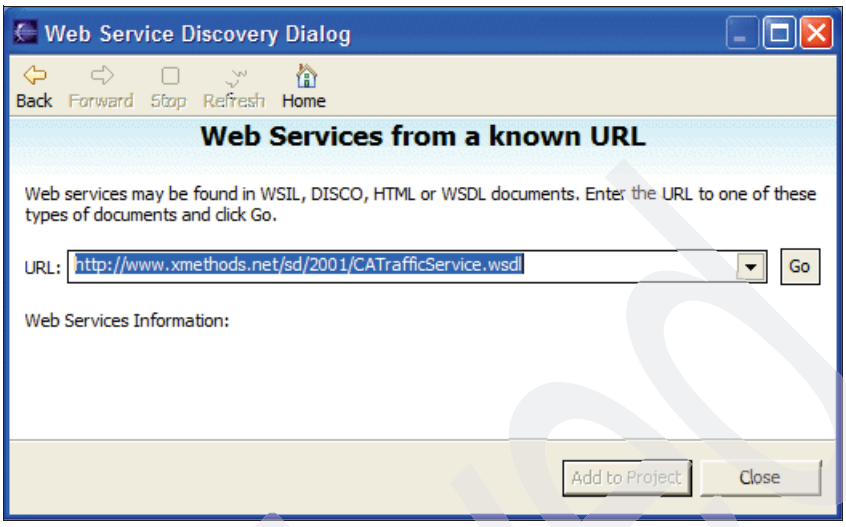


Figure 18-18 Web Services URL

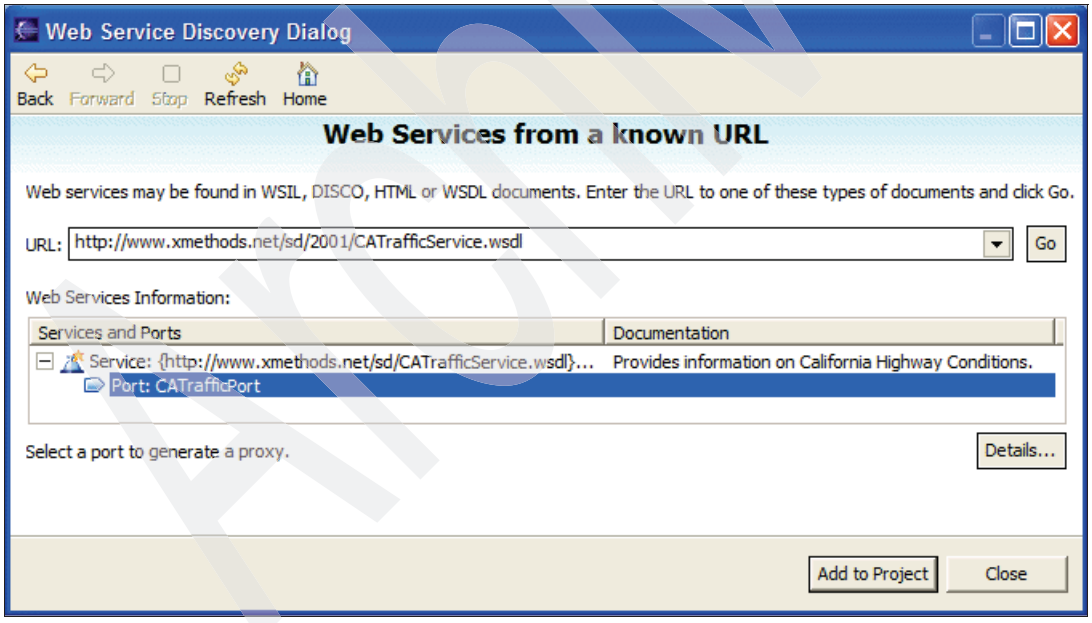


Figure 18-19 The Web Services dialog displaying the Web Services Information window

4. The dialog will display information about the Web service in the Web Services Information window (Figure 18-19). To explore the Web Service, select it and click **Details**. This will launch the Web Services Explorer.

5. Invoke the WSDL operation by clicking **getTraffic** in the WSDL Binding Details window. This will bring the Invoke a WSDL Operation window. Enter a highway number as the `hwynums` parameter (Figure 18-20). The return of the operation will be displayed in the Status window.

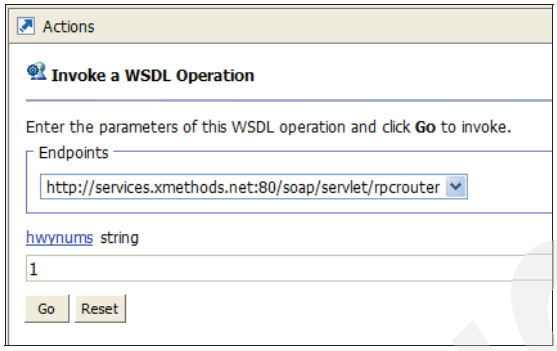


Figure 18-20 Invoking the WSDL operation

6. Click the **Source** link at the right of the Status window (Figure 18-21).

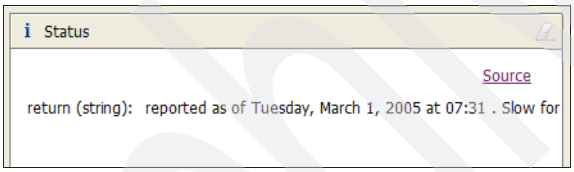


Figure 18-21 The Status window

The details of the SOAP request (Figure 18-22) and response (Figure 18-23 on page 611) are displayed in the Status window.

