



Red Hat JBoss Portal 6.2 Development Guide

For use with Red Hat JBoss Portal 6.2

Jared Morgan

Aakanksha Singh

Red Hat JBoss Portal 6.2 Development Guide

For use with Red Hat JBoss Portal 6.2

Jared Morgan
Red Hat, Ltd. Customer Content Services

Aakanksha Singh
Red Hat, Ltd. Customer Content Services

Legal Notice

Copyright © 2015 Red Hat, Inc..

This document is licensed by Red Hat under the [Creative Commons Attribution-ShareAlike 3.0 Unported License](#). If you distribute this document, or a modified version of it, you must provide attribution to Red Hat, Inc. and provide a link to the original. If the document is modified, all Red Hat trademarks must be removed.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, MetaMatrix, Fedora, the Infinity Logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack Logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

The Development Guide is intended for those developing portlet applications and gadgets for the product, and outlines options and strategies for successful deployment.

Table of Contents

| | |
|---|-----------|
| Preface | 13 |
| 1. Document Conventions | 13 |
| 1.1. Typographic Conventions | 13 |
| 1.2. Pull-quote Conventions | 14 |
| 1.3. Notes and Warnings | 15 |
| 2. Getting Help and Giving Feedback | 15 |
| 2.1. Do You Need Help? | 15 |
| 2.2. We Need Feedback | 16 |
| Part I. Architectural and Design Choices | 17 |
| Chapter 1. Architectural Choices | 18 |
| 1.1. Identity server | 18 |
| 1.2. Storage | 18 |
| 1.3. Cluster | 19 |
| 1.4. SSO | 19 |
| 1.4.1. Summary | 19 |
| Chapter 2. Design Choices | 22 |
| 2.1. Dashboards | 22 |
| 2.2. JCR Index Replication in a Cluster Setup | 22 |
| 2.2.1. JCR Index Replication in a Cluster Setup | 22 |
| 2.2.2. Standalone Index | 23 |
| 2.2.3. Local Index | 23 |
| 2.2.4. Shared Index | 24 |
| Part II. Portal API | 26 |
| 1. About Portal API | 26 |
| Chapter 3. PortalRequest and Portal | 27 |
| Chapter 4. Site | 28 |
| 4.1. Retrieving Sites | 28 |
| 4.2. Creating a Site | 29 |
| 4.2.1. Setting Attributes for a Site | 29 |
| 4.2.2. Setting permissions for a site | 29 |
| 4.3. Deleting a Site | 30 |
| Chapter 5. Navigation | 31 |
| 5.1. Retrieving Navigation Nodes | 31 |
| 5.1.1. NodeVisitor | 32 |
| 5.1.2. Filtering Navigation Nodes | 32 |
| 5.2. Creating a Navigation Node | 33 |
| 5.2.1. Navigation Node Visibility | 34 |
| 5.2.2. Localization | 34 |
| 5.3. Deleting a Navigation Node | 34 |
| 5.4. Moving a Navigation Node | 34 |
| Chapter 6. Page | 36 |
| 6.1. Retrieving Pages | 36 |
| 6.2. Creating a Page | 36 |
| 6.2.1. Setting Permissions for a Page | 37 |
| 6.3. Page Composition | 37 |
| 6.4. Deleting a Page | 38 |

| | |
|---|-----------|
| 6.5. Application | 39 |
| Chapter 7. OAuth Provider API | 40 |
| 7.1. Retrieve an Instance of OAuthProvider | 40 |
| 7.2. OAuthProvider Operations | 40 |
| 7.3. Access to Provider-specific Operations | 42 |
| Chapter 8. REST API | 44 |
| 8.1. Overview | 44 |
| 8.1.1. Base URL | 44 |
| 8.1.2. Authentication | 44 |
| 8.1.3. Content Type | 44 |
| 8.1.4. Localization | 45 |
| 8.2. Resources | 45 |
| 8.2.1. Sites | 45 |
| 8.2.1.1. List Sites | 45 |
| 8.2.1.2. Retrieve a Site | 46 |
| 8.2.1.3. Create a site | 47 |
| 8.2.1.4. Delete a site | 48 |
| 8.2.1.5. Update a site | 48 |
| 8.2.2. Pages | 49 |
| 8.2.2.1. List Pages | 50 |
| 8.2.2.2. Retrieve a page | 51 |
| 8.2.2.3. Create a page | 51 |
| 8.2.2.4. Delete a page | 51 |
| 8.2.2.5. Update a page | 52 |
| 8.2.3. Navigation | 52 |
| 8.2.3.1. Retrieve Navigation | 52 |
| 8.2.3.2. Retrieve a node | 54 |
| 8.2.3.3. Create a node | 56 |
| 8.2.3.4. Delete a node | 56 |
| 8.2.3.5. Update a node | 57 |
| 8.2.4. Working with Templates | 63 |
| 8.2.4.1. Portal Site Template | 63 |
| 8.2.4.2. Group Sites Template | 64 |
| 8.2.4.3. User Sites Template | 65 |
| Chapter 9. Organization API | 68 |
| 9.1. Organization API | 68 |
| Chapter 10. Accessing User Profile | 70 |
| Part III. Portal Development | 71 |
| Chapter 11. Portal Life-cycle | 72 |
| 11.1. Application Server Start and Stop | 72 |
| 11.1.1. Advanced WCI Registration | 72 |
| 11.2. The Command Servlet | 73 |
| Chapter 12. Portal Containers | 76 |
| Chapter 13. Skinning the Portal | 77 |
| 13.1. Skin Components | 77 |
| 13.2. Skin Selection | 78 |
| 13.2.1. Skin Selection Through the User Interface | 78 |
| 13.2.2. Setting the Default Skin within the Configuration Files | 78 |

| | |
|---|------------|
| 13.2.2. Setting the Default Skin within the Configuration Files | 78 |
| 13.3. Skins in Page Markups | 78 |
| 13.4. The Skin Service | 79 |
| 13.4.1. Skin configuration | 80 |
| 13.4.2. Resource Request Filter | 80 |
| 13.5. The Default Skin | 81 |
| 13.6. Creating New Skins | 82 |
| 13.6.1. Creating New Skins | 82 |
| 13.6.2. Creating a New Portal Skin | 82 |
| 13.6.2.1. Portal Skin Configuration | 82 |
| 13.6.2.2. Portal Skin Preview Icon | 82 |
| 13.6.3. Creating a New Window Style | 84 |
| 13.6.3.1. Window Style Configuration | 84 |
| 13.6.3.2. Window Style CSS | 85 |
| 13.6.3.3. How to Set the Default Window Style | 87 |
| 13.6.4. How to Create New Portlet Skins | 88 |
| 13.6.4.1. Define a Custom CSS File | 88 |
| 13.6.4.2. Change Portlet Icons | 90 |
| 13.6.5. Create New Portlet Specification CSS Classes | 90 |
| 13.7. Tips and Tricks | 90 |
| 13.7.1. CSS Hosted on CDN | 90 |
| 13.7.2. Easier CSS Debugging | 91 |
| 13.7.3. Some CSS Techniques | 91 |
| 13.7.3.1. Decorator Pattern | 92 |
| 13.7.3.2. Left Margin Left Pattern | 93 |
| Chapter 14. Portal Extension | 94 |
| 14.1. How the Shadowing Mechanism Works | 94 |
| 14.2. Custom Groovy Template for a Portlet | 97 |
| 14.3. Custom Skin for a Portlet | 98 |
| 14.3.1. gatein-resources.xml | 99 |
| 14.3.2. CSS and Images | 100 |
| 14.4. Custom Navigation and Pages | 100 |
| 14.4.1. portal-configuration.xml | 100 |
| 14.5. Navigation Node Types | 100 |
| 14.6. Internationalization of Navigation Nodes | 105 |
| 14.7. Custom Internationalization Resource Bundles | 106 |
| 14.8. Custom Sign-in Page | 108 |
| Chapter 15. Visual Identity | 110 |
| Chapter 16. Data Import Strategy | 111 |
| 16.1. Import Strategy Overview | 111 |
| 16.1.1. Portal Configuration | 112 |
| 16.1.2. Page Data | 112 |
| 16.1.3. Navigation Data | 112 |
| Chapter 17. Right To Left (RTL) Framework | 117 |
| 17.1. Groovy templates | 117 |
| 17.2. Stylesheet | 117 |
| 17.3. Images | 118 |
| 17.4. Client Side JavaScript | 119 |
| Chapter 18. XML Resources Bundles | 120 |
| 18.1. XML format | 120 |

| | |
|--|------------|
| 18.2. Portal Support | 121 |
| Chapter 19. Navigation Controller | 122 |
| 19.1. Controller Configuration (controller.xml) | 123 |
| 19.1.1. Rendering | 127 |
| 19.1.1.1. Portal URL | 128 |
| 19.1.1.2. Node URL | 129 |
| 19.1.1.3. Component URL | 129 |
| 19.1.1.4. Portlet URL | 130 |
| 19.1.1.5. WebUI URL Builder | 130 |
| 19.1.1.6. Groovy Templates | 130 |
| Part IV. Gadget Development | 132 |
| Chapter 20. Gadgets in Portal | 133 |
| 20.1. Sources to Develop Gadgets | 133 |
| Chapter 21. Portlet Development Resources | 134 |
| 21.1. JSR-168 and JSR-286 overview | 134 |
| 21.1.1. Portal Pages | 134 |
| 21.1.2. Rendering Modes | 135 |
| 21.1.3. Window States | 135 |
| 21.2. Tutorials | 136 |
| 21.2.1. Deploying your first portlet | 136 |
| 21.2.1.1. Compiling | 136 |
| 21.2.1.2. Package Structure | 136 |
| 21.2.1.3. Portlet Class | 137 |
| 21.2.1.4. Application Descriptors | 138 |
| 21.2.2. JavaServer Pages Portlet Example | 140 |
| 21.2.2.1. Package Structure | 141 |
| 21.2.2.2. Portlet Class | 141 |
| 21.2.2.3. JSP files and the Portlet Tag Library | 144 |
| Chapter 22. Portlet Development | 146 |
| 22.1. Starting a Portlet Project | 146 |
| 22.1.1. The Bill of Materials (BOM) Concept | 146 |
| 22.1.2. Using the BOM | 146 |
| 22.2. Building and Deploying Portlets | 148 |
| 22.3. Standard Portlet Development (JSR-286) | 149 |
| 22.3.1. Java Configuration | 149 |
| 22.3.2. portlet.xml | 150 |
| 22.3.3. web.xml | 151 |
| 22.3.4. Building and Deploying Portlets | 151 |
| Chapter 23. Basic JSF Portlet Development | 153 |
| 23.1. Example Code | 153 |
| 23.1.1. pom.xml | 153 |
| 23.1.2. JSF Template Files | 153 |
| 23.1.3. Java Beans | 155 |
| 23.1.4. portlet.xml | 156 |
| 23.1.5. web.xml | 157 |
| 23.1.6. Custom CSS | 158 |
| 23.1.7. Internationalization | 158 |
| 23.2. Further Steps | 159 |
| 23.3. See also | 159 |

| | |
|---|------------|
| Chapter 24. JSF Portlet Development with RichFaces | 160 |
| 24.1. Example Code | 160 |
| 24.1.1. pom.xml | 160 |
| 24.1.2. JSF Template Files | 161 |
| 24.1.3. Java Beans | 161 |
| 24.1.4. portlet.xml | 163 |
| 24.1.5. web.xml | 164 |
| 24.1.6. Custom CSS | 166 |
| 24.1.7. Internationalization | 166 |
| 24.2. Further Steps | 166 |
| 24.3. See also | 166 |
| Chapter 25. Portlets with JSF and CDI | 167 |
| Chapter 26. CDI Portlet Development | 168 |
| 26.1. GenericPortlet and Portlet Filter Injection | 168 |
| 26.2. Portlet CDI Scopes | 168 |
| 26.2.1. @PortletLifecycleScoped | 169 |
| 26.2.2. @PortletRedisplayScoped | 169 |
| Chapter 27. Portlet Filter | 171 |
| 27.1. Global Metadata Elements | 171 |
| 27.1.1. Global Metadata Elements | 171 |
| 27.1.2. Configuring a Portlet Filter | 171 |
| Chapter 28. Portlet Bridge | 173 |
| 28.1. JBoss Portlet Bridge | 173 |
| 28.2. Portlet application | 173 |
| 28.3. Extensions | 173 |
| 28.4. Examples | 173 |
| 28.5. Render Policy Parameters | 174 |
| 28.6. Facelets Configuration | 174 |
| 28.7. JSP-only Configuration | 175 |
| 28.8. RichFaces Local and Remote Portlet Support | 175 |
| 28.9. Sending and Receiving Events | 177 |
| 28.10. Sending Events | 178 |
| 28.11. Receiving Events | 179 |
| 28.12. Public Render Parameters | 180 |
| 28.13. Saving Bridge Request Scope after Render complete | 181 |
| 28.14. PRP portlet configuration | 181 |
| 28.15. Application configuration | 182 |
| 28.16. Portlet Session | 183 |
| 28.17. Resource serving | 184 |
| 28.18. Serving JSF Resources in a Portlet | 184 |
| 28.19. Expression Language Reference | 184 |
| 28.20. Expression Language Configuration | 186 |
| 28.21. Developing Portlets with the Bridge | 187 |
| 28.21.1. Implementing Portlet Bridge | 187 |
| 28.21.2. Declaring Artifact Dependencies | 187 |
| 28.21.3. Declaring Depchain Dependencies | 188 |
| 28.21.4. Deploying Portlet Bridge Portlets | 188 |
| 28.21.5. Disable Automatic Portlet Bridge Injection | 189 |
| 28.21.6. Supported Portlet Tags | 189 |
| 28.21.7. Excluding Attributes from the Bridge Request Scope | 190 |

| | |
|--|------------|
| 28.21.7. Extending Resources from the Page Request Scope | 190 |
| 28.21.8. Prevent Resources Being Added to Portal Page Head | 191 |
| 28.21.9. JSF Facelet View | 191 |
| 28.21.10. Error Handling | 192 |
| 28.21.11. Switching Portlet Modes | 192 |
| 28.21.12. Navigating to a mode's last viewId | 193 |
| 28.21.13. Using Wildcards to Identify the Rule Target | 193 |
| 28.21.14. Clearing the View History when Changing Portlet Modes | 194 |
| 28.21.15. Communication Between Portlets | 194 |
| 28.21.16. Storing Components in PortletSession.APPLICATION_SCOPE | 194 |
| 28.21.17. Using the PortletSession | 195 |
| 28.21.18. Linking to a Facelets page within the Same Portlet | 195 |
| 28.21.19. Redirecting to an External Page or Resource | 195 |
| 28.21.20. Using Provided EL Variables | 196 |
| Chapter 29. Navigation Portlet Using the Public API | 199 |
| 29.1. Example Code | 199 |
| 29.1.1. Required Java Classes | 199 |
| 29.1.2. Required JSP | 200 |
| 29.1.3. Required JavaScript | 202 |
| 29.1.4. Further Steps | 203 |
| 29.2. See also | 203 |
| Chapter 30. Using Data from Social Networks in Portlets | 204 |
| 30.1. Social Portlets Example Code | 204 |
| 30.1.1. Prerequisites | 204 |
| 30.2. Retrieving the Access Token | 204 |
| 30.3. Facebook | 207 |
| 30.4. Google+ | 207 |
| 30.5. Twitter | 208 |
| 30.6. Configuration and Deployment Descriptors | 208 |
| 30.7. Further Steps | 210 |
| Chapter 31. Spring Framework Portlet Development | 211 |
| Part V. JavaScript Development | 213 |
| Chapter 32. JavaScript Modularity | 214 |
| 32.1. JavaScript Modules | 214 |
| 32.2. Introduction to Modules | 214 |
| 32.3. Script Support | 215 |
| Chapter 33. JavaScript in JBoss Portal | 217 |
| 33.1. Declaring a Module | 217 |
| 33.2. Translation to the AMD System | 218 |
| 33.3. Producing Dependencies | 219 |
| Chapter 34. Native AMD Modules | 220 |
| 34.1. Difference in Native AMD and Portal Modules | 220 |
| 34.2. Native AMD Modules hosted on CDN | 220 |
| 34.3. Mapping path entries for AMD Modules | 221 |
| 34.4. Fallback Paths | 221 |
| 34.5. Example to Ensure Portal wide Consistency in mapping paths | 222 |
| Chapter 35. Module Scopes | 223 |
| 35.1. Shared Scope | 223 |

| | |
|--|------------|
| 35.2. Portal Scope | 223 |
| 35.3. Portlet Scope | 223 |
| Chapter 36. Portlet Dynamic Module | 225 |
| Chapter 37. Aliases | 226 |
| Chapter 38. Custom Adapters | 227 |
| Chapter 39. Module Resources | 228 |
| Chapter 40. Load Groups | 230 |
| Chapter 41. Localization | 231 |
| Chapter 42. Non-AMD Scripts | 232 |
| Part VI. JavaScript Code Examples | 233 |
| Chapter 43. Module Code Examples | 234 |
| 43.1. Declare a Module | 234 |
| 43.2. Declare an AMD Module | 235 |
| 43.3. Use jQuery Module | 235 |
| 43.4. Use a Custom jQuery Version | 236 |
| 43.5. jQuery Plug-ins | 237 |
| 43.6. Override Native AMD Module Dependency | 238 |
| 43.7. CommonJS modules | 238 |
| 43.8. Mustache.js module | 240 |
| 43.9. Resource loading with Text.js | 241 |
| Chapter 44. Script Code Examples | 243 |
| 44.1. Accessing a module from a script | 243 |
| 44.2. Exposing jQuery Version Globally | 243 |
| 44.3. Defining a custom global jQuery | 244 |
| 44.4. Using a Global jQuery plugin | 244 |
| Part VII. Advanced Development Concepts | 246 |
| The eXo Kernel | 247 |
| A.1. The eXo Kernel | 247 |
| A.2. Kernel Configuration Namespace | 249 |
| A.3. Configuration Retrieval | 249 |
| A.4. PortalContainer Advanced Concepts | 251 |
| A.4.1. Add new configuration files from a WAR file | 251 |
| A.4.2. Creating PortalContainers from a WAR File | 252 |
| A.4.3. Defining a PortalContainer with its dependencies and its settings | 252 |
| A.4.4. PortalContainer Settings | 259 |
| A.4.5. Dynamically Changing a PortalContainerDefinition | 261 |
| A.4.5.1. PortalContainerDefinitionChange Implementations | 263 |
| A.4.5.1.1. AddDependencies | 263 |
| A.4.5.1.2. AddDependenciesBefore | 264 |
| A.4.5.1.3. AddDependenciesAfter | 266 |
| A.4.5.1.4. AddSettings | 267 |
| A.4.6. Dynamically Disable a Portal Container | 268 |
| A.5. Runtime Configuration Profiles | 270 |
| A.6. Component request life cycle | 272 |
| A.6.1. Component request life cycle contract | 272 |

| | |
|--|-----|
| A.6.2. Request life cycle | 273 |
| A.6.2.1. Scheduling a component request life cycle | 273 |
| A.6.2.2. Scheduling a container request life cycle | 273 |
| A.6.2.3. When request life cycle is triggered | 274 |
| A.6.2.3.1. Portal request life cycle | 274 |
| A.6.2.3.2. JMX request Life Cycle | 274 |
| A.7. Configuring Services | 274 |
| A.7.1. Configuration syntax | 275 |
| A.7.1.1. Components | 275 |
| A.7.1.1.1. RootContainer | 277 |
| A.7.1.1.2. PortalContainer | 278 |
| A.7.1.1.3. External Plug-ins | 279 |
| A.7.1.1.4. Service instantiation | 280 |
| A.7.1.1.5. Service Access | 281 |
| A.7.1.1.6. Includes, and special URLs | 282 |
| A.7.1.1.7. Special variables | 283 |
| A.7.1.2. <init-params> configuration element | 284 |
| A.7.1.2.1. Collection | 290 |
| A.7.1.3. Component Plug-in Priority | 292 |
| A.7.1.4. Configuration Logging | 293 |
| A.7.1.5. Import | 293 |
| A.7.1.6. System properties | 294 |
| A.8. Specific Services | 295 |
| A.8.1. ListenerService | 295 |
| A.8.1.1. Configuring a Listener | 295 |
| A.8.1.2. Registering a Listener | 296 |
| A.8.1.3. Triggering an Event | 296 |
| A.8.1.4. Asynchronous Event Broadcast | 298 |
| A.8.1.5. ListenerService Example | 298 |
| A.8.2. Job Scheduler | 299 |
| A.8.2.1. Using the Job Scheduler Service in the Kernel | 300 |
| A.8.2.2. Samples | 300 |
| A.8.2.2.1. Define a Job | 301 |
| A.8.2.2.2. Configuring a Job | 302 |
| A.8.2.2.3. Run the Project | 304 |
| A.8.2.3. Job Scheduler Reference | 305 |
| A.8.3. The data source provider | 306 |
| A.8.3.1. DataSourceProvider | 306 |
| A.8.3.2. Configuring DataSourceProvider | 306 |
| A.9. Configuring a portal container | 307 |
| A.10. System property configuration | 310 |
| A.10.1. Properties <init-param> | 311 |
| A.10.2. Properties URL <init-param> | 311 |
| A.10.3. System Property configuration of the properties URL | 312 |
| A.10.4. Variable Syntaxes | 312 |
| A.11. The Extension Mechanism and Portal Extensions | 312 |
| A.11.1. Running Multiple Portals | 313 |
| A.12. Manageability | 316 |
| A.12.1. Introduction | 316 |
| A.12.2. Managed framework API | 316 |
| A.12.2.1. Annotations | 317 |
| A.12.2.1.1. @org.exoplatform.management.annotations.Managed annotation | 317 |
| A.12.2.1.2. @org.exoplatform.management.annotations.ManagedDescription | 318 |

| | |
|--|------------|
| A.12.2.1.3. @org.exoplatform.management.annotations.ManagedName | 318 |
| A.12.2.1.4. @org.exoplatform.management.annotations.ManagedBy | 318 |
| A.12.3. JMX Management View | 318 |
| A.12.3.1. JMX Annotations | 318 |
| A.12.3.1.1. @org.exoplatform.management.jmx.annotations.Property annotation | 319 |
| A.12.3.1.2. @org.exoplatform.management.jmx.annotations.NameTemplate annotation | 319 |
| A.12.3.1.3. @org.exoplatform.management.jmx.annotations.NamingContext annotation | 319 |
| A.12.4. Example | 319 |
| A.12.4.1. CacheService example | 319 |
| eXo JCR | 322 |
| B.1. Introduction and Support Scope | 322 |
| B.2. Concepts | 322 |
| B.2.1. Repository and Workspace | 322 |
| B.2.2. Items | 323 |
| B.2.3. The Data Model | 323 |
| B.2.4. Data Abstraction | 323 |
| B.3. eXo JCR Repository Service Components | 324 |
| B.3.1. Workspace Data Model | 325 |
| B.4. Template for JCR configuration file | 327 |
| B.4.1. Portal Configuration for JCR | 328 |
| B.4.1.1. JCR Workspace Configuration | 330 |
| B.4.1.2. JCR Repository Service Configuration | 331 |
| B.4.1.3. Workspace Configuration Parameters | 334 |
| B.4.1.4. Workspace Data Container Configuration Parameters | 335 |
| B.4.1.5. Value Storage plug-in Configuration Parameters | 336 |
| B.4.1.6. Initializer Configuration Parameters | 336 |
| B.4.1.7. Cache Configuration Parameters | 337 |
| B.4.1.8. Query Handler Configuration Parameters | 338 |
| B.4.1.9. Lock Manager Configuration Parameters | 338 |
| B.5. Multi-language Support | 338 |
| B.5.1. Oracle | 339 |
| B.5.2. DB2 | 340 |
| B.5.3. MySQL | 340 |
| B.5.4. PostgreSQL | 341 |
| B.6. Configuring Search | 342 |
| B.6.1. Global Search Index | 345 |
| B.6.1.1. Customized Search Indexes and Analyzers | 346 |
| B.6.1.2. Creating a Customized Query Handler | 347 |
| B.6.1.3. Configuring an application to use the new SearchIndex | 347 |
| B.6.2. Indexing Configuration | 348 |
| B.6.2.1. Node Scope Limit | 348 |
| B.6.2.2. Configuring Index Boost Value | 349 |
| B.6.2.3. Adding Condition to Index Rules | 349 |
| B.6.2.4. Exclusion from the Node Scope Index | 351 |
| B.6.2.5. Characteristics of Node Scope Searches | 351 |
| B.6.3. Advanced features | 353 |
| B.7. Configuring the JDBC Data Container | 353 |
| B.7.1. Introduction | 353 |
| B.7.2. Supported Databases | 354 |
| B.7.3. Configuring the database using SQL-script | 354 |
| B.7.4. Multilanguage support database configuration | 355 |
| B.7.5. Isolated-database Configuration | 356 |
| B.7.6. Multi-database Configuration | 356 |

| | |
|---|-----|
| B.7.6. Multi-database Configuration | 358 |
| B.7.7. Single-database Configuration | 362 |
| B.7.7.1. Configuration Without DataSource | 364 |
| B.7.7.2. Dynamic Workspace Creation | 365 |
| B.7.8. Simple and Complex queries | 365 |
| B.7.9. Force Query Hints | 366 |
| B.7.10. Notes for Microsoft Windows Users | 366 |
| B.8. External Value Storages | 367 |
| B.8.1. Tree File Value Storage | 367 |
| B.8.2. Disabling Value Storage | 369 |
| B.9. Workspace Data Container | 370 |
| B.10. Configuring the Cluster | 373 |
| B.10.1. Launching Cluster | 373 |
| B.10.1.1. Configuring JCR to use external configuration | 373 |
| B.10.2. Requirements | 375 |
| B.10.2.1. Environment requirements | 375 |
| B.10.2.2. Configuration Guidelines | 376 |
| B.11. Configuring JBoss Cache | 378 |
| B.11.1. Indexer lock manager and data container configuration | 378 |
| B.11.2. JGroups configuration | 378 |
| B.11.3. Sharing JBoss Cache instances | 379 |
| B.11.4. Shipped JBoss Cache configuration templates | 379 |
| B.11.4.1. Data container template | 380 |
| B.11.4.2. Lock manager template | 380 |
| B.11.4.3. Query Handler Template | 382 |
| B.12. LockManager | 383 |
| B.12.1. CacheableLockManagerImpl | 383 |
| B.12.2. JBoss Cache Configuration | 384 |
| B.12.3. Configuration of JBoss Cache for LockManager | 384 |
| B.12.4. LockManager Configuration Template | 385 |
| B.12.5. Creating udp-mux.xml | 386 |
| B.12.6. FQN type and node type in different Databases | 389 |
| B.12.7. Lock Migration | 389 |
| B.13. JCR Indexing | 390 |
| B.13.1. Standalone Index | 390 |
| B.13.2. Local Index | 390 |
| B.13.3. Shared Index | 391 |
| B.13.4. RSync-based Index | 392 |
| B.13.5. Query-handler configuration | 393 |
| B.13.5.1. Configuration properties | 393 |
| B.13.5.2. Improve Query Performance with postgresSQL and rdbms-reindexing | 394 |
| B.13.5.3. Improve Query Performance with DB2 and rdbms-reindexing | 395 |
| B.13.5.4. Cluster-ready indexing for shared index | 395 |
| B.13.5.5. System Requirements for RSync Index | 396 |
| B.13.5.6. RSync Index Configuration | 396 |
| B.13.5.7. Cluster-ready indexing for local index | 397 |
| B.13.5.8. Local Index Recovery Filters | 398 |
| B.13.5.9. Filter Implementations | 398 |
| B.13.5.10. JBoss-Cache template configuration | 399 |
| B.13.6. Asynchronous Re-indexing | 400 |
| B.13.6.1. On startup indexing | 400 |
| B.13.6.2. Asynchronous Indexing on startup | 400 |
| B.13.6.3. Hot Asynchronous Workspace Re-indexing using JMX | 401 |
| B.13.7. Indexing | 402 |

| | |
|---|------------|
| B.13.6.4. Notices | 403 |
| B.13.7. Lucene tuning | 403 |
| B.13.7.1. JBossTransactionsService | 404 |
| B.13.7.2. JCR Query Use-cases | 404 |
| B.13.8. Searching Repository Content | 405 |
| B.13.8.1. Bi-directional Rangelterator | 406 |
| B.13.8.2. Fuzzy Searches | 406 |
| B.13.8.3. Synonym Search | 407 |
| B.13.9. Highlighting | 408 |
| B.13.9.1. DefaultXMLExcerpt | 408 |
| B.13.9.2. DefaultHTMLExcerpt | 409 |
| B.13.9.3. Usage | 409 |
| B.13.10. SpellChecker | 410 |
| B.13.10.1. Spell check Usage | 410 |
| B.13.11. Similarity | 411 |
| B.14. Full Text Search And Affecting Settings | 412 |
| B.14.1. Lucene Analyzers | 412 |
| B.14.2. Property Indexing | 413 |
| B.14.3. Different Analyzers | 414 |
| B.15. WebDAV | 415 |
| B.15.1. WebDAV Configuration | 415 |
| B.15.2. WebDAV and JCR Actions | 417 |
| B.15.3. WebDAV Limitation on Windows | 418 |
| B.15.4. WebDAV Limitation for Microsoft Office 2010 | 418 |
| B.16. FTP | 419 |
| B.17. Use External Backup Tool | 421 |
| B.17.1. Repository Suspending | 421 |
| B.17.2. Backup Considerations | 422 |
| B.17.3. Repository Resuming | 423 |
| B.18. eXo JCR statistics | 423 |
| B.18.1. Statistics on the Database Access Layer | 423 |
| B.18.1.1. Database Access Layer Data | 423 |
| B.18.1.2. Enabling Statistics for Database Access Layer | 423 |
| B.18.1.3. Database Access Layer Methods and their alias | 424 |
| B.18.2. Statistics on the JCR API Accesses | 424 |
| B.18.2.1. Enabling Statistics on the JCR API Accesses | 425 |
| B.18.2.2. CSV files used in Statistics | 426 |
| B.18.3. Statistics Manager | 426 |
| B.18.3.1. Accessing Statistics using JMX | 427 |
| B.19. Checking Repository Integrity and Consistency | 428 |
| B.19.1. JMX-based consistency tool | 428 |
| B.20. JCR Performance Tuning Guide | 429 |
| B.20.1. Cluster configuration | 429 |
| B.20.2. JCR Clustered Performance | 430 |
| B.20.3. JBoss Enterprise Application Platform 6 Tuning | 432 |
| B.20.4. JCR Cache Tuning | 432 |
| B.20.5. Clustering Tuning | 432 |
| B.20.6. Declaring the Datasources in the Application Server | 433 |
| B.20.6.1. Binding Datasources | 434 |
| Quickstarts | 436 |
| C.1. Quickstarts | 436 |
| C.2. Quickstarts Downloads Page | 436 |
| C.3. JBoss Developer Studio or Eclipse with JBoss Tools | 436 |

| | |
|---|------------|
| 6.3. JBoss Developer Studio or Eclipse with JBoss Tools | 436 |
| Revision History | 437 |

Preface

1. Document Conventions

This manual uses several conventions to highlight certain words and phrases and draw attention to specific pieces of information.

1.1. Typographic Conventions

Four typographic conventions are used to call attention to specific words and phrases. These conventions, and the circumstances they apply to, are as follows.

Mono-spaced Bold

Used to highlight system input, including shell commands, file names and paths. Also used to highlight keys and key combinations. For example:

To see the contents of the file **my_next_bestselling_novel** in your current working directory, enter the **cat my_next_bestselling_novel** command at the shell prompt and press **Enter** to execute the command.

The above includes a file name, a shell command and a key, all presented in mono-spaced bold and all distinguishable thanks to context.

Key combinations can be distinguished from an individual key by the plus sign that connects each part of a key combination. For example:

Press **Enter** to execute the command.

Press **Ctrl+Alt+F2** to switch to a virtual terminal.

The first example highlights a particular key to press. The second example highlights a key combination: a set of three keys pressed simultaneously.

If source code is discussed, class names, methods, functions, variable names and returned values mentioned within a paragraph will be presented as above, in **mono-spaced bold**. For example:

File-related classes include **filesystem** for file systems, **file** for files, and **dir** for directories. Each class has its own associated set of permissions.

Proportional Bold

This denotes words or phrases encountered on a system, including application names; dialog-box text; labeled buttons; check-box and radio-button labels; menu titles and submenu titles. For example:

Choose **System** → **Preferences** → **Mouse** from the main menu bar to launch **Mouse Preferences**. In the **Buttons** tab, select the **Left-handed mouse** check box and click **Close** to switch the primary mouse button from the left to the right (making the mouse suitable for use in the left hand).

To insert a special character into a **gedit** file, choose **Applications** → **Accessories** → **Character Map** from the main menu bar. Next, choose **Search** → **Find...** from the **Character Map** menu bar, type the name of the character in the **Search** field and click **Next**. The character you sought will be highlighted in the

Character Table. Double-click this highlighted character to place it in the **Text to copy** field and then click the **Copy** button. Now switch back to your document and choose **Edit** → **Paste** from the **gedit** menu bar.

The above text includes application names; system-wide menu names and items; application-specific menu names; and buttons and text found within a GUI interface, all presented in proportional bold and all distinguishable by context.

Mono-spaced Bold Italic* or *Proportional Bold Italic

Whether mono-spaced bold or proportional bold, the addition of italics indicates replaceable or variable text. Italics denotes text you do not input literally or displayed text that changes depending on circumstance. For example:

To connect to a remote machine using ssh, type **ssh *username@domain.name*** at a shell prompt. If the remote machine is **example.com** and your username on that machine is john, type **ssh john@example.com**.

The **mount -o remount *file-system*** command remounts the named file system. For example, to remount the **/home** file system, the command is **mount -o remount /home**.

To see the version of a currently installed package, use the **rpm -q *package*** command. It will return a result as follows: ***package-version-release***.

Note the words in bold italics above: *username*, *domain.name*, *file-system*, *package*, *version* and *release*. Each word is a placeholder, either for text you enter when issuing a command or for text displayed by the system.

Aside from standard usage for presenting the title of a work, italics denotes the first use of a new and important term. For example:

Publican is a *DocBook* publishing system.

1.2. Pull-quote Conventions

Terminal output and source code listings are set off visually from the surrounding text.

Output sent to a terminal is set in **mono-spaced roman** and presented thus:

```
books      Desktop  documentation  drafts  mss    photos  stuff  svn
books_tests Desktop1  downloads      images  notes  scripts svgs
```

Source-code listings are also set in **mono-spaced roman** but add syntax highlighting as follows:

```
static int kvm_vm_ioctl_deassign_device(struct kvm *kvm,
                                       struct kvm_assigned_pci_dev *assigned_dev)
{
    int r = 0;
    struct kvm_assigned_dev_kernel *match;

    mutex_lock(&kvm->lock);

    match = kvm_find_assigned_dev(&kvm->arch.assigned_dev_head,
                                assigned_dev->assigned_dev_id);
    if (!match) {
        printk(KERN_INFO "%s: device hasn't been assigned
```

```

before, "
            "so cannot be deassigned\n", __func__);
        r = -EINVAL;
        goto out;
    }

    kvm_deassign_device(kvm, match);

    kvm_free_assigned_device(kvm, match);

out:
    mutex_unlock(&kvm->lock);
    return r;
}

```

1.3. Notes and Warnings

Finally, we use three visual styles to draw attention to information that might otherwise be overlooked.



Note

Notes are tips, shortcuts or alternative approaches to the task at hand. Ignoring a note should have no negative consequences, but you might miss out on a trick that makes your life easier.



Important

Important boxes detail things that are easily missed: configuration changes that only apply to the current session, or services that need restarting before an update will apply. Ignoring a box labeled “Important” will not cause data loss but may cause irritation and frustration.



Warning

Warnings should not be ignored. Ignoring warnings will most likely cause data loss.

2. Getting Help and Giving Feedback

2.1. Do You Need Help?

If you experience difficulty with a procedure described in this documentation, visit the Red Hat Customer Portal at <http://access.redhat.com>. From the Customer Portal, you can:

- ✧ Search or browse through a knowledge base of technical support articles about Red Hat products.
- ✧ Submit a support case to Red Hat Global Support Services (GSS).
- ✧ Access other product documentation.

Red Hat also hosts a large number of electronic mailing lists for discussion of Red Hat software and technology. You can find a list of publicly available mailing lists at <https://www.redhat.com/mailman/listinfo>. Click the name of any mailing list to subscribe to that list or to access the list archives.

2.2. We Need Feedback

If you find a typographical error in this manual, or if you have thought of a way to make this manual better, we would love to hear from you. Please submit a report in Bugzilla: <http://bugzilla.redhat.com/> against the product Red Hat JBoss Portal.

When submitting a bug report, be sure to mention the manual's identifier: *Development_Guide*

If you have a suggestion for improving the documentation, try to be as specific as possible when describing it. If you have found an error, please include the section number and some of the surrounding text so we can find it easily.

Part I. Architectural and Design Choices

Chapter 1. Architectural Choices

Depending on your environment and goals, you have to decide which components make up the final portal. Some elements may already be in place, such as an identity server, and others may be a matter of choice. This section aims to guide you in making those decisions.

[Report a bug](#)

1.1. Identity server

Red Hat JBoss Portal (JBoss Portal) comes with a component named PicketLink IDM, which allows to store and retrieve users and groups from various identity sources. PicketLink IDM supports three general options:

Database

Users, groups and their relationships are stored in an RDBMS database. Table names and column names can be changed, but the overall relationship between tables remains the same. This solution suits identity servers that are build from scratch and will handle thousands of users.

LDAP

Users, groups and their relationships are stored in an LDAP (or ActiveDirectory) server. The directory's structure can be adapted by configuration to the most common scenarios. This solution is particularly suited to environments that already use an LDAP server, environments that will share the identity server amongst multiple services (the website being one of them) or for very large sets of users (millions). When using LDAP with large number of users, it is recommended to use LDAP tools to do the provisioning of users.

Custom

When retrieving users, groups and their relationship cannot be done by configuration, it is possible to use the Picketlink IDM Service Provider Interface (SPI) to implement custom methods which control retrieving and storing user information.

PicketLink IDM also supports mixed environments, which is very useful when an LDAP infrastructure is provided in read-only mode. Since a website may need to store additional information about users (such as their preferred language), LDAP and database services can be combined so that users' information is stored in an LDAP source and additional properties in a database. During calls to the identity API, the information from both sources is merged transparently.

For more information about PicketLink IDM, see the JBoss Portal *Administration and Configuration Guide*.

[Report a bug](#)

1.2. Storage

The portal framework stores page compositions, portlet preferences and gadget code in a database through a Java Content Repository (JCR) API. A set of database servers and JDBC connectors are tested by Red Hat and details of supported configurations can be found on the JBoss Enterprise Portal Platform Supported Configurations page here;

<http://www.jboss.com/products/platforms/portals/testedconfigurations/> page.

The database schema is automatically created during the initial start up of the portal, at which point

it is required that the database users have sufficient rights to create tables. This privilege can be revoked after the initial start up. The database's content can be exported and imported without **CREATE TABLE** permission. The automatic creation of the database schema minimizes the manual effort involved in the installation of the product.

Since an RDBMS is not suited to the storage of large files, it is recommended to configure eXo JCR to store such files in the filesystem instead of a database, with metadata about the files stored in the database.



Note

If the website is running on a cluster, the filesystem must be accessible from all nodes and an NFS solution is required. For more details see "value storage" in the chapter *eXo JCR*.

[Report a bug](#)

1.3. Cluster

Clustering to meet failover or load-balancing goals requires a more complex configuration than a single node configuration. There is a cost associated with clustering since Red Hat JBoss Portal 6.1 has some optimizations when running on a single node, but the product is designed to scale linearly so that performance is improved with each new node.

In a cluster environment, all critical components are synchronized across all nodes, while less critical components are de-prioritized to achieve better performance. It is important that custom applications behave in the same manner when replicating data across a cluster of nodes.

The number of nodes required will vary according to factors such as: infrastructure, performance criteria, and applications developed. Performance analysis with tools such as JMeter and Grinder is recommended for measuring the impact of heavy loads prior to production implementation.

[Report a bug](#)

1.4. SSO

If a website is a part of a global infrastructure, a deployment of Single-Sign-On (SSO) may be beneficial to end users. Various SSO solutions are supported by Red Hat JBoss Portal, as documented on the "Supported Configurations" page available at <https://access.redhat.com/site/articles/119833>.

In some cases it can be better to have the token manager service on a specific server.

[Report a bug](#)

1.4.1. Summary

The infrastructure required is as follows:

- ✧ A database.
- ✧ LDAP (Optional).
- ✧ NFS (Optional, unless in a cluster configuration with default settings).

- » An SSO token service (Optional)
- » A cluster of nodes (Optional).

An example of the simplest configuration:

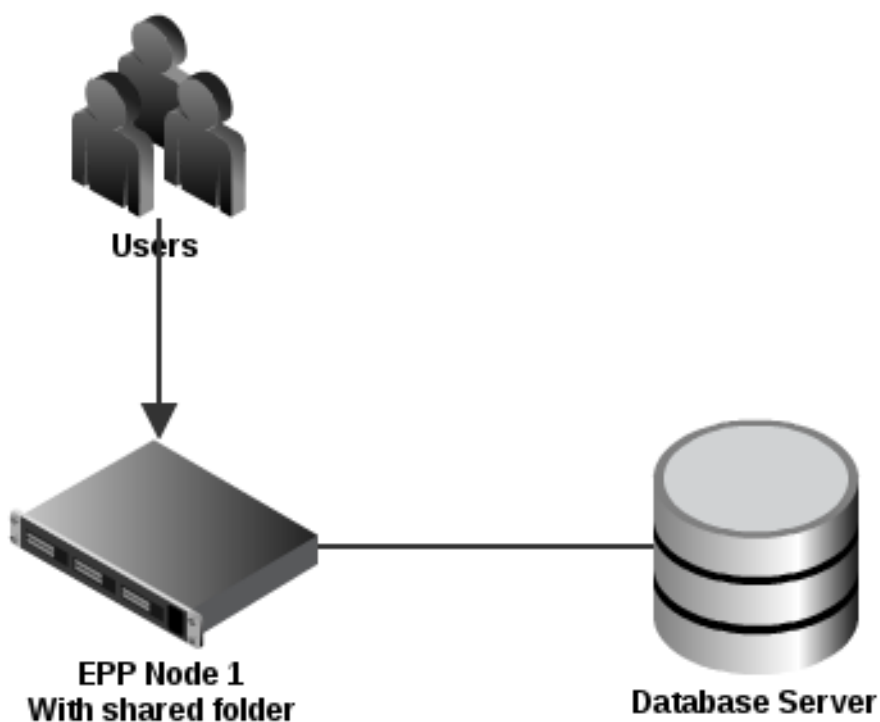


Figure 1.1. Simple SSO Example

An example of a more complex configuration:

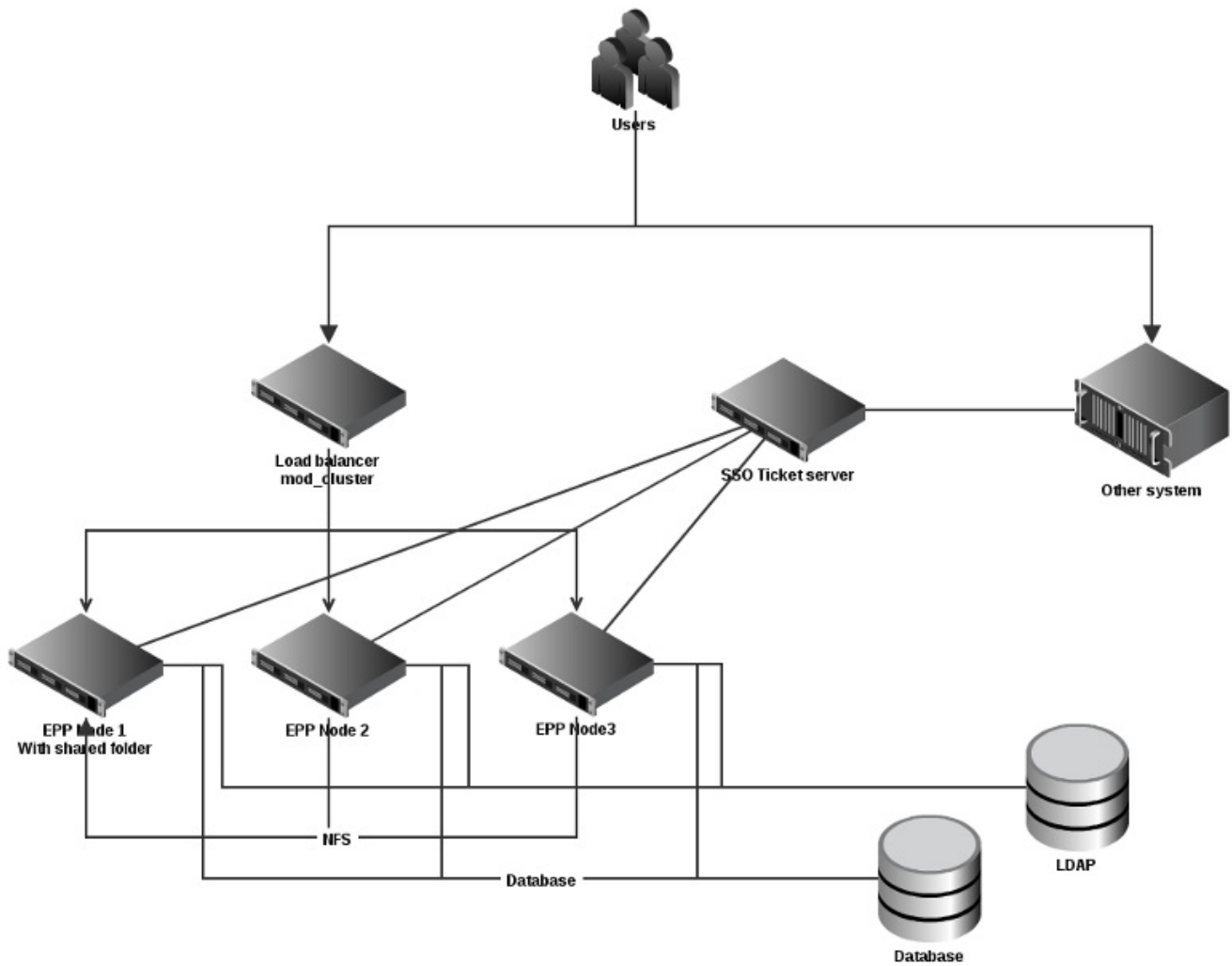


Figure 1.2. Multi-node SSO Example

[Report a bug](#)

Chapter 2. Design Choices

Once the main components of the architecture have been decided, choices must be made on the overall design.