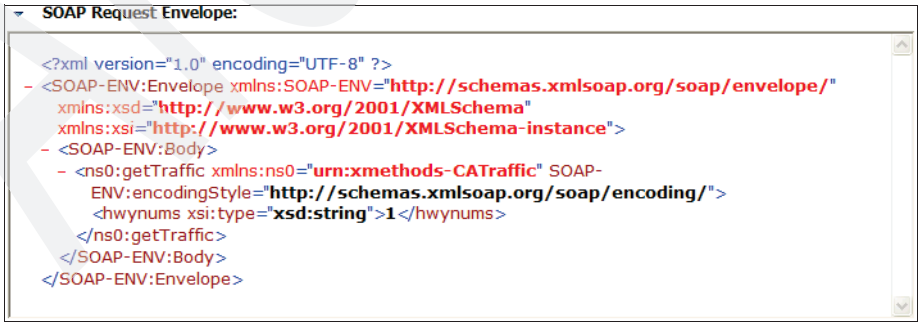


Figure 18-22 The SOAP request

Archived

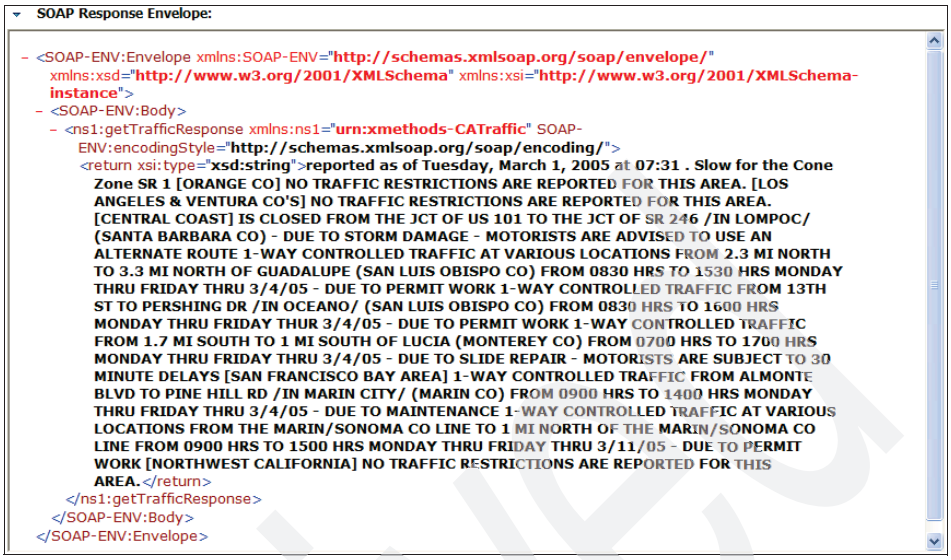


Figure 18-23 The SOAP response

7. Close the Web Services Explorer. In the Services and Ports section of the Web Services Discovery dialog, expand the **Service** element and select the **Port: CATrafficPort**. Click **Add to Project**. This will generate a client proxy that will hide the details of the Web service JAX-RPC API.
8. In the Add Web Service dialog (Figure 18-24 on page 612), select **Finish**.

Archive

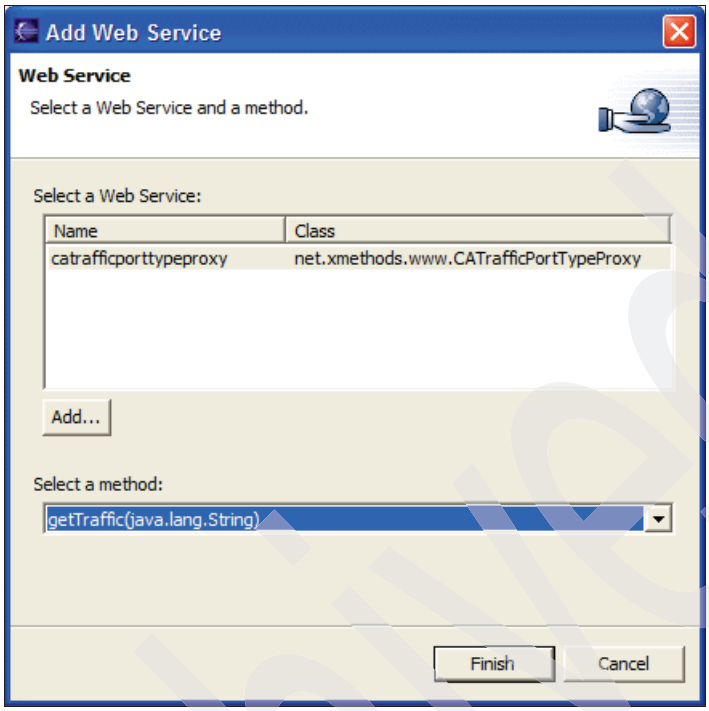


Figure 18-24 The dialog to add a Web service

The published methods available for the Web service will now be visible in the Page Data view (Figure 18-25)

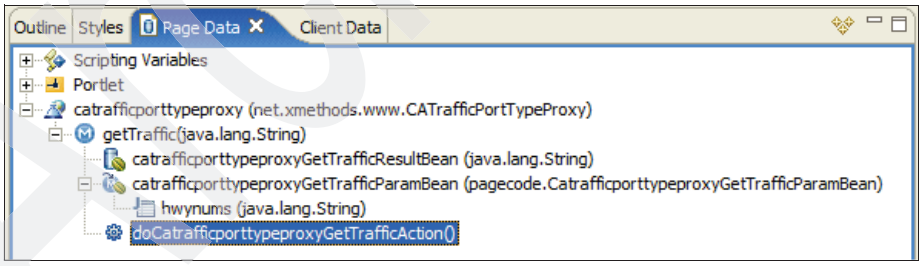


Figure 18-25 The page data window with the new Web service defined

18.2.3 Creating the page layout

Now that you have located and explored the Web service, you will add it to the page, then prepare the page for the Web service components by modifying the default values and adding the correct components.

1. In the TrafficConditionView.jsp file, replace the default text with `Enter a highway number to get traffic conditions:`
2. From the Faces Components palette, drag and drop a Panel - Group Box below the text.
3. In the Select pop-up (Figure 18-26), select **List** and click **OK**.

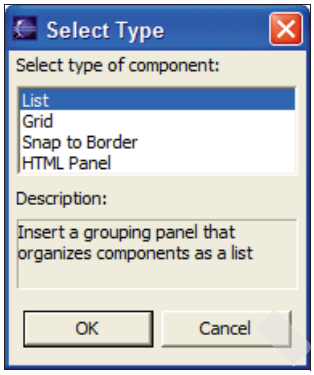


Figure 18-26 Select Type

Now you will add the input and output components and the action button for the Web service to the Faces JSP.

4. In the Page Data view, expand **catrafficporttypeproxyGetTrafficParamBean** and select **hwynums**. Drag and drop it into the Panel - Group Box.
5. In the Insert Web Service dialog, make sure that the Control Type selected is **Input Field** (Figure 18-27 on page 614).

Archived

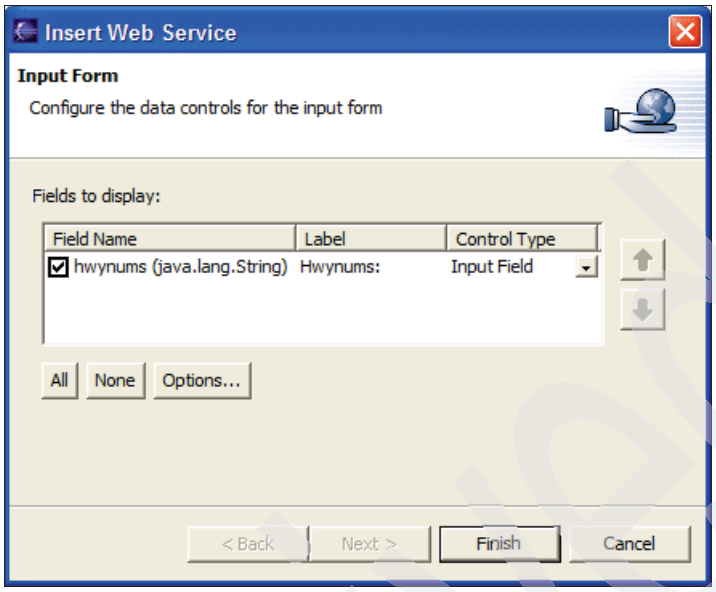


Figure 18-27 The insert Web service dialog

6. Click **Finish**.
7. Bind the **doCatrafficporttypeproxyGetTrafficAction** to the `Submit` button by dragging and dropping the action onto the button (Figure 18-28).

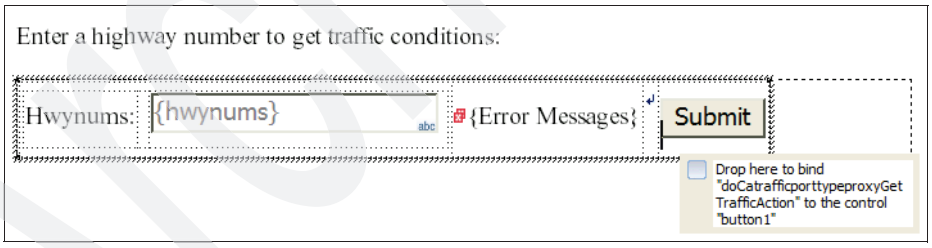


Figure 18-28 Binding the action to the Submit button

8. Insert a break after the Submit button.
9. The `catrafficporttypeproxyGetTrafficResultBean(String)` will return the result from the Web service. Drag and drop it below the break that was just inserted.
10. In the Insert Web Service dialog, make sure that the Control Type selected is **Output Field**. Change the label to `Traffic Report:`, then click **Finish**.

    The page layout should look like Figure 18-29 on page 615:

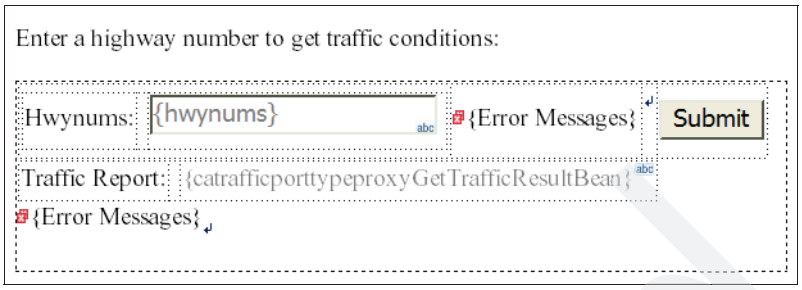


Figure 18-29 The page layout for the Web service client portlet

11. Save and close the file.

18.2.4 Testing the application

1. Select the `TrafficCondition` project, in Project Explorer view and select **Run → Run on Server** from the context menu.
2. Select **IBM/WebSphere Portal V5.1** and click **Finish**.