[Report a bug](#)

2.1. Dashboards

User dashboards may be very costly in a large-scale portal due to the information storage considerations for each user creating a customized page. While some measures have been taken to ameliorate this cost, it, by nature, cannot be avoided completely and should be taken into account when making design decisions in large organizations.

This overhead is difficult to estimate as it is dependent on how users navigate the portal. If only a minority use this functionality; the cost may be negligible. Alternately, the cost could be considerable if many users create custom spaces.

The impact of implementing this feature must be measured by:

1. Estimating the number of dashboards and pages that will be created.
2. Observing the impact on the database (through the JCR) in terms of size.

[Report a bug](#)

2.2. JCR Index Replication in a Cluster Setup

2.2.1. JCR Index Replication in a Cluster Setup

The JCR implementation uses Apache Lucene to index the data. The indexes are used to search for content (page nodes or WCM content for instance).

Lucene is not cluster-ready, but on a cluster, each node must be able to search for content and will need to have access to the Lucene indexes.

When optimizing searching, a balance must be struck between various goals, including:

- ✧ Speed.
- ✧ Fast indexing.
- ✧ Consistent results on a node.
- ✧ Data accuracy and longevity.
- ✧ Impact on system performance.
- ✧ Ease and impact of implementation.

However eXo JCR, the JCR implementation used by Red Hat JBoss Portal, makes it possible to configure the storage and retrieval of indexes according to the architect's decisions on constraint stringency.

See Also:

» [Appendix B, eXo JCR](#)

[Report a bug](#)

2.2.2. Standalone Index

Standalone Index is only suitable for a non-cluster environment but is the least complicated configuration, with a combination of in-memory and file based indexes. There is no replication involved so any entry can be found by a search as soon as it is created.

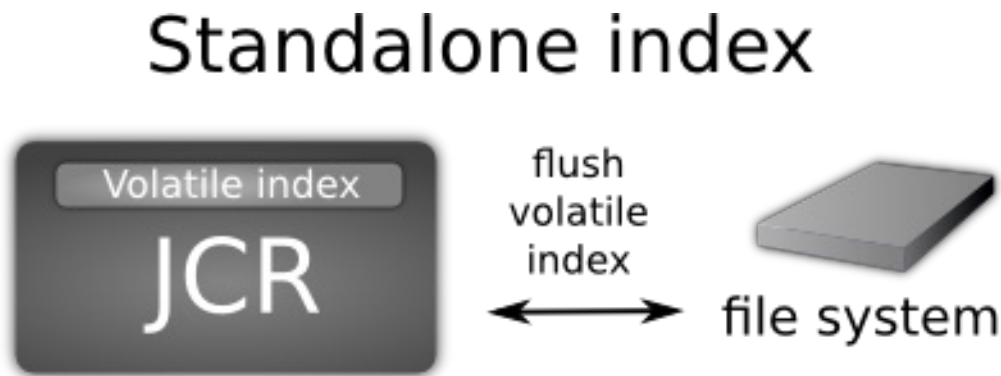


Figure 2.1. Standalone Index

[Report a bug](#)

2.2.3. Local Index

In Local Index setup each node keeps a local copy of the full indexes so that when a search is requested on a node, there is no network communication required. However, the cost of this method is replication; when a node indexes an item, it is required to replicate that index on each node. If a node is unavailable at that time, it may miss an index update request and then the different nodes may be inconsistent.

Also when a node is added, it has to recreate its own full index.

An alternative to this configuration is to configure a node to retrieve the information from a coordinator on each search, which makes the startup of the new node faster, but impacts its performance during operation.

Local index

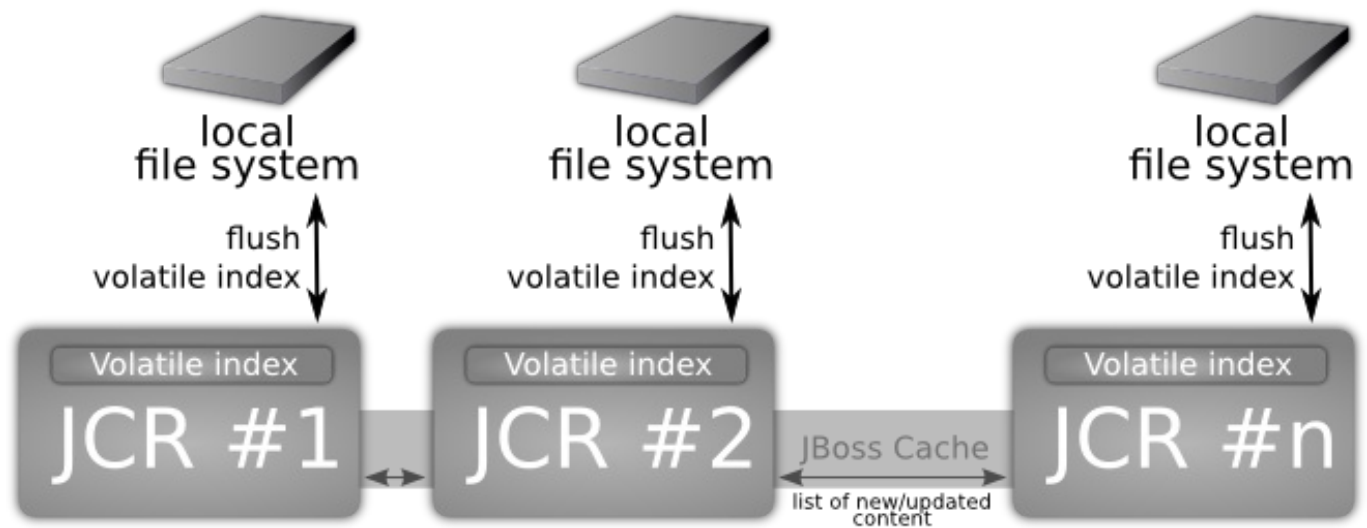


Figure 2.2. Local Index

[Report a bug](#)

2.2.4. Shared Index

In this setup a unique index is created and shared among all the nodes. The Network File System (NFS) must be installed so that all nodes can read the index.

Advantages:

- ✧ Consistency: all the nodes see the same data.

Disadvantages:

- ✧ Requires a highly available NFS setup (NFS 3 is recommended).
- ✧ Increased network communication.

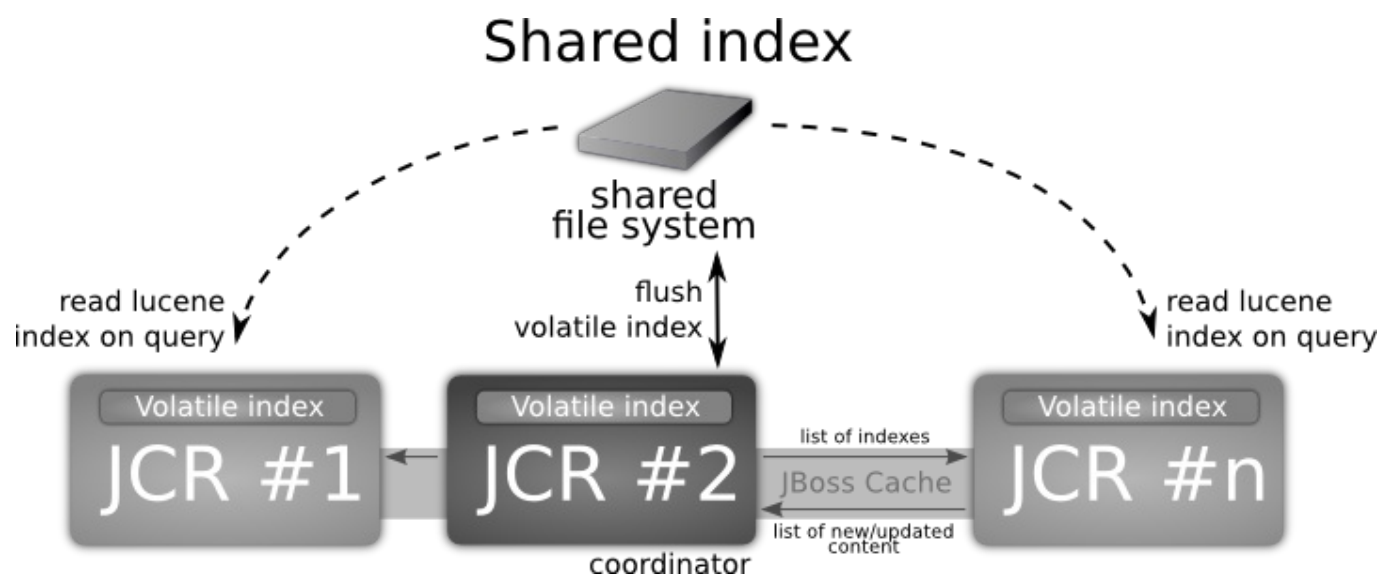


Figure 2.3. Shared Index

[Report a bug](#)

Part II. Portal API

1. About Portal API

The Portal API provides access to retrieving, creating and deleting Portal Sites and Portal Pages. It also provides full support for the navigation and nodes associated with a site, which can be very useful for e.g. creating custom navigation portlets. This section only gives some basic examples of the API usage. For full documentation of all features available, see the [Javadoc](#).

[Report a bug](#)

Chapter 3. PortalRequest and Portal

The **PortalRequest** is the entry point to the API. It provides access to the portal, and it also provides access to information associated with the current request (such as the authenticated user and the page being requested).

Example 3.1. Portal Request

This example shows how to use **PortalRequest** to obtain an instance of the portal.

```
Portal portal = PortalRequest.getInstance().getPortal();
```

[Report a bug](#)

Chapter 4. Site

A Site is comprised of a set of pages, navigation definitions, and can be assigned a unique skin. There are three different types of sites: standard sites, group spaces and user dashboards.

[Report a bug](#)

4.1. Retrieving Sites

A specific site can be retrieved using its **SiteId**. The **SiteId** is constructed by passing either the name for a standard site, the group for a group space or the user for a user dashboard. Alternatively, it can be constructed by passing the **SiteType** and name.

Example 4.1. Retrieving Sites

This example shows how to retrieve sites.

```
Site standardSite = portal.getSite(new SiteId("classic"));
Site groupSpace = portal.getSite(new SiteId(new Group("platform",
"administrators")));
Site userDashboard = portal.getSite(new SiteId(new User("john")));
```

It is also possible to query for sites using the **SiteQuery**. The **SiteQuery** supports filtering, sorting and pagination when querying for sites.



Note

The default number of returned results is 15. If no pagination is set while building the query, the maximum number of results will be 15. Ensure the correct pagination is set while querying.

Example 4.2. Site Query

This example finds standard sites that a user has access permissions to, limited to the first ten results.

```
SiteQuery.Builder qb = new SiteQuery.Builder();

qb.withSiteTypes(SiteType.SITE).withPagination(0, 10).withFilter(new
Filter<Site>() {
    public boolean accept(Site site) {
        return portal.hasPermission(user, site.getAccessPermission());
    }
});

List<Site> sites = portal.findSites(qb.build());
```

[Report a bug](#)

4.2. Creating a Site

The work flow to create a site is described below:

1. Create a new site through the Portal.
2. Set the configuration for the site (such as localized display names and skin).
3. Save the configuration using **Portal**.

Example 4.3. Create a New Standard Site

This example creates a new standard site with the name mysite and a non-localized display name.

The site is not visible (or persisted) until **saveSite** is invoked.

```
Site site = portal.createSite(new SiteId("foo"));
site.setDisplayName("My Site");
portal.saveSite(site);
```

[Report a bug](#)

4.2.1. Setting Attributes for a Site

It is possible to change attributes for the site. The supported attributes are:

sessionAlive

Specifies whether a session is kept alive or not. Valid values are **"onDemand"**, **"always"**, and **"never"** with the default being **"onDemand"**

showPortletInfo

Specifies whether the info bar is shown by default when adding applications to pages. The default is 'false'

Example 4.4. Set Attributes Using the Key

Attributes are set by using the associated **Key**. This example changes the **sessionAlive** attribute.

```
site.getAttributes().put(Site.AttributeKeys.SESSION_BEHAVIOR,
"always");
```

[Report a bug](#)

4.2.2. Setting permissions for a site

Associated with a site is an access permission and an edit permission, which controls which users and groups are allowed to access and edit the site respectively.

Example 4.5. Setting Access and Edit Permissions

This example demonstrates how to make a site accessible by everyone, but only editable by users in the `/platform/administrators` group.

The changed permissions do not take affect until `saveSite` is invoked.

```
site.setAccessPermission(Permission.everyone());
site.setEditPermission(Permission.any("platform", "administrators"));

portal.saveSite(site);
```

[Report a bug](#)

4.3. Deleting a Site

A site is deleted by invoking `removeSite` on the Portal.

[Report a bug](#)

Chapter 5. Navigation

A **Navigation** for a site is retrieved from the Portal and allows accessing the node tree that represents the navigation entries for the site. There is a special root node which is not directly part of the navigation, but it's the parent of the first level of navigation entries. As the root node is not meant to be displayed it doesn't have a display name.

Example 5.1. Retrieving Navigation

This example retrieves the **Navigation** for the classic site.

```
Navigation navigation = portal.getNavigation(new SiteId("classic"));
```

[Report a bug](#)

5.1. Retrieving Navigation Nodes

When retrieving navigation nodes it is possible to either retrieve the root node, or a specific node in the hierarchy. It is also possible to control which if any of the children are loaded.

Example 5.2. Basic Navigation Portlet

This example shows a very simple navigation portlet that displays the top level of entries in the navigation menu.

```
public void doView(RenderRequest request, RenderResponse response)
throws IOException {
    PrintWriter pw = response.getWriter();
    Navigation navigation =
PortalRequest.getInstance().getNavigation();

    pw.print("<ul>");
    for (Node n : navigation.getRootNode(Nodes.visitChildren())) {
        pw.printf("<li><a href='%s'>%s</a></li>", n.getURI(),
n.getDisplayName());
    }
    pw.print("</ul>");
}
```

Example 5.3. Retrieve a Specific Node

This example demonstrates how to retrieve a specific node in the tree by specifying the **NodePath** to the node.

```
Node node = navigation.getNode(NodePath.path("home"));
```



Note

When a node is retrieved, it actually represents a tree of nodes and all operations (for example, save and refresh) act on that entire tree. For example, if a root node with two child nodes is retrieved, and changes are made to the both child nodes but only one node is saved, the other child will be saved automatically because the save operation acts on the entire node tree.

[Report a bug](#)

5.1.1. NodeVisitor

It is important to limit how many levels of navigation nodes are retrieved, especially where there is a large number of nodes. This is controlled by using either one of the built in **NodeVisitor** from the **Nodes** class, or by implementing your own **NodeVisitor**. The **Nodes** class contains the following visitors:

- ✦ visitAll - loads all nodes
- ✦ visitChildren - loads the immediate children
- ✦ visitNone - loads only the node
- ✦ visitNodes(int) - loads a specified depth of descendants

Example 5.4. Retrieve a Node Using visitNodes

This example retrieves the root node and two levels of nodes.

```
Node rootNode = navigation.getRootNode(Nodes.visitNodes(2));
```

Example 5.5. isChildrenLoaded versus visitChildren Method

To correctly confirm whether the child nodes are loaded, use the **isChildrenLoaded** method instead of **visitChildren**.

This would return true:

```
Node rootNode =
navigation.getRootNode(Nodes.visitChildren()).isChildrenLoaded()
```

This would return false:

```
Node rootNode =
navigation.getRootNode(Nodes.visitNone()).isChildrenLoaded()
```

[Report a bug](#)

5.1.2. Filtering Navigation Nodes

Nodes support a filtering mechanism which simplifies displaying nodes that have specific properties.

Example 5.6. Display User-visible Nodes When Creating a Navigation Portlet

This example displays visible nodes where the user has access to view the page.

```
Node filtered = node.filter().showDefault();
```

There are a number of methods available to control what nodes are displayed, and they all start with **show**. For example, **showDefault** is an abbreviation for **showVisible().showHasAccess(PortalRequest.getInstance().getUser())**

Example 5.7. Custom Filter

This example uses a custom filter to only display nodes with a display name that starts with "A".

```
Node filtered = root.filter().show(new Filter<Node>() {
    public boolean accept(Node node) {
        return node.getDisplayName().startsWith("A");
    }
});
```

[Report a bug](#)

5.2. Creating a Navigation Node

Creating a node uses the following workflow:

1. Retrieve the parent node you want to add the node to.
2. Invoke the **addChild** method on the parent node.
3. Set the configuration for the node (such as the display name and the page it should link to).
4. Save it using **saveNode** on **Navigation**.

Example 5.8. Creating a Navigation Node

This example creates a node as a child of the home node.

The node is not visible (or persisted) until **saveNode** is invoked.

```
Node home = navigation.getNode(NodePath.path("home"));
Node child = home.addChild("mynode");
child.setDisplayName("My Node");
child.setPageId(new PageId("classic", "mypage"));
navigation.saveNode(home);
```

[Report a bug](#)

5.2.1. Navigation Node Visibility

Nodes can be visible, hidden or only visible at a specified publication date. By default a new node is visible.

A node can be hidden with `node.setVisibility(false)`, or only shown until a specific date with `node.setVisibility(PublicationDate.endingOn(date))`. It is also possible to set a starting date, or set both a starting date and a finishing date.

Changes to node visibility are not shown until **saveNode** is invoked on the Portal.

[Report a bug](#)

5.2.2. Localization

The display name for a node supports localization.

Example 5.9. Setting and English and French Node

This example sets the display name for a node in English and French.

```
LocalizedString localizedString = node.getDisplayNames();
localizedString.setLocalizedValue(Locale.ENGLISH, "My node");
localizedString.setLocalizedValue(Locale.FRENCH, "Mon noeud");
node.setDisplayNames(localizedString);

navigation.saveNode(node);
```

[Report a bug](#)

5.3. Deleting a Navigation Node

A node is deleted by removing it from the parent node.

Example 5.10. Deleting a Node

This example removes the child with the name mynode.

The node is not removed until **saveNode** is invoked.

```
node.removeChild("mynode");
navigation.saveNode(node);
```

[Report a bug](#)

5.4. Moving a Navigation Node

A node can be moved to a different parent, or it can be moved to a different index in the same parent. When moving to a different parent the new parent is required to be in the same tree.

Example 5.11. Move Child Nodes Between Parent and Index Nodes

Move a node from one parent to another.

```
root.getNode("parent1", "child").moveTo(root.getNode("parent2"));
navigation.saveNode(root);
```

Move a node to a different index in the same parent.

```
root.getNode("parent", "child").moveTo(0);
navigation.saveNode(root);
```

The changes are not visible (or persisted) until **saveNode** is invoked.

A more convenient way to sort children for a parent is to use the **sort** method on the parent.

Example 5.12. Sorting by Display Name

This example demonstrates how to sort the children of the root node by their display names.

The changes are not visible (or persisted) until **saveNode** is invoked.

```
root.sort(new Comparator<Node>() {
    public int compare(Node o1, Node o2) {
        return o1.getDisplayName().compareTo(o2.getDisplayName());
    }
});
navigation.saveNode(root);
```

[Report a bug](#)

Chapter 6. Page

6.1. Retrieving Pages

A specific page can be retrieved by its **PageId**. The **PageId** is constructed by passing either the name for a standard site, the group for a group space or the user for a user dashboard, and the name of the page. Alternatively it can be constructed by passing the **SiteId** and name of the page.

Example 6.1. Retrieving Specific Pages

This example shows how to retrieve specific pages of a portal.

```
Page standardSitePage = portal.getPage("classic", "home");
Page groupSpacePage = portal.getSite(new Group("platform",
"administrators"), "grouppage");
Page userDashboardPage = portal.getSite(new User("john", "johnspage"));
```

It is also possible to query for pages using the **PageQuery**. The **PageQuery** supports filtering, sorting and pagination when querying for pages.

Example 6.2. Retrieve Pages Using Filtering, Sorting, and Pagination

This example finds pages with a display name that starts with "A".

```
PageQuery.Builder qb = new PageQuery.Builder();

qb.withSiteType(SiteType.SITE).withFilter(new Filter<Page>() {
    public boolean accept(Page page) {
        return page.getDisplayName().startsWith("A");
    }
});

List<Page> pages = portal.findPages(qb.build());
```

[Report a bug](#)

6.2. Creating a Page

Creating a page uses the following workflow:

1. Create a new page through the **Portal**.
2. Set the configuration for the page (such as localized display names).
3. Save it using **Portal**.

Example 6.3. Create a Page

This example creates a new page in the classic site with the name mypage and a non-localized display name.

The page is not visible (or persisted) until **portal . savePage** is invoked.

```
Page page = portal.createPage(new PageId("classic", "mypage"));
page.setDisplayName("My Page");
portal.savePage(page);
```

[Report a bug](#)

6.2.1. Setting Permissions for a Page

Associated with a page is an access permission and an edit permission, which controls what users and groups are allowed to access and edit the page respectively.

Example 6.4. Set Permissions for a Page

This example makes a page accessible to everyone, but only editable by users in the **/platform/administrators** group.

The changed permissions do not take affect until **savePage** is invoked.

```
page.setAccessPermission(Permission.everyone());
page.setEditPermission(Permission.any("platform", "administrators"));
portal.savePage(page);
```

[Report a bug](#)

6.3. Page Composition

Red Hat JBoss Portal allows you to add applications and containers to a Page. Applications and containers represent blocks of content visible on a page.

There are three types of application:

- Portlet
- WSRP Portlet
- Gadget

The template for containers renders children in the user interface. The Portal API provides a container with two basic templates for rendering children in rows and columns. A container can hold the following child elements: other containers, and applications.

Example 6.5. Page Containing a Single Application

```

PageBuilder pageBuilder = portal.newPageBuilder();
Application calculator =
portal.getApplicationRegistry().getApplication("Gadgets/Calculator");
Page page =
pageBuilder.child(calculator).siteName("classic").siteType("portal").name("awesome").build();

```

Example 6.6. Container Template Using Two Column Containers Nested in a Row Container

```

PageBuilder pageBuilder = portal.newPageBuilder();
ApplicationRegistry appRegistry = portal.getApplicationRegistry();
Page page = pageBuilder
    .newRowsBuilder()
        .newColumnsBuilder()
            .child(appRegistry.getApplication("Gadgets/Calculator"))

        .child(appRegistry.getApplication("Gadgets/rssAggregator"))
            .buildToParentBuilder()

        .newColumnsBuilder()
            .child(appRegistry.getApplication("Gadgets/Calendar"))
            .child(appRegistry.getApplication("Gadgets/ToDo"))
            .buildToParentBuilder()

    .buildToTopBuilder()
    .siteName("classic")
    .siteType("portal")
    .name("awesome_" + UUID.randomUUID().toString())
    .displayName("Awesome page")
    .showMaxWindow(false)
    .accessPermission(Permission.everyone())
    .editPermission(Permission.any("platform", "administrators"))
    .moveAppsPermission(Permission.everyone())
    .moveContainersPermission(Permission.everyone())
    .build();

```

The **PageBuilder.build()** method returns a page object. The page object is an argument for **Portal.savePage()** method to persist the page.



Note

It is not mandatory to add children initially to a page. A page can be stored without content, and children added later through the JBoss Portal UI or API.

[Report a bug](#)

6.4. Deleting a Page

A page is deleted by invoking **removePage** on the Portal.

[Report a bug](#)

6.5. Application

An application is a read-only representation of a gadget, portlet or a WSRP portlet. Application is used to compose pages.

Application can be accessed through the user interface by clicking **Application Registry**.

To access the complete list of application use the following code:

```
ApplicationRegistry registry = portal.getApplicationRegistry();  
List<Application> applications = registry.getApplications();
```

In case you know the ID of an application use the following code:

```
ApplicationRegistry registry = portal.getApplicationRegistry();  
Application gadgetRss =  
registry.getApplication(applicationIdFromPreviousInteraction):
```

[Report a bug](#)

Chapter 7. OAuth Provider API

The interface **OAuthProvider** is a part of the public API. It is the entry point to perform operations on OAuth providers such as Facebook, Google, and Twitter.

Some additional portal configuration is required to use OAuth providers for authentication of users. See *OAuth - Authentication with Social Network Accounts* chapter in the *Administration and Configuration Guide* for more details on this topic.

Once a user is authenticated (or their account is linked with the OAuth provider), their access token is saved in the IDM database as a part of their User Profile. It is possible to retrieve the user's OAuth access token through the **OAuthProvider** interface and run its operations. It is also possible to revoke or validate existing access tokens or send a request to obtain new access tokens with more scopes (privileges).

The following sections present basic uses for the **OAuthProvider** API. There is an excellent Quickstart available called *Portlets for Integration with Social Networks*, shipped in the Quickstarts binary.

[Report a bug](#)

7.1. Retrieve an Instance of OAuthProvider

You need to retrieve the appropriate instance of **OAuthProvider** from **Portal**.

Currently the portal supports three OAuth providers:

- ✳ **OAuthProvider.FACEBOOK** for Facebook.
- ✳ **OAuthProvider.GOOGLE** for Google+.
- ✳ **OAuthProvider.TWITTER** for Twitter.

Example 7.1. Retrieve Facebook OAuth Instance

This example shows the syntax to use when retrieving the Facebook instance of **OAuthProvider**. This syntax can be adapted for other instances.

```
Portal portal = PortalRequest.getInstance().getPortal();
OAuthProvider facebookProvider =
portal.getOAuthProvider(OAuthProvider.FACEBOOK)
```

[Report a bug](#)

7.2. OAuthProvider Operations

Example 7.2. OAuthProvider Basic Use

This example provides a code snippet that describes some basic uses of the **OAuthProvider** API.


```

// Retrieve instance of Google OAuth provider
OAuthProvider googleProvider =
PortalRequest.getInstance().getPortal().getOAuthProvider(OAuthProvider
.GOOGLE);

// Check if Google was enabled in configuration.properties
if (googleProvider == null) {
    renderResp.getWriter().println("Authentication with Google not
available. Ensure your administrator has enabled the service in the
portal. See the Administration and Configuration Guide for
instructions.");
    return;
}

// Retrieve the key and display name of the social network
String key = googleProvider.getKey();
String friendlyName = googleProvider.getFriendlyName();
renderResp.getWriter().println(friendlyName + " is enabled");

// Retrieve access token of the current user
AccessToken accessToken =
googleProvider.loadAccessToken(renderReq.getRemoteUser());

// Check if access token is available. It's the case when this user was
registered/authenticated into portal
// through Google+ or if he linked his account with Google+
if (accessToken == null) {
    renderResp.getWriter().println("Your account is not linked with
Google+. You can link it in 'Social network' tab of " +
        "user settings or you can authenticate through Google into
portal");
    return;
}

// Check if access token is valid and refresh it if necessary
try {
    accessToken =
googleProvider.validateTokenAndUpdateScopes(accessToken);
} catch (OAuthApiException oauthException) {
    if
(oauthException.getExceptionCode().equals(OAuthApiExceptionCode.ACCESS
_TOKEN_ERROR)) {
        renderResp.getWriter().println("Your access token is invalid or
has been revoked");
    } else if
(oauthException.getExceptionCode().equals(OAuthApiExceptionCode.IO_ERR
OR)) {
        renderResp.getWriter().println("Network error during the
communication with Google");
    }
}

// Check all available scopes
String availableScopes = accessToken.getAvailableScopes();

```

```
// Check if we have scope to call Google+ operations
if
(!availableScopes.contains("https://www.googleapis.com/auth/plus.login"
)) {
    // Redirect to Google+ and ask for plus.login scope

googleProvider.startOAuthWorkflow("https://www.googleapis.com/auth/plus
.login");
    return;
}

// Obtain Google API object to call Google plus API operations
Plus service = googleProvider.getAuthorizedSocialApiObject(accessToken,
Plus.class);

// Retrieve activities from Google+ wall of user
ActivityFeed activityFeed = service.activities().list("me",
"public").execute();
for (Activity activity : activityFeed.getItems()) {
    renderResp.getWriter().println(activity.getTitle());
}

// Revoke the access token. It won't be possible to run any operations
with it anymore.
// And your application will be cleared from Google applications of
current user on page https://plus.google.com/apps
googleProvider.revokeToken(accessToken);

// Remove the token from the UserProfile of the current user
googleProvider.removeAccessToken(request.getRemoteUser());
```

[Report a bug](#)

7.3. Access to Provider-specific Operations

The `oauthProvider.getAuthorizedSocialApiObject()` method is used to obtain access to provider-specific operations. This method usually returns objects from a third-party library. Those objects are always initialized with the access token of the current user and can be used to retrieve data from the related social network.

Google

There are two supported types usable as arguments of this method:

com.google.api.services.plus.Plus

- Google Plus API class, which can be used to call operations on Google Plus - see **GoogleActivitiesPortlet** and **GoogleFriendsPortlet** in the Quickstart.

com.google.api.services.oauth2.Oauth2

OAuth2 class, which provides operations related to the user, such as obtaining their Google user profile details or obtaining information about his access token - see **GoogleUserInfoPortlet** in the Quickstart.

Twitter

There is only one supported type for Twitter: `twitter4j.Twitter`. An instance of this class can be used to retrieve user details, number of tweets, number of friends, last tweets. See `TwitterPortlet` in the Quickstart.

Facebook

There is no supported type for Facebook. In the Quickstart, the third-party library [RestFB](#) is used to perform operations against Facebook.

[Report a bug](#)

Chapter 8. REST API

[Report a bug](#)

8.1. Overview

The portal provides a REST API to portal entities such as Sites, Pages, and Navigation. All data is sent and received as JSON.

[Report a bug](#)

8.1.1. Base URL

There are two base URLs for the REST API. One that will be used if you need to authenticate and one that can be used to access resources anonymously.

Anonymous Base URL

`http://<host>:<port>/rest/managed-components/api`

Authenticated Base URL

`http://<host>:<port>/rest/private/managed-components/api`

[Report a bug](#)

8.1.2. Authentication

The REST API supports basic authentication. Below is an example of using basic authentication in curl command.

```
curl -u <username> http://<host>:<port>/rest/private/managed-components/api
```

This allows users to access resources they have access to depending on the portal entity permissions.



Note

Only members of the group **/platform/administrators** are allowed to add, update, or delete resources.

[Report a bug](#)

8.1.3. Content Type

Since all data is sent and received using JSON, appropriate HTTP headers need to be included in each HTTP request.

For clients receiving data the following HTTP header needs to be included in the request

```
Accept: application/json
```

For clients sending data the following HTTP header needs to be included in the request

```
Content-Type: application/json
```

[Report a bug](#)

8.1.4. Localization

Localization is supported for navigation nodes that have localized displayNames. To specify the language simply add the **Accept-Language** header to the HTTP request.

Example 8.1. French Header

This examples describes the Accept-Language header used for French localization.

```
Accept-Language: fr
```

[Report a bug](#)

8.2. Resources

The following sections describe the resources available in the REST API.



Important

All URLs in the Example responses featured in this section are relative to the Base URL.

[Report a bug](#)

8.2.1. Sites

[Report a bug](#)

8.2.1.1. List Sites

Sites are organized by site type, where **site-type** can be **site**, **space**, or **dashboard**.

Example 8.2. Listing Sites using GET

This example describes the syntax required to list site types.

```
GET /api/{site-type}s/
```

**Note**

The 's' at the end of the URL is important. Space resolves to **/api/spaces/** and dashboard resolves to **/api/dashboards/**.

Table 8.1. Additional Parameters for site-type

| Parameter | Type | Default | Description |
|------------|---------|---------|--|
| emptySites | boolean | false | Indicates to include empty sites (sites w/out pages or navigation) |
| offset | int | 0 | The offset of the results in a paginated request |
| limit | int | 15 | The maximum number of results to return in a paginated request |

Example 8.3. List Sites using GET Request and Response

This example lists the sites that are available and their corresponding resource location. This should be used by clients instead of hard coding every location or URL.

```
GET /api/sites
```

```
HTTP/1.1 200 OK
Content-Type: application/json
[
  {
    "name" : "classic",
    "type" : "site",
    "url" : "/api/sites/classic"
  },
  {
    "name" : "mobile",
    "type" : "site",
    "url" : "/api/sites/mobile"
  }
]
```

**Note**

The **url** element is the full URL but for sake of brevity it is relative for the examples in this document.

[Report a bug](#)

8.2.1.2. Retrieve a Site

To retrieve a site named **site-name** the following syntax applies.

```
GET /api/{site-type}s/{site-name}
```

Example 8.4. Retrieve a Site Request and Response

This example shows how to retrieve the Classic site.

```
GET /api/sites/classic
```

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "name" : "classic",
  "type" : "site",
  "displayName" : "Classic",
  "description" : "GateIn default portal",
  "skin" : "Default",
  "locale" : "en",
  "access-permissions" : ["Everyone"],
  "edit-permissions" : ["*/platform/administrators"],
  "attributes" : [
    {
      "key" : "showPortletInfo",
      "value" : "false"
    },
    {
      "key" : "sessionAlive",
      "value" : "onDemand"
    }
  ],
  "pages" : {"url" : "/api/sites/classic/pages"},
  "navigation" : {"url" : "/api/sites/classic/navigation"}
}
```



Note

The **url** element is the full URL but for sake of brevity it is relative for the examples in this document.

[Report a bug](#)

8.2.1.3. Create a site

To create a site named **site-name**, the following syntax applies:

```
POST /api/{site-type}s/{site-name}
```

Example 8.5. Create a Site Request and Response

This example creates a site named 'foo'.

```
POST /api/sites/foo
```

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "name" : "foo",
  "type" : "site",
  "displayName" : "Basic Portal",
  "description" : "This is basic portal template",
  "skin" : "Default",
  "locale" : "en",
  "access-permissions" : ["Everyone"],
  "edit-permissions" : ["*/platform/administrators"],
  "attributes" : [{
    "key" : "sessionAlive",
    "value" : "onDemand"
  }],
  "pages" : {"url" : "/api/sites/foo/pages"},
  "navigation" : {"url" : "/api/sites/foo/navigation"}
}
```



Note

The **url** element is the full URL but for sake of brevity it is relative for the examples in this document.

[Report a bug](#)

8.2.1.4. Delete a site

To delete a site named **site-name**, the following syntax applies:

```
DELETE /api/{site-type}s/{site-name}
```

Example 8.6. Delete a Site Request and Response

This example deletes a site named 'foo'

```
DELETE /api/sites/foo
```

```
HTTP/1.1 200 OK
Content-Type: application/json
```

[Report a bug](#)

8.2.1.5. Update a site

To update a site named **site-name**, the following syntax applies:


```
PUT /api/{site-type}s/{site-name}
```

Example 8.7. Update a Site Request and Response

This example updates the site 'classic' with a new description "The Classic site shipped with the product".

```
PUT /api/sites/classic
```

```
Content-Type: application/json
```

```
{
  "description" : "The Classic site shipped with the product"
}
```

```
HTTP/1.1 200 OK
```

```
Content-Type: application/json
```

```
{
  "name" : "classic",
  "type" : "site",
  "displayName" : "Classic",
  "description" : "The Classic site shipped with the product !",
  "skin" : "Default",
  "locale" : "en",
  "access-permissions" : ["Everyone"],
  "edit-permissions" : ["*/platform/administrators"],
  "attributes" : [
    {
      "key" : "showPortletInfo",
      "value" : "false"
    },
    {
      "key" : "sessionAlive",
      "value" : "onDemand"
    }
  ],
  "pages" : {"url" : "/api/sites/classic/pages"},
  "navigation" : {"url" : "/api/sites/classic/navigation"}
}
```



Note

The **url** element is the full URL but for sake of brevity it is relative for the examples in this document.

[Report a bug](#)

8.2.2. Pages

[Report a bug](#)

8.2.2.1. List Pages

To list pages in a site named **site-name**, the following syntax applies:

```
GET /api/{site-type}s/{site-name}/pages
```

Table 8.2. list-pages Additional Attributes

| Parameter | Type | Default | Description |
|-----------|------|---------|--|
| offset | int | 0 | The offset of the results in a paginated request |
| limit | int | 15 | The maximum number of results to return in a paginated request |

Example 8.8. List all Pages Request and Response

This example shows how to list all pages for the site 'classic'.

```
GET /api/sites/classic/pages
```

```
HTTP/1.1 200 OK
Content-Type: application/json
[
  {
    "name" : "homepage",
    "siteType" : "site",
    "siteName" : "classic",
    "url" : "/api/sites/classic/pages/homepage"
  },
  {
    "name" : "register",
    "siteType" : "site",
    "siteName" : "classic",
    "url" : "/api/sites/classic/pages/register"
  },
  {
    "name" : "sitemap",
    "siteType" : "site",
    "siteName" : "classic",
    "url" : "/api/sites/classic/pages/sitemap"
  }
]
```



Note

The **url** element is the full URL but for sake of brevity it is relative for the examples in this document.

[Report a bug](#)

8.2.2.2. Retrieve a page

To retrieve a page named **page-name**, the following syntax applies:

```
GET /api/{site-type}s/{site-name}/pages/{page-name}
```

Example 8.9. Retrieve a Page Request and Response

This example shows how to retrieve the page 'homepage'

```
GET /api/sites/classic/pages/homepage
```

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "name" : "homepage",
  "displayName" : "Home Page",
  "description" : null,
  "access-permissions" : ["Everyone"],
  "edit-permissions" : ["*/platform/administrators"]
}
```

[Report a bug](#)

8.2.2.3. Create a page

To create a page named **page-name**, the following syntax applies:

```
POST /api/{site-type}s/{site-name}/pages/{page-name}
```

Example 8.10. Create a New Page Request and Response

The following example creates a page named 'newpage'.

```
POST /api/sites/classic/pages/newpage
```

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "name" : "newpage",
  "displayName" : "newpage",
  "description" : null,
  "access-permissions" : ["Everyone"],
  "edit-permissions" : ["*/platform/administrators"]
}
```

[Report a bug](#)

8.2.2.4. Delete a page

To delete a page named **page-name**, the following syntax applies:

```
DELETE /api/{site-type}s/{site-name}/pages/{page-name}
```

Example 8.11. Delete a Page Request and Response

This example deletes a page named 'newpage'.

```
DELETE /api/sites/classic/pages/newpage
```

```
HTTP/1.1 200 OK
Content-Type: application/json
```

[Report a bug](#)

8.2.2.5. Update a page

To update a page named **page-name**, the following syntax applies:

```
PUT /api/{site-type}s/{site-name}/pages/{page-name}
```

Example 8.12. Update Page Description Request and Response

This example updates the page 'homepage' with a new description.

```
PUT /api/sites/classic/pages/homepage

Content-Type: application/json
{
  "description" : "The default homepage"
}
```

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "name" : "homepage",
  "displayName" : "Home Page",
  "description" : "The default homepage",
  "access-permissions" : ["Everyone"],
  "edit-permissions" : ["*/platform/administrators"]
}
```

[Report a bug](#)

8.2.3. Navigation

[Report a bug](#)

8.2.3.1. Retrieve Navigation

To update a page named **site-name**, the following syntax applies:

```
GET /api/{site-type}s/{site-name}/navigation
```

Table 8.3. Retrieve Navigation Parameters

| Parameter | Type | Default | Description |
|-----------|------|---------|---|
| scope | int | n/a | Specifies how many nodes to load (-1 for all) |

Below is an example of retrieving the navigation for the site 'classic'

Example 8.13. Request

```
GET /api/sites/classic/navigation
```

Example 8.14. Response

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "priority" : 1,
  "siteType" : "site",
  "siteName" : "classic",
  "nodes" : [
    {
      "name" : "home",
      "url" : "/api/sites/classic/navigation/home"
    },
    {
      "name" : "sitemap",
      "url" : "/api/sites/classic/navigation/sitemap"
    }
  ]
}
```



Note

The **url** element is the full URL but for sake of brevity it is relative for the examples in this document.

Example 8.15. Scope Parameter Request and Response

This example uses the scope parameter to load more nodes which will include the actual node representation in the response

```
GET /api/sites/classic/navigation?scope=1
```

```

HTTP/1.1 200 OK
Content-Type: application/json
{
  "priority" : 1,
  "siteType" : "site",
  "siteName" : "classic",
  "nodes" : [
    {
      "name" : "home",
      "uri" : "/portal/classic/home",
      "isVisible" : true,
      "visibility" : {"status" : "VISIBLE"},
      "iconName" : null,
      "displayName" : "Home",
      "displayNames" : [...]
      "children" : null,
      "page" : {
        "pageName" : "homepage",
        "siteName" : "classic",
        "siteType" : "site",
        "url" : "/api/sites/classic/pages/homepage"
      }
    },
    {
      "name" : "sitemap",
      "uri" : "/portal/classic/sitemap",
      "isVisible" : true,
      "visibility" : {"status" : "VISIBLE"},
      "iconName" : null,
      "displayName" : "SiteMap",
      "displayNames" : [...]
      "children" : null,
      "page" : {
        "pageName" : "sitemap",
        "siteName" : "classic",
        "siteType" : "site",
        "url" : "/api/sites/classic/pages/sitemap"
      }
    }
  ]
}

```



Note

The **url** element is the full URL but for sake of brevity it is relative for the examples in this document.

[Report a bug](#)

8.2.3.2. Retrieve a node

To retrieve a node with path **node-path**, the following syntax applies:

```
GET /api/{site-type}s/{site-name}/navigation/{node-path}
```

Table 8.4. Node Path Parameters

| parameter | type | default | description |
|-----------|------|---------|---|
| scope | int | n/a | Specifies how many nodes to load (-1 for all) |

Example 8.16. Retrieve a Node Request and Response

This example retrieves the node 'home'.

```
GET /api/sites/classic/navigation/home
```

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "name" : "home",
  "uri" : "/portal/classic/home",
  "isVisible" : true,
  "visibility" : {"status" : "VISIBLE"},
  "iconName" : null,
  "displayName" : "Home",
  "displayNames" : [...]
  "children" : null,
  "page" : {
    "pageName" : "homepage",
    "siteName" : "classic",
    "siteType" : "site",
    "url" : "/api/sites/classic/pages/homepage"
  }
}
```

You can control the displayName value for localized nodes with the **Accept-Language** header.

Example 8.17. Accept-Language Request and Response

This example retrieves the node 'home' to display the displayName in French.

```
GET /api/sites/classic/navigation/home
Accept-Language: fr
```

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "name" : "home",
  "uri" : "/portal/classic/home",
  "isVisible" : true,
```

```

    "visibility" : {"status" : "VISIBLE"},
    "iconName" : null,
    "displayName" : "Accueil",
    "displayNames" : [...]
    "children" : null,
    "page" : {
      "pageName" : "homepage",
      "siteName" : "classic",
      "siteType" : "site",
      "url" : "/api/sites/classic/pages/homepage"
    }
  }
}

```



Note

The **url** element is the full URL but for sake of brevity it is relative for the examples in this document.

[Report a bug](#)

8.2.3.3. Create a node

To create a node with path **node-path**, the following syntax applies:

```
POST /api/{site-type}s/{site-name}/navigation/{node-path}
```

Example 8.18. Create a Node Request and Response

This example creates a new node 'newnode' under the home navigation node.

```
POST /api/sites/classic/navigation/home/newnode
```

```

HTTP/1.1 200 OK
Content-Type: application/json
{
  "name" : "newnode",
  "uri" : "/portal/classic/home/newnode",
  "isVisible" : true,
  "visibility" : {"status" : "VISIBLE"},
  "iconName" : null,
  "displayName" : "newnode",
  "children" : null,
  "page" : null
}

```

[Report a bug](#)

8.2.3.4. Delete a node

To delete a node with path **node-path**, the following syntax applies:

```
DELETE /api/{site-type}s/{site-name}/navigation/{node-path}
```

Example 8.19. Delete Node Request and Response

This example deletes a node 'newnode'.

```
DELETE /api/sites/classic/navigation/home/newnode
```

```
HTTP/1.1 200 OK
Content-Type: application/json
```

[Report a bug](#)

8.2.3.5. Update a node

To update a node with path **node-path**, the following syntax applies:

```
PUT /api/{site-type}s/{site-name}/navigation/{node-path}
```

Example 8.20. Update Node Request and Response

Below is an example of updating the node 'home' to point to the sitemap page rather than the homepage.

```
PUT /api/sites/classic/navigation/home
```

```
Content-Type: application/json
```

```
{
  "page" : {
    "pageName" : "sitemap",
    "siteName" : "classic",
    "siteType" : "site"
  }
}
```

```
HTTP/1.1 200 OK
```

```
Content-Type: application/json
```

```
{
  "name" : "home",
  "uri" : "/portal/classic/home",
  "isVisible" : true,
  "visibility" : {"status" : "VISIBLE"},
  "iconName" : null,
  "displayName" : "Test",
  "displayNames" : [...],
  "children" : null,
  "page" : {
    "pageName" : "sitemap",
```

```

    "siteName" : "classic",
    "siteType" : "site",
    "url" : "/api/sites/classic/pages/sitemap"
  }
}

```



Note

The **url** element is the full URL but for sake of brevity it is relative for the examples in this document.