   The browser opens and shows a page with the portlet.
3. Enter a highway number and click **Submit**. The Web service is invoked and the result is displayed on the page (Figure 18-30):

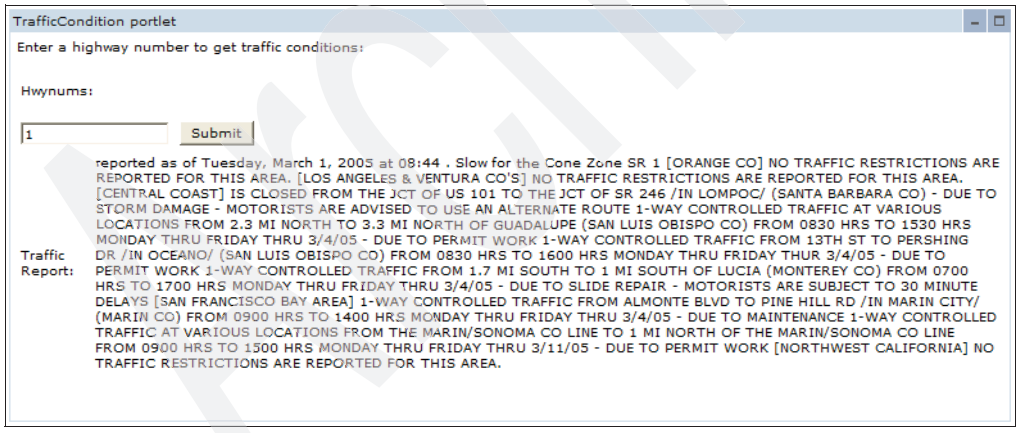


Figure 18-30 Running the application

Redbooks

Archived

19

Portlet services

In this chapter, we will discuss portlet services. WebSphere portal provide the ContentAccessService, CredentialVaultService and the PropertyBrokerService. We will briefly discuss the ContentAccessService here, the CredentialVaultService and the PropertyBrokerService are discussed in their own sections. We will also discuss writing a custom portlet service.

Archived

© Copyright IBM Corp. 2005. All rights reserved.

19.1 Portlet services

Portlet services provide commonly used functions to portlets. Portlet services can be used with both IBM portlets and JSR 168 portlets. IBM portlets access the service by using `PortletContext.getService`. JSR 168 portlets must use a JNDI lookup to access the service. Portlet services can only be invoked from portlets, not themes or skins.

Portlet service interfaces are different for the two APIs. You can, however, write a portlet service implementation you can use for both APIs. You can write your own portlet service and register it in the portal, so that all portlets can use it.

19.1.1 ContentAccessService

ContentAccessService (CAS) provides portlets with access to remote systems and content from remote URLs. When developing portlets, if you are not certain if there will be a firewall when the portlet is deployed you should use the ContentAccessService. URLs on the other side of a proxyserver can be obtained using ContentAccessService.

ContentAccessService provides three methods.

- getMarkup

  This method will return a String of the content returned by the URL.

- include

  The include method will use a RequestDispatcher to include the provided URL.

- getInputStream

  This method can be used if the URL to be requested does not specify the character encoding. You can specify the required encoding, otherwise it is returned as UTF-8.

- getURL

  Returns a URL object representing the provided URL.

Using ContentAccessService in an IBM portlet

IBM portlets can use the getService method as seen in Example 19-1

Example 19-1 IBM portlet using CAS

```
org.apache.jetspeed.portlet.service.ContentAccessService service =
(org.apache.jetspeed.portlet.service.ContentAccessService)getPortletConfig().ge
```

```
tContext().getService(org.apache.jetspeed.portlet.service.ContentAccessService.
class);
service.include(someURL, request, response);
```

Using ContentAccessService in a JSR 168 portlet

JSR portlets use a JNDI look up to get the ContentAccessService as seen in Example 19-2

Example 19-2 JSR 168 portlet using CAS

```
package casexample;

import java.io.IOException;
import java.io.PrintWriter;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.portlet.GenericPortlet;
import javax.portlet.PortletConfig;
import javax.portlet.PortletException;
import javax.portlet.RenderRequest;
import javax.portlet.RenderResponse;

import com.ibm.portal.portlet.service.PortletServiceHome;
import com.ibm.portal.portlet.service.contentaccess.ContentAccessService;

public class CASexamplePortlet extends GenericPortlet {
   private PortletServiceHome contentAccessServiceHome;
   public void init(PortletConfig config) throws PortletException {
      super.init(config);

         try {
            Context ctx = new InitialContext();
            Object home =
ctx.lookup("portletservice/com.ibm.portal.portlet.service.contentaccess.Content
AccessService");
            if(home != null){
               contentAccessServiceHome = (PortletServiceHome)home;
            }
         } catch (NamingException e) {
            e.printStackTrace();
         }
   }
```

Archived

```
   public void doView(RenderRequest request, RenderResponse response) throws
PortletException, IOException {
       // Set the MIME type for the render response
       response.setContentType(request.getResponseContentType());

       if(contentAccessServiceHome != null){
          ContentAccessService service =
(ContentAccessService)contentAccessServiceHome.getPortletService(ContentAccessS
ervice.class);
          service.include(request.getParameter("URL"), request, response,
getPortletConfig().getPortletContext());

       }
   }
}
```

19.1.2 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. 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.

Portlet Services can be written to support both IBM portlets and JSR 168 portlets. Most of the steps for creating portlet services for both APIs are the same, but we will point it out where they are different.

1. Define the service interface

   First, you must define an interface that defines the functionality this service will provide. For JSR 168 portlet service, the custom service interface must extend com.ibm.portal.service.PortletService.

   IBM Portlet service will need to extend org.apache.jetspeed.portlet.service.PortletService.

   The PortletService interface for both APIs is a flag interface and therefore does not define any methods.

Example 19-3 Defining the Service Interface for JSR 168 portlet service.

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

import com.ibm.portal.portlet.service.PortletService;

public interface MailService extends PortletService {
   public boolean sendMail(String to, String from, 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 com.ibm.portal.portlet.service.spi.PortletServiceProvider interface. The PortletServiceProvider defines the init method 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.

   You can use one implementation for both APIs. You would need to implement both the JSR 168 portlet service interface and the IBM portlet service interface defined in step 1.

Example 19-4 Implementing the custom service

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

import java.util.Date;
import java.util.Properties;
import java.util.prefs.Preferences;

import javax.mail.Message;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;

import com.ibm.portal.portlet.service.PortletServiceUnavailableException;
import com.ibm.portal.portlet.service.spi.PortletServiceProvider;

public class MailServiceImpl implements PortletServiceProvider, MailService {

    private String server_name;
    private String serverNotFound = "ServerNotFound";
```

```
public void init(Preferences servicePreferences)
       throws PortletServiceUnavailableException {
// Set Mail Server name based on initialization parameters
server_name = servicePreferences.get("server_name", serverNotFound);

}

public boolean sendMail(String to, String from,String subject, String message)
{
   if(server_name.equalsIgnoreCase(serverNotFound)){
    return false;
   } else {
      return send(to, from, subject, message);
   }
}

private boolean send(String from, String to, String subject, String message)
{
   try
   {
   Properties props = System.getProperties();
   // -- Attaching to default Session, or we could start a new one --
   props.put("mail.smtp.host", server_name);
   Session session = Session.getDefaultInstance(props, null);

   // -- Create a new message --
   Message msg = new MimeMessage(session);

   // -- Set the FROM and TO fields --
   msg.setFrom(new InternetAddress(from));
   msg.setRecipients(MimeMessage.RecipientType.TO, InternetAddress.parse(to,
   false));

   // -- Set the subject and body text --
   msg.setSubject(subject);
   msg.setText(message);

   // -- Set some other header information --
   msg.setHeader("X-Mailer", "LOTONtechEmail");
   msg.setSentDate(new Date());

   // -- Send the message --
   Transport.send(msg);

   return true;
   }
   catch (Exception ex)
   {
   ex.printStackTrace();
```

Archived

```
   return false;
}
}
}
```

3. Using Portlet Object - optional

   If you are creating a portlet service for both portlet APIs, you can use the same implementation class for both. If any method take a portlet object, they will have different signatures for the two. One will be a JSR 168 package and the other will be an IBM package.

   You could provide a single implementation method by converting the IBM portlet objects to JSR 168 portlet object. WebSphere Portal provides the com.ibm.portal.portlet.apiconvert.APIConverterFactory class. This class includes methods for converting the IBM portlet object to the appropriate JSR 168 portlet object. Example 19-5 demonstrates using the APIConverterFactory to convert the PortletRequest and PortletResponse to a RenderRequest and RenderResponse object.

Example 19-5 APIConverterFactory

```
public void message(org.apache.jetspeed.portlet.PortletRequest request,
      org.apache.jetspeed.portlet.PortletResponse response)
      throws IOException {
message(APIConverterFactory.getInstance().getRenderRequest(request),
      APIConverterFactory.getInstance().getRenderResponse(response));

}
```

4. Register the service

   Once the service interface has been defined and the implementation class created, the classes should be packaged into a jar file. This jar should be placed in the <WPS-ROOT>shared\app directory.

   Once the files have been deployed, the service must be registered. Open the PortletServiceRegistryService.properties file in the <WP-ROOT>\shared\app\config\services directory. It is recommended that you make a backup of this file prior to modifying it. The service and its implementation be registered. The first entry will be your custom service and the second entry is your service implementation.

   IBM Portlets will register portlet services as it seen in Example 19-6 on page 624.

Archive

Example 19-6 Registering the service in PortletServiceRegistryService.properties

```
# MailService
com.yourco.services.mailservice.IBMMailService =
com.yourco.services.mailservice.MailServiceImpl
```

JSR 168 portlet services are registered using JNDI. The syntax is `jndi:service_interface = service_implementation`. Example 19-7 show the addition of registering the JSR 168 portlet service.

Note: The service implementation classes are the same for both portlet APIs in my example. In some case you may need to separate the service a implementations for each API.

Example 19-7 Registering JSR 168 Portlet Service

```
# MailService
jndi:com.yourco.services.mailservice.MailService =
com.yourco.services.mailservice.MailServiceImpl
com.yourco.services.mailservice.IBMMailService =
com.yourco.services.mailservice.MailServiceImpl
```

Initialization parameters are also supplied in the PortletServiceRegistryService.properties file as illustrated in Example 19-8. Accessing these parameters is illustrated in Example 19-4 on page 621.

Example 19-8 Setting parameters in PortletServiceRegistryService.properties