Example 8.21. Update displayName Request and Response

This example updates the English localized displayName to 'Home Page'.

```
PUT /api/sites/classic/navigation/home
```

```
Content-Type: application/json
```

```

{
  "displayNames" : [
    {
      "value" : "ホーム",
      "lang" : "ja"
    },
    {
      "value" : "Home Page",
      "lang" : "en"
    },
    {
      "value" : "首頁",
      "lang" : "zh-TW"
    },
    {
      "value" : "Domů",
      "lang" : "cs"
    },
    {
      "value" : "Accueil",
      "lang" : "fr"
    },
    {
      "value" : "主页",
      "lang" : "zh"
    },
    {
      "value" : "Главная",
      "lang" : "ru"
    },
    {
      "value" : "ترحيب",
      "lang" : "ar"
    },
  ],
}

```

```

    {
      "value" : "Trang chủ",
      "lang" : "vi"
    },
    {
      "value" : "Home",
      "lang" : "nl"
    },
    {
      "value" : "Inicio",
      "lang" : "es"
    },
    {
      "value" : "Principal",
      "lang" : "pt-BR"
    },
    {
      "value" : "Startseite",
      "lang" : "de"
    },
    {
      "value" : "홈",
      "lang" : "ko"
    },
    {
      "value" : "Home",
      "lang" : "it"
    },
    {
      "value" : "Додому",
      "lang" : "uk"
    },
    {
      "value" : "ཀློང་པྱུ་ཁོ་",
      "lang" : "ne"
    }
  ]
}

```

```

HTTP/1.1 200 OK
Content-Type: application/json
{
  "name" : "home",
  "uri" : "/portal/classic/home",
  "isVisible" : true,
  "visibility" : {"status" : "VISIBLE"},
  "iconName" : null,
  "displayName" : "Home Page",
  "displayNames" : [
    {
      "value" : "ホーム",
      "lang" : "ja"
    },
    {
      "value" : "Home Page",
      "lang" : "en"
    }
  ]
}

```

```

    },
    {
        "value" : "首頁",
        "lang" : "zh-TW"
    },
    {
        "value" : "Domů",
        "lang" : "cs"
    },
    {
        "value" : "Accueil",
        "lang" : "fr"
    },
    {
        "value" : "主页",
        "lang" : "zh"
    },
    {
        "value" : "Главная",
        "lang" : "ru"
    },
    {
        "value" : "ترحيب",
        "lang" : "ar"
    },
    {
        "value" : "Trang chủ",
        "lang" : "vi"
    },
    {
        "value" : "Home",
        "lang" : "nl"
    },
    {
        "value" : "Inicio",
        "lang" : "es"
    },
    {
        "value" : "Principal",
        "lang" : "pt-BR"
    },
    {
        "value" : "Startseite",
        "lang" : "de"
    },
    {
        "value" : "홈",
        "lang" : "ko"
    },
    {
        "value" : "Home",
        "lang" : "it"
    },
    {
        "value" : "Додому",
        "lang" : "uk"
    }

```

```

    },
    {
        "value" : "गृह पृष्ठ",
        "lang" : "ne"
    }
],
"children" : null,
"page" : {
    "pageName" : "homepage",
    "siteName" : "classic",
    "siteType" : "site",
    "url" : "/api/sites/classic/pages/homepage"
}
}

```



Note

The **url** element is the full URL but for sake of brevity it is relative for the examples in this document.



Note

We have to include all the displayNames since the REST API will overwrite the values on the server with whatever we supply in the request.

Since the REST API overwrites the displayName field we can delete existing displayNames by simply not including them.

Example 8.22. Deleting All Localized Display Names Except for English Request and Response

This example deletes all localized displayNames except for English.

```
PUT /api/sites/classic/navigation/home
```

```
Content-Type: application/json
```

```

{
    "displayNames" : [
        {
            "value" : "Home",
            "lang" : "en"
        }
    ]
}

```

```
HTTP/1.1 200 OK
```

```
Content-Type: application/json
```

```

{
    "name" : "home",

```

```

    "uri" : "/portal/classic/home",
    "isVisible" : true,
    "visibility" : {"status" : "VISIBLE"},
    "iconName" : null,
    "displayName" : "Home",
    "displayNames" : [
      {
        "value" : "Home Page",
        "lang" : "en"
      }
    ],
    "children" : null,
    "page" : {
      "pageName" : "homepage",
      "siteName" : "classic",
      "siteType" : "site",
      "url" : "/api/sites/classic/pages/homepage"
    }
  }
}

```



Note

The **url** element is the full URL but for sake of brevity it is relative for the examples in this document.

Example 8.23. Removing Localized Display Names Request and Response

This example removes all localized displayNames from the home navigation node and replaces them with a non-localized (simple) displayName 'Home'.

```
PUT /api/sites/classic/navigation/home
```

```
Content-Type: application/json
```

```
{
  "displayName" : "Home"
}
```

```
HTTP/1.1 200 OK
```

```
Content-Type: application/json
```

```
{
  "name" : "home",
  "uri" : "/portal/classic/home",
  "isVisible" : true,
  "visibility" : {"status" : "VISIBLE"},
  "iconName" : null,
  "displayName" : "Home",
  "displayNames" : [{"value" : "Home"}]
  "children" : null,
  "page" : {
    "pageName" : "homepage",
    "siteName" : "classic",

```

```

    "siteType" : "site",
    "url" : "/api/sites/classic/pages/homepage"
  }
}

```

[Report a bug](#)

8.2.4. Working with Templates

Portal administrators use templates to update existing portal, group, and user sites. Templates are also used to create new sites.

The scenarios described here explain the objective of templates.

Example 8.24. Portal Site Template

Consider a portal with 100 portal sites created from template A and 50 portal sites created from template B. A portal administrator wants to add two new pages to sites created from template A and three new pages to sites created from template B.

Using templates the portal administrator can perform this operation by using a single template for sites with two new pages created using template A and a single template for sites with three new pages created using template B.

Example 8.25. Group Site Template

Consider a portal with 20 different navigation groups. A portal administrator wants to add three new pages to ten navigation groups.

Using templates, the portal administrator can perform this operation by applying a single template to ten navigation groups.

Example 8.26. User Site Template

Consider a portal with 1000 users with individual dashboards. A portal administrator wants to modify the layout of the dashboards, or add new pages and navigation nodes for all the users.

In the absence of templates, the portal administrator uses the REST management API to perform this operation. The administrator exports all dashboards, modifies each dashboard and imports the modified dashboards with a merge strategy.

Using templates the portal administrator can perform the same operation by using a single template for all the dashboards.

[Report a bug](#)

8.2.4.1. Portal Site Template

To create a Portal site template, download the template using PUT command.

PUT `http://<host>:<port>/rest/private/managed-components/template/portal`
Content-Type: `application/zip` .

The template file is a .zip file with following structure:

```
/portal
/portal/template
/portal/template/<name_of_template1>
/portal/template/<name_of_template1>/navigation.xml
/portal/template/<name_of_template1>/pages.xml
/portal/template/<name_of_template1>/portal.xml
/portal/template/<name_of_template2>
/portal/template/<name_of_template2>/navigation.xml
/portal/template/<name_of_template2>/pages.xml
/portal/template/<name_of_template2>/portal.xml
...
```

Portal site template parameters

importMode

String with default setting as merge.

Possible values: **conserve**, **insert**, **merge**, and **overwrite**

targetSite

String with default value set to all sites.

If this targetSite attribute is not present template is applied in all portals created from templates defined in .zip. This attribute allows to filter and define in which sites this operation is performed. This attribute can have multiple instances of itself.

Example 8.27. Create a Portal Site Using Curl Tool

Run the curl tool command.

```
$ curl -i -H "Content-Type: application/zip" -u root:gtn -X PUT -T
"template-templateA.zip" http://localhost:8080/rest/private/managed-
components/template/portal?
importMode=merge&targetSite=site1&targetSite=site2
```

The output of the command.

Where template-templateA.zip:

```
portal/template/templateA/navigation.xml
portal/template/templateA/pages.xml
portal/template/templateA/portal.xml
```

Result:

Updates only site1 and site2 of sites created from templateA.

[Report a bug](#)

8.2.4.2. Group Sites Template

To create a Group site template, download the template using the PUT command.

PUT `http://<host>:<port>/rest/private/managed-components/template/group`
Content-Type: `application/zip`

The template file is a .zip file with following structure:

```
//group
/group/template
/group/template/navigation.xml
/group/template/pages.xml
/group/template/group.xml
...
```

Group site template parameters

importMode

String with default setting as merge.

Data import strategy. Possible values: **conserve**, **insert**, **merge**, **overwrite**

targetSite

String with default value set to all sites.

If this targetGroup attribute is not present template is applied in all group sites. This attribute allows to filter and define the sites on which this operation is performed. This attribute can have multiple instances of itself.

Example 8.28. Create Group Site using Curl Tool

Run the curl tool command.

```
$ curl -i -H "Content-Type: application/zip" -u root:gtn -X PUT -T
"template-group.zip" http://localhost:8080/rest/private/managed-
components/template/group?
importMode=merge&targetGroup=myorg/mygroup1&targetGroup=myorg/mygroup2
```

The output of the command

Where template-group.zip:

```
group/template/navigation.xml
group/template/pages.xml
group/tempalte/group.xml
```

Result:

"myorg/mygroup1" and "myorg/mygroup2" groups sites created are updated with MERGE strategy using template defined in template-group.zip.

[Report a bug](#)

8.2.4.3. User Sites Template

To create a User site template, download the template using the PUT command.

PUT `http://<host>:<port>/rest/private/managed-components/template/user`
Content-Type: `application/zip`

The template file is a .zip file with following structure:

```
/user
/user/template
/user/template/navigation.xml
/user/template/pages.xml
/user/template/user.xml
...
```

Group site template parameters

importMode

String with default setting as merge.

Data import strategy. Possible values: **conserve**, **insert**, **merge**, and **overwrite**

targetExpr

String with default value set to all users.

This attribute defines a search expression for user sites on which the template operation is performed. This attribute has priority over targetUser attribute.

targetUser

Is a String with default value set to all users.

If targetExpr is not defined, targetUser can define user target for template operation. This attribute can have multiple instances.

dashboardMode

String with default value set to none.

Creates dashboard for user. If this attribute is not present template is applied on users with a dashboard. Possible values: create.

Example 8.29. User Site using Curl Tool

Run the curl tool command.

```
$ curl -i -H "Content-Type: application/zip" -u root:gtn -X PUT -T
"template-user.zip" http://localhost:8080/rest/private/managed-
components/template/user?importMode=merge&targetExpr=user*
```

The output of the command

```
Where template-user.zip:

user/template/navigation.xml
user/template/pages.xml
```

```
user/template/user.xml
```

Result:

All dashboards from users starting with "user" updated with MERGE strategy using template defined in template-user.zip

[Report a bug](#)

Chapter 9. Organization API

9.1. Organization API

The `exo.platform.services.organization` package has five main components:

User

The **User** component contains basic information about a user; such as username, password, first name, last name, and email address.

User Profile

The **User Profile** component contains extra information about a user, such as user's personal information, and business information. You can also add additional information about a user if your application requires it.

Group

The **Group** component contains a group graph.

Membership Type

The **Membership Type** component contains a list of predefined membership types.

Membership

The **Membership** component connects a User, a Group and a Membership Type.

A user can have one or more memberships within a group. For example: *User A* can have the '*member*' and '*admin*' memberships in group */user*. A user belongs to a group if he has at least one membership in that group.

The **OrganizationService** component is an additional component that serves as an entry point into the Organization API. It provides handling functionality for the five components.

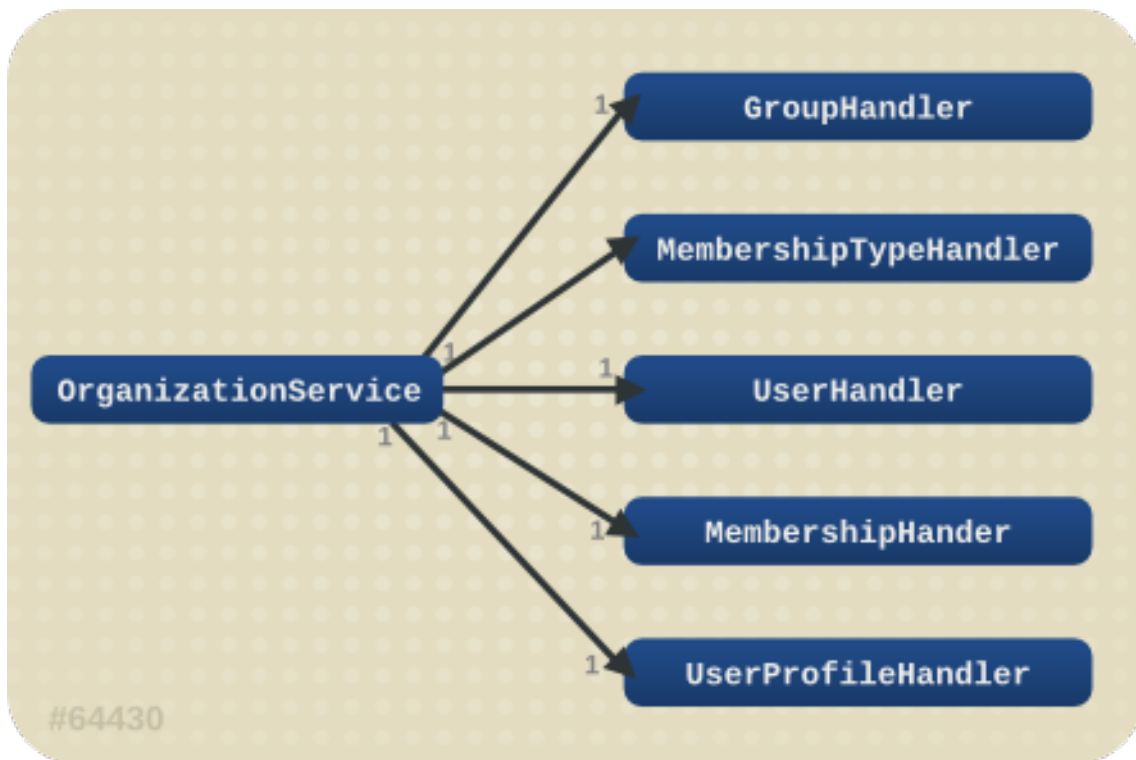


Figure 9.1. Handler Objects Accessible by the OrganizationService Component

By exposing the Organization API, the **OrganizationService** component provides developers with access to handler objects for managing each of the five components:

- ✦ UserHandler
- ✦ UserProfileHandler
- ✦ GroupHandler
- ✦ MembershipTypeHandler
- ✦ MembershipHandler

The five central API components are designed to be similar to persistent entities and handlers are specified similarly to data access objects (DAO).

Organization API simply describes a contract, meaning it is not a concrete implementation. The described components are interfaces, allowing for different concrete implementations. In practical terms the existing implementation can be replaced with a different one.

[Report a bug](#)

Chapter 10. Accessing User Profile

The following code retrieves the details for a logged-in user:

```
// Alternative context: WebuiRequestContext context =
WebuiRequestContext.getCurrentInstance() ;
PortalRequestContext context = PortalRequestContext.getCurrentInstance()
;
// Get the id of the user logged
String userId = context.getRemoteUser();

// Retrieve OrganizationService but it works only from WebUI code. See
variants below in documentation
OrganizationService orgService =
getApplicationComponent(OrganizationService.class) ;

// Request the information from OrganizationService:
if (userId != null)
{
    User user = orgService.getUserHandler().findUserByName(userId) ;
    if (user != null)
    {
        String firstName = user.getFirstName();
        String lastName = user.getLastName();
        String email = user.getEmail();
    }
}
```

Below are two alternatives for retrieving the Organization Service:

- ```
OrganizationService service = (OrganizationService)

ExoContainerContext.getCurrentContainer().getComponentInstanceOfType
(OrganizationService.class);
```
- ```
OrganizationService service = (OrganizationService)

PortalContainer.getInstance().getComponentInstanceOfType(Organizati
onService.class);
```

Both alternatives are probably better than **OrganizationService orgService = getApplicationComponent(OrganizationService.class)** because you can use them from your own portlets or servlet/portlet filters. The variant with **getApplicationComponent** works only from WebUI.

[Report a bug](#)

Part III. Portal Development

Chapter 11. Portal Life-cycle

[Report a bug](#)

11.1. Application Server Start and Stop

A portal instance is a set of web applications deployed as **EAR** and **WAR** archives. Portlets are also part of an enhanced WAR called a portlet application.

Red Hat JBoss Portal (JBoss Portal) does not require any particular setup for portlets in most common scenarios, and the **web.xml** file can remain without any JBoss Portal specific configuration.

During deployment, JBoss Portal will automatically inject a servlet into the portlet application to be able to interact with it. This feature is dependent on the underlying servlet container but will work out of the box on the proposed bundles.

[Report a bug](#)

11.1.1. Advanced WCI Registration

The portal integrates with the web container to perform tasks such as automatic detection and registration of web applications. This is used by the portal container to detect when portlets are deployed and is accomplished through the WCI (Web Container Integration) component.

Some applications, particularly Spring based portlets, may have requirements that specific servlets are started before any portlets are initialized. Although portlets and servlet initialization order are meant to be independent of each other, Red Hat JBoss Portal provides a method to circumvent limitations imposed by these specific third-party applications.

As a workaround to this issue, the following advanced features have been integrated into the WCI component;

Advanced Features

Disabling Automatic registration

WCI registers all web applications by default. The portlet container analyzes the registered applications and initializes any portlets contained within them.

To prevent web applications from being automatically registered by the WCI component, set the ***gatein.wci.native.DisableRegistration*** context-param to **true** in the **web.xml** file of the application.

```
<!-- Disable the Native Application Registration -->
<context-param>
  <param-name>gatein.wci.native.DisableRegistration</param-
name>
  <param-value>true</param-value>
</context-param>
```

Manual application deployment.

If you have disabled the automatic registration of your application in the first step, the portal container will not know about any of the portlets contained and will not be able to

initialize them. WCI does have a servlet which can be used to manually register the web application. Since servlets can specify when they are deployed with regards to other servlets, we can use this to specify that the web application gets registered by WCI after another servlet has already been started. This means that the a servlet, for example the Spring servlet, can be initialized before any of the portlets.

Below is an example **web.xml** file configured to ensure the **MyCustomServlet** will be initialized before the webapp is registered by WCI:

```
<!-- Disable the Native Application Registration -->
<context-param>
  <param-name>gatein.wci.native.DisableRegistration</param-
name>
  <param-value>true</param-value>
</context-param>
```

```
<!-- Register the Web Application Manually -->
<servlet>
  <servlet-name>GateInServlet</servlet-name>
  <servlet-class>org.gatein.wci.api.GateInServlet</servlet-
class>
  <load-on-startup>1</load-on-startup>
</servlet>
```

```
<!-- Custom Servlet which will be initialised before the webapp is
registered in WCI -->
<servlet>
  <servlet-name>MyCustomServlet</servlet-name>
  <servlet-class>my.custom.Servlet</servlet-class>
  <load-on-startup>0</load-on-startup>
</servlet>
```

```
<!-- Servlet Mapping for the Manual Registration -->
<servlet-mapping>
  <servlet-name>GateInServlet</servlet-name>
  <url-pattern>/gateinservlet</url-pattern>
</servlet-mapping>
```

[Report a bug](#)

11.2. The Command Servlet

The **CommandServlet** is called by the portlet container for requests to particular portlets, it also includes some **init** code when the portal is launched. This servlet (**org.gatein.wci.api.GateInServlet**) is automatically added during the deployment of each portlet application and mapped to **/gateinservlet**.

This is equivalent to adding the following into **web.xml**.



Note

The servlet is already configured. This example is for information only.

```

<servlet>
  <servlet-name>GateInServlet</servlet-name>
  <servlet-class>org.gatein.wci.api.GateInServlet</servlet-class>
  <load-on-startup>0</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>GateInServlet</servlet-name>
  <url-pattern>/gateinservlet</url-pattern>
</servlet-mapping>

```

It is possible to filter on the **GateInServlet** by filtering the URL pattern used by the servlet mapping.

This example would create a servlet filter that calculates the time of execution of a portlet request.

The filter class:

```

package org.example;

import java.io.IOException;

import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class MyFilter implements javax.servlet.Filter {

    public void doFilter(ServletRequest request, ServletResponse response,
        FilterChain chain) throws IOException, ServletException
    {
        long beforeTime = System.currentTimeMillis();
        chain.doFilter(request, response);
        long afterTime = System.currentTimeMillis();
        System.out.println("Time to execute the portlet request (in ms): " +
            (afterTime - beforeTime));
    }

    public void init(FilterConfig config) throws ServletException
    {
    }

    public void destroy()
    {
    }
}

```

The Java EE web application configuration file (**web.xml**) of the portlet is the file on which we want to know the time to serve a portlet request.

As mentioned above nothing specific to Red Hat JBoss Portal needs to be included, only the URL pattern to set has to be known.

```
<?xml version="1.0"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
  version="2.5">

  <filter>
    <filter-name>MyFilter</filter-name>
    <filter-class>org.example.MyFilter</filter-class>
  </filter>

  <filter-mapping>
    <filter-name>MyFilter</filter-name>
    <url-pattern>/gateinservlet</url-pattern>
    <dispatcher>INCLUDE</dispatcher>
  </filter-mapping>

</web-app>
```



Note

It is important to set **INCLUDE** as dispatcher as the portal will always hit the **GateInServlet** through a request dispatcher. Without this, the filter will not be triggered, unless direct access to a resource (such as an image) is set.

[Report a bug](#)

Chapter 12. Portal Containers

In a single instance (or cluster) of JBoss Portal, multiple portals can be running and share resources with other portals with two levels of granularity:

title

Portal Containers

A portal container can host multiple sites, and a JBoss Portal instance can host multiple portal containers

Site

A site can have a unique identity, with its own skin applied to a set of pages.

The highest-level component of Red Hat JBoss Portal (JBoss Portal) is the **portal container**. A portal container can host multiple **Sites**. Those two components have a unique identifier that can be found in the default URL mapping according to the following scheme:

<http://localhost:8080/<portalcontainer>/<site>>

When creating a website, you can either create a portal container or extend an existing one. Extending an existing portal container, such as the default one provided with JBoss Portal, is the recommended option because you only need to customize it to suit your requirements. Another benefit of the extension method is that upgrades consist of copying the newer distribution archives in place of the originals. This upgrade method is not possible if the distribution archives have been modified.

While running multiple portal containers is possible, it is recommended to keep those on separate installations. Note that multiple websites can run in a single portal container and share some services.

The procedure for creating portal containers and extending existing portal containers is similar: create an enterprise archive (EAR) containing configuration details, runnable code and static resources.

[Report a bug](#)

Chapter 13. Skinning the Portal

A portal visual identity is made of HTML produced as a result of portal aggregation (the components that make a page like columns and rows combined with the content produced by the portlets) and associated CSS files.

Red Hat JBoss Portal (JBoss Portal) allows to deploy multiple skins consisting of CSS files, which makes it possible to apply styling to the page compositions and components of a page (portlets). Different skins can be applied to the different websites, also if made available to the users, they can choose their preferred skin.

JBoss Portal provides robust skinning support for the entire portal User Interface (UI). This includes support for skinning all of the common portal elements as well as being able to provide custom skins and window decoration for individual portlets. This has been designed with common graphic resource reuse and ease of development in mind.

[Report a bug](#)

13.1. Skin Components

The skin of a page is composed of three separate parts:

Portal Skin

The portal skin contains the CSS styles for the portal and its various UI components. This must include all UI components, except for window decorators and portlet-specific styles.

Window Styles

The CSS styles associated with the portlet window decorators. The window decorators contain the control buttons and borders surrounding each portlet. Individual portlets can have their own window decorator selected or be rendered without one.

Portlet Skins

The portlet skins dictate how portlets are rendered on the page. There are two ways they can be implemented:

Portlet Specification CSS Classes

The portlet specification defines a set of CSS classes available to portlets. The portal provides these classes as part of the portal skin. This allows each portal skin to define its own look and feel for these default values.

Portlet Skins

The portal provides a means for portlet CSS files to be loaded based on the current portal skin. This allows a portlet to provide different CSS styles to better match the current portal look and feel. Portlet skins provide a much more customizable CSS experience in contrast to using the portlet specification CSS classes.



CSS Classes

The window decorators and the default portlet specification CSS classes are separate types of skinning components. They must be included as part of the overall portal skin otherwise they will not be displayed correctly.

A portlet skin does not need to be included as part of the portal skin, and can be included within the portlet's web application.

[Report a bug](#)

13.2. Skin Selection

[Report a bug](#)

13.2.1. Skin Selection Through the User Interface

The easiest way to change a skin is to select it through the user interface. An administrator can change the default skin for the portal, or a logged in user can select which skin they would prefer to be displayed.

For more information about changing skins, see the *User Guide*.

[Report a bug](#)

13.2.2. Setting the Default Skin within the Configuration Files

The default skin can also be configured using the portal configuration files. This allows the portal to have the new default skin ready for use when first started.

The default skin of the portal is called **Default**. To change this value add a **skin** tag in the **\$JPP_HOME/gatein/gatein.ear/portal.war/WEB-INF/conf/portal/portal/classic/portal.xml** configuration file.

Example 13.1. Changing a Skin Name

To change the skin to **MySkin** make the following changes:

```
<portal-config>
  <portal-name>classic</portal-name>
  <locale>en</locale>
  <access-permissions>Everyone</access-permissions>
  <edit-permission>*:/platform/administrators</edit-permission>
  <skin>MySkin</skin>
  ...
</portal-config>
```

[Report a bug](#)

13.3. Skins in Page Markups

A skin contains CSS styles for the portal's components, but also shares components that may be reused in portlets. When the portal generates a portal page markup, it inserts stylesheet links in the page's **head** tag.

There are two main types of CSS links that will appear in the **head** tag: a link to the portal skin CSS file and a link to the portlet skin CSS files.

Portal Skin

The portal skin will appear as a single link to a CSS file. This link will contain contents from all the portal skin classes merged into one file. This allows the portal skin to be transferred as a single file instead of multiple smaller files.

Portlet Skin

Each portlet on a page may contribute its own style. The link to the portlet skin will only appear on the page if that portlet is loaded on the current page. A page may contain many portlet skin CSS links or none.

In the code fragment below you can see the two types of links:

```
<head>
...
<!-- The portal skin -->
<link id="CoreSkin" rel="stylesheet" type="text/css"
href="/eXoResources/skin/Stylesheet.css" />

<!-- The portlet skins -->
<link id="web_FooterPortlet" rel="stylesheet" type="text/css" href=
"/web/skin/portal/webui/component/UIFooterPortlet/DefaultStylesheet.css"
/>
<link id="web_NavigationPortlet" rel="stylesheet" type="text/css" href=
"/web/skin/portal/webui/component/UINavigationPortlet/DefaultStylesheet.c
ss" />
<link id="web_HomePagePortlet" rel="stylesheet" type="text/css" href=
"/portal/templates/skin/webui/component/UIHomePagePortlet/DefaultStylese
et.css" />
<link id="web_BannerPortlet" rel="stylesheet" type="text/css" href=
"/web/skin/portal/webui/component/UIBannerPortlet/DefaultStylesheet.css"
/>
...
</head>
```



Note

Window styles and the portlet specification CSS classes are included within the portal skin.

[Report a bug](#)

13.4. The Skin Service

The skin service manages the various types of skins. It is responsible for discovering and deploying skins into the portal.

[Report a bug](#)

13.4.1. Skin configuration

The portal automatically discovers web archives that contain a file descriptor for skins (**\$JPP_HOME/gatein/gatein.ear/portal.war/WEB-INF/gatein-resources.xml**). This file is responsible for specifying the portal, portlet and window decorators to be deployed into the skin service.

The full schema is available from http://www.gatein.org/xml/ns/gatein_resources_1_2.

Example 13.2. Sample Skin

This example shows where to define a skin (**MySkin**) with its CSS location, and how to specify some window decorator skins.

```
<gatein-resources>
  <portal-skin>
    <skin-name>MySkin</skin-name>
    <css-path>/skin/myskin.css</css-path>
    <overwrite>false</overwrite>
  </portal-skin>
</gatein-resources>

<!-- window style -->
<window-style>
  <style-name>MyThemeCategory</style-name>
  <style-theme>
    <theme-name>MyThemeBlue</theme-name>
  </style-theme>
  <style-theme>
    <theme-name>MyThemeRed</theme-name>
  </style-theme>
  ...
</window-style>
```

[Report a bug](#)

13.4.2. Resource Request Filter

Because of the portal's Right-To-Left support, all CSS files need to be retrieved through a Servlet filter and the web application must be configured to activate this filter. This is already done for the **\$JPP_HOME/gatein/gatein.ear/eXoResources.war** web application which contains the default skin.

Any new web applications containing skinning CSS files will need to have the following added to the **web.xml** file:

```
<filter>
  <filter-name>ResourceRequestFilter</filter-name>
  <filter-
class>org.exoplatform.portal.application.ResourceRequestFilter</filter-
class>
  </filter>
```



```
<filter-mapping>
<filter-name>ResourceRequestFilter</filter-name>
<url-pattern>*.css</url-pattern>
</filter-mapping>
```



Note

The **display-name** element must be specified in the **web.xml** for the skinning service to work properly with the web application.

[Report a bug](#)

13.5. The Default Skin

The default skin is located as part of the **\$JPP_HOME/gatein/gatein.ear/eXoResources.war**. The main files associated with the skin are:

WEB-INF/gatein-resources.xml

For the default portal skin, this file contains definitions for the portal skin, the window decorations that this skin provides as well as defining some JavaScript resources which are not related to the skin. The default portal skin does not directly define portlet skins. Portlet skins must be provided by the portlets themselves.

WEB-INF/web.xml

For the default portal skin, the **web.xml** of the **eXoResources.war** contains information which is mostly irrelevant to portal skinning.

skin/Stylesheet.css

This file is the main portal skin stylesheet. It is the main entry point to the CSS class definitions for the skin. The main content points of this file are:

```
/* Skin for the main portal page */
@import
url(DefaultSkin/portal/webui/component/UIPortalApplicationSkin.css
);
/* Skins for various portal components */
@import url(DefaultSkin/webui/component/Stylesheet.css);
/* Window decoration skins */
@import url(PortletThemes/Stylesheet.css);
/* The portlet specification CSS classes */
@import url(Portlet/Stylesheet.css);
```

This method imports other CSS stylesheet files (some of which may also import further CSS stylesheets) instead of defining all the CSS classes in this one file. Splitting the CSS classes between multiple files allows new skins to reuse parts of the default skin.

To reuse a CSS stylesheet from the default portal skin you would need to reference the default skin from **eXoResources**. For example; to include the window decorators from the default skin within a new portal skin you would need to use this import:

```
@import url(/eXoResources/skin/Portlet/Stylesheet.css);
```



Note

When the portal skin is added to the page, it merges all the CSS stylesheets into a single file.

[Report a bug](#)

13.6. Creating New Skins

13.6.1. Creating New Skins

[Report a bug](#)

13.6.2. Creating a New Portal Skin

New portal skins will need to be added to the portal through the skin service. Therefore, the web application which contains the skins will need to be properly configured for the skin service to discover them. This means properly configuring the **ResourceRequestFilter** and **gatein-resources.xml**.

[Report a bug](#)

13.6.2.1. Portal Skin Configuration

The **gatein-resources.xml** will need to specify the new portal skin. This will include the name of the new skin, where to locate its CSS stylesheet file and whether to overwrite an existing portal theme with the same name.

```
<gatein-resources>
  <portal-skin>
    <skin-name>MySkin</skin-name>
    <CSS-path>/skin/myskin.css</CSS-path>
    <overwrite>false</overwrite>
  </portal-skin>
</gatein-resources>
```

The default portal skin and window styles are defined in **\$JPP_HOME/gatein/gatein.ear/eXoResources.war/WEB-INF/gatein-resources.xml**.



CSS

The CSS for the portal skin needs to contain the CSS for all the window decorations and the portlet specification CSS classes.

[Report a bug](#)

13.6.2.2. Portal Skin Preview Icon

It is possible to see a preview of what the portal will look like when selecting a new skin. This functionality relies on the current skin being updated with skin icons for all other available skins. Otherwise it will not be able to show the previews.

It is recommended that preview icons of any other skins are included when creating a new portal skin and that the other skins are updated with your new portal skin preview.

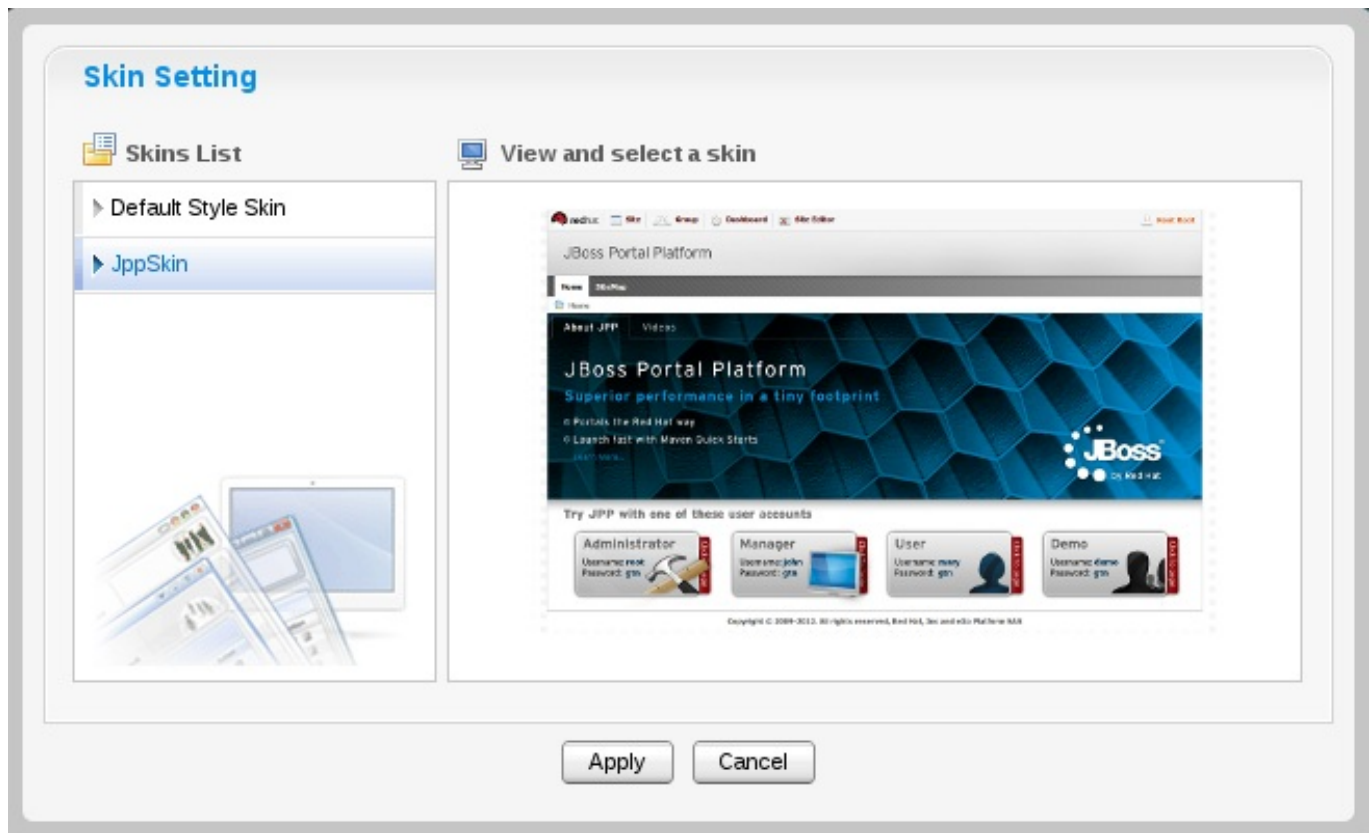


Figure 13.1. Portal_Skin_Change

The portal skin preview icon is specified through the CSS of the portal skin. In order for the current portal skin to be able to display the preview it must specify a specific CSS class and set the icon as the background.

For a portal named **MySkin**, it must define the following CSS class:

```
.UIChangeSkinForm .UIItemSelector .TemplateContainer .MySkinImage
```

In order for the default skin to display the skin icon for a new portal skin, the preview screenshot must be placed in

\$JPP_HOME/gatein/gatein.ear/eXoResources.war/skin/DefaultSkin/portal/webui/component/customization/UIChangeSkinForm/background.

The CSS stylesheet for the default portal must have the following updated with the preview icon CSS class. For a skin named **MySkin**, the following must be updated

\$JPP_HOME/gatein/gatein.ear/eXoResources.war/skin/DefaultSkin/portal/webui/component/customization/UIChangeSkinForm/Stylesheet.css.

```
.UIChangeSkinForm .UIItemSelector .TemplateContainer .MySkinImage {
    margin: auto;
    width: 329px; height:204px;
    background: url('background/MySkin.jpg') no-repeat top;
```

```

    cursor: pointer ;
}

```

[Report a bug](#)

13.6.3. Creating a New Window Style

Window styles are the CSS applied to window decorations. An administrator can decide which style of decoration is applied to the window when they add a new application or gadget to a page.

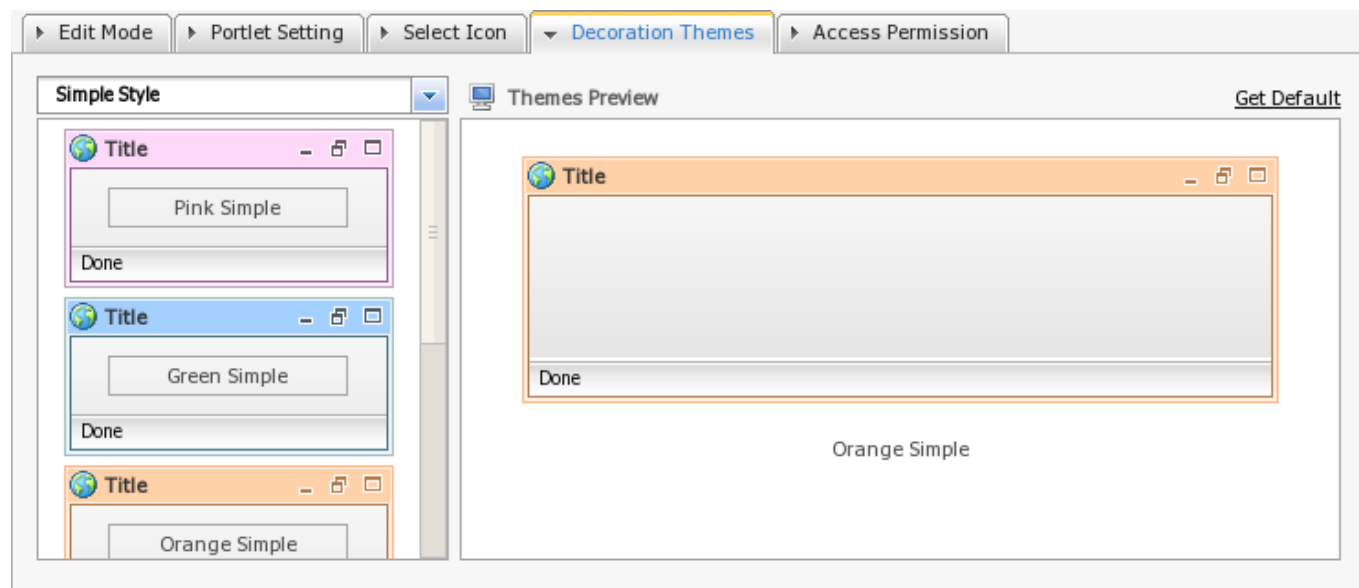


Figure 13.2. Window Styles

[Report a bug](#)

13.6.3.1. Window Style Configuration

Window Styles are defined within a **gatein-resources.xml** file which is used by the skin service to deploy the window style into the portal. Window styles can belong in a window style category. This category and the window styles will need to be specified in resources file.

The following **gatein-resources.xml** fragment will add **MyThemeBlue** and **MyThemeRed** to the **MyTheme** category.

```

<window-style>
  <style-name>MyTheme</style-name>
  <style-theme>
    <theme-name>MyThemeBlue</theme-name>
  </style-theme>
  <style-theme>
    <theme-name>MyThemeRed</theme-name>
  </style-theme>
</window-style>

```

The windows style configuration for the default skin is configured in:

\$JPP_HOME/gatein/gatein.ear/eXoResources.war/WEB-INF/gatein-resources.xml.



Window Styles and Portal Skins

When a window style is defined in **gatein-resources.xml** file, it will be available to all portlets regardless of whether the current portal skin supports the window decorator or not.

It is recommended that when a new window decorator is added that it be added to all portal skins or that all portal skins share a common stylesheet for window decorators.

[Report a bug](#)

13.6.3.2. Window Style CSS

In order for the skin service to display the window decorators, it must have CSS classes specifically named in relation to the window style name. The service will try and display CSS based on this naming convention. The CSS class must be included as part of the current portal skin for the window decorators to be displayed.

The location of the window decorator CSS classes for the default portal theme is located at: **JPP_HOME/gatein/gatein.ear/eXoResources.war/skin/PortletThemes/Stylesheet.css**.

Create the CSS file:

```
/*---- MyTheme ----*/
.MyTheme .WindowBarCenter .WindowPortletInfo {
    margin-right: 80px; /* orientation=lt */
    margin-left: 80px; /* orientation=rt */
}

.MyTheme .WindowBarCenter .ControlIcon {
    float: right; /* orientation=lt */
    float: left; /* orientation=rt */
    width: 24px;
    height: 17px;
    cursor: pointer;
    background-image: url('background/MyTheme.png');
}

.MyTheme .ArrowDownIcon {
    background-position: center 20px;
}

.MyTheme .OverArrowDownIcon {
    background-position: center 116px;
}

.MyTheme .MinimizedIcon {
    background-position: center 44px;
}

.MyTheme .OverMinimizedIcon {
    background-position: center 140px;
}
```

```
.MyTheme .MaximizedIcon {
    background-position: center 68px;
}

.MyTheme .OverMaximizedIcon {
    background-position: center 164px;
}

.MyTheme .RestoreIcon {
    background-position: center 92px;
}

.MyTheme .OverRestoreIcon {
    background-position: center 188px;
}

.MyTheme .NormalIcon {
    background-position: center 92px;
}

.MyTheme .OverNormalIcon {
    background-position: center 188px;
}

.MyTheme .Information {
    height: 18px; line-height: 18px;
    vertical-align: middle; font-size: 10px;
    padding-left: 5px; /* orientation=lt */
    padding-right: 5px; /* orientation=rt */
    margin-right: 18px; /* orientation=lt */
    margin-left: 18px; /* orientation=rt */
}

.MyTheme .WindowBarCenter .WindowPortletIcon {
    background-position: left top; /* orientation=lt */
    background-position: right top; /* orientation=rt */
    padding-left: 20px; /* orientation=lt */
    padding-right: 20px; /* orientation=rt */
    height: 16px;
    line-height: 16px;
}

.MyTheme .WindowBarCenter .PortletName {
    font-weight: bold;
    color: #333333;
    overflow: hidden;
    white-space: nowrap;
}

.MyTheme .WindowBarLeft {
    padding-left: 12px;
    background-image: url('background/MyTheme.png');
    background-repeat: no-repeat;
    background-position: left -148px;
}
```

```

.MyTheme .WindowBarRight {
    padding-right: 11px;
    background-image: url('background/MyTheme.png');
    background-repeat: no-repeat;
    background-position: right -119px;
}

.MyTheme .WindowBarCenter {
    background-image: url('background/MyTheme.png');
    background-repeat: repeat-x;
    background-position: left -90px;
    height: 21px;
    padding-top: 8px;
}

.MyTheme .MiddleDecoratorLeft {
    padding-left: 12px;
    background: url('background/MMyTheme.png') repeat-y left;
}

.MyTheme .MiddleDecoratorRight {
    padding-right: 11px;
    background: url('background/MMyTheme.png') repeat-y right;
}

.MyTheme .MiddleDecoratorCenter {
    background: #ffffff;
}

.MyTheme .BottomDecoratorLeft {
    padding-left: 12px;
    background-image: url('background/MyTheme.png');
    background-repeat: no-repeat;
    background-position: left -60px;
}

.MyTheme .BottomDecoratorRight {
    padding-right: 11px;
    background-image: url('background/MyTheme.png');
    background-repeat: no-repeat;
    background-position: right -30px;
}

.MyTheme .BottomDecoratorCenter {
    background-image: url('background/MyTheme.png');
    background-repeat: repeat-x;
    background-position: left top;
    height: 30px;
}

```

[Report a bug](#)

13.6.3.3. How to Set the Default Window Style

To set the default window style to be used for a portal you will need to specify the CSS classes for a theme called **DefaultTheme**.



Note

You do not need to specify the **DefaultTheme** in **gatein-resources.xml**.

[Report a bug](#)

13.6.4. How to Create New Portlet Skins

Portlets often require additional styles that may not be defined by the portal skin. The portal allows portlets to define additional stylesheets for each portlet and will append the corresponding **link** tags to the **head**.

The link ID will be of the form **{portletAppName}{PortletName}**.

For example: **ContentPortlet** in **content.war**, will give **id="contentContentPortlet"**.

To define a new CSS file to include whenever a portlet is available on a portal page, the following fragment is required in **gatein-resources.xml**.

```

<portlet-skin>
  <application-name>portletAppName</application-name>
  <portlet-name>PortletName</portlet-name>
  <skin-name>Default</skin-name>
  <css-path>/skin/DefaultStylesheet.css</css-path>
</portlet-skin>

<portlet-skin>
  <application-name>portletAppName</application-name>
  <portlet-name>PortletName</portlet-name>
  <skin-name>OtherSkin</skin-name>
  <css-path>/skin/OtherSkinStylesheet.css</css-path>
</portlet-skin>

```

This will load the **DefaultStylesheet.css** when the Default skin is used and the **OtherSkinStylesheet.css** when the **OtherSkin** is used.



Updating Portlet Skins

If the current portal skin is not defined as one of the supported skins, then the portlet CSS class will not be loaded. It is recommended that portlet skins are updated whenever a new portal skin is created.

[Report a bug](#)

13.6.4.1. Define a Custom CSS File

The portal does not serve CSS files directly, but uses a filter as well as a skin service in order to:

- Cache the CSS files.
- Merge them into one file if possible.

- » Add support for Right-To-Left (RTL) languages.

This causes the portal to create a non-functioning CSS link in `html-src-code`.

Procedure 13.1. Resolving a non-functioning CSS link

- » Add the following files to the custom portlet application:

WEB-INF/gatein-resources.xml:

```
<portlet-skin>
  <application-name>custom</application-name>
  <portlet-name>test</portlet-name>
  <skin-name>Default</skin-name>
  <css-path>/css/main.css</css-path>
</portlet-skin>
```

The value of the `<application-name>` element must match the value of the `<display-name>` element in **web.xml**.

The value of the `<portlet-name>` element must match the value of the `<portlet-name>` element in **\$JPP_HOME/standalone/configuration/gatein/portlet.xml**.