```
# MailService
jndi:com.yourco.services.mailservice.MailService =
com.yourco.services.mailservice.MailServiceImpl
com.yourco.services.mailservice.IBMMailService =
com.yourco.services.mailservice.MailServiceImpl
com.yourco.services.mailservice.MailServiceImpl.server_name =
your_server_name
```

5. Test the service

   In order for the service to become available in the Portal, the Portal Server must be restarted. Restart the WebSphere Portal Application Server. Example 19-9 on page 625 shows a simple JSR 168 portlet making use of the MailService service.

Example 19-9 Using the MailService service

```
package jsrmailportlet;

import java.io.IOException;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.portlet.ActionRequest;
import javax.portlet.ActionResponse;
import javax.portlet.GenericPortlet;
import javax.portlet.PortletConfig;
import javax.portlet.PortletException;
import javax.portlet.PortletRequestDispatcher;
import javax.portlet.RenderRequest;
import javax.portlet.RenderResponse;

import com.ibm.portal.portlet.service.PortletServiceHome;
import com.yourco.services.mailservice.MailService;

public class JSRMailPortletPortlet extends GenericPortlet {




   public static final String VIEW_JSP     = "JSRMailPortletPortletView";
   public static final String FORM_SUBMIT   =
"JSRMailPortletPortletFormSubmit";
   public static final String FORM_To       = "JSRMailPortletPortletFormTo";
   public static final String FORM_From     =
"JSRMailPortletPortletFormFrom";
   public static final String FORM_Subject   =
"JSRMailPortletPortletFormSubject";
   public static final String FORM_Message   =
"JSRMailPortletPortletFormMessage";
   private PortletServiceHome mailServiceHome;

public void init(PortletConfig config) throws PortletException {
      super.init(config);
      try {
         Context ctx = new InitialContext();
         Object home =
ctx.lookup("portletservice/com.yourco.services.mailservice.MailService");
         if(home != null){
            mailServiceHome = (PortletServiceHome)home;
         }
      } catch (NamingException e) {
         e.printStackTrace();
```

Archived

```
      }

   }

   public void doView(RenderRequest request, RenderResponse response) throws
PortletException, IOException {
      // Set the MIME type for the render response
      response.setContentType(request.getResponseContentType());

       //test for failure
      if(request.getParameter("error") == null){
      // Invoke the JSP to render
      PortletRequestDispatcher rd =
getPortletContext().getRequestDispatcher("/jsrmailportlet/jsp/html/"+VIEW_JSP+"
.jsp");
      rd.include(request,response);
      } else if(request.getParameter("error").equalsIgnoreCase("true")){
         response.getWriter().println("Error in sending mail!");
      }
   }

   public void processAction(ActionRequest request, ActionResponse response)
throws PortletException, java.io.IOException {
      if( request.getParameter(FORM_SUBMIT) != null ) {
         // Set form text in the session bean
         if(mailServiceHome != null){
            MailService mailService =
(MailService)mailServiceHome.getPortletService(MailService.class);
            String to      = request.getParameter(FORM_To);
            String from    = request.getParameter(FORM_From);
            String subject = request.getParameter(FORM_Subject);
            String message = request.getParameter(FORM_Message);

            if(!mailService.sendMail(to, from, subject, message)){
               //set render parameter
               response.setRenderParameter("error", "true");
            }
         }

      }
   }
}
```

19.2 Accessing portlet services

Because of the inherit differences in the two portlet APIs, accessing portlet services in each API is different

19.2.1 Accessing a portlet service in an IBM portlet

The IBM portlet API is tightly coupled with WebSphere Portal. The IBM portlet API has direct access to certain features that the JSR 168 API does not. To access a service in an IBM portlet you need to use the getService method on the PortletContext. Example 19-10 shows an example of accessing the MailService from an IBM portlet.

Example 19-10 Accessing MailService in an IBM portlet

```
IBMMailService mailService =
(IBMMailService)getPortletConfig().getContext().getService(IBMMailService.class
);
```

19.2.2 Accessing a portlet service in a JSR 168 portlet

JSR 168 portlets do not have direct access to the services like IBM portlets. Instead JSR 168 portlets must get the service by using JNDI. The first step is defining a class variable of type PortletServiceHome. Then in the init method your should perform the lookup. If the portlet service is found, you should then set the PortletServiceHome variable. To use the service in your code, first test to make sure that service was found by testing PortletServiceHome for null. If it is not null, you can use the PortletServiceHome object to get your portlet service. Once you have the service you will use in the same manner as in IBM portlets. Example 19-11 shows an example of accessing the MailService from a JSR 168 portlet.

Example 19-11 Accessing MailService in a JSR 168 portlet

```
....
private PortletServiceHome mailServiceHome;
....
public void init(PortletConfig config) throws PortletException {
   super.init(config);
   try {
   Context ctx = new InitialContext();
   Object home =
   ctx.lookup("portletservice/com.yourco.services.mailservice.MailService");
      if(home != null){
         mailServiceHome = (PortletServiceHome)home;
      }
```

```
   } catch (NamingException e) {
      e.printStackTrace();
   }
}
.....