WEB-INF/web.xml:

```
<display-name>custom</display-name>

<filter>
  <filter-name>ResourceRequestFilter</filter-name>
  <filter-
class>org.exoplatform.portal.application.ResourceRequestFilter<
/filter-class>
</filter>

<filter-mapping>
  <filter-name>ResourceRequestFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

The value of the `<display-name>` element must match the value of the `<application-name>` element in **gatein-resources.xml**.

The **ResourceRequestFilter** must be added to the custom portlet application for proper CSS file handling within the Portal container.

WEB-INF/portlet.xml:

```
<portlet-name>test</portlet-name>
```

The value of the `<portlet-name>` element must match the value of the `<portlet-name>` element in **gatein-resources.xml**.

The final portlet application will be structured as illustrated below:

```
custom.war
```

```

├── CSS
│   └── main.css
├── WEB-INF
│   ├── classes
│   │   [...] custom portlet class [...]
│   ├── gatein-resources.xml
│   ├── portlet.xml
│   └── web.xml

```

[Report a bug](#)

13.6.4.2. Change Portlet Icons

Each portlet can be registered by a unique icon in the portlet registry or page editor. This icon can be changed by adding an image to the directory of the portlet web application:

✱ **skin/DefaultSkin/portletIcons/*portlet_name*.png**.

To be used correctly the icon must be named after the portlet.

For example; the icon for an account portlet named **AccountPortlet** would be located at:

✱ **skin/DefaultSkin/portletIcons/AccountPortlet.png**



Portlet Icons Directory

You must use **skin/DefaultSkin/portletIcons/** for the directory to store the portlet icon regardless of what skin is going to be used.

[Report a bug](#)

13.6.5. Create New Portlet Specification CSS Classes

The portlet specification defines a set of default CSS classes that should be available for portlets. These classes are included as part of the portal skin. Please see the portlet specification for a list of the default classes that should be available.

For the default portal skin, the portlet specification CSS classes are defined in:

\$JPP_HOME/gatein/gatein.ear/eXoResources.war/skin/Portlet/Stylesheet.css.

[Report a bug](#)

13.7. Tips and Tricks

[Report a bug](#)

13.7.1. CSS Hosted on CDN

Red Hat JBoss Portal permits Cascading Stylesheet (CSS) to be loaded from external sources using a Content Delivery Network (CDN).

Declaring external CSS sources is achieved by using **@import** statements in CSS files that reference the external source (CDN).



Important

Import statements can not be declared in the `<css-path>` element of the `gatein-resources.xml`, and must be declared using `@import` statements directly in CSS files.

Example 13.3. Example CSS Import Statements

In this example, the URLs map to the page protocol that embeds the Cascading Stylesheet

```
@import url('//my-fast-cdn.com/my-favorite-framework/1.2.3/cool-theme.css');
```

```
@import url('//another-cdn.com/another-dir/another-style.css');
```

[Report a bug](#)

13.7.2. Easier CSS Debugging

By default, CSS files are processed during deployment so that `@import` and `url(...)` statements are replaced by the content returned by the imported URLs. These imports are merged into a single CSS file that is stored in a cache and used by **SkinService**. This reduces the number of HTTP requests from the browser to the server.

Although the optimization is useful for a production environment, it may be easier to deactivate this optimization while debugging stylesheets. Set the Java system property `exo.product.developing` to `true` to disable the optimization.

For example, the property can be passed as a JVM parameter with `-D` option when running the portal.



Warning

This option may cause display bugs in some browsers.

```
./bin/standalone.sh -Dexo.product.developing=true
```

[Report a bug](#)

13.7.3. Some CSS Techniques

It is recommended that users have some experience with CSS before studying Red Hat JBoss Portal (JBoss Portal) CSS.

JBoss Portal relies heavily on CSS to create the layout and effects for the user interface. Some common techniques for customizing JBoss Portal CSS are explained below.

[Report a bug](#)

13.7.3.1. Decorator Pattern

The decorator is a pattern to create a contour or a curve around an area. In order to achieve this effect you need to create nine cells. The **BODY** is the central area to decorate. The other eight cells are distributed around the **BODY** cell. You can use the width, height and background image properties to achieve any desired decoration effect.

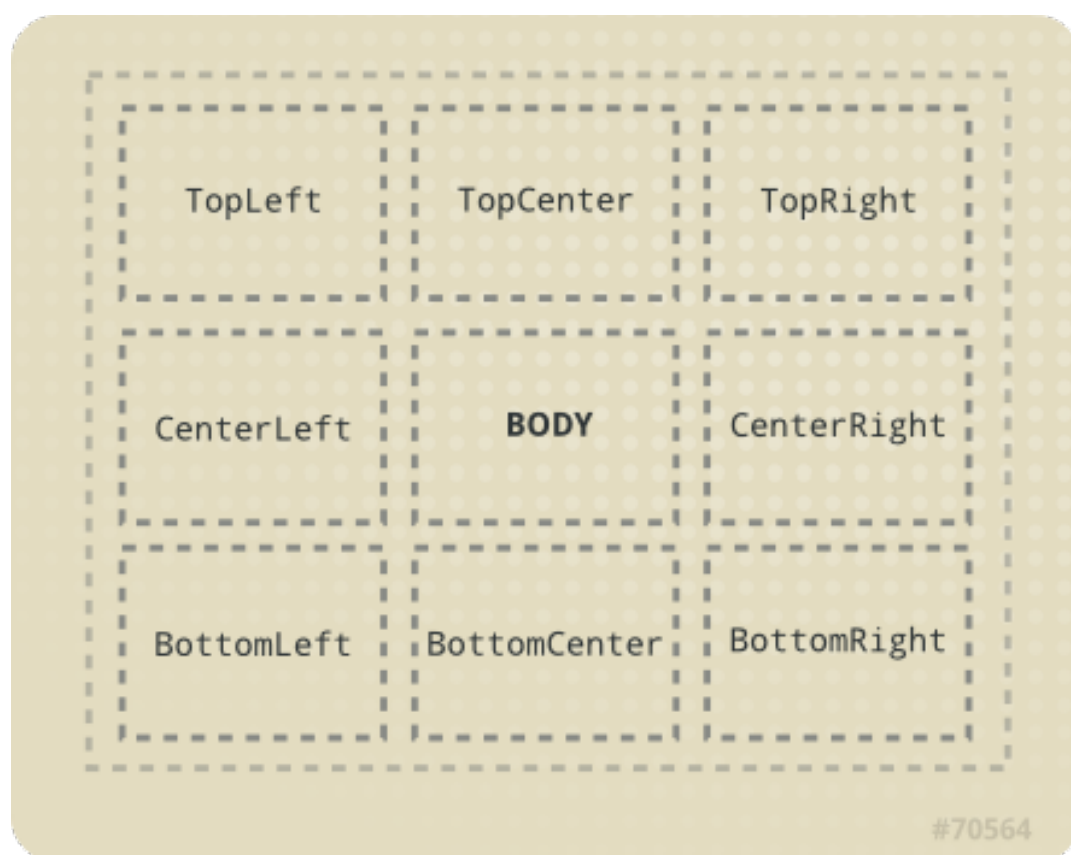


Figure 13.3. Decorator Pattern

```
<div class="Parent">
  <div class="TopLeft">
    <div class="TopRight">
      <div class="TopCenter"><span></span></div>
    </div>
  </div>
  <div class="CenterLeft">
    <div class="CenterRight">
      <div class="CenterCenter">BODY</div>
    </div>
  </div>
  <div class="BottomLeft">
    <div class="BottomRight">
```

```

    <div class="BottomCenter"><span></span></div>
  </div>
<div>
</div>

```

[Report a bug](#)

13.7.3.2. Left Margin Left Pattern

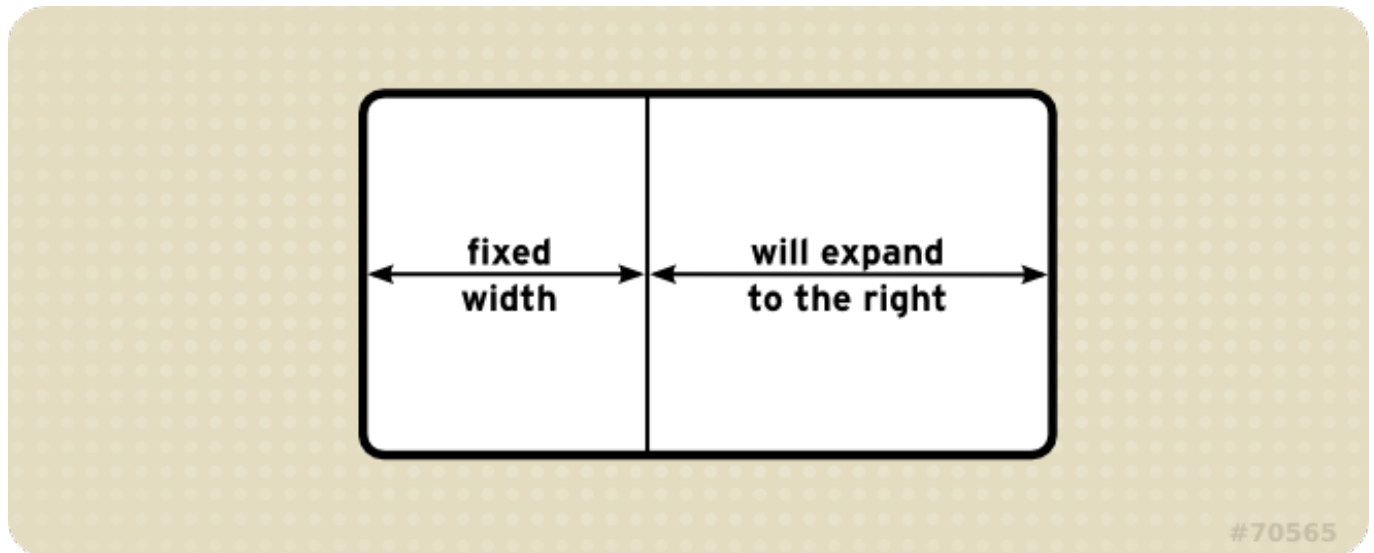


Figure 13.4. Left Margin Left Pattern

Left margin left pattern is a technique to create two blocks side by side. The left block will have a fixed size and the right block will take the rest of the available space. When the user resizes the browser the added or removed space will be taken from the right block.

```

<div class="Parent">
  <div style="float: left; width: 100px">
  </div>
  <div style="margin-left: 105px;">
  <div>
  <div style="clear: left"><span></span></div>
</div>

```

[Report a bug](#)

Chapter 14. Portal Extension

When extending an existing portal container, the name of the portal in the extension configuration is the same as the name of the existing portal container. The configuration (and other aspects) of the existing portal container can therefore be shadowed by the extension. Using this approach, many aspects of an available portal container can be customized, such as,

- » Customize groovy template for a portlet.
- » Customize skin for a portlet.
- » Customize CSS and images.
- » Customize navigation and pages.
- » Internationalization of navigation nodes.
- » Customize sign-in page.

[Report a bug](#)

14.1. How the Shadowing Mechanism Works

The following diagram illustrates how the shadowing mechanism works.

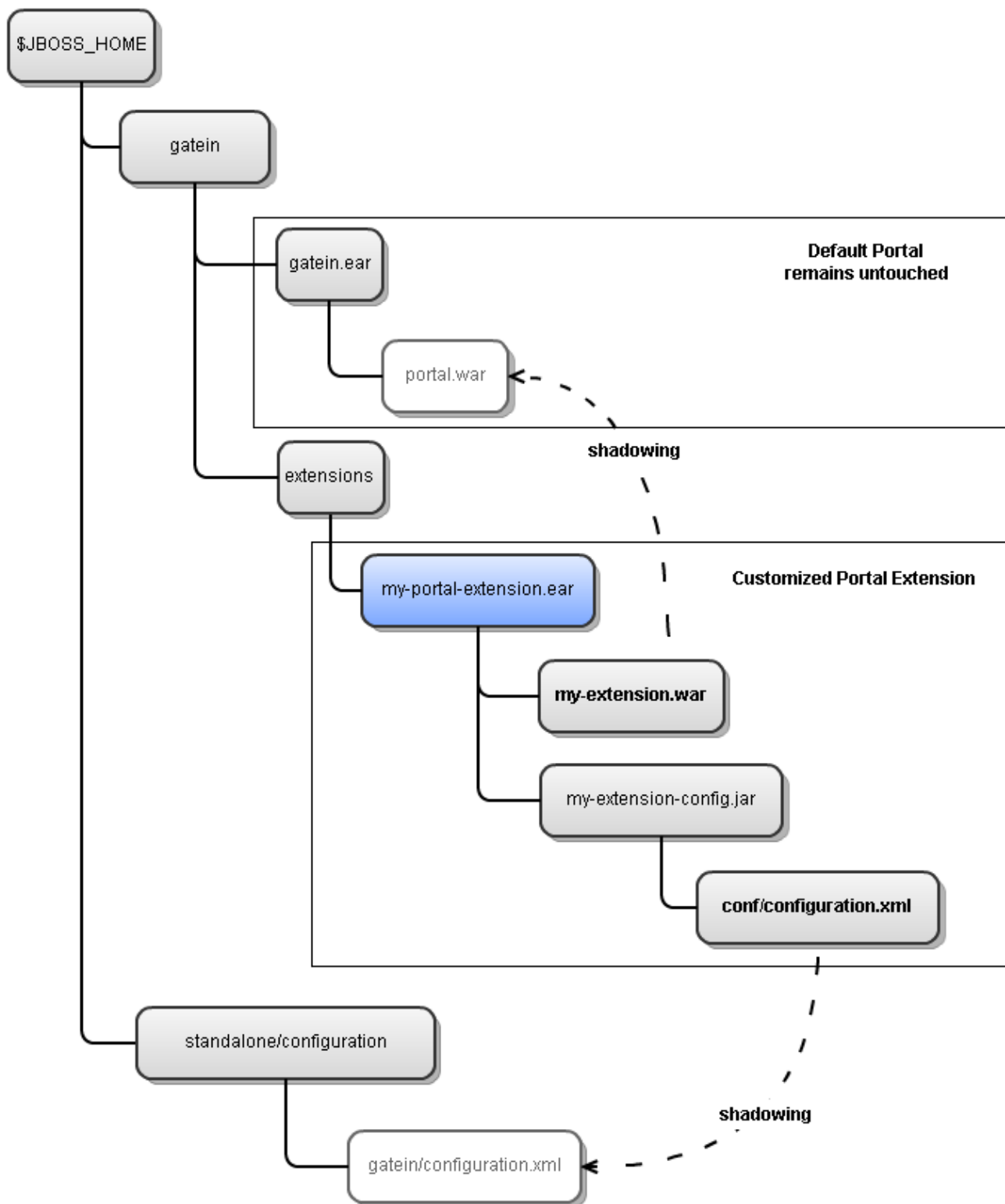


Figure 14.1. Shadowing Mechanism

The extension's EAR `conf/configuration.xml` file plays a crucial role in shadowing.

Example 14.1. configuration.xml

```

<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.exoplaform.org/xml/ns/kernel_1_2.xsd
http://www.exoplaform.org/xml/ns/kernel_1_2.xsd"

```

```

xmlns="http://www.exoplaform.org/xml/ns/kernel_1_2.xsd">
  <external-component-plugins>
    <!-- The full qualified name of the PortalContainerConfig -->
    <target-
component>org.exoplaform.container.definition.PortalContainerConfig</
target-component>
    <component-plugin>
      <!-- The name of the plugin -->
      <name>Add PortalContainer Definitions</name>
      <!-- The name of the method to call on the
PortalContainerConfig in order to register the
PortalContainerDefinitions -->
      <set-method>registerPlugin</set-method>
      <!-- The full qualified name of the
PortalContainerDefinitionPlugin -->

      <type>org.exoplaform.container.definition.PortalContainerDefinitionPl
ugin</type>
      <init-params>
        <object-param>
          <name>portal</name>
          <object
type="org.exoplaform.container.definition.PortalContainerDefinition">
            <!-- The name of the Portal Container: note that
a Portal Container called "portal"
already exists in the default GateIn
installation. Therefore, we actually redefine
that existing Portal Container here. -->
            <field name="name">
              <string>portal</string>
            </field>
            <!-- The name of the context name of the rest web
application -->
            <field name="restContextName">
              <string>rest</string>
            </field>
            <!-- The name of the realm -->
            <field name="realmName">
              <string>gatein-domain</string>
            </field>
            <field name="externalSettingsPath">
              <string>configuration.properties</string>
            </field>
            <!--
The list of all the context names that are
needed to initialize the Portal Container properly.
The order of the dependencies will define the
initialization order and also the order for
loading resources. When a resource with the
same path, say /dir/subdir/resource, is available
in more than one of the listed contexts, the
one from the context closest to the end of this list
will be chosen. Here we want the resources
available in gatein-portal-extension to win over all
other resources. Therefore we have added
gatein-portal-extension as the last element of the list.

```



```

-->
<field name="dependencies">
  <collection type="java.util.ArrayList">
    <value>
      <string>eXoResources</string>
    </value>
    <value>
      <string>portal</string>
    </value>
    <value>
      <string>dashboard</string>
    </value>
    <value>
      <string>exoadmin</string>
    </value>
    <value>
      <string>eXoGadgets</string>
    </value>
    <value>
      <string>gwtGadgets</string>
    </value>
    <value>
      <string>eXoGadgetServer</string>
    </value>
    <value>
      <string>rest</string>
    </value>
    <value>
      <string>web</string>
    </value>
    <value>
      <string>gatein-portal-
extension</string>
    </value>
  </collection>
</field>
</object>
</object-param>
</init-params>
</component-plugin>
</external-component-plugins>
</configuration>

```

In this code extract, the **name** parameter of the **PortalContainerDefinition** is set to **"portal"**. Because a portal container called "portal" already exists in the default portal installation, the container name is redefined in the above file.

Other resources available in an existing portal container can be customized in a similar way. See the following sections for more details.

[Report a bug](#)

14.2. Custom Groovy Template for a Portlet

Shadowing of built-in Groovy templates is controlled by the order of **<dependencies>** in the **conf/configuration.xml** file.

If a template with the same path (for example, **/dir/subdir/template.gtmpl**) is available in more than one of the dependencies, the template from the context closest to the end of the list of dependencies is chosen. To ensure that the template created in [Procedure 14.1, “Customizing HomePagePortlet”](#) is chosen, **gatein-portal-extension** is added as the last element in the list.



Note

This section mentions code from the Portal Extension Examples, which is available for download as part of a valid Red Hat subscription.

Procedure 14.1. Customizing HomePagePortlet

Complete this procedure to change the text and layout used in the default HomePagePortlet shipped with the platform in a custom extension.

1. Copy **\$JPP_HOME/gatein/gatein.ear/portal.war/templates/groovy/webui/component/UIHomePagePortlet.gtmpl**
2. Paste the file into the **war/src/main/webapp/templates/groovy/webui/component/** folder of the extension EAR.



Important

The file location in the custom extension must mirror the original location of the file for shadowing to work correctly.

3. Make the desired changes to the text and layout of the HomePagePortlet in the custom extension.
4. Deploy the extension EAR to the portal platform. The HomePagePortlet mirrors the changes made in the custom extension.

See Also:

- » [Section 14.1, “How the Shadowing Mechanism Works”](#)

[Report a bug](#)

14.3. Custom Skin for a Portlet



Note

This section mentions code from the Portal Extension Examples available for download as part of a valid Red Hat subscription.

The same guidelines that apply when creating a new portlet skin also apply when creating a custom skin for a built-in portlet distributed with the portal.

Declare the skin first in the **gatein-resources.xml** file, and then create the CSS and other supporting files such as backgrounds.

Shadowing of built-in Groovy templates is controlled by the order of **<dependencies>** in the configuration of the Portal Container (or Extension).

When the **HomePagePortlet** skin defined in **web.war** is available in more than one of the dependencies, the template from the context closest to the end of the list of dependencies is chosen. To ensure that the skin available in **gatein-portal-extension** is chosen, **gatein-portal-extension** is added as the last element in the list.

See Also:

✱ [Section 14.1, “How the Shadowing Mechanism Works”](#)

[Report a bug](#)

14.3.1. gatein-resources.xml

gatein-resources.xml is located in the **war/src/main/webapp/WEB-INF** sub-folder of the Portal Extension Quickstart project.

Example 14.2. gatein-resources.xml

```
<gatein-resources xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.gatein.org/xml/ns/gatein_resources_1_3
http://www.gatein.org/xml/ns/gatein_resources_1_3"
xmlns="http://www.gatein.org/xml/ns/gatein_resources_1_3">

<!-- The platform will load this CSS for HomePagePortlet instead of the
default one. -->
<portlet-skin>
  <!--
    The name of the web application containing the portlet.xml
    file in which
    BannerPortlet is defined. The application name is usually the
    same as
    the name of the WAR through which the application was
    deployed.
  -->
  <application-name>web</application-name>
  <!-- <portlet-name> value from the portlet.xml referenced above --
>
  <portlet-name>HomePagePortlet</portlet-name>
  <skin-name>Default</skin-name>
  <css-
path>/templates/skin/webui/component/UIHomePagePortlet/DefaultStyleshee
t.css</css-path>
</portlet-skin>
```

[Report a bug](#)

14.3.2. CSS and Images

In the `gatein-resources.xml` file in [Section 14.3.1, “gatein-resources.xml”](#), the custom skin is set to use the CSS file

`/templates/skin/webui/component/UIHomePagePortlet/DefaultStylesheet.css`

which translates to the Portal Extension Quickstart project

`war/src/main/webapp/templates/skin/webui/component/UIHomePagePortlet/DefaultStylesheet.css` file. In the CSS file itself, image files use relative paths as demonstrated in [Example 14.3, “DefaultStylesheet.css”](#):

Example 14.3. DefaultStylesheet.css

```
.UIHomePagePortlet .TRContainer .DotLine {
  background: url("DefaultSkin/background/Line.gif") no-repeat -2px top;
  height: 1px;
  width: 182px;
  margin: 7px auto;
}
```

See Also:

» [Chapter 13, Skinning the Portal](#)

[Report a bug](#)

14.4. Custom Navigation and Pages



Note

This section mentions code from the Portal Extension Examples available for download as part of a valid Red Hat subscription.

[Report a bug](#)

14.4.1. portal-configuration.xml

This configuration file is located in the `war/src/main/webapp/WEB-INF/conf/sample-ext/portal` directory of the Portal Extension quickstart. It is the starting point for navigation customization. There is normally no need to change any configuration in this file, although it is useful to know the location of this file.

For more information about `portal-configuration.xml`, see the "Portal Navigation Configuration" section in the *Administration and Configuration Guide*.

[Report a bug](#)

14.5. Navigation Node Types

There are three types of navigation nodes, as described in [Figure 14.2, “Types of Navigation Nodes”](#) :

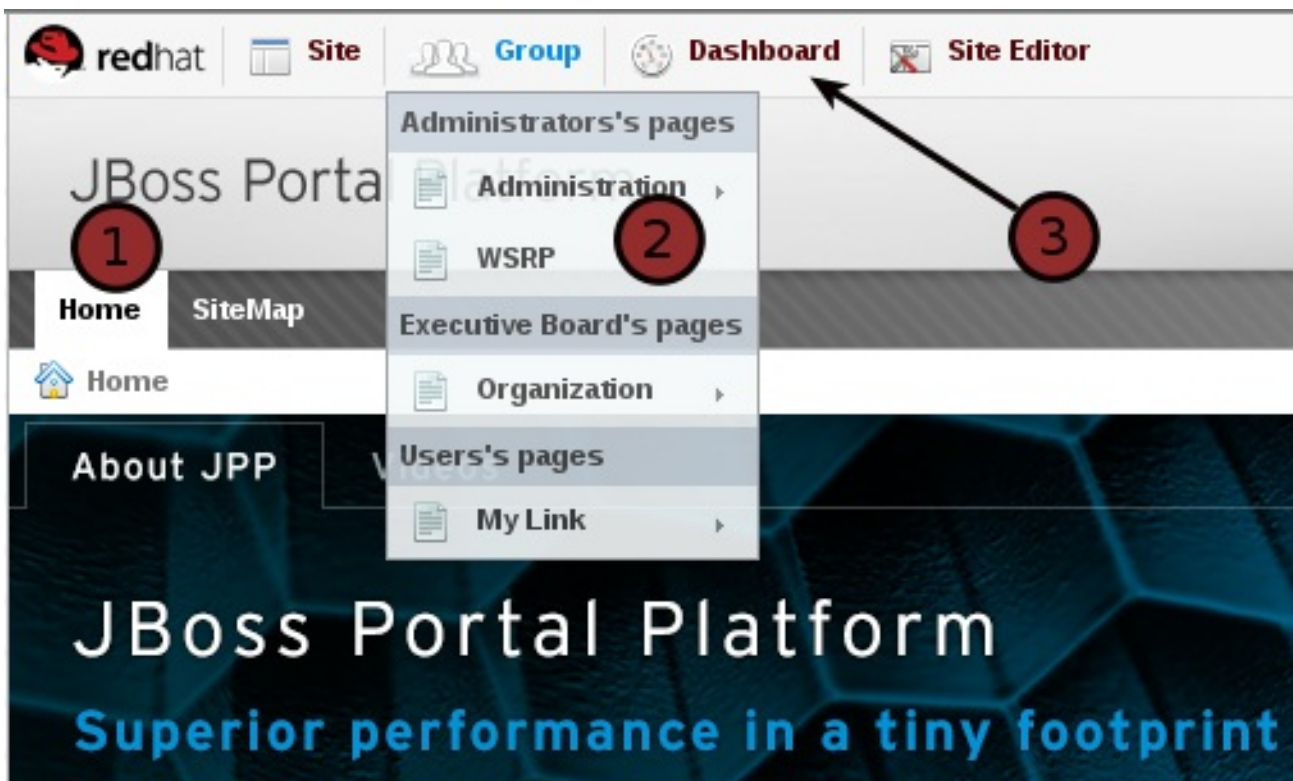


Figure 14.2. Types of Navigation Nodes

1 (Site)

Navigation nodes of the current site.

2 (User Group)

Navigation nodes of *user groups* which the current user is a member.

3 (User)

Navigation nodes of the current user.

Example 14.4. Adding a Navigation Node to a Site Navigation

The file `war/src/main/webapp/WEB-INF/conf/sample-ext/portal/portal/classic/navigation.xml` contains the classic site navigation which can be modified.

```
<node-navigation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.gatein.org/xml/ns/gatein_objects_1_3
http://www.gatein.org/xml/ns/gatein_objects_1_3"
xmlns="http://www.gatein.org/xml/ns/gatein_objects_1_3">
  <!-- Priority has no significance in site navigations. But it is
  required. See GTNPORTAL-2751 -->
  <priority>1</priority>
```

```

<!--
  In this file, we are adding a navigation node to the navigation
  of the site called "classic".
  The site-wide navigation is used e.g. in NavigationPortlet.
  Visibility of navigation nodes depends on <access-permissions> set
  in the sibling pages.xml file.
  See also portal.configuration section of the file ../../portal-
  configuration.xml.
-->
<page-nodes>
  <node>
    <name>page-just-added</name>
    <!--
      #{portal.extension.justAddedPage} is place holder for an
      internationalized string.
      See WEB-
      INF/classes/locale/navigation/portal/classic_en.properties
    -->
    <label>#{portal.extension.justAddedPage}</label>
  </node>
</page-nodes>
</node-navigation>

```

By appending the **page-just-added** navigation node into the portal navigation, the new site navigation node is now visible in the navigation bar.



The **page-just-added** navigation node refers to **portal::classic::justAddedPage** which is defined in the **pages.xml** file located in the same directory as the **navigation.xml** file.

```

<page-set xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.gatein.org/xml/ns/gatein_objects_1_3
http://www.gatein.org/xml/ns/gatein_objects_1_3"
xmlns="http://www.gatein.org/xml/ns/gatein_objects_1_3">
  <page>
    <name>justAddedPage</name>
    <title>Page Just Added</title>
    <!-- This page and any navigation nodes referring to it will be
    visible to all visitors (signed in or anonymous). -->
    <access-permissions>Everyone</access-permissions>
    <edit-permission>*/platform/administrators</edit-permission>
    <portlet-application>
      <portlet>
        <application-ref>gatein-portal-extension</application-
ref>
        <portlet-ref>JustAddedPortlet</portlet-ref>
      </portlet>
    <title>Just Added Portlet</title>
    <access-permissions>Everyone</access-permissions>
    <show-info-bar>false</show-info-bar>
    <show-application-state>false</show-application-state>
  </page>

```

```

        <show-application-mode>false</show-application-mode>
    </portlet-application>
</page>
</page-set>

```

Note that **page-just-added** navigation node and the related pages are visible to all portal visitors (signed in or anonymous) due to **<access-permissions>** set to **Everyone** in the above file.

Example 14.5. Adding a Navigation Node to a Group Navigation

To add a new navigation node visible to the members of **/platform/administrators** group, create a **navigation.xml** file located in **war/src/main/webapp/WEB-INF/conf/sample-ext/portal/group/platform/administrators/navigation.xml**. The name of the user group **/platform/administrators** forms part of the path.

```

<node-navigation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.gatein.org/xml/ns/gatein_objects_1_3
http://www.gatein.org/xml/ns/gatein_objects_1_3"
xmlns="http://www.gatein.org/xml/ns/gatein_objects_1_3">
<!-- Priority is currently ignored for importMode merge. But it is
required. See GTNPORTAL-2751 -->
<priority>1</priority>

<page-nodes>
    <node>
        <name>admins-sitemap</name>
        <!--
            #{platform.administrators.adminsSitemap} is place holder
for an internationalized string.
            See WEB-
INF/classes/locale/navigation/group/platform/administrators_en.properti
es
        -->
        <label>#{platform.administrators.adminsSitemap}</label>
        <page-
reference>group::/platform/administrators::adminsSitemap</page-
reference>
    </node>
</page-nodes>
</node-navigation>

```

The navigation node **admins-sitemap** refers to **adminsSitemap** page defined in the sibling **pages.xml** file:

```

<page-set xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.gatein.org/xml/ns/gatein_objects_1_3
http://www.gatein.org/xml/ns/gatein_objects_1_3"
xmlns="http://www.gatein.org/xml/ns/gatein_objects_1_3">
<!--
    adminsSitemap page contains just SiteMapPortlet. There is nothing
special about it.
    It is here just to demonstrate the adding of nodes to group
navigation in the sibling

```

```

navigation.xml file.
-->
<page>
  <name>adminsSitemap</name>
  <title>Admins' Sitemap</title>
  <access-permissions>*:/platform/administrators</access-permissions>
  <edit-permission>*:/platform/administrators</edit-permission>
  <portlet-application>
    <portlet>
      <application-ref>web</application-ref>
      <portlet-ref>SiteMapPortlet</portlet-ref>
    </portlet>
    <title>Sitemap</title>
    <access-permissions>*:/platform/administrators</access-
permissions>
    <show-info-bar>false</show-info-bar>
  </portlet-application>
</page>
</page-set>

```

Example 14.6. Adding a Navigation Node to a User Navigation

To add a new navigation node visible to a particular user, for example **root** create a **navigation.xml** file located in **war/src/main/webapp/WEB-INF/conf/sample-ext/portal/user/root/navigation.xml**. Note that the name of the user **root** is a part of the path.

```

<node-navigation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.gatein.org/xml/ns/gatein_objects_1_3
http://www.gatein.org/xml/ns/gatein_objects_1_3"
xmlns="http://www.gatein.org/xml/ns/gatein_objects_1_3">
  <!-- Priority has no significance in user navigations. But it is
required. See GTNPORTAL-2751 -->
  <priority>1</priority>

  <page-nodes>
    <node>
      <name>roots-extra-dashboard</name>
      <label>#{user.root.RootsExtraDashboard}</label>
      <page-reference>user::root::roots-dashboard</page-reference>
    </node>
  </page-nodes>
</node-navigation>

```

The navigation node **roots-extra-dashboard** refers to **roots-dashboard** page defined in the sibling **pages.xml** file:

```

<page-set xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.gatein.org/xml/ns/gatein_objects_1_3
http://www.gatein.org/xml/ns/gatein_objects_1_3"
xmlns="http://www.gatein.org/xml/ns/gatein_objects_1_3">
  <!--
    roots-dashboard page contains just DashboardPortlet. There is
nothing special about it.

```



```

    It is here just to demonstrate the adding of nodes to user
    navigation in the sibling
    navigation.xml file.
-->
<page>
  <name>roots-dashboard</name>
  <title>Root's Extra Dashboard</title>
  <access-permissions>*:/platform/users</access-permissions>
  <edit-permission>*:/platform/administrators</edit-permission>
  <portlet-application>
    <portlet>
      <application-ref>dashboard</application-ref>
      <portlet-ref>DashboardPortlet</portlet-ref>
    </portlet>
    <title>Root's Extra Dashboard</title>
    <access-permissions>*:/platform/users</access-permissions>
    <show-info-bar>false</show-info-bar>
  </portlet-application>
</page>
</page-set>

```

[Report a bug](#)

14.6. Internationalization of Navigation Nodes

As you can see in the above **navigation.xml** files the labels of navigation nodes can be internationalized. The files containing translation resources live under **war/src/main/webapp/WEB-INF/classes/locale/navigation**. The directory layout is similar to the one used for **navigation.xml** and **pages.xml** files:

Note that the navigation resource bundles also need to be named in **init.resources** parameter of **BaseResourceBundlePlugin** in **war/src/main/webapp/WEB-INF/conf/sample-ext/common/common-configuration.xml**:

Example 14.7. common-configuration.xml containing declarations of navigation resource bundles

```

<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.exoplaform.org/xml/ns/kernel_1_2.xsd
http://www.exoplaform.org/xml/ns/kernel_1_2.xsd"
xmlns="http://www.exoplaform.org/xml/ns/kernel_1_2.xsd">
<external-component-plugins>
  <!-- The full qualified name of the ResourceBundleService -->
  <target-
component>org.exoplatform.services.resources.ResourceBundleService</tar
get-component>
  <component-plugin>
    <!-- The name of the plugin -->
    <name>Sample ResourceBundle Plugin</name>
    <!-- The name of the method to call on the
ResourceBundleService in order to register the ResourceBundles -->
    <set-method>addResourceBundle</set-method>
    <!-- The full qualified name of the BaseResourceBundlePlugin --

```

```

>

<type>org.exoplatform.services.resources.impl.BaseResourceBundlePlugin<
/type>
  <init-params>
    <values-param>
      <name>init.resources</name>
      <description>Initiate the following resources during
the first launch.</description>
      <value>locale.portal.extension</value>
      <value>locale.navigation.user.root</value>
      <!--
        Note that we actually do not need to name
        locale.navigation.portal.classic and
        locale.navigation.group.platform.administrators
        here as they are in init.resources
        of the default GateIn installation. But it makes no
        harm to include them once again here.
      -->
      <value>locale.navigation.portal.classic</value>

<value>locale.navigation.group.platform.administrators</value>
    </values-param>
  </init-params>
</component-plugin>

```

[Report a bug](#)

14.7. Custom Internationalization Resource Bundles



Note

This section mentions code from the Portal Extension Examples available for download as part of a valid Red Hat subscription.

There are two resource bundle customization scenarios possible with a Portal Extension:

- ✧ to add new resource bundle items to the ones available in the default portal installation.
- ✧ to assign new values to resource bundle items available in default portal installation.

Both scenarios can be demonstrated using the **locale.portal.extension** resource bundle. To use this bundle, include both **init.resources** and **portal.resource.names** parameters for the **BaseResourceBundlePlugin<type>** directive in **war/src/main/webapp/WEB-INF/conf/sample-ext/common/common-configuration.xml**.

Example 14.8. common-configuration.xml containing declarations of navigation resource bundles

```

<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd
http://www.exoplatform.org/xml/ns/kernel_1_2.xsd"

```

```

xmlns="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd">
<external-component-plugins>
  <!-- The full qualified name of the ResourceBundleService -->
  <target-
component>org.exoplatform.services.resources.ResourceBundleService</tar
get-component>
  <component-plugin>
    <!-- The name of the plugin -->
    <name>Sample ResourceBundle Plugin</name>
    <!-- The name of the method to call on the
ResourceBundleService in order to register the ResourceBundles -->
    <set-method>addResourceBundle</set-method>
    <!-- The full qualified name of the BaseResourceBundlePlugin --
  >

  <type>org.exoplatform.services.resources.impl.BaseResourceBundlePlugin<
/type>
    <init-params>
      <values-param>
        <name>init.resources</name>
        <description>Initiate the following resources during
the first launch.</description>
        <value>locale.portal.extension</value>
        <value>locale.navigation.user.root</value>
        <!--
          Note that we actually do not need to name
          locale.navigation.portal.classic and
          locale.navigation.group.platform.administrators
          here as they are in init.resources
          of the default GateIn installation. But it makes no
          harm to include them once again here.
        -->
        <value>locale.navigation.portal.classic</value>

        <value>locale.navigation.group.platform.administrators</value>
      </values-param>
      <values-param>
        <name>portal.resource.names</name>
        <description>These resources are merged to a single
resource bundle which is accessible from anywhere
in GateIn. All these keys are located in the same
bundle, which is separated from the navigation
resource bundles.</description>
        <value>locale.portal.extension</value>
      </values-param>
    </init-params>
  </component-plugin>
</external-component-plugins>
</configuration>

```

The English version of `locale.portal.extension` is located in `war/src/main/webapp/WEB-INF/classes/locale/portal/extension_en.properties`.

Example 14.9. English version of `locale.portal.extension`

```

UIHomePagePortlet.Label.Slogan=Congratulations!
UIHomePagePortlet.Label.SubSlogan=You have just installed the GateIn
Portal Extension
UIHomePagePortlet.Label.Title=Sign in as:

```

UIHomePagePortlet.Label.SubSlogan is a new key which is not normally available in the default portal installation. However, **UIHomePagePortlet.Label.Slogan** is redefined in **extension_en.properties** shown above.

In **\$JPP_HOME/gatein/gatein.ear/web.war/WEB-INF/classes/locale/portlet/web/GroovyPortlet_en.properties** it is already defined as:

```

UIHomePagePortlet.Label.Slogan=The Best of eXo and JBoss
Portal<div>GateIn #{gatein.version}</div>

```

Within the Portal Extension, the new value **Congratulations!** is appended to it.

See Also:

✎ [Section 14.5, “Navigation Node Types”](#)

[Report a bug](#)

14.8. Custom Sign-in Page

It is possible to customize the sign-in page displayed to a user by configuring an extension.

Procedure 14.2. Creating a Custom Sign-in Page

1. Copy **\$JPP_HOME/gatein/gatein.ear/portal.war/login/jsp/login.jsp**.
2. Create a custom extension, and paste the file into **Custom_Extension.ear/Custom-WAR.war/login/jsp/login.jsp**.



Important

All logic relating to the sign-in page must be contained within the login page. Failing to do this may cause the portal to operate incorrectly.



Note

The file path must be the same in both the primary path and the extension path for mirroring to work correctly.

3. Edit the **login.jsp** file with the customization required for the extension.

Procedure 14.3. Creating a Custom Modal Window

1. Copy
`$JPP_HOME/gatein/portal.war/groovy/portal/webui/UILoginForm.gtmpl`.
2. In the custom extension created in [Procedure 14.2, "Creating a Custom Sign-in Page"](#), paste the file into **`Custom_Extension.ear/Custom-WAR.war/groovy/portal/webui/UILoginForm.gtmpl`**.



Note

The file path must be the same in both the primary path and the extension path for mirroring to work correctly.

3. Edit the **`UILoginForm.gtmpl`** file with the customization required for the extension.

Once these modifications are made, deploy the custom extension and test the functionality of the sign in page.

[Report a bug](#)

Chapter 15. Visual Identity

Visual Identity of GateIn Portal is given by applied skins. Skins are packages consisting of CSS files, background images, etc. Skins can be applied to whole Sites or Portlets. It is not only possible to deploy multiple skins and apply different skins to individual Sites and Portlets. The skin selection can also be made accessible to the users so that they can choose their preferred skin.



Note

See [Chapter 13, *Skinning the Portal*](#) for further details.

[Report a bug](#)

Chapter 16. Data Import Strategy

In the portal extension mechanism, developers can define an extension that portal data can be customized by.

There are several cases when it can be useful to define how to customize the Portal data; for example modifying, overwriting or inserting data into the data defined by the Portal. The portal also defines several modes for each case so that a developer only has to clarify the use-case and configure the extensions.

[Report a bug](#)

16.1. Import Strategy Overview

The 'Portal Data' term referred to in this section can be classified into three types of object data:

- ✧ Portal Configuration
- ✧ Page Data
- ✧ Navigation Data

Each type of object data has some differences in the import strategy.

The modes for the import strategy include the following:

- ✧ **CONSERVE**
- ✧ **MERGE**
- ✧ **INSERT**
- ✧ **OVERWRITE**

Each mode indicates how the portal data is imported. The import mode value is set whenever **NewPortalConfigListener** is initiated. If the mode is not set, the default value will be used. The default value is configurable as a `UserPortalConfigService` initial parameter. For example, the configuration below means that the default value is **MERGE**.

```
<component>
  <key>org.exoplatform.portal.config.UserPortalConfigService</key>
  <type>org.exoplatform.portal.config.UserPortalConfigService</type>
  <component-plugins>
    .....
  </component-plugins>
  <init-params>
    <value-param>
      <name>default.import.mode</name>
      <value>merge</value>
    </value-param>
  </init-params>
</component>
```

The way the import strategy works with the import mode will be clearly demonstrated in next sections for each type of data.

[Report a bug](#)

16.1.1. Portal Configuration

PortalConfig defines the portal name, permission, layout and some properties of a site. These information are configured in the *portal.xml*, *group.xml* or *user.xml*, depending on the site type. The PortalConfig importer performs a strategy that is based on the mode defined in NewPortalConfigListener, including **CONSERVE**, **INSERT**, **MERGE** or **OVERWRITE**. Let's see how the import mode affects the process of portal data performance:

- **CONSERVE**: There is nothing to be imported. The existing data will be kept without any changes.
- **INSERT**: When the portal configuration does not exist, create the new portal defined by the portal configuration definition. Otherwise, do nothing.
- **MERGE** and **OVERWRITE** have the same behavior. The new portal configuration will be created if it does not exist or update portal properties defined by the portal configuration definition.

[Report a bug](#)

16.1.2. Page Data

The import mode affects the page data import as the same as PortalConfig.



Note

If the Import mode is **CONSERVE** or **INSERT**, the data import strategy acts as if in the **MERGE** mode in the first data initialization of the Portal.

[Report a bug](#)

16.1.3. Navigation Data

The navigation data import strategy is processed at the import mode level as follows:

title

CONSERVE

If the navigation exists, leave it untouched. Otherwise, import data.

INSERT

Insert the missing description data, but add only new nodes. Other modifications remain untouched.

MERGE

Merge the description data, add missing nodes and update existing nodes.

OVERWRITE

Always destroy the previous data and recreate it.

In the portal navigation structure, each navigation can be referred to a tree which each node links to a page content. Each node contains some description data, such as label, icon, page reference, and

more. Therefore, the portal provides a way to insert or merge new data to the initiated navigation tree or a subtree.

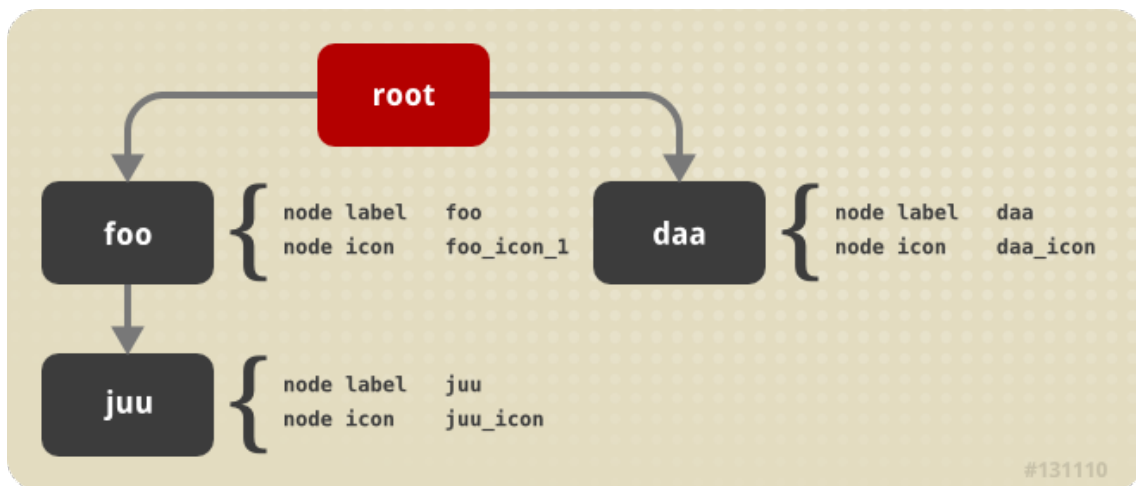


Figure 16.1. Data Import Strategy Navigation1

The merge strategy performs the recursive comparison of child nodes between the existing persistent nodes of a navigation and the transient nodes provided by a descriptor:

1. Start with the root nodes (which is the effective root node or another node if the parent URI is specified).
2. Compare the set of child nodes and insert the missing nodes in the persistent nodes.
3. Proceed recursively for each child having the same name.

Let's see the example with two navigation nodes in each import mode. In this case, there are two navigation definitions:

```

<node-navigation>
  <page-nodes>
    <node>
      <name>foo</name>
      <icon>foo_icon_1</icon>
    <node>
      <name>juu</name>
      <icon>juu_icon</icon>
    </node>
  </page-nodes>
</node-navigation>

```

```

<node-navigation>
  <page-nodes>
    <node>
      <name>foo</name>
      <icon>foo_icon_2</icon>
    </node>
  </page-nodes>
</node-navigation>

```

```

</node>
<node>
  <name>daa</name>
  <icon>daa_icon</icon>
</node>
</page-nodes>
</node-navigation>

```

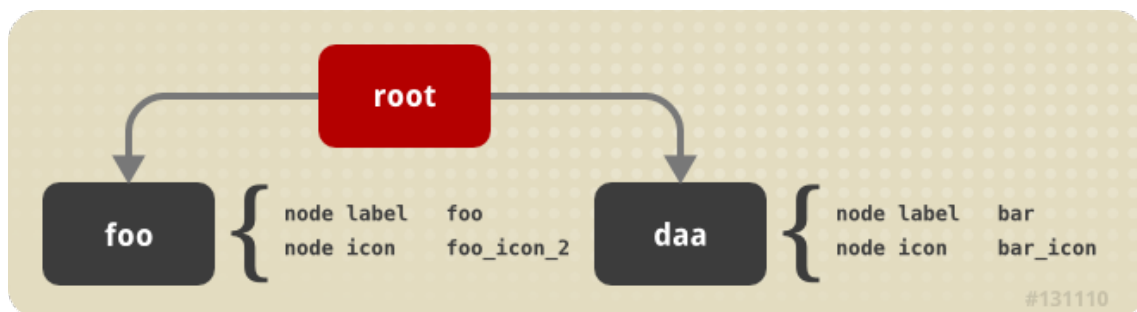


Figure 16.2. Data Import Strategy Navigation2

For example, the *navigation1* is loaded before *navigation2*. The Navigation Importer processes on two navigation definitions, depending on the Import Mode defined in portal configuration.

Import Mode Cases

Case 1: CONSERVE

With the **CONSERVE** mode, data are only imported when they do not exist. So, if the navigation has been created by the *navigation1* definition, the *navigation2* definition does not affect anything on it. We have the result as following

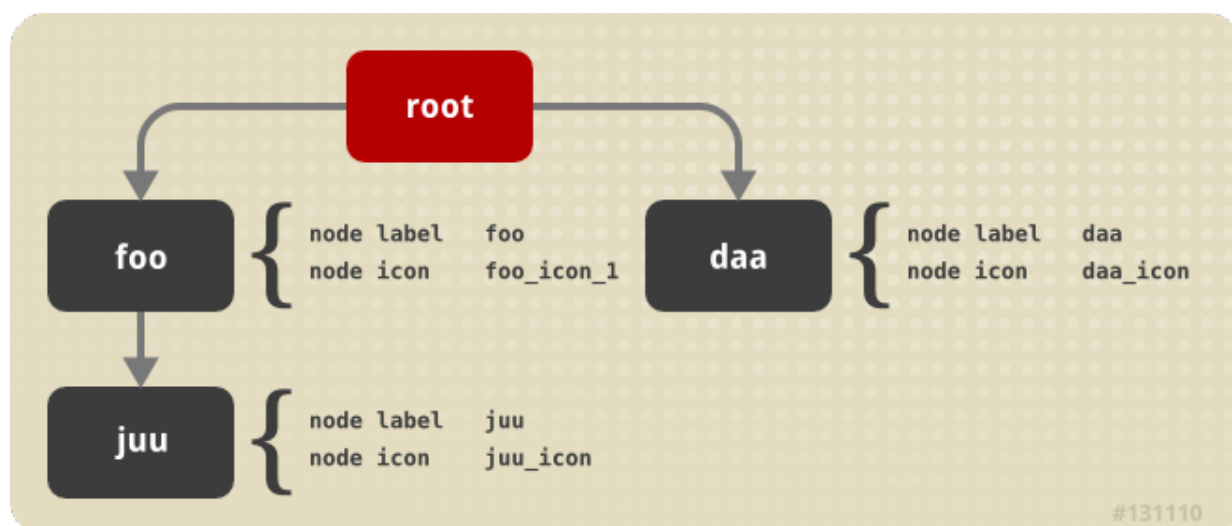


Figure 16.3. Data Import Strategy Navigation1

Case 2: INSERT

If a node does not exist, the importer will add new nodes to the navigation tree. You will see the following result:

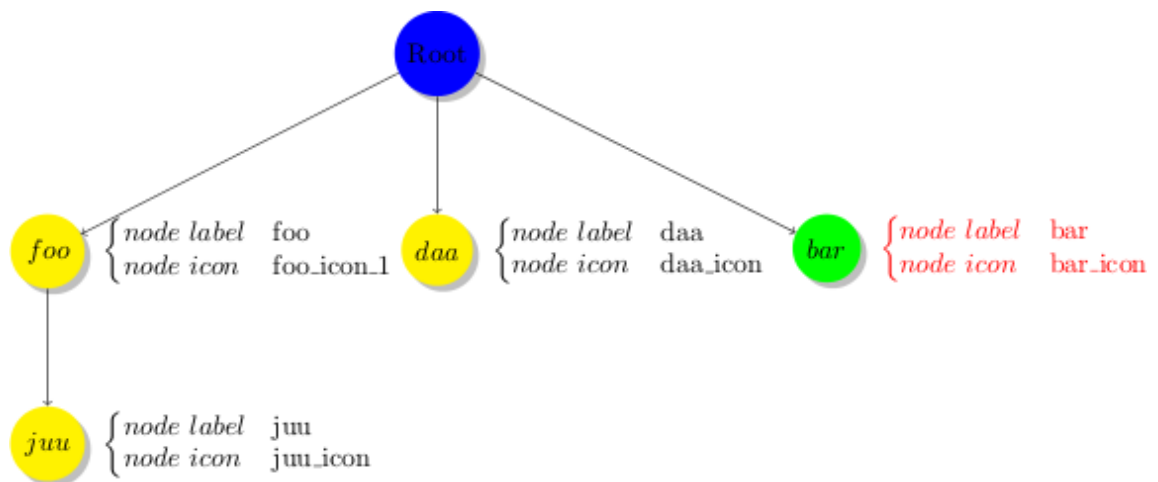


Figure 16.4. Data Import Strategy Navigation_Insert

Hereafter, the node 'bar' is added to the navigation tree, because it does not exist in the initiated data. Other nodes are kept in the import process.

Case 3: MERGE

The **MERGE** mode indicates that a new node is added to the navigation tree, and updates the node data (such node label and node icon in the example) if it exists.

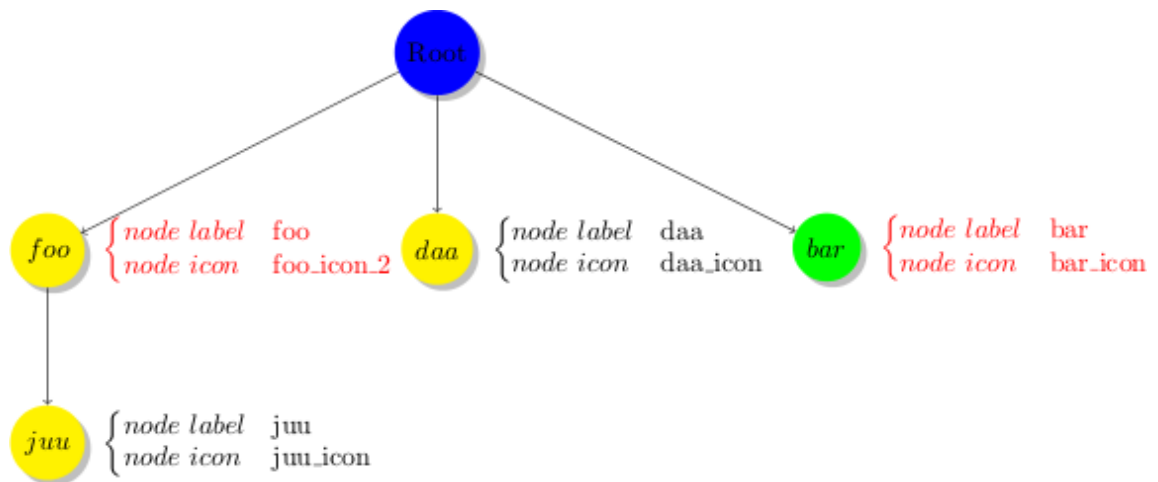


Figure 16.5. Data Import Strategy Navigation_Merge

Case 4: OVERWRITE

Everything will be destroyed and replaced with new data if the **OVERWRITE** mode is used.

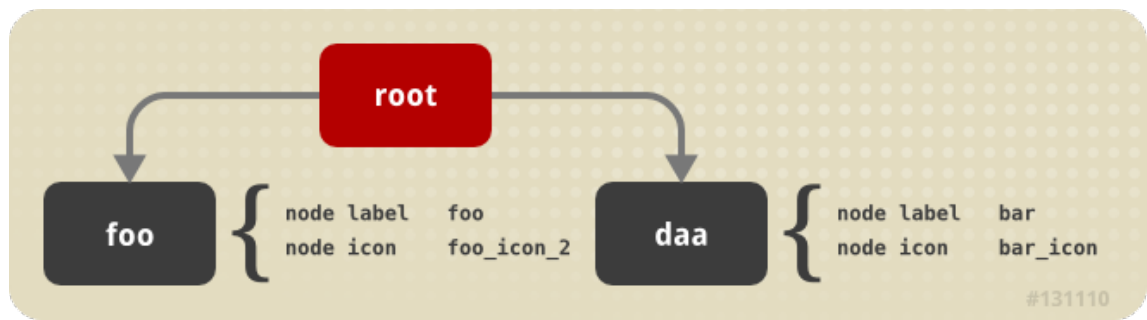


Figure 16.6. Data Import Strategy Navigation_2

[Report a bug](#)

Chapter 17. Right To Left (RTL) Framework

The text orientation depends on the current locale setting. The orientation is a Java 5 enum that provides a set of functionalities:

```
LT, // Western Europe
RT, // Middle East (Arabic, Hebrew)
TL, // Japanese, Chinese, Korean
TR; // Mongolian
public boolean isLT() { ... }
public boolean isRT() { ... }
public boolean isTL() { ... }
public boolean isTR() { ... }
```

[Report a bug](#)

17.1. Groovy templates

Orientation is defined by implicit variables passed into the groovy binding context:

Orientation

The current orientation as an Orientation.

isLT

The value of `orientation.isLT()`.

isRT

The value of `orientation.isRT()`.

dir

The string 'ltr' if the orientation is LT or the string 'rtl' if the orientation is RT.

[Report a bug](#)

17.2. Stylesheet

The skin service handles stylesheet rewriting to accommodate the orientation.

It works by appending -lt or -rt to the stylesheet name.

For instance:

`$JPP_HOME/gatein/gatein.ear/portal.war/templates/skin/portal/webui/component/UIFooterPortlet/DefaultStylesheet-rt.css` will return the same stylesheet as `$JPP_HOME/gatein/gatein.ear/portal.war/templates/skin/portal/webui/component/UIFooterPortlet/DefaultStylesheet.css` but processed for the RT orientation. The `-lt` suffix is optional.

Stylesheet authors can annotate their stylesheet to create content that depends on the orientation.

In This example we need to use the orientation to modify the float attribute that will make the horizontal tabs either float on left or on right:

■

Example 17.1. Example 1

```
float: left; /* orientation=lt */
float: right; /* orientation=rt */
font-weight: bold;
text-align: center;
white-space: nowrap;
```

The LT produced output will be:

```
float: left; /* orientation=lt */
font-weight: bold;
text-align: center;
white-space: nowrap;
```

The RT produced output will be:

```
float: right; /* orientation=rt */
font-weight: bold;
text-align: center;
white-space: nowrap;
```

In this example we need to modify the padding according to the orientation:

Example 17.2. Example 2

```
color: white;
line-height: 24px;
padding: 0px 5px 0px 0px; /* orientation=lt */
padding: 0px 0px 0px 5px; /* orientation=rt */
```

The LT produced output will be:

```
color: white;
line-height: 24px;
padding: 0px 5px 0px 0px; /* orientation=lt */
```

The RT produced output will be:

```
color: white;
line-height: 24px;
padding: 0px 0px 0px 5px; /* orientation=rt */
```

[Report a bug](#)

17.3. Images

Sometimes it is necessary to create an RT version of an image that will be used from a template or from a stylesheet. However symmetric images can be automatically generated, avoiding the necessity to create a mirrored version of an image and further maintenance costs.

The web resource filter uses the same naming pattern as the skin service. When an image ends with the -rt suffix the portal will attempt to locate the original image and create a mirror of it.

For instance: requesting the image

`$JPP_HOME/gatein/gatein.ear/eXoResources.war/skin/DefaultSkin/webui/component/UITabSystem/UITabs/background/NormalTabStyle-rt.gif` returns a mirror of the image

`$JPP_HOME/gatein/gatein.ear/eXoResources.war/skin/DefaultSkin/webui/component/UITabSystem/UITabs/background/NormalTabStyle.gif`.



Note

It is important to consider whether the image to be mirrored is symmetrical as this will impact its final appearance.

Here is an example combining stylesheet and images:

```
line-height: 24px;
background: url('background/NavigationTab.gif') no-repeat right top; /*
orientation=lt */
background: url('background/NavigationTab-rt.gif') no-repeat left top; /*
orientation=rt */
padding-right: 2px; /* orientation=lt */
padding-left: 2px; /* orientation=rt */
```

[Report a bug](#)

17.4. Client Side JavaScript

The `eXo.core.I18n` object provides the following parameters for orientation:

`getOrientation()`

Returns either the string lt or rt

`getDir()`

Returns either the string ltr or rtl

`isLT()`

Returns true for LT

`isRT()`

Returns true of RT

[Report a bug](#)

Chapter 18. XML Resources Bundles

Resource bundles are usually stored in property files. However, as property files are plain files, issues with the encoding of the file may arise. The XML resource bundle format has been developed to provide an alternative to property files.

- The XML format declares the encoding of the file. This avoids use of the **native2ASCII** program which can interfere with encoding.
- Property files generally use **ISO 8859 -1** character encoding which does not cover the full unicode charset. As a result, languages such as Arabic would not be natively supported.
- Tooling for XML files is better supported than the tooling for Java property files and thus the XML editor copes well with the file encoding.

[Report a bug](#)

18.1. XML format

The XML format is very simple and has been developed based on the *Do not Repeat Yourself* (DRY) principle. Usually resource bundle keys are hierarchically defined and we can leverage the hierarchic nature of the XML for that purpose. Here is an example of turning a property file into an XML resource bundle file:

```
UIAccountForm.tab.label.AccountInputSet = ...
UIAccountForm.tab.label.UIUserProfileInputSet = ...
UIAccountForm.label.Profile = ...
UIAccountForm.label.HomeInfo= ...
UIAccountForm.label.BusinessInfo= ...
UIAccountForm.label.password= ...
UIAccountForm.label.Confirmpassword= ...
UIAccountForm.label.email= ...
UIAccountForm.action.Reset= ...
```

```
<?xml version="1.0" encoding="UTF-8"?>
<bundle>
  <UIAccountForm>
    <tab>
      <label>
        <AccountInputSet>...</AccountInputSet>
        <UIUserProfileInputSet>...</UIUserProfileInputSet>
      </label>
    </tab>
    <label>
      <Profile>...</Profile>
      <HomeInfo>...</HomeInfo>
      <BusinessInfo>...</BusinessInfo>
      <password>...</password>
      <Confirmpassword>...</Confirmpassword>
      <email>...</email>
    </label>
    <action>
```



```
<Reset>...</Reset>
</action>
</UIAccountForm>
</bundle>
```

[Report a bug](#)

18.2. Portal Support

In order to be loaded by the portal at runtime (actually the resource bundle service), the name of the file must be the same as a property file and it must use the **.xml** suffix.

For example; for the Account Portlet to be displayed in Arabic, the resource bundle would be **AccountPortlet_ar.xml** rather than **AccountPortlet_ar.properties**.

[Report a bug](#)

Chapter 19. Navigation Controller

The navigation controller now provides non ambiguous URLs for portal managed resources as opposed to previous releases where different resources were available for a single URL, for example, navigation. These resources were dependent on private navigation such as groups and dashboard.

Navigation controller provides a flexible and configurable mapping by decoupling the **http** request from the portal request. Previously, both the requests were tightly coupled, for instance the URL for a site had to begin with **/public/{sitename}** or **/private/{sitename}**.

Navigation controller allows portal administrators to create user-friendly URL by configuring the **http** requests.

The **WebAppController** converts **http** request into a portal request by decoupling the http request to create a portal request.

The mapping engine performs two essential tasks:

- ✧ It creates a `Map<QualifiedName, String>` from an incoming **http** request.
- ✧ It renders a `Map<QualifiedName, String>` as an **http** URL.

The http request data can be encoded in the request path or as a query parameter. The controller allows the portal to route a request based on a set of parameters instead of the servlet request.

The controller configuration is declared in an XML file to allow easy configuration of the routing table. It is processed into an internal data structure that is used to perform resolution (routing or rendering).

The routing rules for controller configuration are located in **controller-configuration.xml**. The location of the configuration file is determined by the **controller.config** property.

The **WebAppController** loads and initializes the controller as shown:

```
<!-- conf/portal/controller.xml of portal.war -->
<component>
  <type>org.exoplatform.web.WebAppController</type>
  <init-params>
    <value-param>
      <name>controller.config</name>
      <value>${gatein.portal.controller.config}</value>
    </value-param>
  </init-params>
</component>
```

Based on the extension mechanism the extension project can define its own routing table.

The **controller.xml** can be modified and reloaded at runtime to test different configurations and provides insight into the routing engine (the **findRoutes** operation).

The **WebAppController** is annotated with **@Managed** annotations and is bound under the **view=portal, service=controller** JMX name and the **portalcontroller** REST name.

Attributes and Operations

configurationPath

Attribute (read only) which specifies the configuration path of the controller XML file.

Attribute (read only), which specifies the configuration path of the controller XML file.

loadConfiguration

Operation, which loads a new configuration file from a specified XML path.

reloadConfiguration

Operation, which reloads the configuration file.

findRoutes

Operation, which routes the request argument through the controller and returns a list of parameter map resolution.

The argument is a request URI such as `/g/:platform:administrators/administration/registry`. It returns a string representation (`List<Map>`) of the matched routes.

[Report a bug](#)

19.1. Controller Configuration (controller.xml)

Most of the controller configuration cares about defining rules (Routing table - contains routes object) that will drive the resolution. Routes are processed during the controller initialization to give a tree of nodes. For every node the following is true:

- ✦ It is related to its parent with a matching rule that can either be an **exact string matching** or a **regular expression matching**.
- ✦ It is associated with a set of parameters.

A parameter is defined by a qualified name. There are three types of parameters: Route, Path, and Request.

Route parameters define a fixed value associated with a qualified name.

- ✦ Routing: route parameters allow the controller to distinguish branches easily and route the request accordingly.
- ✦ Rendering: the system will select a route to render an URL if all route parameters are always matched.

```
<route path="/foo">
  <route-param qname="gtn:handler">
    <value>portal</value>
  </route-param>
</route>
```

This configuration matches the request path `/foo` to the map (`gtn:handler=portal`) and it renders the (`gtn:handler=portal`) map as the `/foo` URL. This example demonstrates two concepts, exact path matching (`/foo`) and route parameters (`gtn:handler`).

Path parameters allow to associate a portion of the request path with a parameter. Such parameter will match any non empty portion of text except the `/` character (that is the `[^/]+` regular expression) otherwise they can be associated with a regular expression for matching specific patterns. Path parameters are mandatory for matching since they are part of the request path, however it is allowed

to write regular expression matching an empty value.

- ✦ Routing: route is accepted if the regular expression is matched.
- ✦ Rendering: selection occurs when the regular expression matches the parameter.

Encoding

Path parameters may contain '/' character which is a reserved char for URI path. This case is specially handled by the navigation controller by using a special character to replace '/' literals. By default the character is the semi colon : and can be changed to other possible values (see controller XML schema for possible values) to give a greater amount of flexibility.

This encoding is applied only when the encoding performed for parameter having a mode set to the **default-form** value, for instance it does not happen for navigation node URI (for which / are encoded literally). The separator escape char can still be used but under it's percent escaped form, so by default a path parameter value containing : would be encoded as %3A and conversely the %3A value will be decoded as :.

```
<route path="/{gtn:path}">
</route>
```

No pattern defined, used the default one [^/]+

```
Routing and Rendering
Path "/foo"    <--> the map (gtn:path=foo)

Path "/foo:bar" <--> the map (gtn:path=foo/bar)
```

If the request path contains another "/" char it will not work, default encoding mode is: **default-form**. For example: "/foo/bar" --> not matched, return empty parameter map

However this could be solved with the following configuration:

```
<route path="/{gtn:path}">
  <path-param encoding="preserve-path" qname="gtn:path">
    <pattern>.*</pattern>
  </path-param>
</route>
```

1. The .* declaration allows to match any char sequence.
2. The preserve-path **encoding** tells the engine that the "/" chars should be handled by the path parameter itself as they have a special meaning for the router. Without this special encoding, "/" would be rendered as the ":" character and conversely the ":" character would be matched as the "/" character.

Request parameters are matched from the request parameters (GET or POST). The match can be optional as their representation in the request allows it.

- ✦ Routing
 - route is accepted when a required parameter is present and matched in the request.
 - route is accepted when an optional parameter is absent or matched in the request.
- ✦ Rendering:

- selection occurs for required parameters when the parameter is present and matched in the map.
- selection occurs for optional parameters when the parameter is absent or matched in the map.

```
<route path="/">
  <request-param name="path" qname="gtn:path"/>
</route>
```

Request parameters are declared by a **request-param** element and by default will match any value. A request like `"/?path=foo"` is mapped to the `(gtn:path=foo)` map. The **name** attribute of the **request-param** tag defines the request parameter value. This element accepts more configuration

- ✧ a **value** or a **pattern** child element to match a constant or a pattern
- ✧ a **control-mode** attribute with the value **optional** or **required** to indicate if matching is mandatory or not
- ✧ a **value-mapping** attribute with the possible values **canonical**, **never-empty**, **never-null** can be used to filter values after matching is done. For instance a parameter configured with **value-mapping="never-empty"** and matching the empty string value will not put the empty string in the map.

The order of route declaration is important as it influences how rules are matched. Sometimes the same request could be matched by several routes and the routing table is ambiguous.

```
<route path="/foo">
  <route-param qname="gtn:handler">
    <value>portal</value>
  </route-param>
</route>
<route path="{gtn:path}">
  <path-param encoding="preserve-path" qname="gtn:path">
    <pattern>.*</pattern>
  </path-param>
</route>
```

In that case, the request path `"/foo"` will always be matched by the first rule before the second rule. This can be misleading since the map `(gtn:path=foo)` would be rendered as `"/foo"` as well and would not be matched by the first rule. Such ambiguity can happen, it can be desirable or not.

Route nesting is possible and often desirable as it helps to

- ✧ factor common parameters in a common rule
- ✧ perform more efficient matching as the match of the common rule is done once for all the sub routes

```
<route path="/foo">
  <route-param qname="gtn:handler">
    <value>portal</value>
  </route-param>
  <route path="/bar">
    <route-param qname="gtn:path">
```

```

        <value>bar</value>
      </route-param>
    </route>
    <route path="/juu">
      <route-param qname="gtn:path">
        <value>juu</value>
      </route-param>
    </route>
  </route>

```

- » The request path `"/foo/bar"` is mapped to the `(gtn:handler=portal,gtn:path=bar)` map
- » The request path `"/foo/juu"` is mapped to the `(gtn:handler=portal,gtn:path=juu)` map
- » The request path `"/foo"` is not mapped as non leaf routes do not perform matches.

When an HTML error code is returned as a response to a URL request (for example, 404 Not Found), the portal redirects users to a default error page where the error code is displayed together with an explanatory message. To replace the default error page with a customized one, implement the configuration described in the procedure.

Procedure 19.1. Configuring redirection to custom error pages

1. Create the actual error pages and place them in the **`$JPP_HOME/gatein/gatein.ear/portal.war/`** directory.
2. For each error code requiring a custom error page, add the **`<error-page>`** element to the **`$JPP_HOME/gatein/gatein.ear/portal.war/WEB-INF/web.xml`** file. This element specifies what page is displayed when the particular error code is returned.

The sample code below ensures the **`my404.html`** page is displayed when the 404 error code is returned.

```

<error-page>
  <error-code>404</error-code>
  <location>/my404.html</location>
</error-page>

```

3. Specify the error page locations as static resources in the **`$JPP_HOME/standalone/configuration/gatein/controller.xml`** file. The code sample below demonstrates this configuration for the **`/my404.html`** path.

```

<route path="/my404.html">
  <route-param qname="gtn:handler">
    <value>staticResource</value>
  </route-param>
</route>

```

Without this configuration, the portal tries to resolve **`/my404.html`** as a name of a resource. This results in unwanted redirection to **`/portal/my404.html`**. It is therefore necessary to configure the error page locations as static resources.

4. When all the previous steps are performed, users are redirected to the specified custom error pages when the respective error codes are returned as a response to a URL request.

The portal defines a set of parameters in the routing table. For each client request, the mapping engine processes the request path and returns the defined parameters with their values as a `Map<QualifiedName, String>`.

gtn:handler

The `gtn:handler` name is one of the most important qualified names. It determines which handler will process the request immediately after the controller has determined the parameter map. The handler value is used to make a lookup in the handler map of the controller. A handler is a class that extends the **`WebRequestHandler`** class and implements the **`execute(ControllerContext)`** method. Several handlers are available by default:

- **portal**

Process aggregated portal requests.

- **upload / download**

process file upload and file download

- **legacy**

Handle legacy URL redirection.

- **default**

http redirection to the default portal of the container.

- **staticResource**

Serve static resources such as image, CSS or javascriptfiles in **portal.war**.

gtn:sitetype / gtn:sitename / gtn:path

The qualified names drive a request for the portal handler. They are used to determine which site to show and which path to resolve against a navigation. For instance the (`gtn:sitetype=portal,gtn:sitename=classic,gtn:path=home`) instruct the portal handler to show the home page of the classic portal site.

gtn:lang

Language in the URL for the portal handler. The language can be specified in URL so that the user can bookmark the URL in the particular locale or change the locale by modifying the URL address.

gtn:componentid / gtn:action / gtn:objectid

WebUI parameters used by the portal handler for managing WebUI component URLs for portal applications. These are not used for portlet applications.

[Report a bug](#)

19.1.1. Rendering

The controller is designed to render a `Map<QualifiedName, String>` as a **http** URL according to the controller's routing table. Additional components are required to integrate the controller into the WebUI Framework of the portal.

[Report a bug](#)

19.1.1.1. Portal URL

PortalURL has a similar role at the portal level: its main role is to abstract the creation of a URL for a resource managed by the portal.

```
public abstract class PortalURL<R, U extends PortalURL<U>>
{
    ...
}
```

The **PortalURL** declaration may seem a bit strange at first sight with two generic types **U** and **R** and the circular recursion of the **U** generic parameter, but it's because most of the time you will not use the **PortalURL** object but instead subclasses.

- ✧ The **R** generic type represents the type of the resource managed by the portal
- ✧ The **U** generic type is also described as **self bound generic type**. This design pattern allows a class to return subtypes of itself in the class declaring the generic type. Java Enums are based on this principle (`class Enum<E extends Enum<E>>`)

A portal URL has various methods but certainly the most important method is the **toString()** method that generates a URL representing that will target the resource associated with the URL. The remaining methods are getter and setter for mutating the URL configuration, those options will affect the URL representation when it is generated.

- ✧ resource: the mandatory resource associated with the URL
- ✧ locale: the optional locale used in the URL allowing the creation of bookmarkable URL containing a language
- ✧ confirm: the optional confirm message displayed by the portal in the context of the portal UI
- ✧ ajax: the optional Ajax option allowing an Ajax invocation of the URL

Obtaining a PortalURL

PortalURL objects are obtained from **RequestContext** instance such as the **PortalRequestContext** or the **PortletRequestContext**. Usually those are obtained thanks to **getCurrentInstance** method of the **RequestContext** class:

```
RequestContext ctx = RequestContext.getCurrentInstance();
```

PortalURL is created with the **createURL** method that has a resource type as its input parameter. A resource type is a constant and a type-safe object that allows to retrieve **PortalURL** subclasses:

```
RequestContext ctx = RequestContext.getCurrentInstance();
PortalURL<R, U> URL = ctx.createURL(type);
```

In reality you will use a concrete type constant and have instead more concrete code like:

```
RequestContext ctx = RequestContext.getCurrentInstance();
NodeURL URL = ctx.createURL(NodeURL.TYPE);
```




Note

The **NodeURL.TYPE** is actually declared as **new ResourceType<NavigationResource, NodeURL>()** that can be described as a **type literal** object emulated by a Java anonymous inner class. Such literals were introduced by Neil Gafter as Super Type Token and popularized by Google Guice as Type Literal. It's an interesting way to create a literal representing a kind of Java type.

[Report a bug](#)

19.1.1.2. Node URL

The class **NodeURL** is a subclass of the **PortalURL** specialized for navigation node resources:

```
public class NodeURL extends PortalURL<NavigationResource, NodeURL>
{
    ...
}
```

The good news is that the **NodeURL** does not carry any generic type of its super class, which means that a **NodeURL** is type safe and one does not have to worry about generic types.

Using a **NodeURL** is pretty straightforward:

```
NodeURL URL =
    RequestContext.getCurrentInstance().createUrl(NodeURL.TYPE);
URL.setResource(new NavigationResource("portal", "classic", "home"));
String s = URL.toString();
```

The **NodeURL** subclass contains specialized setter to make its usage even easier:

```
UserNode node = ...;
NodeURL URL =
    RequestContext.getCurrentInstance().createUrl(NodeURL.TYPE);
URL.setNode(node);
String s = URL.toString();
```

[Report a bug](#)

19.1.1.3. Component URL

The **ComponentURL** subclass is another specialization of **PortalURL** that allows the creation of WebUI components URLs. **ComponentURL** is commonly used to trigger WebUI events from client side:

```
<% def componentURL = uicomponent.event(...); /*or uicomponent.URL(...)
*/ %>
<a href=$componentURL>Click me</a>
```

Normally you should not have to deal with it as the WebUI framework has already an abstraction for managing URL known as **URLBuilder**. The **URLBuilder** implementation delegates URL creation to **ComponentURL** objects.

[Report a bug](#)

19.1.1.4. Portlet URL

Portlet URLs API implementation delegates to the portal **ComponentURL** (via the portlet container SPI). It is possible to control the language in the URL from a **PortletURL** object by setting a property named **gtn:lang**:

- when the property value is set to a value returned by **Locale#toString()** method for locale objects having a non null language value and a null variant value, the URL generated by the **PortletURL#toString()** method will contain the locale in the URL.
- when the property value is set to an empty string, the generated URL will not contain a language. If the incoming URL was carrying a language, this language will be erased.
- when the property value is not set, it will not affect the generated URL.

```
PortletURL URL = resp.createRenderURL();
URL.setProperty("gtn:lang", "fr");
writer.print("<a href='" + URL + "'>French</a>");
```

[Report a bug](#)

19.1.1.5. WebUI URL Builder

This internal API for creating URL works as before and delegates to the **PortletURL** API when the framework is executed in a portlet and to a **ComponentURL** API when the framework is executed in the portal context. The API has been modified to take in account the language in URL with two properties on the builder:

- locale: a locale for setting on the URL
- removeLocale: a boolean for removing the locale present on the URL

[Report a bug](#)

19.1.1.6. Groovy Templates

Within a Groovy template the mechanism is the same, however a splash of integration has been done to make creation of NodeURL simpler. A closure is bound under the **nodeurl** name and is available for invocation anytime. It will simply create a NodeURL object and return it:

```
UserNode node = ...;
NodeURL URL = nodeurl();
URL.setNode(node);
String s = URL.toString();
```

The closure **nodeurl** is bound to Groovy template in **WebuiBindingContext**

```
// Closure nodeurl()
put("nodeurl", new Closure(this)
{
    @Override
    public Object call(Object[] args)
    {
        return context.createURL(NodeURL.TYPE);
    }
});
```

[Report a bug](#)

Part IV. Gadget Development

Chapter 20. Gadgets in Portal

A gadget is a mini web application, embedded in a web page and running on an application server platform. These small applications help users perform various tasks. Red Hat JBoss Portal Platform supports gadgets, such as Todo, Calendar, Calculator, Weather Forecasts, and RSS Reader.

In the context of the portal, gadgets are defined by the Google OpenSocial specifications. The portal framework includes Apache Shindig 2.0, which supports version 0.9 and 1.0 of OpenSocial.

Within a portal, it is possible to embed any OpenSocial gadget in a page, or in a user's dashboard.

Gadgets can be added to the application registry, and links can be added to the mini-composer (see the *User Guide* for more information about this feature).

[Report a bug](#)

20.1. Sources to Develop Gadgets

OpenSocial gadgets are built using standard HTML and JavaScript. The container offers an API. API documentation is available at <http://opensocial-resources.googlecode.com/svn/spec/1.0/Core-Gadget.xml>

A gadget has limited knowledge of its context (the portal) so the gadget integration within the portal is limited, for example, visual integration.

Google Web Toolkit (GWT) applications can be used as gadgets. GWT simplifies the process of creating user friendly applications.



Important

GWT applications usage is not recommended as they are not part of the standard Red Hat JBoss Portal support program.

For more information on gadget configuration and usage see *Administration and Configuration guide*.

[Report a bug](#)

Chapter 21. Portlet Development Resources

[Report a bug](#)

21.1. JSR-168 and JSR-286 overview

The Java Community Process (**JCP**) uses Java Specification Requests (**JSRs**) to define proposed specifications and technologies designed for the Java platform.

Portlet Specifications aim at defining portlets that can be used by any [JSR-168 \(Portlet 1.0\)](#) or [JSR-286 \(Portlet 2.0\)](#) portlet container.

The Red Hat JBoss Portal includes a container that supports both versions.

This chapter gives a brief overview of the Portlet Specifications. Portlet developers are strongly encouraged to read the JSR-286 Portlet Specification, which is available at <http://www.jcp.org/en/jsr/detail?id=286>.

The portal offers complete JSR-286 compliance. Any compliant JSR-168 or JSR-286 portlet operates inside the portal as mandated by the respective specifications.

[Report a bug](#)

21.1.1. Portal Pages

A portal can be considered as a series of web pages with different *areas* within them. Those areas contain different *windows* and each *window* contains a *portlet*:

The diagram below visually represents this nesting:

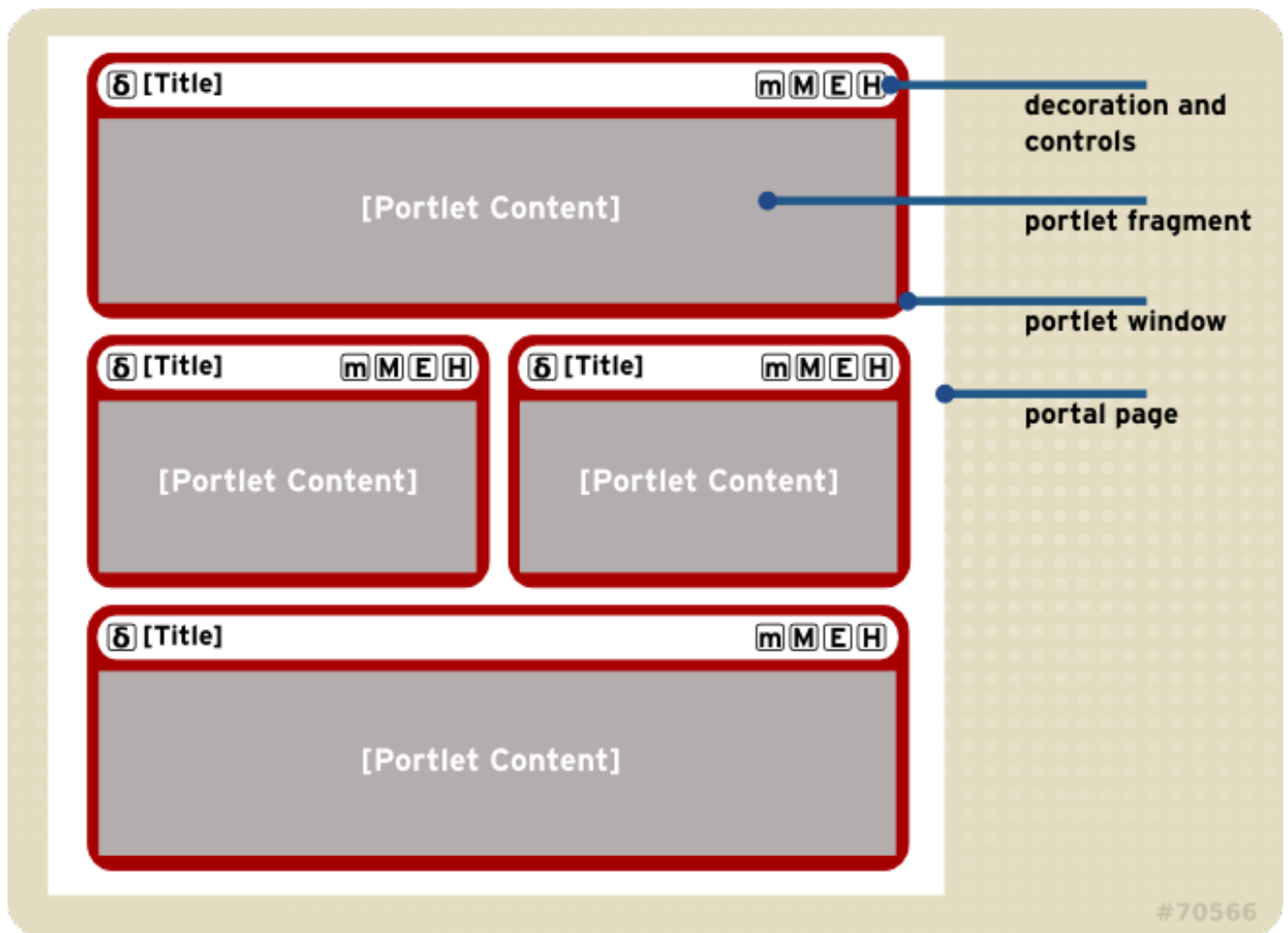


Figure 21.1. Portal Specification

[Report a bug](#)

21.1.1.2. Rendering Modes

A portlet can have different view modes. Three modes are defined by the JSR-286 specification:

View

Generates markup reflecting the current state of the portlet.

Edit

Allows a user to customize the behavior of the portlet.

Help

Provides information to the user as to how to use the portlet.

[Report a bug](#)

21.1.1.3. Window States

Window states are an indicator of how much page space a portlet consumes on any given page. The three states defined by the JSR-286 specification are:

Normal

A portlet shares this page with other portlets.

Minimized

A portlet may show very little information, or none at all.

Maximized

A portlet may be the only portlet displayed on this page.

[Report a bug](#)

21.2. Tutorials

The tutorials contained in this chapter are targeted toward portlet developers. It is also recommended that developers read and understand the [JSR-286 Portlet Specification](#).



Maven

This example is using Maven to compile and build the web archive. Maven versions can be downloaded from maven.apache.org

[Report a bug](#)

21.2.1. Deploying your first portlet

This section describes how to deploy a portlet in the Red Hat JBoss Portal (JBoss Portal).

An example portlet called **SimplestHelloWorld** is available in the `/jboss-jpp-VERSION-src/portal/examples/portlets/` directory of the JBoss Portal sources package.

[Report a bug](#)

21.2.1.1. Compiling

To compile and package the application:

1. Navigate to the application directory and execute:

```
mvn package
```

2. If the example is successfully packaged, the result will be available in: **gatein-simplest-helloworld-6.1.0.GA.war**.
3. Copy the package file into **\$JPP_HOME/standalone/deployments**.
4. Start the portal (if it is not already running).
5. Add the new portlet to the Application Registry.
6. Create a new portal page and add the portlet to it.

[Report a bug](#)

21.2.1.2. Package Structure

Like other Java EE applications, the Red Hat JBoss Portal portlets are packaged in **WAR** files. A typical portlet **WAR** file can include servlets, resource bundles, images, HTML, JavaServer Pages (JSP), and other static or dynamic files.

The following is an example of the directory structure of the **SimplestHelloWorld** portlet.

Example 21.1. Portlet Directory Structure Explanation

```
| -- SimplestHelloWorld-0.0.1.war
|   |-- WEB-INF
|       |-- classes
|           |-- org
|               |-- jboss
|                   |-- portal
|                       |-- portlet
|                           |-- samples
|                               |-- SimplestHelloWorldPortlet.class
|       |-- portlet.xml
|       |-- web.xml
```

Directory Structure Elements

SimplestHelloWorldPortlet.class

The compiled Java class, which implements *javax.portlet.Portlet* through *javax.portlet.GenericPortlet*.

portlet.xml

The mandatory descriptor file for portlets, which is used during deployment.

web.xml

The mandatory descriptor for web applications.

[Report a bug](#)

21.2.1.3. Portlet Class

Below is the Java source for an example portlet named **SimplestHelloWorldPortlet**.



Note

Portlets are responsible for generating markup fragments, as they are included on a page and are surrounded by other portlets. This means that a portlet outputting HTML must not output any markup that cannot be found in a **<body>** element.

Example 21.2. SimplestHelloWorldPortlet Explanation

```
package org.jboss.portal.portlet.samples;
```

```

import java.io.IOException;
import java.io.PrintWriter;
// Comment #1
import javax.portlet.GenericPortlet;
import javax.portlet.RenderRequest;
import javax.portlet.RenderResponse;

public class SimplestHelloWorldPortlet extends GenericPortlet
{
    // Comment #2
    public void doView(RenderRequest request, RenderResponse response)
    throws IOException
    {
        // Comment #3
        PrintWriter writer = response.getWriter();
        // Comment #4
        writer.write("Hello World !");
        // Comment #5
        writer.close();
    }
}

```

Comment #1

All portlets must implement the **javax.portlet.Portlet** interface. The **GenericPortlet** class provides a default implementation for the Portlet interface.

The **javax.portlet.GenericPortlet** class implements the **render** method to dispatch to abstract mode-specific methods. This makes it easier to support the standard portlet modes.

GenericPortlet also provides a default implementation for the **processAction**, **init** and **destroy** methods. It is recommended to extend **GenericPortlet** for most cases.

Comment #2

If only the **view** mode is required, then only the **doView** method needs to be implemented. The **GenericPortlet** render implementation calls our implementation when the **view** mode is requested.

Comment #3

To obtain a writer that can produce content, use the **response.getWriter()** method from the **RenderResponse** object.

Comment #4

Write the markup to display.

Comment #5

Closing the writer.

[Report a bug](#)

21.2.1.4. Application Descriptors

The portal requires certain descriptors to be included in a portlet **WAR** file. These descriptors are defined by the Java EE (**web.xml**) and Portlet Specification (**portlet.xml**).

Below is an example of the **SimplestHelloWorldPortlet/WEB-INF/portlet.xml** file. This file must adhere to its definition in the JSR-286 Portlet Specification. More than one portlet application may be defined in this file.

Example 21.3. The portlet.xml File Explanation

```
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-
app_2_0.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-
app_2_0.xsd http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
  version="2.0">
  <portlet>
    <!-- Comment #1 -->
    <portlet-name>SimplestHelloWorldPortlet</portlet-name>
    <!-- Comment #2 -->
    <portlet-class>
org.jboss.portal.portlet.samples.SimplestHelloWorldPortlet
    </portlet-class>
    <!-- Comment #3 -->
    <supports>
      <mime-type>text/html</mime-type>
    </supports>
    <!-- Comment #4 -->
    <portlet-info>
      <title>Simplest Hello World Portlet</title>
    </portlet-info>
  </portlet>
</portlet-app>
```

Notes

Comment #1

Define the portlet name. It does not have to be the class name.

Comment #2

The Fully Qualified Name (**FQN**) of your portlet class must be declared here.

Comment #3

The **<supports>** element declares all of the markup types that a portlet supports in the **render** method. This is accomplished via the **<mime-type>** element, which is required for every portlet.

The declared MIME types must match the capability of the portlet. It allows administrators to pair which modes and window states are supported for each markup type. This does not have to be declared as all portlets must support the **view** portlet mode.

Use the **<mime-type>** element to define which markup type the portlet supports. In the example above this is **text/html**. This section tells the portal to only output **HTML**.

Comment #4

When rendered, the portlet's title is displayed as the header in the portlet window, unless it is overridden programmatically. In the example above the title would be **Simplest Hello World Portlet**.

[Report a bug](#)

21.2.2. JavaServer Pages Portlet Example

This section discusses:

1. Adding more features to the previous example.
2. Using a JSP page to render the markup.
3. Using the portlet tag library to generate links to the portlet in different ways.
4. Using the other standard portlet modes.



Note

The example used in this section is available in the directory of the portal sources package.

Procedure 21.1. Compile JavaServer Pages Portlet

1. Obtain the Red Hat JBoss Portal sources package from the Customer Support portal.
2. Move to `/jboss-jpp-VERSION-src/portal/examples/portlets/jsphellouser`
3. Execute `mvn package`.
4. Copy `jsphellouser/target/gatein-jsp-hellouser-6.1.0.GA.war` to the `deploy` directory of JBoss Application Server.
5. Add the new portlet to the Application Registry.
6. Create a new portal page and add the portlet to it.

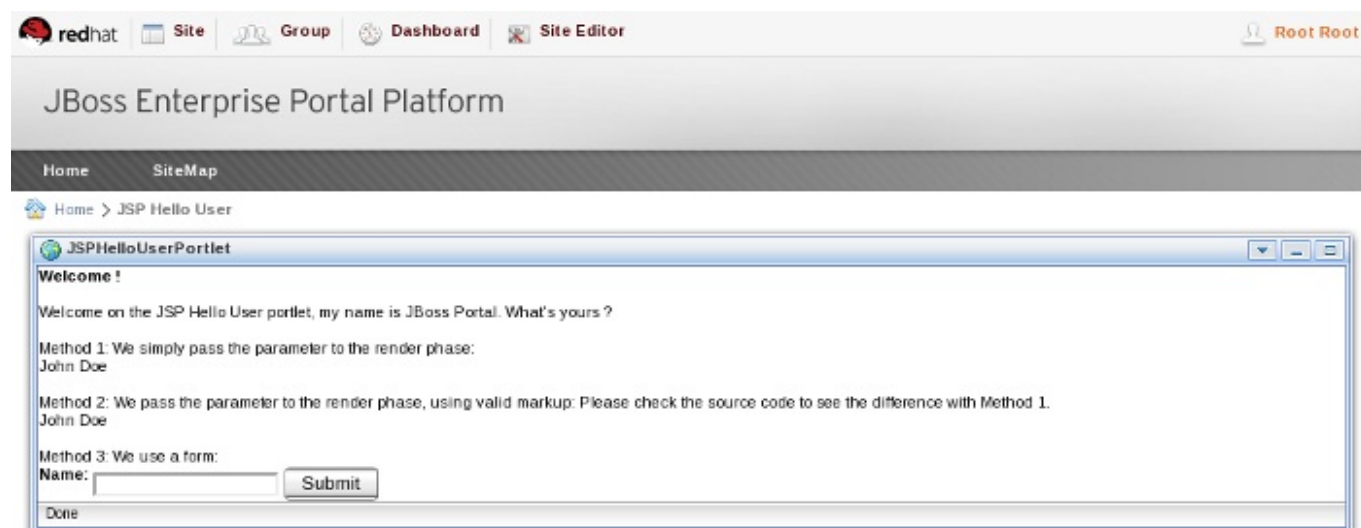


Figure 21.2. Create New Portal Page and Add Portlet

[Report a bug](#)

21.2.2.1. Package Structure

The package structure in this tutorial does not differ greatly from the previous example, with the exception of adding some JSP files which are detailed later.

The **JSPHelloUser** portlet contains the mandatory portlet application descriptors. The following is an example of the directory structure of the **JSPHelloUser** portlet:

```
gatein-jsp-hellouser->1.0.0-GA-SNAPSHOT.war
|-- META-INF
|   |-- MANIFEST.MF
|-- WEB-INF
|   |-- classes
|   |   |-- org
|   |   |   |-- jboss
|   |   |   |   |-- portal
|   |   |   |   |   |-- portlet
|   |   |   |   |   |   |-- samples
|   |   |   |   |   |   |   JSPHelloUserPortlet.class
|   |-- portlet.xml
|   |-- web.xml
|-- jsp
|   |-- edit.jsp
|   |-- hello.jsp
|   |-- help.jsp
|   |-- welcome.jsp
```

[Report a bug](#)

21.2.2.2. Portlet Class

The code below is from the **jsphellouser/src/main/java/org/jboss/portal/portlet/samples/JSPHelloUserPortlet.java** Java source. It is split in different pieces.

Example 21.4. JSPHelloUserPortlet Explanation

```
package org.jboss.portal.portlet.samples;

import java.io.IOException;

import javax.portlet.ActionRequest;
import javax.portlet.ActionResponse;
import javax.portlet.GenericPortlet;
import javax.portlet.PortletException;
import javax.portlet.PortletRequestDispatcher;
import javax.portlet.RenderRequest;
import javax.portlet.RenderResponse;
import javax.portlet.UnavailableException;

public class JSPHelloUserPortlet extends GenericPortlet
{
```

```

// Comment #1
public void doView(RenderRequest request, RenderResponse response)
    throws PortletException, IOException
{
// Comment #2
    String sYourName = (String) request.getParameter("yourname");
    if (sYourName != null)
    {
        request.setAttribute("yourname", sYourName);
// Comment #3
        PortletRequestDispatcher prd =
            getPortletContext().getRequestDispatcher("/jsp/hello.jsp");
// Comment #4
        prd.include(request, response);
    }
    else
    {
//Code split between lines. Direct copy will result in parse errors.
        PortletRequestDispatcher prd = getPortletContext().
            getRequestDispatcher("/jsp/welcome.jsp");
        prd.include(request, response);
    }
}
...

```

Comment #1

Override the *doView* method (as in the first tutorial).

Comment #2

This entry attempts to obtain the value of the render parameter named **yourname**. If defined it should redirect to the **hello.jsp** JSP page, otherwise to the **welcome.jsp** JSP page.

Comment #3

Get a request dispatcher on a file located within the web archive.

Comment #4

Perform the inclusion of the markup obtained from the JSP.

As well as the **VIEW** portlet mode, the specification defines two other modes; **EDIT** and **HELP**.

These modes need to be defined in the **portlet.xml** descriptor. This will enable the corresponding buttons on the portlet's window.

The generic portlet that is inherited dispatches the different views to the methods: **doView**, **doHelp** and **doEdit**.