if(mailServiceHome != null){
   MailService mailService =
(MailService)mailServiceHome.getPortletService(MailService.class);
.....
```

Archived

20

Credential Vault Service

In this chapter, we will discuss the Credential Vault service, its purpose and the APIs involved.

Archived

© Copyright IBM Corp. 2005. All rights reserved.

20.1 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, hiding the login challenge from 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

20.1.1 Credentials

Portlets running on WebSphere Portal may need to access remote applications that require some form of authentication by using appropriate 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 20-1.

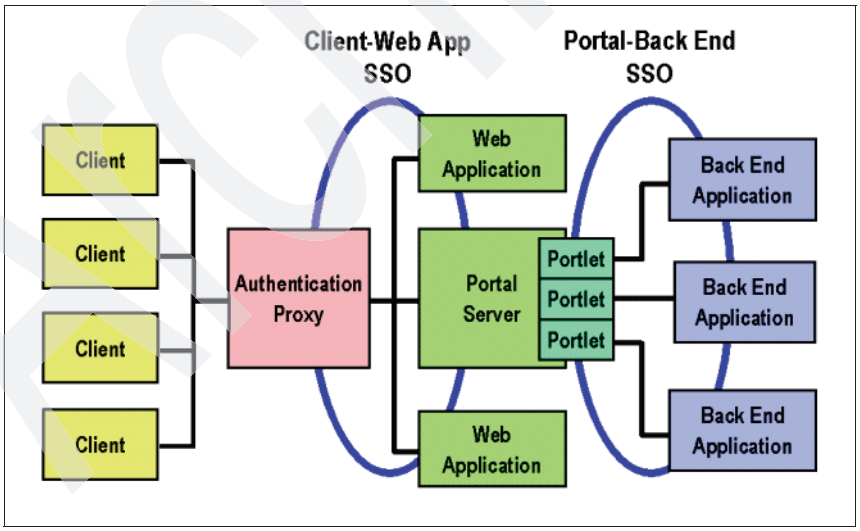


Figure 20-1 Credential Vault in action

The Credential Vault provides this functionality and portlets can use it through the Credential Vault Service.

20.2 Credential Vault organization

The organization of Credential Vault in WebSphere Portal consists of vault segments and credential slots. Figure 20-2 shows an overview of these components.

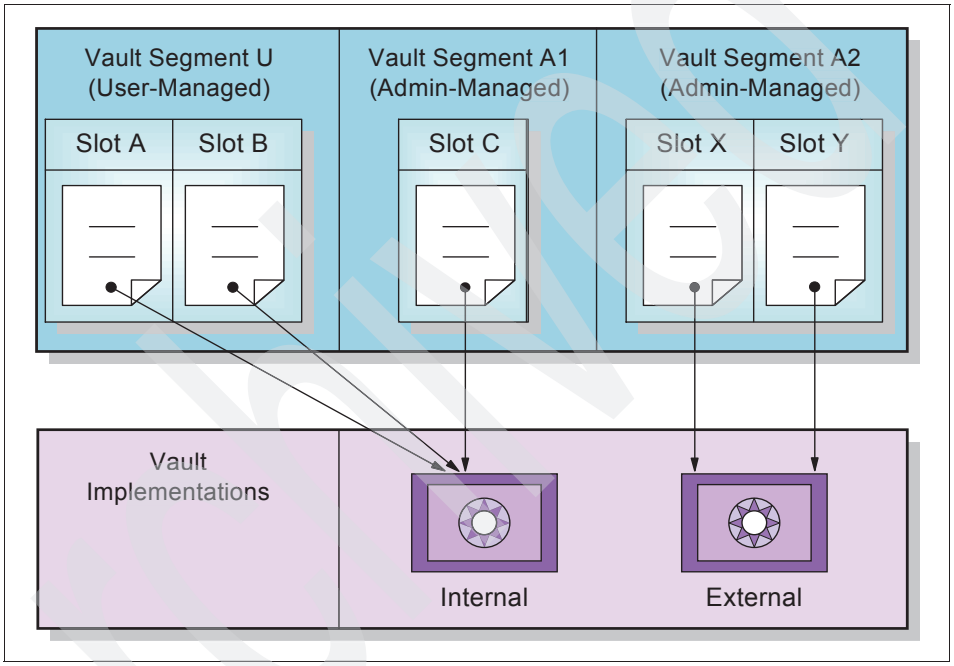


Figure 20-2 Credential Vault organization

The portal server's credential vault is organized as follows:

- The portal administrator can partition the vault into several vault segments. Vault segments can be created and configured only by portal administrators.
- A vault segment contains one or more vault slots. Vault slots are the "drawers" where portlets store and retrieve a user's credentials. Each slot holds one credential.
- A vault slot is linked to a resource in a vault implementation.
  - A vault implementation is the place where users credentials are actually stored. Examples of vault implementations include the WebSphere Portal's default database vault or the Tivoli Access Manager lock box.

– The resource within the vault implementation corresponds to an application or backend system that requires its own authentication. Examples of resources include Lotus Notes, personnel records, or a bank account.

20.2.1 Vault segments

The Credential Vault is partitioned into segments and a vault segment contains one or more vault slots. Vault segments can be created and configured only by portal administrators.

There are two different types of vault segments:

- Administrator-managed vault segments: in this type of vault segment, the creation of new slots is restricted to the portlet administrator.
- User-managed vault 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.

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.

Figure 20-3 on page 633 presents the relationship between Vault segments and Vault implementations. Notice that there is only one user-managed vault segment and it resides in the vault provided by WebSphere Portal.

Archived

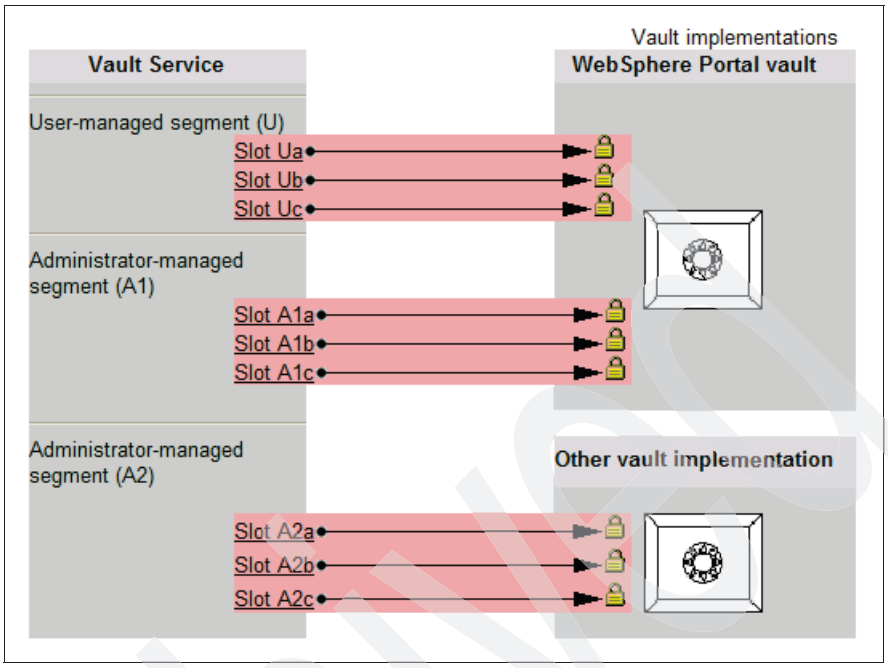


Figure 20-3 Vault segments and Vault implementations

20.2.2 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. It is a shared slot that belongs to an administrative segment.
- An administrative slot allows each user to store a secret for an administrator-defined resource (for example, Lotus Notes). It is an unshared slot that belongs to an administrative segment.
- A shared slot stores user credentials that are shared among the user's portlets. It is a shared slot that belongs to the user segment.
- A portlet private slot stores user credentials that are not shared among portlets. It is an unshared slot that belongs to the user segment.

You will find an example of using private slots in Chapter 21, “The Credential Vault” on page 641.

20.3 Working with the CredentialVaultService

To work with credentials, the first step is to acquire a reference to the `CredentialVaultService`. This section explains how to acquire a reference to a `CredentialVaultService` and its most important methods.

There are in fact two `CredentialVaultService` classes. One for each portlet API. In IBM Portlet API you will use the `com.ibm.wps.portletservice.credentialvault.CredentialVaultService`, whereas in JSR 168 you will use the `com.ibm.portal.portlet.service.credentialvault.CredentialVaultService`.

20.3.1 Acquiring a reference to the CredentialVaultService

There is a slight difference in the way you retrieve a reference to the `CredentialVaultService`. In JSR 168, you must use a JNDI lookup to access the service, as in Example 20-1.

Example 20-1 Acquiring a CredentialVaultService in JSR 168 API