```

...
protected void doHelp(RenderRequest rRequest, RenderResponse
rResponse) throws PortletException, IOException,
    UnavailableException
{
    rResponse.setContentType("text/html");
}

```

```

        PortletRequestDispatcher prd =
getPortletContext().getRequestDispatcher("/jsp/help.jsp");
        prd.include(rRequest, rResponse);
    }

    protected void doEdit(RenderRequest rRequest, RenderResponse
rResponse) throws PortletException, IOException,
        UnavailableException
    {
        rResponse.setContentType("text/html");
        PortletRequestDispatcher prd =
getPortletContext().getRequestDispatcher("/jsp/edit.jsp");
        prd.include(rRequest, rResponse);
    }
    ...

```

Portlet calls happen in one or two phases. One when the portlet is rendered and two when the portlet is actioned *then* rendered.

An action phase is a phase where a state changes. The render phase will have access to render parameters that will be passed each time the portlet is refreshed (with the exception of caching capabilities).

The code to be executed during an action has to be implemented in the *processAction* method of the portlet.

Example 21.5. processAction Explanation

```

...
// Comment #1
    public void processAction(ActionRequest aRequest,
ActionResponse aResponse) throws PortletException, IOException,
UnavailableException
    {
// Comment #2
        String sYourname = (String) aRequest.getParameter("yourname");
// Comment #3
        aResponse.setRenderParameter("yourname", sYourname);
    }
...

```

Comment #1

processAction is the method from **GenericPortlet** to override for the *action* phase.

Comment #2

Here the parameter is retrieved through an *action URL*.

Comment #3

The value of **yourname** is kept to make it available in the rendering phase. The previous line simply copies an action parameter to a render parameter for this example.

21.2.2.3. JSP files and the Portlet Tag Library

The **help.jsp** and **edit.jsp** files are very simple. Note that CSS styles are used as defined in the portlet specification. This ensures that the portlet will render well within the theme and across portal vendors.

```
<div class="portlet-section-header">Help mode</div>
<div class="portlet-section-body">This is the help mode, a convenient
place to give the user some help information.</div>
```

```
<div class="portlet-section-header">Edit mode</div>
<div class="portlet-section-body">This is the edit mode, a convenient
place to let the user change his portlet preferences.</div>
```

The landing page contains the links and form to call our portlet.

Example 21.6. Landing Page Explanation

```
<!-- Comment #1 -->
<%@ taglib uri="http://java.sun.com/portlet" prefix="portlet" %>

<div class="portlet-section-header">Welcome !</div>

<br/>

<div class="portlet-font">Welcome on the JSP Hello User portlet,
my name is GateIn Portal. What's yours ?</div>

<br/>

<div class="portlet-font">Method 1: We simply pass the parameter to the
render phase:<br/>
<!-- Comment #2 -->
<a href="<portlet:renderURL><portlet:param name="yourname" value="John
Doe"/>
        </portlet:renderURL>">John Doe</a></div>

<br/>

<div class="portlet-font">Method 2: We pass the parameter to the render
phase, using valid XML:
Please check the source code to see the difference with Method 1.
<!-- Comment #3 -->
<portlet:renderURL var="myRenderURL">
    <portlet:param name="yourname" value='John Doe' />
</portlet:renderURL>
<br/>
<!-- Comment #4 -->
<a href="<%= myRenderURL %>">John Doe</a></div>

<br/>

<div class="portlet-font">Method 3: We use a form:<br/>
<!-- Comment #5 -->
```



```

<portlet:actionURL var="myActionURL"/>
<!-- Comment #6 -->
<form action="<%= myActionURL %>" method="POST">
    <span class="portlet-form-field-label">Name:</span>
    <input class="portlet-form-input-field" type="text"
name="yourname"/>
    <input class="portlet-form-button" type="Submit"/>
</form>
</div>

```

Comment #1

The portlet taglib. This is required.

Comment #2

The first method showed here is the simplest one. **portlet:renderURL** will create a URL that calls the render phase of the current portlet and append the result at the place of the markup (within a tag). A parameter is also added directly to the URL.

Comment #3

In this method the **var** attribute is used. This avoids having one XML tag within another. Instead of printing the URL the **portlet:renderURL** tag will store the result in the referenced variable (**myRenderURL**).

Comment #4

The variable **myRenderURL** is used like any other JSP variable.

Comment #5

The third method mixes form submission and action request. Again, a temporary variable is used to put the created URL into.

Comment #6

The action URL is used in HTML form.

In the third method, the action phase is triggered first then the render phase is triggered, which outputs some content back to the web browser based on the available render parameters.

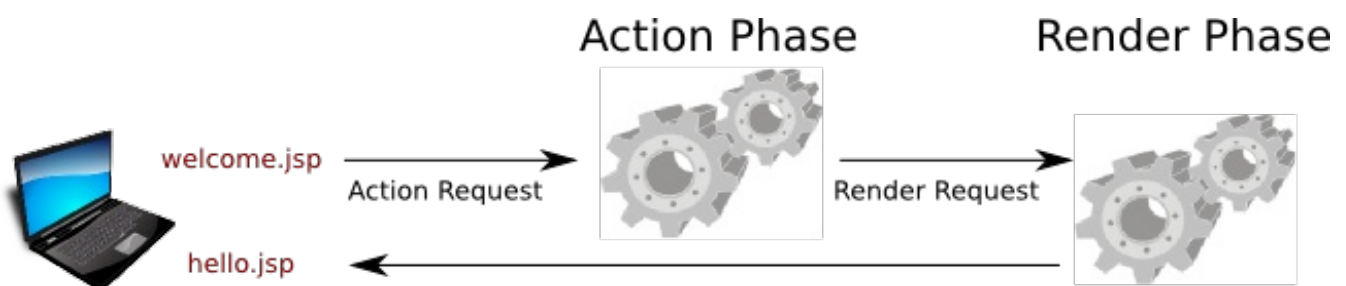


Figure 21.3. Render Phase Process

[Report a bug](#)

Chapter 22. Portlet Development

The Red Hat JBoss Portal (JBoss Portal) interface is fully customizable with applications called portlets. Application development can be done by using the plain Portlet specification JSR-286. It is also possible to use the JBoss Portlet Bridge to write applications with JavaServerFaces (JSF), RichFaces or Seam.

Learn how to set up your project in a robust and effective way by using the JBoss Portal Bill of Materials (BOM).

See Also:

- » [Section 21.1, “JSR-168 and JSR-286 overview”](#)
- » [Section 28.1, “JBoss Portlet Bridge”](#)
- » [Chapter 1, *Architectural Choices*](#)

[Report a bug](#)

22.1. Starting a Portlet Project

The Red Hat JBoss Portal (JBoss Portal) interface is fully customizable with applications called portlets. Application development can be done by using the plain Portlet specification JSR-286. It is also possible to use the JBoss Portlet Bridge to write applications with JavaServerFaces (JSF), RichFaces or Seam.

Learn how to set up your project in a robust and effective way by using the JBoss Portal Bill of Materials (BOM).

[Report a bug](#)

22.1.1. The Bill of Materials (BOM) Concept

To make managing dependencies easier, the BOM needed for developing typical portlet applications is available. The BOM is a Maven **pom.xml** file which specifies the versions of dependencies compatible with or provided by the Red Hat JBoss Portal (JBoss Portal).



Note

The JBoss Portal BOM is closely tied to JBoss EAP for maximum compatibility with the Red Hat JBoss Middleware stack.

[Report a bug](#)

22.1.2. Using the BOM

Below is the **pom.xml** file from the "Simplest Hello World" example portlet. This file contains all the necessary details.

In the **<dependencyManagement>** section, it declares **gatein-3.5-bom** as a **<dependency>** with **import** as the defined **<scope>**.

It indicates that the dependency will *de facto* be replaced with the dependencies in its ***dependencyManagement*** section.

As a result it is possible to declare the **`javax.portlet:portlet-api`** dependency in the **`<dependencies>`** section of the "Simplest Hello World" **`pom.xml`**, without specifying its **`<version>`**, **`<type>`** or **`<scope>`**. All of those parameters are managed by ***gatein-3.5-bom***.

Example 22.1. Simplest Hello World Portlet

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
<modelVersion>4.0.0</modelVersion>

<artifactId>simplest-hello-world-portlet</artifactId>
<groupId>org.gatein.portal.quickstarts</groupId>
<name>Simplest Hello World Portlet</name>
<version>3.5.0.Final</version>
<packaging>war</packaging>
<description>The very essence of every possible portlet.</description>
<url>http://www.gatein.org</url>
<licenses>
  <license>
    <name>Apache License, Version 2.0</name>
    <distribution>repo</distribution>
    <url>http://www.apache.org/licenses/LICENSE-2.0.html</url>
  </license>
</licenses>

<properties>
  <!-- GateIn Bill of Materials (BOM) version -->
  <org.jboss.bom.gatein-
bom.version>1.0.0.Final</org.jboss.bom.gatein-bom.version>

  <!-- Plugin versions and settings -->
  <jboss.as.plugin.version>7.1.1.Final</jboss.as.plugin.version>
  <!-- maven-compiler-plugin -->

  <maven.compiler.plugin.version>2.5.1</maven.compiler.plugin.version>
  <maven.compiler.target>1.6</maven.compiler.target>
  <maven.compiler.source>1.6</maven.compiler.source>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>

<dependencyManagement>
  <dependencies>
    <!--
      Define the version of JBoss Portal we build for. In its
      dependencyManagement,
      JBoss Portal Bill of Materials (BOM) specifies the
      versions, types and scopes
      of dependencies which are granted to be compatible with
      (or indeed in many cases
```

```

        provided by) JBoss Portal.
    -->
    <dependency>
        <groupId>org.jboss.bom</groupId>
        <artifactId>gatein-3.5-bom</artifactId>
        <version>${org.jboss.bom.gatein-bom.version}</version>
        <type>pom</type>
        <scope>import</scope>
    </dependency>
</dependencies>
</dependencyManagement>

<dependencies>
    <!--
        The versions, scopes and types of these dependencies are
        managed in gatein-*-bom.
        You need to name only groupId and artifactId here.
        Name only those artifacts you refer to in your code.
        Look at gatein-*-bom POM file for the complete list of
        available artifacts.
    -->
    <dependency>
        <groupId>javax.portlet</groupId>
        <artifactId>portlet-api</artifactId>
    </dependency>
</dependencies>

<build>
    <finalName>${project.artifactId}</finalName>
    <plugins>
        <plugin>
            <groupId>org.jboss.as.plugins</groupId>
            <artifactId>jboss-as-maven-plugin</artifactId>
            <version>${jboss.as.plugin.version}</version>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>${maven.compiler.plugin.version}</version>
            <configuration>
                <source>${maven.compiler.source}</source>
                <target>${maven.compiler.target}</target>
            </configuration>
        </plugin>
    </plugins>
</build>

</project>

```

Further steps, after you have set up the **pom.xml** file for your project, depend on the technology you have chosen for writing portlets.

[Report a bug](#)

22.2. Building and Deploying Portlets

1. Ensure that the instance of your JBoss Portal is running.
2. Open a command line and navigate to the root directory of your portlet project.
3. Type this command to build and deploy the archive: **mvn clean package jboss-as:deploy**

To deploy to other than default localhost:9999 JBoss instance, copy the following configuration just after `<version>${jboss.as.plugin.version}</version>` in the **pom.xml** file and adjust it to suit your needs. Note that `<username>` and `<password>` elements can be omitted sometimes, depending on your JBoss security settings.

```
<configuration>
  <hostname>127.0.0.1</hostname>
  <port>9999</port>
  <username>admin</username>
  <password>secret</password>
</configuration>
```

This will deploy **target/simplest-hello-world-portlet.war** to the running instance of the portal.

[Report a bug](#)

22.3. Standard Portlet Development (JSR-286)



Note

For an introduction to Portlet development, see [Section 21.1, “JSR-168 and JSR-286 overview”](#)

JSR-286 information is available from the *Java Community Process Program* located at: <http://jcp.org/en/jsr/detail?id=286>.

[Report a bug](#)

22.3.1. Java Configuration

After configuring the Maven **pom.xml** file, continue implementing a basic JSR-286 compatible portlet. An example of such a portlet is contained in the portal Quickstart "Simplest Hello World Portlet"

Example 22.2. SimplestHelloWorldPortlet.java

```
package org.jboss.portal.portlet.samples;

import java.io.IOException;
import java.io.PrintWriter;
```

```

import javax.portlet.GenericPortlet;
import javax.portlet.RenderRequest;
import javax.portlet.RenderResponse;

/**
 * The simplest possible Portlet.
 *
 * @author Peter Palaga
 */
public class SimplestHelloWorldPortlet extends GenericPortlet {
    /**
     * Serves the VIEW mode. Writes "Hello World !" to the response writer.
     *
     * @see
     * javax.portlet.GenericPortlet#doView(javax.portlet.RenderRequest,
     * javax.portlet.RenderResponse)
     */
    @Override
    public void doView(RenderRequest request, RenderResponse response)
        throws IOException {
        PrintWriter writer = response.getWriter();
        writer.write("Hello World !");
        writer.close();
    }
}

```

Two important things have occurred:

- ✱ **javax.portlet.GenericPortlet** was extended from the **javax.portlet:portlet-api** artifact.
- ✱ The **doView()** method has been overridden.

Edit and help portlet modes are not supported in this portlet. To add them, override the **doEdit()** and **doHelp** from **javax.portlet.GenericPortlet** and configure the **portlet.xml** file accordingly.

[Report a bug](#)

22.3.2. portlet.xml

The **portlet.xml** file for a plain JSR- 286 portlet is as follows:

Example 22.3. portlet.xml

```

<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-
app_2_0.xsd" version="2.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-
app_2_0.xsd http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd">
<portlet>
    <description>Simplest Hello World Portlet is the very essence of
every possible Portlet.</description>

```

```

<portlet-name>SimplestHelloWorldPortlet</portlet-name>
<display-name>Simplest Hello World Portlet</display-name>
<portlet-
class>org.jboss.portal.portlet.samples.SimplestHelloWorldPortlet</portl
et-class>
  <supports>
    <mime-type>text/html</mime-type>
    <portlet-mode>view</portlet-mode>
    <!-- You can uncomment the other modes when your portlet-class
supports them
    <portlet-mode>edit</portlet-mode>
    <portlet-mode>help</portlet-mode>
    -->
  </supports>
  <portlet-info>
    <title>Simplest Hello World Portlet</title>
  </portlet-info>
</portlet>
</portlet-app>

```

[Report a bug](#)

22.3.3. web.xml

There is no need to configure filters, servlets or mapping for a plain JSR- 286 portlet to work. However the **maven-war-plugin** artifact requires the **web.xml** by default. It can suffice to include a **web.xml** file which contains only the root element:

Example 22.4. web.xml

```

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  version="2.5">
  <!--
    There is no need to configure any filters, servlets, mapping &
co. for this demo to work
    but maven-war-plugin by default requires this file.
  -->
</web-app>

```

[Report a bug](#)

22.3.4. Building and Deploying Portlets

Build and deploy the portlet by following the included procedure.

Import the portlet and add the portlet to a page to test its functionality. See the *User Guide* for instructions.

Procedure 22.1. Build and Deploy the Portlet

1. Ensure the portal instance is running.
2. Open a command line and navigate to the root directory of the portlet project.
3. To deploy to portal instances not listening on **localhost: 9999** paste the following configuration immediately after `<version>${jboss.as.plugin.version}</version>` in the **pom.xml** file

```
<configuration>
  <hostname>127.0.0.1</hostname>
  <port>9999</port>
  <username>admin</username>
  <password>secret</password>
</configuration>
```

Adjust the configuration to suit individual requirements. `<username>` and `<password>` may be omitted depending on the portal's security settings.

This will deploy **target/simplest-hello-world-portlet.war** to the running instance of the portal.

4. Run **mvn clean package jboss-as:deploy**

[Report a bug](#)
.....

Chapter 23. Basic JSF Portlet Development

JSF stands for JavaServer Pages. The JSF version delivered by the built-in Portlet Bridge is 2.1.

[Report a bug](#)

23.1. Example Code

This section cites code from JSF2 Hello World Portlet from the [Quickstarts Collection](#).

[Report a bug](#)

23.1.1. pom.xml

There is only one noticeable difference in **pom.xml** against what we have shown as a general case in the [Starting a Portlet Project](#) section: we only need to add JSF-specific dependencies:

Example 23.1. pom.xml

```
<dependencies>
  <!--
    The versions, scopes and types of these dependencies are managed in
    gatein-*-bom.
    You need to name only groupId and artifactId here.
    Name only those artifacts you refer to in your code.
    Look at gatein-*-bom POM file for the complete list of available
    artifacts.
  -->
  <dependency>
    <groupId>org.jboss.spec.javafx.faces</groupId>
    <artifactId>jboss-jsf-api_2.1_spec</artifactId>
  </dependency>
  <dependency>
    <groupId>org.jboss.portletbridge</groupId>
    <artifactId>portletbridge-api</artifactId>
  </dependency>
</dependencies>
```

[Report a bug](#)

23.1.2. JSF Template Files

Example 23.2. Beginning of a JSF portlet template

This example shows the start of a typical JSF portlet template declaration. It is taken from **main.xhtml** file located in the **src/main/webapp/pages** directory.

```
<f:view xmlns="http://www.w3.org/1999/xhtml"
  xmlns:f="http://java.sun.com/jsf/core"
```

```

xmlns:h="http://java.sun.com/jsf/html">
  <!-- the f:view above prevents an erroneous <html> within <body>
  coming from portal. -->
  <!-- <h:head /> and <h:body> in the following are always needed
  otherwise JSF Facelets will not work -->
  <h:head />
  <h:body styleClass="jsf2HelloWorldPortlet">
    <h:outputStylesheet library="css" name="jsf2-hello-world-
    portlet.css" />
    <h2>JSF2 Hello World Portlet</h2>

```

Note that **<f:view>** as a root element of the portlet template above prevents an erroneous **<html>** within **<body>** coming from portal. Moreover, the **<h:head>** and **<h:body>** elements are always needed for JSF Facelets to work.

Example 23.3. A form in a JSF portlet template

This example demonstrates how to use several JSF elements, such as **<h:outputLabel>**, **<h:inputText>** and **<h:commandButton>** within **<h:form>**.

```

<h:form id="jsf2HelloWorldPortlet">
  <h:outputLabel value="#{msgs.Name}" for="nameInput"/>
  <h:inputText id="nameInput" value="#{helloBean.name}">
    <f:ajax render="output" event="keyup"/>
  </h:inputText>
  <br />

  <p>
    <h:panelGroup id="output">
      <strong>
        <h:outputFormat value="#{msgs.Hello_0}" rendered="#{not
        empty helloBean.name}">
          <f:param value="#{helloBean.name}" />
        </h:outputFormat>
      </strong>
    </h:panelGroup>
  </p>

  <h:commandButton id="reset" value="#{msgs.Reset}" actionListener="#
  {helloBean.reset}">
    <f:ajax render="@form" />
  </h:commandButton> - #{msgs.ResetComment}
  <br />
  <h:commandButton id="reload" value="#{msgs.Reload}" /> - #
  {msgs.ReloadComment}
  <br />

  <h:messages />
</h:form>

```

The complete source code of the above template is located in **src/main/webapp/pages/main.xhtml** of JSF2 Hello World Portlet project.

[Report a bug](#)

23.1.3. Java Beans

In the JSF template file shown above, we reference the **helloBean** bean. This bean is implemented in Java as follows:

Example 23.4. helloBean Implementation

```
/**
 * {@link HelloBean} is the JSF backing bean for the application,
 * holding the input data to be re-displayed.
 */
@ManagedBean(name = "helloBean")
@SessionScoped
public class HelloBean implements Serializable {

    private static final long serialVersionUID = -6239437588285327644L;

    /**
     * Stores the name which will be used to greet the application
     user.
     */
    private String name;

    /**
     * Initializes {@link #name} with the value {@code "World"}.
     */
    @PostConstruct
    public void postConstruct() {
        this.name = "World";
    }

    /**
     * Returns {@link #name}.
     *
     * @return {@link #name}
     */
    public String getName() {
        return name;
    }

    /**
     * Set {@link #name}.
     *
     * @param name
     */
    public void setName(String name) {
        this.name = name;
    }

    /**
     * Resets {@link #name} to the default value {@code "World"}.
     */
}
```

```

    * @param ae ignored
    */
    public void reset(ActionEvent ae) {
        this.name = "World";
    }
}

```

The **@ManagedBean** and **@SessionScoped** annotations allow for omitting of equivalent declarations in the **faces-config.xml** file.

[Report a bug](#)

23.1.4. portlet.xml

portlet.xml is the place where we tie the JSF templates with the portlet.

Example 23.5. portlet.xml

```

<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-
app_2_0.xsd" version="2.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-
app_2_0.xsd
    http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd">
    <portlet>
        <description>A simple JSF2 portlet.</description>
        <portlet-name>jsf2HelloWorldPortlet</portlet-name>
        <display-name>JSF2 Hello World Portlet</display-name>
        <portlet-
class>javax.portlet.faces.GenericFacesPortlet</portlet-class>
        <init-param>
            <name>javax.portlet.faces.defaultViewId.view</name>
            <value>/pages/main.xhtml</value>
        </init-param>
        <init-param>
            <name>javax.portlet.faces.defaultViewId.edit</name>
            <value>/pages/edit.xhtml</value>
        </init-param>
        <init-param>
            <name>javax.portlet.faces.defaultViewId.help</name>
            <value>/pages/help.xhtml</value>
        </init-param>
        <init-param>
            <name>javax.portlet.faces.preserveActionParams</name>
            <value>true</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>

```

```

    </supports>
    <portlet-info>
        <title>JSF2 Hello World Portlet</title>
    </portlet-info>
    <container-runtime-option>
        <name>org.gatein.pc.remotable</name>
        <value>true</value>
    </container-runtime-option>
</portlet>
</portlet-app>

```

Note that the `javax.portlet.faces.defaultViewId.*init-params` are used to bind the JSF templates with the respective portlet view modes. `javax.portlet.faces.GenericFacesPortlet` as a `<portlet-class>` will serve the purpose for most JSF portlets.

[Report a bug](#)

23.1.5. web.xml

JSF portlets require a few tweaks in the `web.xml` file:

Example 23.6. web.xml

```

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" version="2.5">
    <display-name>jsf2-hello-world-portlet</display-name>
    <context-param>
        <description>See
https://docs.jboss.org/author/display/PBR/Render+Policy</description>
        <param-name>javax.portlet.faces.RENDER_POLICY</param-name>
        <param-value>ALWAYS_DELEGATE</param-value>
    </context-param>

    <!-- The following params are documented here:
http://myfaces.apache.org/core21/myfaces-impl/webconfig.html -->
    <context-param>
        <param-name>javax.faces.FACELETS_VIEW_MAPPINGS</param-name>
        <param-value>*.xhtml</param-value>
    </context-param>
    <context-param>
        <param-name>facelets.DEVELOPMENT</param-name>
        <param-value>>false</param-value>
    </context-param>
    <context-param>
        <param-name>javax.faces.DEFAULT_SUFFIX</param-name>
        <param-value>.xhtml</param-value>
    </context-param>

```

```

<servlet>
  <servlet-name>Faces Servlet</servlet-name>
  <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>Faces Servlet</servlet-name>
  <url-pattern>*.faces</url-pattern>
</servlet-mapping>
</web-app>

```

[Report a bug](#)

23.1.6. Custom CSS

Portlet Bridge supports loading of CSS resources in the "JSF way". Just use the `<h:outputStylesheet library="css" name="jsf2-hello-world-portlet.css" />` as we do in the `main.xhtml` file above. The `jsf2-hello-world-portlet.css` file needs to be placed in `resources/css` folder of the web application.

Note that relative paths like `url('css/background/jsf-logo.png')` do not work when used in CSS loaded via `<h:outputStylesheet ...>`. Rather a JSF Expression Language expression `#{resource['/css/background/jsf-logo.png']}` needs to be used. This expression is dynamically evaluated to a proper public URL. The path `'/css/background/jsf-logo.png'` used in the expression is relative to `resources` folder of the web application. See also [Resource Serving](#).

Example 23.7. jsf2-hello-world-portlet.css

```

17. div.jsf2HelloWorldPortlet {
18.     padding: 10px;
19.     /* In the following we use a JSF Expression Language expression
20.        rather than plain relative path.
21.        Plain relative paths do not work in JSF portlets. The
22.        expression #{resource['...']} is
23.        dynamically evaluated to a proper public URL. The path
24.        '/css/background/jsf-logo.png'
25.        used in the expression is relative to resources folder of
26.        this web application.
27.        See
28.        https://docs.jboss.org/author/display/PBR/Resource+Serving */
29.     background: url("#{resource['/css/background/jsf-logo.png']}")
30.     no-repeat;
31.     background-position-x: 753px;
32.     background-position-y: 10px;
33. }
34. div.jsf2HelloWorldPortlet p {
35.     width: 713px;
36. }

```

[Report a bug](#)

23.1.7. Internationalization

Internationalization is supported via standard Java Resource bundles. In our example project the ***.property** files of **org.jboss.as.quickstarts.jsf.messages** resource bundle are stored under **src/main/resources/org/jboss/as/quickstarts/jsf**. The following settings in **faces-config.xml** are needed so that this bundle can be used in JSF templates.

Example 23.8. faces-config.xml

```
<faces-config xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-facesconfig_2_0.xsd"
version="2.0">
  <application>
    <!-- Declare the internationalization resources -->
    <resource-bundle>
      <!-- The resource bundle property files are in
src/main/resources/org/jboss/as/quickstarts/jsf -->
      <base-name>org.jboss.as.quickstarts.jsf.messages</base-name>
      <var>msgs</var>
    </resource-bundle>
    <locale-config>
      <default-locale>en</default-locale>
      <supported-locale>de</supported-locale>
    </locale-config>
  </application>
</faces-config>
```

Note that in the above **faces-config.xml**, we have made our bundle visible in JSF templates under the variable **msgs**. Its individual entries can then be accessed using the usual Expression Language dot notation: e.g. **#{msgs.Name}**.

[Report a bug](#)

23.2. Further Steps

After having done all the above, it is time to [build and deploy the portlet](#), [import it](#) and [add it to a page](#) so that you can test its functionality.

[Report a bug](#)

23.3. See also

➤ [Portlet Primer](#)

[Report a bug](#)

Chapter 24. JSF Portlet Development with RichFaces

As we have already noted, RichFaces (RF) is just a component library for JavaServer Faces (JSF). Therefore, everything said in [Section 23.1, “Example Code”](#) applies here too.

[Report a bug](#)

24.1. Example Code

This section cites code from JSF2+RF4 Hello World Portlet from the [Quickstarts Collection](#).

[Report a bug](#)

24.1.1. pom.xml

We need to add several RF-specific dependencies to the general JSF ones:

Example 24.1. pom.xml

```
<dependencies>
  <!--
    The versions, scopes and types of these dependencies are managed in
    gatein-*-bom.
    You need to name only groupId and artifactId here.
    Name only those artifacts you refer to in your code.
    Look at gatein-*-bom POM file for the complete list of available
    artifacts.
  -->
  <!-- General JSF dependencies -->
  <dependency>
    <groupId>org.jboss.spec.java.faces</groupId>
    <artifactId>jboss-jsf-api_2.1_spec</artifactId>
  </dependency>
  <dependency>
    <groupId>org.jboss.portletbridge</groupId>
    <artifactId>portletbridge-api</artifactId>
  </dependency>

  <!-- RF-specific dependencies -->
  <dependency>
    <groupId>org.jboss.portletbridge</groupId>
    <artifactId>portletbridge-extension-richfaces</artifactId>
  </dependency>
  <dependency>
    <groupId>org.richfaces.ui</groupId>
    <artifactId>richfaces-components-api</artifactId>
  </dependency>
  <dependency>
    <groupId>org.richfaces.ui</groupId>
    <artifactId>richfaces-components-ui</artifactId>
  </dependency>
  <dependency>
    <groupId>org.richfaces.core</groupId>
```



```

    <artifactId>richfaces-core-impl</artifactId>
  </dependency>
</dependencies>

```

[Report a bug](#)

24.1.2. JSF Template Files

We use `<rich: *>` components in the templates:

Example 24.2. Form with rich: components in main.xhtml

```

</p>
<h:form id="jsf2HelloWorldPortlet">
  <h:panelGrid columns="2">
    <h:outputLabel value="#{msgs.Greeting}" for="greeting"/>
    <rich:select id="greeting" value="#{helloBean.greeting}">
<f:selectItems value="#{helloBean.greetings}" />
<f:ajax render="output" event="selectitem"/>
    </rich:select>

    <h:outputLabel value="#{msgs.Name}" for="nameInput"/>
    <h:inputText id="nameInput" value="#{helloBean.name}">
<f:validateLength minimum="1" maximum="50" />
<f:ajax render="output" event="keyup"/>
    </h:inputText>
  </h:panelGrid>
  <p>
    <h:panelGroup id="output">
<strong><h:outputText value="#{helloBean.greeting} #
{helloBean.name}!" rendered="#{not empty helloBean.name}"/></strong>
    </h:panelGroup>
  </p>
  <p>
    <h:commandButton id="reset" value="#{msgs.Reset}"
actionListener="#{helloBean.reset}">
<f:ajax render="@form" />
    </h:commandButton> - #{msgs.ResetComment}
  </p>
  <p>
    <h:commandButton id="reload" value="#{msgs.Reload}" /> - #
{msgs.ReloadComment}
  </p>

```

The complete source code of the above template can be found in `src/main/webapp/pages/main.xhtml` of JSF2+RF4 Hello World Portlet quickstart.

[Report a bug](#)

24.1.3. Java Beans

The **HelloBean** presented in the [Section 23.1, “Example Code”](#) chapter was extended firstly to provide a list of greeting phrases selectable in the drop-down box on the `main.xhtml` page and secondly to be able to store the greeting phrase selected in the drop-down box.

Example 24.3. HelloBean.java

```
/**
 * Static list of greetings. Contains {@code "Hello"} and {@code "Hi"}.
 */
private static final List<SelectedItem> GREETINGS;

static {
    List<SelectedItem> l = new ArrayList<SelectedItem>(2);
    l.add(new SelectItem("Hello"));
    l.add(new SelectItem("Hi"));
    GREETINGS = Collections.unmodifiableList(l);
}

/**
 * Stores the greeting phrase which will be used to greet the
 * application user.
 */
private String greeting;
```

Example 24.4. HelloBean.java

```
/**
 * Returns {@link #greeting}.
 *
 * @return {@link #greeting}
 */
public String getGreeting() {
    return greeting;
}

/**
 * Set {@link #greeting}.
 *
 * @param greeting
 */
public void setGreeting(String greeting) {
    this.greeting = greeting;
}

/**
 * Returns {@link #GREETINGS}.
 *
 * @return {@link #GREETINGS}
 */
public List<SelectedItem> getGreetings() {
    return GREETINGS;
}
```

```

/**
 * Resets {@link #name} to the default value {@code "World"} and
 * {@link #greeting} with the default value {@code "Hello"}.
 *
 * @param ae ignored
 */
public void reset(ActionEvent ae) {
    this.name = "World";
    this.greeting = "Hello";
}
}

```

[Report a bug](#)

24.1.4. portlet.xml

There is no substantial change in **portlet.xml** against [Section 23.1, “Example Code”](#). Only **<description>**, **<portlet-name>**, **<display-name>** and **<title>** have been changed.

Example 24.5. portlet.xml

```

<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-
app_2_0.xsd" version="2.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-
app_2_0.xsd
    http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd">
    <portlet>
        <description>A simple portlet usinf JSF2 and RF4.</description>
        <portlet-name>jsf2Rf4HelloWorldPortlet</portlet-name>
        <display-name>JSF2+RF4 Hello World Portlet</display-name>
        <portlet-
class>javax.portlet.faces.GenericFacesPortlet</portlet-class>
        <init-param>
            <name>javax.portlet.faces.defaultViewId.view</name>
            <value>/pages/main.xhtml</value>
        </init-param>
        <init-param>
            <name>javax.portlet.faces.defaultViewId.edit</name>
            <value>/pages/edit.xhtml</value>
        </init-param>
        <init-param>
            <name>javax.portlet.faces.defaultViewId.help</name>
            <value>/pages/help.xhtml</value>
        </init-param>
        <init-param>
            <name>javax.portlet.faces.preserveActionParams</name>
            <value>true</value>
        </init-param>
        <expiration-cache>0</expiration-cache>
    </portlet>
</portlet-app>

```

```

    <supports>
      <mime-type>text/html</mime-type>
      <portlet-mode>VIEW</portlet-mode>
      <portlet-mode>EDIT</portlet-mode>
      <portlet-mode>HELP</portlet-mode>
    </supports>
    <portlet-info>
      <title>JSF2+RF4 Hello World Portlet</title>
    </portlet-info>
    <container-runtime-option>
      <name>org.gatein.pc.remotable</name>
      <value>true</value>
    </container-runtime-option>
  </portlet>
</portlet-app>

```

[Report a bug](#)

24.1.5. web.xml

We set a few more **init-params** in **web.xml** for RichFaces components to work:

Example 24.6. web.xml

```

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" version="2.5">
  <display-name>jsf2-rf4-hello-world-portlet</display-name>
  <context-param>
    <description>See
https://docs.jboss.org/author/display/PBR/Installing+Portlet+Bridge#Ins
tallingPortletBridge-
DisableautomaticinclusionofPortletBridge</description>
    <param-
name>org.gatein.portletbridge.WAR_BUNDLES_PORTLETBRIDGE</param-name>
    <param-value>true</param-value>
  </context-param>
  <context-param>
    <description>See
https://docs.jboss.org/author/display/PBR/Render+Policy</description>
    <param-name>javax.portlet.faces.RENDER_POLICY</param-name>
    <param-value>ALWAYS_DELEGATE</param-value>
  </context-param>

  <!-- The following params are documented here:
http://myfaces.apache.org/core21/myfaces-impl/webconfig.html -->
  <context-param>
    <param-name>javax.faces.FACELETS_VIEW_MAPPINGS</param-name>
    <param-value>*.xhtml</param-value>
  </context-param>
  <context-param>

```

```

        <param-name>facelets.DEVELOPMENT</param-name>
        <param-value>>false</param-value>
    </context-param>
    <context-param>
        <param-name>javax.faces.DEFAULT_SUFFIX</param-name>
        <param-value>.xhtml</param-value>
    </context-param>
    <context-param>
        <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
        <param-value>server</param-value>
    </context-param>
    <context-param>
        <param-name>javax.faces.PROJECT_STAGE</param-name>
        <!-- Change to Production to compress js files, etc. -->
        <param-value>Development</param-value>
    </context-param>

    <context-param>

<description>http://docs.jboss.org/richfaces/latest_4_X/Developer_Guide
/en-US/html/chap-Developer_Guide-
Skinning_and_theming.html</description>
        <param-name>org.richfaces.skin</param-name>
        <param-value>#{skinBean.skin}</param-value>
    </context-param>
    <context-param>
        <description>See
http://docs.jboss.org/richfaces/latest_4_X/Developer_Guide/en-
US/html/chap-Developer_Guide-Advanced_features.html</description>
        <param-name>org.richfaces.resourceOptimization.enabled</param-
name>
        <param-value>true</param-value>
    </context-param>

    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>*.faces</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>*.jsf</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>/faces/*</url-pattern>
    </servlet-mapping>

    <mime-mapping>
        <extension>xcss</extension>

```

```
<mime-type>text/css</mime-type>
</mime-mapping>

</web-app>
```

[Report a bug](#)

24.1.6. Custom CSS

Fully analogous with basic JSF portlets. See [Chapter 23, Basic JSF Portlet Development](#) for more information about basic JSF Portlet Development.

[Report a bug](#)

24.1.7. Internationalization

Fully analogous with basic JSF portlets. See [Chapter 23, Basic JSF Portlet Development](#) for more information about JSF portlet development.

[Report a bug](#)

24.2. Further Steps

After having done all the above, it is time to [build and deploy the portlet](#), [import it](#) and [add it to a page](#) so that you can test its functionality.

[Report a bug](#)

24.3. See also

✎ [Portlet Primer](#)

[Report a bug](#)

Chapter 25. Portlets with JSF and CDI

See the *CDI Portlet with JSF* Example in the Quickstarts Collection. Review the comments in the Quickstart files for necessary details.

[Report a bug](#)

Chapter 26. CDI Portlet Development

Red Hat JBoss Portal supports portlet development using Contexts and Dependency Injection (CDI) as specified by [JSR 299](#). You can perform CDI injection directly into `GenericPortlet` instances and Portlet Filters. There are two new CDI scopes to specifically support the portlet lifecycle.

[Report a bug](#)

26.1. GenericPortlet and Portlet Filter Injection

To activate CDI in portlet applications, the `WEB-INF/beans.xml` file must be present inside the WAR, as defined by the [JSR 299](#) specification. There is no additional configuration required to use CDI within a portlet or portlet filter.

Even if a portlet class is not managed by the CDI container, it can have CDI beans injected into it. These classes include any portlet class extending `GenericPortlet` or implementing the `Portlet` interface. Additionally, filters including `PortletFilter`, `RenderFilter`, `ActionFilter`, `EventFilter`, or `ResourceFilter` can use CDI injection. As with any kind of injection, use `@Inject` and specify the type of the bean to inject.

The point at which the injection is performed on the portlet class or filter, some contexts such as session will not be active. This will cause `ContextNotActiveException` to appear. Such a situation could arise if the injected bean is produced by a bean that is in session scope.



Note

The examples for CDI Generic Portlet are available in the Quickstarts Collection. See [Section C.1, “Quickstarts”](#). The quickstarts show a basic portlet, and portlet filter using CDI.

The CDI Portlet with JSF quickstart shows how to use CDI in combination with JSF.

[Report a bug](#)

26.2. Portlet CDI Scopes



Note

The example titled Portlet Using CDI Scopes project is available in the Quickstarts Collection, see [Section C.1, “Quickstarts”](#). The quickstart demonstrates the usage of CDI Scopes `@PortletLifecycleScoped` and `@PortletRedisplayScoped`.

To support the complex Portlet Lifecycle within CDI, `@PortletLifecycleScoped` and `@PortletRedisplayScoped` are available for use with either JSF portlets or portlets extending `GenericPortlet`. These scopes are present within the portal artifact, and are available to your portlet project by adding the Maven dependency described in the example API Maven Dependency.

Example 26.1. API Maven Dependency

There is no `<version>` necessary in `<dependency>` if the Bill of Materials (BOM) is imported as

described in [Section 22.1, “Starting a Portlet Project”](#).

```
<dependency>
  <groupId>org.gatein.api</groupId>
  <artifactId>gatein-api</artifactId>
  <scope>provided</scope>
</dependency>
```

[Report a bug](#)

26.2.1. @PortletLifecycleScoped

A bean annotated with **@PortletLifecycleScoped** is active for the full Portlet Lifecycle, from Action to Render.

If you set a value from within an **ActionRequest** or **EventRequest**, then retrieve that value from the bean within a **RenderRequest**, the value that was previously set is returned.

However, any further **RenderRequest** events that occur after the initial Portlet Lifecycle has completed will not contain any values that may have been set as part of an **ActionRequest** or **EventRequest**.

The **@PortletLifecycleScoped** is best suited to situations where it does not matter if the value is only present for the first render, such as wanting to display a message on the portlet to the user as a result of some action that was performed.



Note

A **ResourceRequest** is considered separate from the regular Portlet Lifecycle, therefore any changes to a CDI bean within a **ResourceRequest** will not be reflected in the instance of the bean used during an **ActionRequest**, **EventRequest** or **RenderRequest**, just as any changes to bean state within these requests will not be reflected in the instance that a **ResourceRequest** sees.

[Report a bug](#)

26.2.2. @PortletRedisplayScoped

A bean annotated with **@PortletRedisplayScoped** is active until a subsequent Portlet Lifecycle triggered by an **ActionRequest** or **EventRequest** is initiated.

This takes **@PortletLifecycleScoped** a step further by supporting the use case of needing to display the same data each time the portlet is refreshed without any new actions or events being triggered.

It does not matter how many **RenderRequest** are made, the data will be retained for redisplay until a new **ActionRequest** or **EventRequest** is triggered, causing the beans to be re-initialized to their default state.



Important

Because the **@PortletRedisplayScoped** bean is passivation capable, it must implement **Serializable**.

A **ResourceRequest** has any **@PortletRedisplayScoped** annotated bean initialized to its default state, however any bean that has its methods called will overwrite any existing instances of that bean upon completion of the **ResourceRequest**. This enables a subsequent **RenderRequest** to reflect the values that were set as part of a **ResourceRequest**. The typical use case for this scenario is Ajax calls from the browser.



Important

As **@PortletRedisplayScoped** beans have a CDI client proxy injected into the beans instead of the actual instance, it's important to remember that calling a **get** method on a bean during a **ResourceRequest** will result in its initialized state overwriting whatever is present in the instance that will be used for a **RenderRequest**.

Therefore it is recommended to only call beans methods during **RenderRequest** to avoid unexpected data changes on a portlet refresh.

[Report a bug](#)

Chapter 27. Portlet Filter

The Java Portlet Specification introduces `PortletFilter` as a standard approach to extend the behaviors of portlet objects. For example, a filter can transform the content of portlet requests and portlet responses.

According to the Portlet Specification, there are normally three steps in setting up a portlet filter:

1. Implement a `PortletFilter` object.
2. Define the filter in portlet application deployment descriptor.
3. Define the filter mapping in portlet definitions.

While the first two steps are quite straightforward, the third requires developers or administrators to replicate the filter mapping in many portlet definitions. This can be tedious and opens the potential for input errors. The global portlet feature is designed to mitigate these concerns.

Example 27.1. Contents of `portlet.xml`

Global portlet metadata is declared in the `portlet.xml` file and conforms with the Portlet 2.0 XSD.

```
<portlet-app version="1.0"
xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-
app_2_0.xsd http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd">
</portlet-app>
```

The path to the global `portlet.xml` is the value of `gatein.portlet.config` in the `configuration.properties.xml` file. By default, the file path is `$JPP_HOME/standalone/configuration/gatein/portlet.xml`

[Report a bug](#)

27.1. Global Metadata Elements

27.1.1. Global Metadata Elements

The global `$JPP_HOME/standalone/configuration/gatein/portlet.xml` file conforms, with some restrictions, to the portlet deployment descriptor schema defined in the Portlet Specification. In this file, the following elements are supported:

1. Portlet Filter
2. Portlet Mode
3. Window State

[Report a bug](#)

27.1.2. Configuring a Portlet Filter

Portlet filter mappings are declared in the global **portlet.xml** file and applied across portlet applications.

Example 27.2. Configuring a filter

The filter **ApplicationMonitoringFilter** is involved in request handling on any deployed portlet.

```
<filter>
<filter-
name>org.exoplatform.portal.application.ApplicationMonitoringFilter</f
ilter-name>
<filter-
class>org.exoplatform.portal.application.ApplicationMonitoringFilter</f
ilter-class>
  <life-cycle>ACTION_PHASE</life-cycle>
  <life-cycle>RENDER_PHASE</life-cycle>
  <life-cycle>EVENT_PHASE</life-cycle>
  <life-cycle>RESOURCE_PHASE</life-cycle>
</filter>
```

Application Monitoring Filter supports four life-cycle phases:

1. ACTION_PHASE
2. EVENT_PHASE
3. RENDER_PHASE
4. RESOURCE_PHASE

The Application Monitoring Filter records statistic information about deployed portlets. The filter alternates the actual monitoring mechanism in WebUI Framework.

[Report a bug](#)

Chapter 28. Portlet Bridge

A portlet bridge is a mediator that allows non-native application frameworks to run in a portal environment, independent to the underlying portlet API, or portlet concept. This functionality provides a developer flexibility to continue writing applications in a preferred language, and allows a controlled transition to new technologies.

[Report a bug](#)

28.1. JBoss Portlet Bridge

The JBoss Portlet Bridge is an implementation of the JSR-329 specification. It supports the JSF 2.0 runtime within a JSR 168 or 286 portlet. Version 3.x of the bridge is compatible with JSF 2.0 and RichFaces 4.x.

JBoss Portlet Bridge allows JSF applications to run using supported JBoss frameworks such as **RichFaces**. Seam 3.2 support is not yet available for JSF2, however support may be included in a future release.

The bridge is used to execute **Faces** requests on behalf of the portlet. During each request, the **Faces** environment is setup and handled by the bridge.

Part of this implementation acts as a **Faces** controller, much like the FacesServlet does in the direct client request environment.

The other part of this implementation is provided by implementing a variety of (standard) **Faces** extensions.

[Report a bug](#)

28.2. Portlet application

A portlet application is defined as a single web archive (WAR).

All portlets that are part of the same WAR are considered to form part of the same portlet application.

[Report a bug](#)

28.3. Extensions

Portlet extensions sit atop the portlet bridge framework. They extend the functionality of other JBoss portlet applications, and are critical in JSF portlet development.

[Report a bug](#)

28.4. Examples

Portlet Bridge provides example JSF2 portlets, which provide a good starting point for including the framework in portlets, as well as demonstrating the functionality available in Portlet Bridge.

**Important**

The provided examples are not officially supported by Red Hat.

JSF2 Portlet

<https://github.com/jbossportletbridge/jbossportletbridge/tree/master/examples/jsf2portlet>

This example provides a basic JSF2 portlet demonstrating Ajax functionality. This example provides a solid foundation for basic JSF2 functionality.

RichFaces 4 Simple

<https://github.com/jbossportletbridge/jbossportletbridge/tree/master/examples/richfaces-simple>

This example demonstrates a simple RichFaces 4 form with ajax submission and an extended data table showing the submitted data. This example provides a solid foundation for basic RichFaces 4 functionality.

RichFaces 4 Showcase

This example demonstrates a RichFaces 4 application, with portlet-specific changes. This example is worth deploying for advanced RichFaces 4 functionality, and shows the full spectrum of what can be achieved with RichFaces 4 and Portlet Bridge

[Report a bug](#)

28.5. Render Policy Parameters

Different JSF View Declaration Languages require different behavior from the JBoss Portlet Bridge when it comes to rendering views. Instructing the Bridge on which policy to use is done using the `javax.portlet.faces.RENDER_POLICY<context-param>` directive in **web.xml**.

RenderPolicy Options**ALWAYS_DELEGATE**

Indicates the bridge should not render the view itself, but rather always delegate the rendering.

NEVER_DELEGATE

Indicates the bridge should always render the view itself and never delegate.

DEFAULT

Directs the bridge to first delegate the render. If an exception is thrown, the bridge renders the view based on its own logic. If the configuration parameter is not present or has an invalid value the bridge renders using default behavior as it would if DEFAULT was set.

[Report a bug](#)

28.6. Facelets Configuration

The following **web.xml** setting is only for **Facelets** based applications.

Example 28.1. Facelets web.xml Configuration

```
<context-param>
<param-name>javax.portlet.faces.RENDER_POLICY</param-name>
<param-value>ALWAYS_DELEGATE</param-value>
</context-param>
```

[Report a bug](#)

28.7. JSP-only Configuration

The following **web.xml** setting is only for JSP based applications. Download the demonstration application [here](#).

Example 28.2. web.xml

```
<context-param>
<param-name>javax.portlet.faces.RENDER_POLICY</param-name>
<param-value>NEVER_DELEGATE</param-value>
</context-param>
```

[Report a bug](#)

28.8. RichFaces Local and Remote Portlet Support

Table 28.1. RichFaces Feature Support Exceptions

| RichFaces Component | Local Portlet | Remote Portlet using WSRP | Additional Comments |
|-------------------------|---------------|---------------------------|---|
| rich:editor | Yes | Yes | Some issues with editor icons over WSRP may be encountered when the Consumer and Producer are on different servers because the URLs are relative and are generated through JavaScript |
| rich:focus | Yes | Untested | Available since RichFaces 4.3.0.M3 |
| rich:placeholder | Yes | Untested | Available since RichFaces 4.3.0.M3 |

| RichFaces Component | Local Portlet | Remote Portlet using WSRP | Additional Comments |
|---------------------|---------------|---------------------------|---|
| a4j:push | Yes | No | Not supported for WSRP in Red Hat JBoss Portal Platform 6.2 |

Table 28.2. RichFaces Feature Status

| RichFaces Component | Supported as Local Portlet | Supported as Remote Portlet using WSRP |
|--|----------------------------|--|
| a4j:actionListener | Yes | Yes |
| a4j:ajax | Yes | Yes |
| a4j:attachQueue | Yes | Yes |
| a4j:commandButton | Yes | Yes |
| a4j:commandLink | Yes | Yes |
| a4j:jsFunction | Yes | Yes |
| a4j:log | Yes | Yes |
| a4j:mediaOutput | Yes | Yes |
| a4j:outputPanel | Yes | Yes |
| a4j:param | Yes | Yes |
| a4j:poll | Yes | Yes |
| a4j:queue | Yes | Yes |
| a4j:region | Yes | Yes |
| a4j:repeat | Yes | Yes |
| a4j:status | Yes | Yes |
| rich:accordion | Yes | Yes |
| rich:accordionItem | Yes | Yes |
| rich:autoComplete | Yes | Yes |
| rich:calendar | Yes | Yes |
| rich:collapsiblePanel | Yes | Yes |
| rich:collapsibleSubTable | Yes | Yes |
| rich:collapsibleSubTableToggler | Yes | Yes |
| rich:column | Yes | Yes |
| rich:columnGroup | Yes | Yes |
| rich:componentControl | Yes | Yes |
| rich:contextMenu | Yes | Yes |
| rich:dataGrid | Yes | Yes |
| rich:dataTable | Yes | Yes |
| rich:dragIndicator | Yes | Yes |
| rich:dragSource | Yes | Yes |
| rich:dropDownMenu | Yes | Yes |
| rich:dataScroller | Yes | Yes |
| rich:dropTarget | Yes | Yes |
| rich:editor | Yes | Yes |
| rich:extendedDataTable | Yes | Yes |
| rich:fileUpload | Yes | Yes |
| rich:focus | Yes | Untested |

| RichFaces Component | Supported as Local Portlet | Supported as Remote Portlet using WSRP |
|--|----------------------------|--|
| rich:graphValidator | Yes | Yes |
| rich:hashParam | Yes | Yes |
| rich:hotKey | Yes | Yes |
| rich:inplaceInput | Yes | Yes |
| rich:inplaceSelect | Yes | Yes |
| rich:inputNumberSlider | Yes | Yes |
| rich:inputNumberSpinner | Yes | Yes |
| rich:jQuery | Yes | Yes |
| rich:list | Yes | Yes |
| rich:menuGroup | Yes | Yes |
| rich:menuItem | Yes | Yes |
| rich:menuSeparator | Yes | Yes |
| rich:message | Yes | Yes |
| rich:messages | Yes | Yes |
| rich:notify | Yes | Yes |
| rich:notifyMessage | Yes | No |
| rich:notifyMessages | Yes | Yes |
| rich:notifyStack | Yes | Yes |
| rich:orderingList | Yes | Yes |
| rich:panel | Yes | Yes |
| rich:panelMenu | Yes | Yes |
| rich:panelMenuGroup | Yes | Yes |
| rich:panelMenuItem | Yes | Yes |
| rich:inputNumberSlider | Yes | Yes |
| rich:pickList | Yes | Yes |
| rich:placeholder | Yes | Untested |
| rich:popupPanel | Yes | Yes |
| rich:progressBar | Yes | Yes |
| rich:select | Yes | Yes |
| rich:tab | Yes | Yes |
| rich:tabPanel | Yes | Yes |
| rich:toggleControl | Yes | Yes |
| rich:togglePanel | Yes | Yes |
| rich:togglePanelItem | Yes | Yes |
| rich:toolbar | Yes | Yes |
| rich:toolbarGroup | Yes | Yes |
| rich:tooltip | Yes | Yes |
| rich:tree | Yes | Yes |
| rich:treeModelAdaptor | Yes | Yes |
| rich:treeModelRecursive Adaptor | Yes | Yes |
| rich:treeNode | Yes | Yes |
| rich:validator | Yes | Yes |

[Report a bug](#)

28.9. Sending and Receiving Events

The bridge considers a portlet event a *model event*, which means the event is targeted to the application's data model, not its view.

Because JSF events primarily concern its view, the bridge processes the portlet events manually. Provisions are made to ensure that any model changes resulting from processing the event are updated in the view.

Because event payloads are arbitrarily complex however, the manual data processing is left primarily to the portlet application to support.

[Report a bug](#)

28.10. Sending Events

The following configuration uses information relating to a generic Booking Event portlet. The concepts contained within each example can be applied to other event sending scenarios.

Procedure 28.1. Sending Events

1. Define the `autoDispatchEvents` `<init-param>` in **portlet.xml** to specify the `GenericFacesPortlet` should override event handling and dispatch all events to the bridge.



Important

If the application is written entirely in JSF (as opposed to a mix of view technologies), this `<init-param>` must be set to **true**.

```
<init-param>
  <name>javax.portlet.faces.autoDispatchEvents</name>
  <value>true</value>
</init-param>
```

2. Define the `<supported-publishing-event>` in **portlet.xml** to enable the portlet to publish an event to the portal.

```
<supported-publishing-event>
  <qname
xmlns:jbp="urn:jboss:portal:samples:event">jbp:BookingEvent</qname>
</supported-publishing-event>
```

3. Define the `<event-definition>` directive in **portlet.xml** to specify how the event namespace (`<qname>`) is linked to an actual type within the application.

```
<event-definition>
  <qname
xmlns:jbp="urn:jboss:portal:samples:event">jbp:BookingEvent</qname>
  <value-type>org.jboss.example.booking.BookingEvent</value-type>
</event-definition>
```

Example 28.3. Event type example

To publish an event, define an event type that represents the actual object that will be attached to the event.

The type requires the `@XmlRootElement` annotation to allow the event to be serialized into a JAXB object for publishing.

```
@XmlRootElement
public class BookingEvent implements Serializable {

    private String id;
    public static final QName QNAME = new
QName("urn:jboss:portal:samples:event", "BookingEvent");

    public BookingEvent(String id) {
        this.id = id;
    }

    public String getId() {
        return id;
    }
}
```

Example 28.4. Dispatch event example

To dispatch an event to other portlets within the portal, one approach is to use a bean method triggered by an action.

```
Object response =
FacesContext.getCurrentInstance().getExternalContext().getResponse();
if (response instanceof StateAwareResponse) {
    String id = "an id";
    StateAwareResponse stateResponse = (StateAwareResponse) response;
    stateResponse.setEvent(BookingEvent.QNAME, new BookingEvent(id));
}
```

[Report a bug](#)

28.11. Receiving Events

The following configuration uses information relating to a generic Booking Event portlet. The concepts contained within each example can be applied to other event receiving scenarios.

Procedure 28.2.

1. Define the `bridgeEventHandler` `<init-param>` directive in **portlet.xml** to specify the portlet can receive an event from Portlet Bridge.

```
<init-param>
    <name>javax.portlet.faces.bridgeEventHandler</name>
    <value>org.jboss.example.booking.BookingEventHandler</value>
</init-param>
```

2. Define the `<supported-processing-event>` directive in `portal.xml` to specify the portlet can also receive an event from the portal.

```
<supported-processing-event>
  <qname
xmlns:jbp="urn:jboss:portal:samples:event">jbp:BookingEvent</qname>
</supported-processing-event>
```

3. Define the `<event-definition>` directive in the portlet that will be receiving the event.

```
<event-definition>
  <qname
xmlns:jbp="urn:jboss:portal:samples:event">jbp:BookingEvent</qname>
  <value-type>org.jboss.example.booking.BookingEvent</value-type>
</event-definition>
```

To process a received event within a portlet, specify in the portlet code that the event handler implements the `BridgeEventHandler`.

```
public class BookingEventHandler implements BridgeEventHandler {
    public EventNavigationResult handleEvent(FacesContext context, Event
event) {
        // Process event payload as appropriate
    }
}
```

[Report a bug](#)

28.12. Public Render Parameters

Public Render Parameters (or PRPs) are one of the most powerful and simple Portlet 2.0 features. Several portlets (JSF or otherwise) can share the same PRPs. This feature can be used to present a cohesive UI to the user across all portlets on the page. An example would be using an employee ID to display relative data.

The bridge automates public render parameter processing.

A public render parameter can be mapped to an object's accessor (**get/set** method) designed to handle a String representation of the value through a **Faces ValueExpression**.

Unlike Events, PRPs can only be used to set/get a string value, and not an object.

When a new public render parameter value is received in a request, the bridge sets the value by calling the **ValueExpression's** `setValue()`.

The bridge maps a render parameter to a backing bean using settings in the `faces-config.xml` and `portlet.xml` files.

At the end of a request, if the current value of any mapped public render parameter does not match the current incoming value, the bridge sets the new value in an outgoing public render parameter (if feasible in the given phase).

[Report a bug](#)

28.13. Saving Bridge Request Scope after Render complete

By default, the Bridge Request Scope is not retained after completion of a Render Request. To save Bridge Request Scope after Render complete will enable developers to focus on Ajax requests and interactions as opposed to full page refreshes.

To return to the behavior of previous Portlet Bridge versions, you need to set the following initialization parameter in portlet.xml for a specific portlet:

```
<init-param>

<name>org.jboss.portletbridge.BRIDGE_SCOPE_PRESERVED_POST_RENDER</name>
  <value>true</value>
</init-param>
```

You can also set it for all portlets present within an archive by setting the following context parameter in web.xml:

```
<context-param>
  <param-
name>org.jboss.portletbridge.BRIDGE_SCOPE_PRESERVED_POST_RENDER</param-
name>
    <param-value>true</param-value>
</context-param>
```

[Report a bug](#)

28.14. PRP portlet configuration

The following configuration uses information relating to a generic Booking Event portlet. The concepts contained within each example can be applied to other PRP portlet configuration.

PRPs must be configured in both the portlet that sends (sets) the PRP, and the portlet that retrieves (gets) the PRP.

```
<portlet>
<!-- Unnecessary configuration information removed for clarity -->
  <supported-public-render-parameter>hotelName</supported-public-render-
parameter>
</portlet>

<public-render-parameter>
  <identifier>hotelName</identifier>
  <qname xmlns:j="http://jboss.org/params">j:hotelName</qname>
</public-render-parameter>
```

In the portlet that retrieves the PRP, a PRP handler must be specified as an `<init-param>` directive.

```
<init-param>

<name>javax.portlet.faces.bridgePublicRenderParameterHandler</name>
  <value>org.jboss.example.booking.BookingPRPHandler</value>
</init-param>
```

[Report a bug](#)

28.15. Application configuration

The following configuration uses information relating to a generic Booking Event portlet. The concepts contained within each example can be applied to other PRP portlet configuration.

Example 28.5. Set Parameter

To set the PRP, create a Bean method for the portlet setting the parameter that is triggered from a UI action, and sets the PRP onto the response.

```
Object response =
FacesContext.getCurrentInstance().getExternalContext().getResponse();
if (response instanceof StateAwareResponse) {
    StateAwareResponse stateResponse = (StateAwareResponse) response;
    stateResponse.setRenderParameter(""hotelName";", ""Name
of Hotel");
}
```

Example 28.6. Retrieve parameter

The retrieving portlet requires a Bean with a getter/setter for the parameter, to enable Portlet Bridge to set the parameter.

```
public class BookingPRP {
    private String hotelName;

    public String getHotelName() {
        return hotelName;
    }

    public void setHotelName(String hotelName) {
        this.hotelName = hotelName;
    }
}
```

To set the value on the bean, the Bridge needs to be informed which PRP to retrieve and which Bean to set it on.

If the getter/setter Bean is defined with a EL name of bookingPRP, the following configuration is required in **faces-config.xml**.

```
<application>
  <application-extension>
    <bridge:public-parameter-mappings>
      <bridge:public-parameter-mapping>
        <parameter>[bookingMapPortlet]:[hotelName]</parameter>
        <model-el>#{bookingPRP.hotelName}</model-el>
      </bridge:public-parameter-mapping>
    </bridge:public-parameter-mappings>
  </application-extension>
</application>
```



Important

Specifying the portlet name restricts model updates to the specified portlet only. If the portlet name is omitted and only the render parameter specified, every portlet within the web application that supports that render parameter will have its model updated.

[bookingMapPortlet]

The arbitrary name of the portlet, as set in **portlet.xml**

[hotelName]

The arbitrary name of the render parameter.

The handler specified in the **portlet.xml** requires a bean to process the updates to the model, as a result of the Public Render Parameter.

```
public class BookingPRPHandler implements
BridgePublicRenderParameterHandler {
    public void processUpdates(FacesContext context) {
        ELContext elContext = context.getELContext();
        BookingPRPBean bean = (BookingPRPBean)
elContext.getELResolver().getValue(elContext, null,
        "bookingPRP");

        if(null != bean) {
            System.out.println("*****processUpdates from
BookingPRPHandler: " + bean.getHotelName());
        }
    }
}
```

[Report a bug](#)

28.16. Portlet Session

To share data with other portlets within the same portlet application, use name/value pairs within the `PortletSession`.

```
Object objSession =
```

```

FacesContext.getCurrentInstance().getExternalContext().getSession(false);
try {
    if (objSession instanceof PortletSession) {
        PortletSession portalSession = (PortletSession)objSession;
        portalSession.setAttribute("your parameter name", "parameter
value", PortletSession.APPLICATION_SCOPE);
    }
}

```

In the JSP or Facelets page, the value string can be retrieved using the following code.

```
#{httpSessionScope['your parameter name']}
```

[Report a bug](#)

28.17. Resource serving

When using resources from a JSF portlet, it is important to ensure that they are accessed in a way that allows JBoss Portlet Bridge to construct an appropriate Portal URL to the resource being requested.

The correct way to reference a resource is to locate it within the web application, in the / **resources** directory. This placement allows the resource to be retrieved using JSF2 resource handling.

```
#{resource['/stylesheet.css']}
```



Important

`#{resource}` is particularly important when using `@import` to retrieve content within CSS files.

In this instance, `stylesheet.css` would be present in the root of the /**resources** directory, because it does not specify a resource library. For resources that do specify a library, these must be placed in the library sub-directory.

[Report a bug](#)

28.18. Serving JSF Resources in a Portlet

The bridge deals with portlet served resources in one of two ways:

- ✧ If the request is for a non-JSF resource, the bridge handles the request by acquiring a request dispatcher and forwarding the request to the named resource.
- ✧ If the request is for a JSF resource, the bridge runs the full JSF life cycle ensuring that data is processed and the resource (markup) is rendered.

[Report a bug](#)

28.19. Expression Language Reference

Some examples demonstrating how to use **EL** and a simple bean are provided in the `ResourceBean.java` file of the Richfaces Portlet example.

```
package org.richfaces.demo.common;

import java.util.Collection;
import java.util.Collections;
import java.util.Map;
import java.util.Set;

import javax.faces.bean.ManagedBean;
import javax.faces.context.ExternalContext;
import javax.faces.context.FacesContext;
import javax.portlet.MimeResponse;
import javax.portlet.ResourceURL;

/**
 * @author <a href="http://community.jboss.org/people/kenfinni">Ken
Finnigan</a>
 */
@ManagedBean(name = "portletRes")
public class PortletResource implements Map<String, String> {

    @Override
    public void clear() {
    }

    @Override
    public boolean containsKey(Object arg0) {
        return true;
    }

    @Override
    public boolean containsValue(Object arg0) {
        return true;
    }

    @Override
    public Set<java.util.Map.Entry<String, String>> entrySet() {
        return Collections.emptySet();
    }

    @Override
    public String get(Object resourceKey) {
        FacesContext context = FacesContext.getCurrentInstance();
        String resourceUrl = null;

        if (null != resourceKey) {
            ExternalContext extCon = context.getExternalContext();
            MimeResponse response = (MimeResponse) extCon.getResponse();
            ResourceURL resUrl = response.createResourceURL();
            resUrl.setResourceID(resourceKey.toString());
            resourceUrl = resUrl.toString();
        }
        return resourceUrl;
    }
}
```

```

@Override
public boolean isEmpty() {
    return false;
}

@Override
public Set<String> keySet() {
    return Collections.emptySet();
}

@Override
public String put(String arg0, String arg1) {
    return null;
}

@Override
public void putAll(Map<? extends String, ? extends String> arg0) {
}

@Override
public String remove(Object arg0) {
    return null;
}

@Override
public int size() {
    return 0;
}

@Override
public Collection<String> values() {
    return Collections.emptySet();
}
}

```

See Also:

✱ [Section 28.21.20, “Using Provided EL Variables”](#)

[Report a bug](#)

28.20. Expression Language Configuration

When you have the normal `/images`, `/styles` and other resource folders in your web application, you can use the following **EL** expression to serve them in your JSF application.

```
{resource['/img/the-path-to-my-image.png']}
```

Copy the **ResourceBean.java** code described in [Section 28.19, “Expression Language Reference”](#), and add an entry to the **faces-config.xml** for the bean.

```
<managed-bean>
```

```
<managed-bean-name>resource</managed-bean-name>
<managed-bean-class>org.richfaces.demo.common.ResourceBean</managed-bean-
class>
<managed-bean-scope>application</managed-bean-scope>
</managed-bean>
```

[Report a bug](#)

28.21. Developing Portlets with the Bridge

28.21.1. Implementing Portlet Bridge

Portlet Bridge supports JSF2 and RichFaces 4 portlets. There are subtle differences in the way these portlets require Portlet Bridge to be implemented.

JSF2 and RichFaces 4 portlets must be JSF2 portlets for this version of the Portlet Bridge to function as expected. There are no other specific requirements.

[Report a bug](#)

28.21.2. Declaring Artifact Dependencies

A JSF2 portlet can inherit the correct dependencies by configuring the Maven repository. This will include all dependencies for all Portlet Bridge JSF2 scenarios.

Procedure 28.3. Artifact Dependencies

1. Add the following **pom.xml** dependencies to inherit the correct JSF2 artifacts.

```
<dependency>
  <groupId>org.jboss.portletbridge</groupId>
  <artifactId>portletbridge-api</artifactId>
  <version>3.1.2.Final</version>
</dependency>
<dependency>
  <groupId>org.jboss.portletbridge</groupId>
  <artifactId>portletbridge-impl</artifactId>
  <version>3.1.2.Final</version>
  <scope>runtime</scope>
</dependency>
```

2. For RichFaces 4 portlets, add the following additional dependency to inherit the additional RichFaces dependencies.

```
<dependency>
  <groupId>org.jboss.portletbridge</groupId>
  <artifactId>portletbridge-extension-richfaces</artifactId>
  <version>3.1.2.Final</version>
  <scope>runtime</scope>
</dependency>
```

[Report a bug](#)

28.21.3. Declaring Depchain Dependencies

It is possible to reduce the **pom.xml** configuration to a single dependency, using the Portlet Bridge Depchain **pom.xml** configuration.

Procedure 28.4. JSF2 Depchain Dependencies

1. Add the following dependencies to inherit the correct artifacts using the **pom.xml** Depchain.

```
<dependency>
  <groupId>org.jboss.portletbridge</groupId>
  <artifactId>jsf2-depchain</artifactId>
  <version>3.1.2.Final</version>
  <type>pom</type>
</dependency>
```

2. For RichFaces 4 portlets, add the following additional dependency to inherit the correct artifacts using the **pom.xml** Depchain.

```
<dependency>
  <groupId>org.jboss.portletbridge</groupId>
  <artifactId>jsf2-depchain</artifactId>
  <version>3.1.2.Final</version>
  <type>pom</type>
</dependency>
<dependency>
  <groupId>org.jboss.portletbridge</groupId>
  <artifactId>richfaces4-depchain</artifactId>
  <version>3.1.2.Final</version>
  <type>pom</type>
</dependency>
```

[Report a bug](#)

28.21.4. Deploying Portlet Bridge Portlets

Red Hat JBoss Portal injects Portlet Bridge functionality into all JSF2 portlets by default, therefore portlets developed for the portal do not need to include the Portlet Bridge as part of the application.

If the Portlet Bridge API is included in a JSF2 or RichFaces 4 portlet, the provided scope must be correctly declared.

Procedure 28.5. Setting the Provided Scope for JSF2 and RichFaces Portlets that use Portlet Bridge API

1. Add the following **pom.xml** dependencies to declare the JSF2 provided scope.

```
<dependency>
  <groupId>org.jboss.portletbridge</groupId>
  <artifactId>portletbridge-api</artifactId>
  <version>3.1.2.Final</version>
  <scope>provided</scope>
</dependency>
```

- For RichFaces 4 portlets, add the following additional dependency to declare the RichFaces 4 runtime scope.



Note

The RichFaces 4 extension is not included in the portal by default.

```
<dependency>
  <groupId>org.jboss.portletbridge</groupId>
  <artifactId>portletbridge-extension-richfaces</artifactId>
  <version>portletbridge-extension-richfaces- [VERSION]</version>
  <scope>runtime</scope>
</dependency>
```

Where *[VERSION]* is the **portletbridge-extension-richfaces** version in **\$JPP_HOME/modules/org/jboss/portletbridge/api/main**

[Report a bug](#)

28.21.5. Disable Automatic Portlet Bridge Injection

In some cases, it is appropriate to prevent the portal from injecting Portlet Bridge into the application. For example, if a different version of Portlet Bridge is required for the application.

Procedure 28.6. Disabling automatic Portlet Bridge injection

- Add the following **web.xml** dependency to prevent injecting the packaged Portlet Bridge implementation into a portlet application.



Note

If a portlet application specifies either **WAR_BUNDLES_JSF** or **JSF_CONFIG_NAME** context parameters in the **web.xml**, Portlet Bridge will not be automatically injected.

```
<context-param>
  <param-
    name>org.gatein.portletbridge.WAR_BUNDLES_PORTLETBRIDGE</param-name>
    <param-value>true</param-value>
  </context-param>
```

[Report a bug](#)

28.21.6. Supported Portlet Tags

Portlet Bridge supports the following tags from section PLT.26 of the Portlet 2.0 Spec (JSR 286):

- ✱ **actionURL**
- ✱ **renderURL**

- » resourceURL
- » namespace
- » param
- » property

When using the tag library, the following namespace must be added to the facelet page of the application.

```
xmlns:pbr="http://jboss.org/portletbridge"
```

Example 28.7. renderURL example

Generate a Portlet Render URL that enables switching between Portlet modes.

```
<pbr:renderURL var="renderUrl" portletMode="edit">
</pbr:renderURL>
<h:outputLink value="#{renderUrl}">Edit Mode</h:outputLink>
```

Example 28.8. namespace example

This portlet tag is particularly useful for prefixing JavaScript functions within a given portlet. This ensures that the JavaScript function name does not clash with a name from another portlet, or from the same portlet displayed on the same page multiple times.

An example for defining a JavaScript method is:

```
<script type='text/javascript'>
  function <pbr:namespace />DoSomething() {
  }
</script>
```

[Report a bug](#)

28.21.7. Excluding Attributes from the Bridge Request Scope

When your application uses request attributes on a per request basis and you do not need that particular attribute to be managed in the extended bridge request scope, you must use the following configuration in your **faces-config.xml**.

In the code sample below, any attribute namespaced as **foo.bar**, or any attribute beginning with **foo.baz**. (**wild-card**) will be excluded from the bridge request scope and only be used for that application's request.

```
[<application>
  <application-extension>
    <bridge:excluded-attributes>
      <bridge:excluded-attribute>foo.bar</bridge:excluded-attribute>
```

```

    <bridge:excluded-attribute>foo.baz.*</bridge:excluded-attribute>
  </bridge:excluded-attributes>
</application-extension>
</application>

```

[Report a bug](#)

28.21.8. Prevent Resources Being Added to Portal Page Head

Portlet Bridge will add any JSF Resources for the portlet into the <head> section of the portal page the portlet is on, providing the portlet container supports marking up the head of a page.

To prevent Portlet Bridge from adding resources into the <head> section of the portal page, override the default behavior by declaring the following directive in the portlet's **web.xml** file.

```

<context-param>
  <param-name>org.jboss.portletbridge.markupHead.enabled</param-name>
  <param-value>>false</param-value>
</context-param>

```

[Report a bug](#)

28.21.9. JSF Facelet View

When creating a JSF Facelet view document, it is common to see content wrapped with the following code.

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" ">
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:h="http://java.sun.com/jsf/html"
xmlns:f="http://java.sun.com/jsf/core"
    xmlns:ui="http://java.sun.com/jsf/facelets" >
<!-- Unnecessary content removed for clarity -->
</html>

```

Because a single portlet only reflects a potentially small portion of the HTML markup for a page, a JSF portlet returning the above markup for each portlet can be distracting, and potentially problematic.

The recommended way to wrap the content of a JSF Facelet view document for a portlet is to use the <f:view> element. Using <f:view> results in the relevant portlet markup being returned to the page as required.

```

<f:view xmlns="http://www.w3.org/1999/xhtml"
xmlns:h="http://java.sun.com/jsf/html"
xmlns:f="http://java.sun.com/jsf/core"
    xmlns:ui="http://java.sun.com/jsf/facelets" >
<!-- Unnecessary content removed for clarity -->
</f:view>

```

**Note**

While not specifically relevant to portlet, it is important to include `<h:head>` and `<h:body>` elements to allow JSF to process the facelet correctly.

[Report a bug](#)

28.21.10. Error Handling

To display error pages for specific exceptions in a JSF portlet, the following code is required in the `web.xml` file.

```
<error-page>
  <exception-type>javax.servlet.ServletException</exception-type>
  <location>/faces/error.xhtml</location>
</error-page>
<error-page>
  <exception-
type>javax.faces.application.ViewExpiredException</exception-type>
  <location>/faces/error.xhtml</location>
</error-page>
```

The above error page definitions are appropriate for a JSF portlet that has a `FacesServlet` mapping such as `/faces/`.

If the `FacesServlet` mapping was `**.jsf`, the location would be `/error`, `/error.jsf` or `/error.xhtml`.

There is no requirement when using `FacesServlet` suffix mapping to append an extension, the name of the view is all that is required for the page to be found

**Note**

Remember to add a link from any error page to the normal flow of the JSF application, otherwise the same error page will be constantly displayed.

[Report a bug](#)

28.21.11. Switching Portlet Modes

A **PortletMode** represents a distinct render path within an application. There are three standard modes: *view*, *edit*, and *help*.

The bridge's **ExternalContext.encodeActionURL** recognizes the query string parameter **javax.portlet.faces.PortletMode** and uses this parameter's value to set the portlet mode on the underlying portlet **actionURL** or response. Once processed it then removes this parameter from the query string.

Example 28.9. /edit.xhtml navigation rule

This navigation rule causes one to render the `/edit.xhtml` viewId in the portlet edit mode.


```
<navigation-rule>
  <from-view-id>/register.xhtml</from-view-id>
  <navigation-case>
    <from-outcome>edit</from-outcome>
    <to-view-id>/edit.xhtml?javax.portlet.faces.PortletMode=edit</to-view-id>
  </navigation-case>
</navigation-rule>
```

[Report a bug](#)

28.21.12. Navigating to a mode's last viewId

By default a mode change will start in the mode's default view without any (prior) existing state. One common portlet pattern when returning to a mode left after entering another mode (e.g.. view -> edit -> view) is to return to the last view (and state) of this original mode.

The bridge will explicitly encode the necessary information so that when returning to a prior mode it can target the appropriate view and restore the appropriate state.

The session attributes maintained by the bridge are intended to be used by developers to navigate back from a mode to the last location and state of a prior mode. As such, a developer must describe a dynamic navigation: "From view *X* return to the last view of mode *Y*".

This is most easily expressed via an **EL** expression. For example:

Example 28.10. /register.xhtml viewId navigation rule

This navigation rule causes one to render the `/edit.xhtml` viewId in the portlet edit mode.

```
<navigation-rule>
  <from-view-id>/edit.xhtml*</from-view-id>
  <navigation-case>
    <from-outcome>view</from-outcome>
    <to-view-id>#{
sessionScope['javax.portlet.faces.viewIdHistory.view']}</to-view-id>
  </navigation-case>
</navigation-rule>
```

[Report a bug](#)

28.21.13. Using Wildcards to Identify the Rule Target

When using values from session scoped attributes, or viewIds which may contain query string parameters, it may be necessary to use the wild-card syntax when identifying the rule target.

In [Section 28.21.12, "Navigating to a mode's last viewId"](#) the `<to-view-id>` expression returns a **viewId** of the form

```
/viewId?javax.portlet.faces.PortletMode=view&....
```

Without wild-carding, when a subsequent navigation occurs from this new view, the navigation rules would not resolve because there would not be an exact match.

Similarly, the `edit.jspx<from-view-id>` is wild-carded because there are navigation rules that target it that use a query string:

```
<to-view-id> /edit.jspx?javax.portlet.faces.PortletMode=edit </to-view-id>
```

Developers are encouraged to use this wild card method to ensure they execute properly in the broadest set of bridge implementations.

[Report a bug](#)

28.21.14. Clearing the View History when Changing Portlet Modes

By default the bridge remembers the view history when you switch to a different portlet mode (like "Help" or "Edit"). You can use the following parameter in your `portlet.xml` to use the default viewId each time you switch modes.

```
<init-param>
  <name>javax.portlet.faces.extension.resetModeViewId</name>
  <value>true</value>
</init-param>
```

[Report a bug](#)

28.21.15. Communication Between Portlets

There are four different ways to send messages, events, and parameters between portlets which are contained in different `ears/wars` or contained in the same `war`.

Having two portlets in the same `war` or having them separated does not affect the Portlet Container because each portlet has a different `HttpSession`.

The recommended way to share a parameter or event payload between two or more portlets with the Portlet 2.0 specification are the [Section 28.12, "Public Render Parameters"](#) and [Section 28.9, "Sending and Receiving Events"](#) mechanisms.

This allows you to decouple your application from surgically managing objects in the `PortletSession.APPLICATION_SCOPE`.

However, if these do not meet your use case or you have a different strategy, you can use one of the following methods.

[Report a bug](#)

28.21.16. Storing Components in PortletSession.APPLICATION_SCOPE

Sometimes it is beneficial to store your `Seam` components in the portlet `APPLICATION_SCOPE`.

By default, these objects are stored in the `PORTLET_SCOPE` but with the annotation below, this class can be pulled out of the `PortletSession` and its values used in other portlets across different `Seam` applications.

```
@PortletScope(PortletScope.ScopeType.APPLICATION_SCOPE)
```

Then you would pull the stateful object from the session:

```
YourSessionClass yourSessionClass =
(YourSessionClass)getRenderRequest().getAttribute("javax.portlet.p./default/seamproject/seamprojectPortletWindow?textHolder");
```

This method is demonstrated in this video: [Lesson 2: Portlet 1.0 Advanced Seam and RichFaces](#)

[Report a bug](#)

28.21.17. Using the PortletSession

If you only need to access the **PortletSession** to share a parameter or value across multiple portlets, you can use the following:

```
Object objSession =
FacesContext.getCurrentInstance().getExternalContext().getSession(false);
try
{
    if (objSession instanceof PortletSession)
    {
        PortletSession portalSession = (PortletSession)objSession;
        portalSession.setAttribute("your parameter name", "parameter
value", PortletSession.APPLICATION_SCOPE);
        ...
    }
}
```

Then, in your JSP or Facelets page, you can use:

```
<#{httpSessionScope['your parameter name']}
```

[Report a bug](#)

28.21.18. Linking to a Facelets page within the Same Portlet

To link any JSF/Facelets page within a portlet web application, use the following code.

```
<h:outputLink value="#"
{facesContext.getExternalContext().requestContextPath}/home.xhtml">
    <f:param name="javax.portlet.faces.ViewLink" value="true"/>
    navigate to the test page
</h:outputLink>
```

[Report a bug](#)

28.21.19. Redirecting to an External Page or Resource

To link to a non JSF view, *jboss.org* for example, you can use the following code.

```
<h:commandLink actionListener="#{yourBean.yourListener}">
    <f:param name="javax.portlet.faces.DirectLink" value="true"/>
    navigate to the test page
</h:commandLink>
```

In the backing bean, a ***redirect()*** must be called.

```
public class YourBean {
    public void yourListener() {

FacesContext.getCurrentInstance().getExternalContext().redirect("http://
www.jboss.org");
    }
}
```

[Report a bug](#)

28.21.20. Using Provided EL Variables

All **EL** variables found in the JSR-329 (Portlet 2.0) specification are available in the JBoss Portlet Bridge.

Table 28.3.

| | |
|----------------|--|
| portalConfig | Object of type javax.portlet.PortletConfig |
| actionRequest | Object of type javax.portlet.ActionRequest (only accessible when processing an ActionRequest) |
| actionResponse | Object of type javax.portlet.ActionResponse (only accessible when processing an ActionResponse) |
| eventRequest | Object of type javax.portlet.EventRequest (only accessible when processing an EventRequest) |
| eventResponse | Object of type javax.portlet.EventResponse (only accessible when processing an EventRequest) |
| renderRequest | Object of type javax.portlet.RenderRequest (only accessible when processing an RenderResponse) |

| | |
|---------------------------------|---|
| renderResponse | Object of type javax.portlet.RenderResponse (only accessible when processing an RenderResponse) |
| resourceRequest | Object of type javax.portlet.ResourceRequest (only accessible when processing an ResourceRequest) |
| resourceResponse | Object of type javax.portlet.ResourceResponse (only accessible when processing an ResourceResponse) |
| portletSession | Current PortletSession object |
| portletSessionScope | Map of PortletSession attributes in PORTLET_SCOPE. JSP Expression returns immutable Map, but Faces Expression returns mutable Map. |
| httpSessionScope | Mutable Map of PortletSession attributes in APPLICATION_SCOPE |
| portletPreferences | Current PortletPreferences object |
| portletPreferencesValues | Immutable Map containing entries equivalent to PortletPreferences.getMap() |
| mutablePortletPreferencesValues | Mutable Map of type Map<String, javax.portlet.faces.preference.Preference>. This EL variable provides read/write access to each portlet preference. |

Example 28.11.

This code allows you to edit the portlet preferences on the UI

```
<h:form>
  <h:inputText id="pref" required="true" value="#
{mutablePortletPreferencesValues['userName'].value}" />
  <h:commandButton actionListener="#{myBean.savePref}" value="Save
Preferences" />
</h:form>
```

In the backing bean, call the PortletPreferences.store() method.

```
Object request =
FacesContext.getCurrentInstance().getExternalContext().getRequest();
PortletRequest portletRequest = (PortletRequest)request;
if (request instanceof PortletRequest) {
```

```
try {  
    PortletPreferences portletPreferences =  
    portletRequest.getPreferences();  
    portletPreferences.store();
```

[Report a bug](#)

Chapter 29. Navigation Portlet Using the Public API

Portal API can be used to implement a Navigation Portlet.

See Also:

- » [Section 22.1, “Starting a Portlet Project”](#)
- » [Section 22.3, “Standard Portlet Development \(JSR-286\)”](#)

[Report a bug](#)

29.1. Example Code

The only difference in the `pom.xml` standard portlet project code is the `gatein-api` dependency.

Example 29.1. pom.xml

```
<dependency>
  <groupId>org.gatein.api</groupId>
  <artifactId>gatein-api</artifactId>
</dependency>
```

This section cites code from Navigation API Portlet from the Quickstarts Collection (<https://docs.jboss.org/author/pages/viewpage.action?pageId=62587128>).

See Also:

- » [Section 22.1, “Starting a Portlet Project”](#)

[Report a bug](#)

29.1.1. Required Java Classes

Two classes are required to implement the Navigation Portlet: **NavigationPortlet** and **NavigationNodeBean**.

The **NavigationPortlet.doView(RenderRequest, RenderResponse)** method is responsible for rendering the main persistent menu.

Example 29.2. The doView method

```
protected void doView(RenderRequest request, RenderResponse response)
throws PortletException, IOException {

    PortalRequest portalRequest = PortalRequest.getInstance();

    Navigation navigation = PortalRequest.getInstance().getNavigation();

    // Diving two levels so the information about children count of
    children nodes is available
    Node rootNode = navigation.getRootNode(Nodes.visitNodes(2));
```

```

navigationRootNodeBean = new NavigationNodeBean(rootNode);

List<NavigationNodeBean> rootNodeChildrenList =
navigationRootNodeBean.getChildren();

/* Setting the 1st node to be active when accesing the root node "/"
*/
if (!rootNodeChildrenList.isEmpty() &&
portalRequest.getNodePath().equals(NodePath.root())) {
    navigationRootNodeBean.setFirstActive();
}

request.setAttribute("navigationRootNode", navigationRootNodeBean);

PortletRequestDispatcher prd =
getPortletContext().getRequestDispatcher("/jsp/navigation.jsp");
prd.include(request, response);
}

```

The **NavigationPortlet.serveResource(ResourceRequest, ResourceResponse)** method serves the AJAX requests to render sub-menus. For a given sub-menu URI, it returns a HTML fragment containing the sub-menu.

Example 29.3. The serveResource method

```

public void serveResource(ResourceRequest request, ResourceResponse
response) throws PortletException, IOException {

    Navigation navigation = PortalRequest.getInstance().getNavigation();

    String chosenNodeURI = request.getParameter("uri");

    Node chosenNode =
navigation.getNode(NodePath.fromString(chosenNodeURI),
Nodes.visitNodes(2));

    request.setAttribute("parentNode", new
NavigationNodeBean(chosenNode));

    PortletRequestDispatcher prd =
getPortletContext().getRequestDispatcher("/jsp/node.jsp");
prd.include(request, response);
}

```

The **NavigationNodeBean** serves as a model and contains all the information being rendered on the page. In the **NavigationPortlet.doView()** method, the **navigationRootNodeBean** field is representing the root node of the navigation and contains main menu (top-menu) elements (Home and Sitemap by default) as children nodes.

[Report a bug](#)

29.1.2. Required JSP

JSP (JavaServer Pages) is used for rendering.

The **navigation.jsp** file encapsulates the **node.jsp** output into an element with unique id identifier.

Example 29.4. Top Portlet JSP Source Code

```
<div id="id<portlet:namespace/>_gtnQuickstartNavigationPortlet"
class="gtnQuickstartNavigationPortlet ">
    <!-- Render the main menu, if nodes are available --%>
    <c:if test="${fn:length(navigationRootNode.children) > 0}">
        <c:set var="parentNode" value="${navigationRootNode}"
scope="request" />
        <c:set var="menuType" value="topmenu" scope="request" />
        <jsp:include page="node.jsp" />
    </c:if>
</div>
```

The **node.jsp** file is responsible for rendering the menus.



Note

The HTML comment tag does not disable the JSP code. It is used to remove the whitespace for more precise CSS styling.

Example 29.5. Menu JSP Source Code

```
<ol class="${menuType} collapsibleContent"><!--
    <c:forEach var="child" items="${parentNode.children}">
<c:if test="${!child.system}">
    --><li class="menuitem <c:if test='${child.active}'>active</c:if>
<c:if test='${(child.parent) & & (child.page)}'>
multilevel</c:if>">
    <portlet:resourceURL var="ajaxResourceUrl">
<portlet:param name="uri" value="${child.path}" />
    </portlet:resourceURL>
    <c:choose>
<!-- Node is a clickable page and contains children nodes --%>
<c:when test="${child.page & & child.parent}">
    <a href="${child.URI}"><span>${child.name}</span></a><!--
    --><a href="#${ajaxResourceUrl}" class="caret menuhandler">
<i>${resourceBundle.getString("label.children")}</i></a>
</c:when>
<!-- Node is a clickable page but doesn't contain any children nodes --
%>
<c:when test="${child.page & & !child.parent}">
    <a href="${child.URI}"><span>${child.name}</span></a>
</c:when>
<!-- Node is not a clickable page but contains children nodes, it's a
"category" node --%>
<c:when test="${!child.page & & child.parent}">
```

```

    <a href="#"${ajaxResourceUrl}" class="menuhandler">
<span>${child.name}</span><i
class="caret">${resourceBundle.getString("label.children")}</i></a>
</c:when>
<!-- Do nothing with non-clickable "category" nodes without children
nodes -->
<c:otherwise>
    <span>${child.name}</span>
</c:otherwise>
    </c:choose>
</li>
<!--
</c:if>
</c:forEach>
-->
</ol>

```

[Report a bug](#)

29.1.3. Required JavaScript

The JavaScript part was implemented as a JQuery plug-in. It has several roles:

- * AJAX support
- * Basic on-click drop-down support
- * Dynamically respond to window changes, position menus within the browser window

The asynchronous requests use the JQuery **ajax()** method:

Example 29.6. The JQuery plugin for drop-down menu