```
com.ibm.portal.portlet.service.credentialvault.CredentialVaultService vaultService = null;
Context ctx = new InitialContext();
PortletServiceHome cvsHome =
(PortletServiceHome)ctx.lookup("portletservice/com.ibm.portal.portlet.service.credentialvault.C
redentialVaultService"); //$NON-NLS-1$
if (cvsHome != null) {
vaultService=(CredentialVaultService)cvsHome.getPortletService(CredentialVaultService.class);
}
```

In IBM Portlet API, the service is accessed by using `PortletContext.getService`, as in Example 20-2.

Example 20-2 Acquiring a CredentialVaultService in IBM Portlet API

```
com.ibm.wps.portletservice.credentialvault.CredentialVaultService vaultService =
      (CredentialVaultService)config.getContext().getService(CredentialVaultService.class);
```

20.3.2 Using the CredentialVaultService

The `CredentialVaultService` allows to retrieve credentials from the credential vault and to manage vault segments and slots. The most important methods provided by the `CredentialVaultService` are the following. For a complete list of all methods, refer to the API documentation:

- getCredential

This method returns a specific credential to be used in the application. It expects to receive the following parameters:

- `String` slotId: Identifies the slot where the credential will be retrieved from.
- `String` type: Identifies the type of credential to be retrieved. The valid types are the ones described in 20.4, “Credential objects” on page 635.
- `Map` config: List of name/value pairs. Typically, you will pass a `new HashMap()`.
- `PortletRequest` request: The `PortletRequest` object of the API being used.

- getUserSubject

  This method returns the user's JAAS subject. This is a special case of the getCredential call. It expects the `PortletRequest` as parameter.

- getCredentialTypes

  This method Returns an Iterator of Strings that represent all Credential Types that are registered in the Credential Type Registry.

- getAccessibleSlots

  This method returns a Iterator of all credential slots that a portlet is authorized to use. It expects the `PortletRequest` as parameter.

- setCredentialSecretBinary

  This method sets a credential's binary secret. It expects the slotId, the secret, and the `PortletRequest` as parameters

- setCredentialSecretUserPassword

  This method sets a credential's user/password secret. It expects the slotId, the user, the password, and the `PortletRequest` as parameters

- getAllVaultSegments

  This method returns a List of all Vault Segments.

Note: For details about implementing a solution using the CredentialVaultService, refer to Chapter 21, “The Credential Vault” on page 641.

20.4 Credential objects

The CredentialVault PortletService returns credentials in the form of credential objects. All credentials must implement the `Credential` interface. WebSphere Portal differentiates between passive and active credential objects.