```

$.ajax({
    type: "POST",
    url: $(this).attr('href').substring(1),
    cache: false,
    dataType: "text",
    success: function(data)
    {
        menuItem.append(data);
        openSubmenu(menuItem);
    },
    error: function(XMLHttpRequest, textStatus, errorThrown)
    {
        console && console.log("Ajax error");
    }
});

```

For more details about the implementation, see the comments in the **dropdown.jquery.js** file.

[Report a bug](#)

29.1.4. Further Steps

Read and complete the following sections to test the functionality:

- ✦ [Section 22.2, “Building and Deploying Portlets”](#)
- ✦ *Import Portlets and Gadgets* section in the *User Guide*.
- ✦ *Manage Pages* section in the *User Guide*.

[Report a bug](#)

29.2. See also

- ✦ [Section 21.1, “JSR-168 and JSR-286 overview”](#)

[Report a bug](#)

Chapter 30. Using Data from Social Networks in Portlets

This chapter describes how to consume data from social networks such as Google+, Facebook and Twitter in portlets. While the kind of data accessible through public APIs varies among the providers, it is generally possible to read data from social networks, and in some cases, write data (in the case of Facebook).

[Report a bug](#)

30.1. Social Portlets Example Code

This section cites code from Social Portlets example project from the [Quickstarts Collection](#) . There is a portlet that displays the most recent Facebook wall posts, a portlet that displays a list of Google+ circles (and who is in the circles), and a portlet that shows user account details from Twitter.

[Report a bug](#)

30.1.1. Prerequisites

Prerequisites:

- ✧ [Section 22.1, “Starting a Portlet Project”](#)
- ✧ [Section 22.3, “Standard Portlet Development \(JSR-286\)”](#)
- ✧ [Chapter 26, *CDI Portlet Development*](#)
- ✧ [Chapter 7, *OAuth Provider API*](#)

Configure the portal with a specific OAuth provider.

Finally, the given portal user must have the OAuth access token saved as part of their User Profile in the portal. To retrieve the access token the user must either:

- ✧ Authenticate or register on the portal with the specific OAuth provider (social network),
- ✧ Link their account with the OAuth Provider - this can be done through the dedicated tab of User Info screen in the portal UI or programmatically from portlets.

Once the user has an OAuth access token saved in their profile, it can be used to call various operations in the particular social network.

[Report a bug](#)

30.2. Retrieving the Access Token

The OAuth Provider API is used to retrieve the current user's access token. In the Social Portlets Quickstart, there is an **OAuthProviderFilter** portlet filter, which retrieves the social network access token of the current user and saves the token into the **RequestContext** bean. Alternative flows result in one of the following messages:

- ✧ OAuth provider is not enabled in portal configuration.
- ✧ User is not logged in.
- ✧ User's access token does not exist.

- » User's access token is invalid or expired.

These messages are presented by **error.jsp** or **token.jsp**. In case the access token is unavailable or invalid, the **token.jsp** will show a link, which the user can follow to link their GateIn Portal account with the given social network. Once linked, the user will obtain the access token.

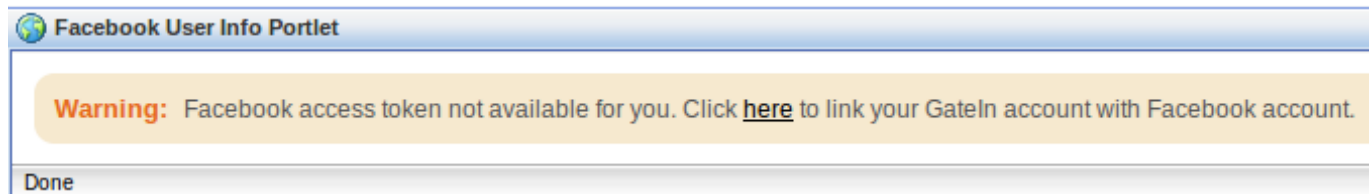


Figure 30.1. Token Unavailable Message

Example 30.1. Declaration of OAuthProviderFilter in portlet.xml

```
<filter>
  <filter-name>FacebookFilter</filter-name>
  <filter-
class>org.gatein.security.oauth.portlet.OAuthPortletFilter</filter-
class>
  <lifecycle>ACTION_PHASE</lifecycle>
  <lifecycle>RENDER_PHASE</lifecycle>
  <init-param>
    <name>accessTokenValidation</name>
    <value>SKIP</value>
  </init-param>
  <init-param>
    <name>oauthProviderKey</name>
    <value>FACEBOOK</value>
  </init-param>
</filter>
```

Example 30.2. Mapping of OAuthProviderFilter in portlet.xml

```
<filter-mapping>
  <filter-name>FacebookFilter</filter-name>
  <portlet-name>Facebook*</portlet-name>
</filter-mapping>
```

Mapping of OAuthProviderFilter in portlet.xml, specifies that all portlets with the name starting with "**Facebook**" are handled by this filter. This same mapping configuration can be applied to Google+ and Twitter.

All portlets in the Social Portlets Quickstart extend the **AbstractSocialPortlet**. It is a subclass of the standard portlet API's **GenericPortlet**. It contains some basic functionality and helper methods useful for all social portlets, one of which is handling the action to redirect to the Social login.

Example 30.3. actionRedirectToOAuthFlow in AbstractSocialPortlet

```
@ProcessAction(name = ACTION_OAUTH_REDIRECT)
public void actionRedirectToOAuthFlow(ActionRequest aReq,
    ActionResponse aResp) throws IOException {
    String customScope = aReq.getParameter(PARAM_CUSTOM_SCOPE);
    getOAuthProvider().startOAuthWorkflow(customScope);
}
```

If **customScope** is null, the portal will request scopes only configured in **configuration.properties**. Otherwise, the portal will request this scope in addition to scopes from **configuration.properties**. This allows portlets to ask for more scopes than scopes from **configuration.properties**.

Example 30.4. Widening the Scope On-demand for OAuth Token Permissions

Portlets can request wider scope *on demand* in cases when wider scope is required to complete a task.

In **configuration.properties**, Facebook is enabled only with the default scope email.

In **FacebookStatusUpdatePortlet**, a user wants to publish messages on their Facebook Wall. For this purpose, an access token with scope `publish_message` is required.

Before the user is able to publish a message on their Facebook wall, the portlet displays an error message that the present scope is insufficient, and offers the user an opportunity to correct the issue.

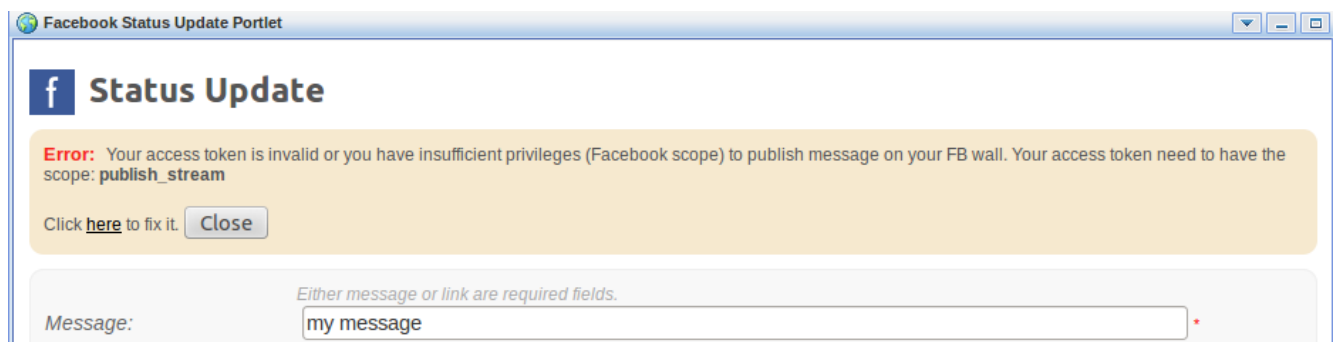


Figure 30.2. Insufficient Privileges Error

Clicking the link starts an OAuth2 flow. The user is redirected to Facebook and is asked to permit the `publish_message` scope. After permitting the scope change, the user is able to publish messages.

In this scenario, CDI is used to store the access token. The **RequestContext** class is annotated as **@RequestScoped**. Each portlet contains only logic specific to its purpose, and does not need to specifically handle retrieving the access token.

See Also:

- » [Chapter 7, OAuth Provider API](#)
- » [Section 30.3, "Facebook"](#)

[Report a bug](#)

30.3. Facebook

There are three portlets that demonstrate a basic integration with Facebook.

title

FacebookUserInfoPortlet

A simple portlet showing some details of the current user's profile on Facebook, such as first name, last name, username, e-mail and profile picture.

FacebookFriendsPortlet

An advanced portlet showing friends of the current user and their pictures. It also supports pagination (only 10 friends are on screen, but you can switch to different page size to see more friends). When clicking a friend, the portlet displays the last few status messages from the given friend's Facebook Wall.

FacebookStatusUpdatePortlet

A portlet with the capability to publish messages to the Wall of the current user.

All three portlets leverage the third-party library Rest FB(<http://restfb.com>), which provides the capability to send requests to the Facebook Graph API and convert JSON responses to plain Java objects (POJOs).

To decouple logic from the UI and handle error situations, the helper class **FacebookClientWrapper** offers multiple methods to retrieve objects from the Facebook Graph API and convert them to POJOs or beans relevant to the portal. These converted methods can be dispatched by **PortletRequestDispatcher** to JSP pages, which handle the HTML markup rendering.

The example portlets are using a small subset of the Facebook Graph API. It is possible use the whole spectrum of the Facebook Graph API operations in your own portlets. The complete documentation of Graph API is available on the Facebook Developers site, which is located at <http://developers.facebook.com/docs/reference/api/>.

[Report a bug](#)

30.4. Google+

There are three example portlets for showing integration with Google+. All portlets leverage Google SDK libraries.

- ✦ **GoogleUserInfoPortlet** - a simple portlet which shows some details about the current user from Google+, such as their first name, last name, e-mail, and profile picture. See <https://developers.google.com/accounts/docs/OAuth2WebServer#callinganapi> for more details.
- ✦ **GoogleFriendsPortlet** - a simple portlet for displaying Google+ friends and their photos. It supports pagination (only 10 friends are displayed on screen each time). See <https://developers.google.com/+/api/latest/people/list> for details.
- ✦ **GoogleActivitiesPortlet** - a simple portlet for displaying the latest activities from Google+ Stream of the current user. See <https://developers.google.com/+/api/latest/activities/list> for details.

GoogleUserInfoPortlet leverages [Google OAuth2 API library](#) and it uses an **OAuth2** object to call the operation.

GoogleFriendsPortlet and **GoogleActivitiesPortlet** leverage [Google+ API library](#) and they use **Plus** object to call the operations. Portlets have available access token and they obtain the required object through a call to **oauthProvider.getAuthorizedAPIObject** as described in [Chapter 7, OAuth Provider API](#).

[Report a bug](#)

30.5. Twitter

There is one simple portlet for Twitter: **TwitterPortlet**. It shows details about the current user, such as their picture, last tweet, number of followers, and other user data commonly available on a twitter user's profile.

TwitterPortlet leverages [Twitter4J library](#) to interact with Twitter. Given that the access token is available, the portlet demonstrates how to obtain the **Twitter** client object from Twitter4J library through calling **oauthProvider.getAuthorizedAPIObject()** as described in [Section 7.1, "Retrieve an Instance of OAuthProvider"](#).

[Report a bug](#)

30.6. Configuration and Deployment Descriptors

pom.xml

In **pom.xml** file of our Social Portlets Quickstart, we need to declare dependencies on JSR 286 portlet API, GateIn Portal API and CDI API:

Example 30.5. Declaration of basic dependencies in pom.xml

```
<!--
    The versions of these dependencies are managed in gatein-*-bom.
    Note that artifacts managed in gatein-*-bom are
    <scope>provided</scope> in most cases.
119.    Name only those artifacts you refer to in your code.
    Look at gatein-*-bom POM file for the complete list of
    available artifacts.
-->
<dependency>
  <groupId>javax.portlet</groupId>
  <artifactId>portlet-api</artifactId>
</dependency>
<dependency>
  <groupId>org.gatein.api</groupId>
  <artifactId>gatein-api</artifactId>
</dependency>
<dependency>
```



```

    <groupId>javax.enterprise</groupId>
    <artifactId>cdi-api</artifactId>
    <scope>provided</scope>
  </dependency>

```

We also need to declare dependencies on 3rd party libraries to be able to access the public APIs of Facebook, Twitter and Google+:

Example 30.6. Declaration of 3rd party dependencies in pom.xml

```

<!-- Google -->
<dependency>
  <groupId>com.google.apis</groupId>
  <artifactId>google-api-services-plus</artifactId>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>com.google.apis</groupId>
  <artifactId>google-api-services-oauth2</artifactId>
  <scope>provided</scope>
</dependency>
<!-- Twitter -->
<dependency>
  <groupId>org.twitter4j</groupId>
  <artifactId>twitter4j-core</artifactId>
  <scope>provided</scope>
</dependency>
<!-- Facebook (not provided for now as it's not needed in
Portal and so it's bundled in this portlet WAR) -->
<dependency>
  <groupId>com.restfb</groupId>
  <artifactId>restfb</artifactId>
</dependency>

```

You may have noticed that **com.restfb** dependency is not declared as **provided**. This is because this library is not used by GateIn Portal itself, and therefore it is not bundled therein as well. We need to provide it in our WAR ourselves. Dependencies on Twitter4J and Google API libraries are **provided** by GateIn Portal and there is no need to package them inside our portlet WAR.

jboss-deployment-structure.xml

This file is specific for JBoss AS7 and JBoss EAP. In this file, dependencies on JBoss AS7/EAP modules are declared which are required by our portlet application.

Example 30.7. Declaration of dependencies in jboss-deployment-structure.xml

```

<deployment>
  <dependencies>
    <module name="com.google.apis" />

```

```
<module name="org.twitter4j" />
</dependencies>
</deployment>
```

Modules **org.twitter4j** and **com.google.apis** are part of GateIn Portal JBoss AS7 packaging and they contain the 3rd party libraries from Google+ and Twitter.

beans.xml

The presence of **beans.xml** file activates CDI for the application. There is nothing useful in it.

Example 30.8. beans.xml

```
<beans xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://jboss.org/schema/cdi/beans_1_0.xsd">
<!--
```

portlet.xml

It contains declaration of all social portlets. Very important is also declaration of **OAuthPortletFilter** and it's filter mapping.

gatein-resources.xml

Declaration of CSS files used by our portlets.

See Also:

- » [Section 13.6.4, “How to Create New Portlet Skins”](#)

[Report a bug](#)

30.7. Further Steps

Import Portlets and Gadgets and *Manage Pages* in the User Guide for further guidance about deploying and testing the portlets.

See Also:

- » [Section 22.2, “Building and Deploying Portlets”](#)

[Report a bug](#)

Chapter 31. Spring Framework Portlet Development

Support for Spring Portlet Model-View-Controller (MVC) is limited in the Red Hat JBoss Portal. It is recommended that other fully supported options such as JSP, JSF or JSF with RichFaces are considered as replacements to Spring Portlet MVC.



Important

Using other Spring components must be carefully evaluated as their support in the Portal environment may be limited.

See the Portlet MVC Framework section (<http://static.springsource.org/spring/docs/3.2.x/spring-framework-reference/html/portlet.html>) of the Spring Framework Reference Documentation for detailed Spring-based portlet project configuration instructions.



Important

When generating an Action URL using the portlet tag library, ensure that **escapeXml="false"** is included in the definition of **portlet:actionURL**, as described in [Example 31.1, "Action URL"](#). A page not found issue due to the Action URL not being able to map the URL to a Spring controller is likely if this advice is not followed.

Example 31.1. Action URL

```
<portlet:actionURL var="myAction" escapeXml="false" />
```

Deploying a Spring-based portlet, importing it, and adding it to a page uses the same process as any other portlet.

See *Import Portlets and Gadgets* and *Manage Pages* in the User Guide for further guidance about using the Web interface to import and manage portlets.



Limitation for Spring portlet on version 2.5

During development or migration of a Spring Portlet on version 2.5 for the platform, the configuration **<context:component-scan base-package="org.classname.etc"/>** fails to locate the beans for scanning. Spring 2.5 cannot interpret the VFS file system used by Red Hat JBoss Enterprise Application Platform therefore it cannot scan the JAR files to find the beans. Spring 3.x versions do not have this limitation. To resolve the problem, replace the **<component-scan>** tag with the bean tags for each required bean.

Spring 2.5 only supports JSR 168 (Portlet Spec 1.0), it does not include support for ResourceRequest which was introduced by JSR 286 (Portlet Spec 2.0). Without the ability to use a ResourceRequest, the means by which Ajax calls are handled in a portlet, it is not possible to use Ajax calls with a Spring 2.5 portlet. If your portlet requires Ajax calls, it must be upgraded to Spring 3.x.

See Also:

- » [Section 22.3.4, “Building and Deploying Portlets”](#)

[Report a bug](#)

Part V. JavaScript Development

Chapter 32. JavaScript Modularity

[Report a bug](#)

32.1. JavaScript Modules

The Red Hat JBoss Portal JavaScript handling improvements are built upon the concept of the JavaScript *module*. JavaScript does not provide native support for namespacing and so the notion of modules was designed to solve this problem. These modules provide namespacing and more; the module pattern allows developers to create dependencies between modules and resolve them at runtime, enabling on demand and parallel loading of JavaScript resources.

For more information see the RequireJS documentation at <http://requirejs.org/>. An in-depth article about JavaScript module patterns is available at <http://www.adequatelygood.com/2010/3/JavaScript-Module-Pattern-In-Depth>.

[Report a bug](#)

32.2. Introduction to Modules

A module can be thought of as:

- An identifier or name;
- A list of the module's dependencies;
- Packaged code, usually expressed as a self-executing function.
- The product, an object produced by the module that is usually consumed by other modules.

At runtime the dependency system defines a graph of functions to execute, the product of each module being injected in the other modules. It can be seen as a simple dependency injection system able to load modules in an asynchronous and parallel fashion, providing parallel loading, namespacing, and dependency management.

The portal provides support for different module formats:

GateIn Module Definition (GMD)

This format is the most appropriate way to define a JavaScript module in the portal. The module is expressed as a simple function, the list of dependencies being expressed in the **gatein-resources.xml** file. This format clearly decouples the module's code in a JavaScript file from the module's dependencies, expressed in a single XML file, and is the best compromise for integrating JavaScript in the portal.

Asynchronous Module Definition (AMD)

This is the native module format supported by **RequireJS**.

Adapted Module

This format is the most flexible as it adapts to other module formats such as CommonJS [modules](#).

To explore the benefits of modules (beyond namespacing) the following example uses a module consuming the jQuery library:

```
(function($) {
    $("body").on("click", "mybutton", function(event) {
        alert("Button clicked");
    });
})(jQuery);
```

The JavaScript component of our module is a function that takes the *jQuery* **\$** object as an argument:

- ✧ The jQuery object is provided as a module dependency expressed as a function argument **\$**. The module code can use any name for jQuery; in our example we use the **\$** name.

\$ jQuery can be used without needing to use the **\$. ready** function because jQuery will be already loaded and initialized when the function is executed.

- ✧ The module is scoped to the entire page and this code will react to any click on the **.mybutton** selector. Since a single event listener handles as many portlets as are on the page, this benefits performance and simplicity.

To integrate this module, it is declared in the **gatein-resources.xml** file of the WAR archive:

```
<gatein-resources
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.gatein.org/xml/ns/gatein_resources_1_3
http://www.gatein.org/xml/ns/gatein_resources_1_3"
xmlns="http://www.gatein.org/xml/ns/gatein_resources_1_3">
  <portlet>
    <name>MyPortlet</name>
    <module>
      <script>
        <path>/mymodule.js</path>
      </script>
      <depends>
        <module>jquery</module>
        <as>$</as>
      </depends>
    </module>
  </portlet>
</gatein-resources>
```

The module tag declares the module and its dependencies. In this case it is declared that jQuery is the only dependency and that it is named **\$** when the module is executed.

A dependency relationship between the module and the jQuery module has been created such that:

- ✧ The module will be loaded only when the portlet is displayed;
- ✧ The portal loads the jQuery module before loading our module;
- ✧ Several different versions of jQuery can coexist in the browser without conflicting.

This introductory example outlines the important features provided by modules: performance, isolation, dependency management and ease of use.

[Report a bug](#)

32.3. Script Support

While modules are preferred whenever possible, they cannot be used in every instance. For example, when an existing portlet uses an inline script it will access the global namespace to find functions or objects. To accommodate these instances, the traditional method of using JavaScript with a simple script declaration in the head section of a web page continues to be supported.

Dependencies between scripts can be created to control the order of the scripts in the head section and resolve dependency problems as much as possible.

[Report a bug](#)

Chapter 33. JavaScript in JBoss Portal

33.1. Declaring a Module

A JavaScript module is the declaration of a JavaScript self-executing function and its declaration in the **gatein-resources.xml** descriptor of the web application.

The descriptor defines the module and its dependencies. The portal builds a graph of resources and their dependencies at runtime. When a client requires a particular module, it invokes a JavaScript function that calls *RequireJS* to resolve dependencies.

Example 33.1. foo.js

```
(function ($) {
  // Do something with jQuery
})(jQuery)
```

This module is declared as a *self-executing function* in the **foo.js** file.

The **foo.js** file is then declared in the **gatein-resources.xml** file:

Example 33.2. gatein-resources.xml

```
...
<module>
  <name>foo</name>
  <script>
    <path>/foo.js</path>
  </script>
  <depends>
    <module>jquery</module>
    <as>jQuery</as>
  </depends>
</module>
...
```

The self-executing function declares:

- » The parameter for the function: **\$**.
- » The arguments of the function's invocation: **jQuery**.

The self-executing function argument must match the dependencies.

- » Functions do not need to match the XML dependencies order.
- » Functions can use a parameter subset; a dependency can be declared in the XML but not consumed as an argument.
- » Parameters must be aliased in dependencies with the **<as>** XML tag.



Note

The self-executing function argument is a JavaScript construct. The purpose of this construct is to pass arguments to the function, let the function name the arguments and override the current scope of execution.

For example, **jQuery** uses the following:

```
(function(window, undefined) { .... })(window);
```

Resources are related by dependency relationships, which specify how scripts are related to each other and how the modular system should be honored.

In our example, the **foo** module uses **jQuery**. They are in relationship: **foo** *depends* on **jQuery**. When the **foo** module is loaded, the **jQuery** module must be available to it.

[Report a bug](#)

33.2. Translation to the AMD System

The portal will translate logical identifiers into an AMD module:

```
define("SHARED/foo", ["SHARED/jquery"], function(jQuery) {
    return (function($) {
        // Do something with jQuery
    })(jQuery);
});
```

✧ Logical identifiers are translated to AMD identifiers:

- The *foo* module is translated to the **SHARED/foo** AMD module identifier; the *SHARED* prefix is added by the portal for scoping the name.

Other existing scopes include *PORTLET* and *PORTAL*.

- The dependency on the **jquery** module is translated to a [**SHARED/jquery**] AMD module dependency.

✧ The module function is wrapped twice:

- Once by the portal module wrapper which delegates to the module's function. The goal of this function is to provide a ? evaluation of the module's self-executing function.
- Then by the **define** AMD function that manages loading the module properly.

✧ The self-executing function module's arguments must match the module dependencies expressed in the XML declaration.

- The **jquery** dependency is aliased to the **jQuery** name in the XML **as** tag. As a consequence the portal function wrapper parameter is called **jQuery**.
- The module's self-executing function argument is named **jQuery** to match the declared alias.

- The module's self-executing function parameter is named **\$** and thus redefines the *jquery* dependency to **\$** locally.

At runtime, the process happens in the following order of events:

- ✦ The defined function is invoked and declares its dependencies.
- ✦ Once the dependencies are resolved, the module wrapper is invoked with the **jQuery** argument containing the *jquery* dependency.
- ✦ The module wrapper evaluates the self-executing function that resolves the **jQuery** argument to the *jquery* dependency.
- ✦ The self-executing function is invoked and the *jquery* dependency is available under the name **\$**.

[Report a bug](#)

33.3. Producing Dependencies

The previous example illustrated that a module is able to consume dependencies as arguments. This section explains how a module can *produce* a dependency.

When a module self-executing function is evaluated it can return an object. You can modify the previous example to return a value.

Example 33.3. foo.js

```
(function ($) {  
    // Do something with jQuery  
    return {hello:"world"}  
})($)
```

In this second example, we return a simple JavaScript object. This object will be stored by the dependency system of AMD and provided as arguments of modules that consume the *foo* module.

[Report a bug](#)

Chapter 34. Native AMD Modules

The support for native AMD modules is added to Red Hat JBoss Portal Platform. Native AMD modules exhibit three main differences against JBoss Portal Modules:

- ✳ Simplified declaration in `gatein-resources.xml`.
- ✳ No server-side wrapping of JavaScript code.
- ✳ Shared scope only.

[Report a bug](#)

34.1. Difference in Native AMD and Portal Modules

Despite the great flexibility of Red Hat JBoss Portal Platform Modules, there are situations where they cannot provide an appropriate solution. For example, to use a large 3rd party JavaScript framework such as which complies with the AMD specification and it should be possible to declare Dojo modules as JBoss Portal Modules, but for certain reasons is not possible:

1. Large Dojo 1.9.2 contains as many as 4899 JavaScript files. You declare them in `gatein-resources.xml` using a tool. But such a tool will have to provide the dependency tree which is not trivial.
2. Extreme AMD. although AMD compliant consist of constructs, that JBoss Portal cannot handle such as:
 - a. Relative module names. If you have a dependency graph as: `dojo/sniff` depends on `dojo/has`. Given that both `dojo/sniff` and `dojo/has` live in the same directory named `dojo`, the `dojo/sniff.js` file can start with `define(["./has"], function(has){`, but relative module path names such as `"./has"` cannot be handled by JBoss Portal modules.
 - b. Cyclic dependencies occur in AMD Modules but JBoss Portal capitulates with an error when seeing them.
 - c. Inline calls such as, `require([id == 'platform' ? platformId : defId], function(provider){...});` are allowed in AMD but cannot be declared in JBoss Portal modules. There are two problems with such calls, you do not know at development time what dependency to declare in `gatein-resources.xml` and the JBoss Portal module implementation does server-side wrapping of all the modules which causes that these inline `require()` calls result in a request for a nonexistent resource.
 - d. Loader Plugins. Specific loader plug-ins are supported in JBoss Portal. In AMD, the string following the `!` separator, which is the resource ID is interpreted by the loader plug-in, so it can be interpreted incorrectly as a URL, name of another module, and so on. But the JBoss Portal module implementation is able to interpret them as resource paths.

[Report a bug](#)

34.2. Native AMD Modules hosted on CDN

Red Hat JBoss Portal supports AMD modules hosted outside of the portal, for example modules hosted on a Content Delivery Network (CDN).



Important

The code snippets used in the section are under development at ArcGis Gears Portlet hosted on GitHub: <https://github.com/ppalaga/arcgis-gears-portlet/commits/master>

This feature is implemented using **require.js** path mappings listed at, <http://requirejs.org/docs/api.html#config-paths> The implementation allows to map a module ID prefix to an URL, instead of the prefix.

[Report a bug](#)

34.3. Mapping path entries for AMD Modules

The mappings entries are declared in the **<amd>** section of the **gatein-resources.xml** file:

Example 34.1. AMD path entries in gatein-resources.xml

```
<gatein-resources xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.gatein.org/xml/ns/gatein_resources_1_5
http://www.gatein.org/xml/ns/gatein_resources_1_5"
xmlns="http://www.gatein.org/xml/ns/gatein_resources_1_5">

  <amd>
    <path-entry>
      <prefix>dojo</prefix><target-
path>//js.arcgis.com/3.9/js/dojo/dojo</target-path>
    </path-entry>
    <path-entry>
      <prefix>dijit</prefix><target-
path>//js.arcgis.com/3.9/js/dojo/dijit</target-path>
    </path-entry>
    <path-entry>
      <prefix>dojox</prefix><target-
path>//js.arcgis.com/3.9/js/dojo/dojox</target-path>
    </path-entry>
    <path-entry>
      <prefix>esri</prefix><target-
path>//js.arcgis.com/3.9/js/esri</target-path>
    </path-entry>
  </amd>
```

[Report a bug](#)

34.4. Fallback Paths

You can assign multiple target paths to a given prefix. These multiple paths are interpreted as fallback paths.

For example, you can insert a third-party CDN URL as the first choice and add a URL on your own server as a fallback URL.

Example 34.2. Adding Fallback paths to gatein-resources.xml

```
...
    <amd>
        <path-entry>
            <prefix>dojo</prefix>
            <target-path>//js.arcgis.com/3.9/js/dojo/dojo</target-
path>
            <target-path>//mycompany.com/frameworks/dojo/1.9.1</target-
path>
        </path-entry>
        ...
    </amd>
    ...
```

For more information on fallback URL, see <http://requirejs.org/docs/api.html#pathsfallbacks>

See Also:

- » [Section 13.7.1, “CSS Hosted on CDN”](#)

[Report a bug](#)

34.5. Example to Ensure Portal wide Consistency in mapping paths

There is one global path mappings collection for each Portal instance. When a portlet application is deployed on JBoss Portal, the Portal can only accept path mappings which are consistent with the paths declared earlier by other applications.

For example, application **app2** wants to register the AMD module ID prefix **/dojo** to be mapped to target path <http://fastcdn.com/dojo/1.7.0>

The prefix **/dojo** was previously registered by another application **app1** to point to a different target path <http://othercdn.com/dojo/1.9.1>.

Adding the prefix **/dojo** to the available prefixes could break **app1** and no JavaScript or CSS resources from **app2** will be accepted by the portal.

But to resolve this scenario, you can add the same mapping multiple times as shown below:

- » Application **app2** wants to register the AMD module ID prefix **/dojo** to be mapped to target path <http://fastcdn.com/dojo/1.7.0>
- » The prefix **/dojo** was previously registered by another application **app1** to point to the same target path <http://fastcdn.com/dojo/1.7.0>.

The path mapping from **app2.war** is accepted by the portal because it does not introduce any inconsistency.

[Report a bug](#)

Chapter 35. Module Scopes

The name of a logical portal module translates into an AMD name with a prefix of **SHARED**, **PORTLET** or **PORTAL**. This prefix fully identifies a module by its logical name and its scope. Scopes ensure that a module (and therefore the underlying web resource) is loaded at the right time when it is effectively required.

[Report a bug](#)

35.1. Shared Scope

The shared scope does not related to a specific portal entity, instead a shared module should be consumed by other modules. It is declared in **gatein-resources.xml** with the top level **module** tag:

```
<module>
  <name>core</name>
  <script>
    <path>/core.js</path>
  </script>
  <depends>
    <module>base</module>
  </depends>
</module>
```

[Report a bug](#)

35.2. Portal Scope

The module is related to a portal instance and is loaded when the related portal is accessed:

```
<portal>
  <name>classic</name>
  <module>
    <script>
      <path>/classic.js</path>
    </script>
    <depends>
      <module>core</module>
    </depends>
  </module>
</portal>
```

[Report a bug](#)

35.3. Portlet Scope

The module is loaded when the corresponding portlet is accessed:

```
<portlet>
  <name>sitemap</name>
  <module>
    <script>
      <path>/sitemap.js</path>
    </script>
    <depends>
      <module>core</module>
    </depends>
  </module>
</portlet>
```



Note

A module can only depend on a shared module, therefore any **depends** tags implicitly reference a shared module.

[Report a bug](#)

Chapter 36. Portlet Dynamic Module

As explained previously, portlet dependencies (including JavaScript files) can be declared in the **gatein-resources.xml** file, which forces declaration of portlet dependencies at packaging time when the corresponding war file is created.

The JSR286 specification provides a mechanism for modifying portal headers, which can also be used to load dependency JavaScript files. Although this mechanism is portable, it has the following drawbacks:

- ✧ A script can be loaded multiple times, specially if two portlets in two different war files load the same scripts since the only way to identify a script is by its URL.
- ✧ Scripts have to be loaded by the head section of the portal, impacting front-end performance.

The portal combines both approaches by allowing to create dynamic dependencies of a portlet at runtime, during the render phase of the portlet:

```
public void render(RenderRequest req, RenderResponse resp) throws
PortletException, IOException {
    resp.setProperty("org.gatein.javascript.dependency", "base");
    resp.addProperty("org.gatein.javascript.dependency", "common");
}
```

The code above is equivalent to the following declaration in the **gatein-resources.xml** file:

```
<portlet>
  <name>MyPortlet</name>
  <module>
    <depends>
      <module>base</module>
    </depends>
    <depends>
      <module>common</module>
    </depends>
  </module>
</portlet>
```

[Report a bug](#)

Chapter 37. Aliases

It is possible to define an alias for each module declared in the **gatein-resources.xml** file. The alias is added to the module declaration using the **<as>** element:

```
<module>
  <name>foo</name>
  <as>Foo</as>
  ...
</module>
<module>
  <name>bar</name>
  ...
  <depends>
    <module>foo</module>
  </depends>
</module>
```

The alias can then be used instead of the module's name in the argument of the respective self-executing JavaScript function:

```
(function(Foo) {
})(Foo)
```

Similarly, the **<as>** element can also be used inside the **<depends>** element to define a local alias for the dependency:

```
<module>
  <name>foo</name>
  ...
</module>
<module>
  <name>bar</name>
  ...
  <depends>
    <module>foo</module>
    <as>Foo</as>
  </depends>
</module>
```

[Report a bug](#)

Chapter 38. Custom Adapters

Until ECMAScript 6, there had been no standardized format of a JavaScript module declaration. However, several proposals of such a format have emerged, all revolving around the same module pattern. The AMD (Asynchronous Module Definition) format was chosen because it is suitable for the Web, and its asynchronous nature.

As a consequence, you sometimes have to deal with included scripts that do not match the self-executing function declaration format. To adapt such scripts to the expected format without editing the actual code, it is possible to declare an adapter that will wrap the original scripts and make them reusable.

The jQuery library is a good example of how a custom adapter is useful. Thanks to the adapter, it is possible to reuse the jQuery library without any modifications, which facilitates the integration and maintenance of the library in the portal. jQuery uses the following construct to define itself:

```
(function(window, undefined) {
})(window)
```

The issue with this construct is that it will bind jQuery to the window, and that it will not return any value as expected by the dependency system. Thanks to the custom adapter, we can integrate JQuery the following way:

```
<module>
  <name>jquery</name>
  <script>
    <adapter>
      (function() {
        <include>/jquery.js</include>
        return jQuery.noConflict(true);
     })();
    </adapter>
  </script>
</module>
```

The **<adapter>** element can encapsulate any code, and the **<include>** element will perform a simple string inclusion of the original jQuery script:

```
define("SHARED/jquery", [], function() {
  return (function() {
    (function(window, undefined) {
    })(window);
    return jQuery.noConflict(true);
  })();
});
```

[Report a bug](#)

Chapter 39. Module Resources

The AMD format allows to define dependencies on a resource loaded by a module based on the [loader plug-in](#).

A module that uses a loader plug-in uses the exclamation mark character (!) to delineate between the loader plug-in module to use and the target resource to load. Loader plug-ins are useful because they can load resources by means of the AMD loading mechanism and benefit from its performance and parallel loading.

There are some useful loader plug-ins:

- ✧ The <https://github.com/requirejs/text> plug-in for loading of text resources such as stylesheets or templates
- ✧ The <http://requirejs.org/docs/api.html#18n> plug-in for internationalization bundle support

Procedure 39.1. Configuring a Module Resource

To demonstrate the configuration of an AMD loader plug-in's target resource, we need some HTML code generated by JavaScript. The **text** AMD loader plug-in can be used for this purpose. The plug-in will load a template as its resource and pass it as a parameter into a module definition callback.

1. Declare **text** as a module. It is written as a native module that depends on the predefined RequireJS module called **module**. The native support mechanism allows this configuration works transparently without modifying the third-party library.

```
<module>
  <name>text</name>
  <script>
    <path>/requirejs/js/plugins/text.js</path>
  </script>
  <depends>
    <!-- module means RequireJS itself -->
    <module>module</module>
  </depends>
</module>
```

2. Define the **foo** module that requires the **text** plug-in to load the template by means of the **<resource>** XML element.

```
<module>
  <name>foo</name>
  ...
  <depends>
    <module>text</module>
    <as>myTemplate</as>
    <resource>/path/to/template.tpl</resource>
  </depends>
</module>
```

3. The **foo** module template is loaded by the **text** plug-in.

```
(function(myTemplate) {  
  //parse the 'myTemplate' then append to DOM  
})(myTemplate);
```

[Report a bug](#)

Chapter 40. Load Groups

When self-executing functions and XML descriptors are loaded by a browser, they are served as web resources by the portal. By default, a module is served as a single resource, which can cause performance issues in production systems. The load group feature solves these issues by decoupling the logical modules and the JavaScript resource serving.

The **<load-group>** element can be used to group modules into a single web resource. When a module contained in a load group is requested, the web resource containing all the grouped modules is loaded. For instance, the code below shows configuration of the **foo** and **bar** modules grouped into the **foobar** load group:

Example 40.1. load-group Example

```
<module>
  <name>foo</name>
  <load-group>foobar</load-group>
  ...
</module>
<module>
  <name>bar</name>
  <load-group>foobar</load-group>
  ...
</module>
```

When the **foo** module is requested, the AMD loader will load the whole **foobar** group. This will cause that the **bar** module will be loaded as well. On any subsequent request for the **bar** module, no more load requests will be made as the module will already have been loaded.

[Report a bug](#)

Chapter 41. Localization

The portal can localize JavaScript modules on the server. The JavaScript code of each module can be rewritten and localization keys replaced by values from resource bundles for individual locales. Each localized script is available under a different URL, which ensures optimal resource caching.

Script localization is activated by the **<resource-bundle>** and **<path>** elements. Resource bundles are declared by the **<supported-locale>** element and must be available in the web application's classpath.

Example 41.1. supported-locale Example

```
<module>
  <name>greetings</name>
  <supported-locale>de</supported-locale>
  <supported-locale>en</supported-locale>
  <supported-locale>fr</supported-locale>
  <supported-locale>vi</supported-locale>
  <supported-locale>ru</supported-locale>
  <script>
    <path>/path/to/greetings.js</path>
    <resource-bundle>greetings_bundle</resource-bundle>
  </script>
  ...
</module>
```

The portal escapes all keys in the **\${key}** format and looks up its value in the resource bundles. For instance, a JSON object containing localized values can be built by the following code:

```
(function() {
  return {
    "hello": "${hello}",
    "goodbye": "${goodbye}"
  };
})();
```

At runtime, different versions of this script will be served by the portal, with the keys replaced by the respective values obtained from the localization bundles.

[Report a bug](#)

Chapter 42. Non-AMD Scripts

Whenever possible, JavaScript code should be written as an AMD module. Alternatively, the custom adapter mechanism can be used to adapt scripts to the AMD format.

While this is the recommended practice, it may sometimes be necessary to load scripts as traditional non-AMD modules. This approach is possible, providing the following conditions are met:

- Traditional scripts must be declared using the **<scripts>** element instead of **<module>**.
- Dependencies can be declared using the **<depends>** element, but the scripts can only depend on other scripts, not on modules.

This is also true for modules: a module can only depend on another module, not on a script.

- JavaScript code declared in the **<scripts>** element is pushed to the HTML head. This means the script will be loaded and executed before a DOM is created, and will likely have an impact on front-end performance.

```
<scripts>
  <name>foo</name>
  <script>
    <path>/path/to/foo.js</path>
  </script>
  <depends>
    <scripts>bar</scripts>
  <depends>
</scripts>
```

It is also possible to use a script that is available at a public URL. This can be achieved using the **<url>** element instead of **<script>**:

```
<scripts>
  <name>foo</name>
  <url>http://path.to/foo.js</url>
</scripts>
```

See Also:

- [Chapter 38, Custom Adapters](#)

[Report a bug](#)

Part VI. JavaScript Code Examples

Chapter 43. Module Code Examples

[Report a bug](#)

43.1. Declare a Module

Module declaration requires integrating the [Highlight.js](#) library. This example is part of the examples found in `amd-js.war`.

Highlight.js is a jQuery plugin. jQuery plugins are the best examples as they are native portal modules. They follow the self-invoking pattern that accepts the *jquery* dependency as `$`.

```
(function($) {  
    ...  
})(jQuery)
```



Note

For an overview of the *Highlight.js* source code, see <https://github.com/isagalaev/highlight.js/blob/master/src/highlight.js>.

The highlight module is integrated using the XML declaration:

```
<module>  
  <name>highlight</name>  
  <script>  
    <path>/highlight/highlight.js</path>  
  </script>  
  <depends>  
    <module>jquery</module>  
    <as>jQuery</as>  
  </depends>  
</module>
```

- ✧ The module **highlight** uses the `/highlight/highlight.js` source code bundled in the war file.
- ✧ The **depends** tag creates a dependency on the *jquery* module. The dependency is aliased as `$` using the **as** tag to match the **Highlight.js** self-executing function argument `$`.



Note

The *jquery* module provides jQuery 1.7.1. The *jquery* plug-ins are not a part of the portal and do not need to be declared.

[Report a bug](#)

43.2. Declare an AMD Module

Asynchronous module declaration is explained in the RequireJS documentation available at <http://requirejs.org/docs/api.html#define>.

The pattern for AMD modules is as follows:

```
define("module", ["dependency1", ..., "dependencyN"],
function(dep1, ..., depN) {
});
```

The portal allows independent usage of modules such as AMD, where some parts are overridden by the XML declaration:

- ✧ The **"module"** name is ignored and replaced by the declared module name.
- ✧ The module dependencies ranging from **"dependency1"** to **"dependencyN"** are declared with the same name in XML (use the **as** tag to override the dependency name).

After declaring the dependencies from **dependency1** to **dependencyN** in XML, the module definition is declared as follows:

```
<module>
  <name>MyModule</name>
  ...
  <depends>
    <module>dependency1</module>
  </depends>
  ...
  <depends>
    <module>dependencyN</module>
  </depends>
</module>
```

See Also:

- ✧ [Section 43.1, "Declare a Module"](#)

[Report a bug](#)

43.3. Use jQuery Module

The portal provides the jQuery library as a *jquery* module. The module configuration is found in **eXoResources.war** file.

- ✧ To reuse the jQuery version you need to declare a dependency over it.

```
<portlet>
  <name>RequireJSPortlet</name>
  <module>
```

```
<depends>
  <module>jquery</module>
</depends>
</portlet>
```

- ✱ The default *jquery* module alias is **\$** so while using it, it should be named **\$** in the self-executing function:

```
(function($) {
  ...
})($);
```

- ✱ The XML `<as>` tag is required if your library uses a different name such as **jQuery**:

```
<portlet>
  <name>RequireJSPortlet</name>
  <module>
    <depends>
      <module>jquery</module>
      <as>jQuery</as>
    </depends>
  </module>
</portlet>
```

- ✱ The self-executing function for jQuery is as follows:

```
(function($) {
  ...
})(jQuery);
```

[Report a bug](#)

43.4. Use a Custom jQuery Version

The portal allows integration of other version of jQuery. Products built to run on the Red Hat JBoss Portal may depend on third-party JavaScript frameworks, which in turn depend on other versions of jQuery libraries. For these reasons, deploying other jQuery libraries is sometimes unavoidable. Multiple jQuery instances within a web page conflict over global variables. But the module system allows to use such a library.

For example, a *jQueryPortlet* uses jQuery version 1.6.4 to configure two modules.

```
<module>
  <name>jquery-1.6.4</name>
  <script>
    <adapter>
      (function() {
        <include>/javascript/jquery-1.6.4.js</include>
        return jQuery.noConflict(true);
      })();
    </adapter>
  </script>
</module>
```

```

</script>
</module>
<portlet>
  <name>jQueryPortlet</name>
  ...
  <depends>
    <module>jquery-1.6.4</module>
  </depends>
</portlet>

```

[Report a bug](#)

43.5. jQuery Plug-ins

To configure a jQuery plugin and use it in jQueryPluginPortlet portlet.

For example, the plugin code is a minimal plugin:

Example 43.1. "jQuery plugin"

```

(function($) {
  $.fn.doesPluginWork = function() {
    alert('YES, it works!');
  };
})(jQuery);

```

- ✦ The plugin is then declared as a module:

```

<module>
  <name>jquery-plugin</name>
  <as>jqPlugin</as>
  <script>
    <path>/jqueryPlugin/jquery-plugin.js</path>
  </script>
  <depends>
    <module>jquery</module>
    <as>jQuery</as>
  </depends>
</module>

```

- ✦ This plugin is ready for use in your portlet:

```

<portlet>
  <name>jQueryPluginPortlet</name>
  <module>
    <script>
      <path>/jqueryPlugin/jqueryPluginPortlet.js</path>
    </script>
    <depends>

```

```

    <module>jquery</module>
    <as>$</as>
  </depends>
  <depends>
    <module>jquery-plugin</module>
  </depends>
</module>
</portlet>

```

Your portlet module should depend on the *jquery* and the *jquery-plugin* modules even if it does not use the plugin itself because:

- » Declaring the dependency on *jquery* allows you to use the jQuery object.
- » Declaring the dependency on *jquery-plugin* ensures that the plugin is loaded in the *jquery* dependency before it is injected in the portlet module.

[Report a bug](#)

43.6. Override Native AMD Module Dependency

In case of a mismatch between a module declared in the native module and the module system of the portal, the **as** tag is used to rename the dependencies.

For example, a **foo.js** file defines an AMD module named **foo** with two dependencies "**dep1**", "**dep2**":

```

define("foo", ["dep1", "dep2"], function(a1, a2) {
  // The module
});

```

The dependencies are declared as *module1* and *module2* in the portal, with mismatch in names. To override this, use the **as** tag and rename the dependencies:

```

<module>
  <name>foo</name>
  <script>
    <path>/path/to/foo.js</path>
  </script>
  <depends>
    <module>module1</module>
    <as>dep1</as>
  </depends>
  <depends>
    <module>module2</module>
    <as>dep2</as>
  </depends>
</module>

```

[Report a bug](#)

43.7. CommonJS modules

CommonJS defines its own module format. See the CommonJS wiki for detailed information, which is available at <http://wiki.commonjs.org/wiki/Modules/1.1>. CommonJS module is not supported natively by the Red Hat JBoss Portal, therefore the **adapter** format is used to adapt CommonJS modules to work in the portal.

Two simple CommonJS modules are as follows:

Example 43.2. math.js

```
exports.add = function() {
  var sum = 0, i = 0, args = arguments, l = args.length;
  while (i < l) {
    sum += args[i++];
  }
  return sum;
};
```

Example 43.3. increment.js

```
var add = require('math').add;
exports.inc = function(val) {
  return add(val, 1);
};
```

CommonJS modules uses the **require** function which conflicts with the same function in RequireJS. To resolve the conflict in AMD enabled environment, these modules are wrapped and injected to *predefined* modules such as *require*, *exports*, and *module* provided by RequireJS.

The portal wraps the code using the <adapter> format as per the configuration:

```
<module>
  <name>math</name>
  <script>
    <adapter>
      define(["require", "exports"], function(require, exports) {
        <include>/commonjs/math.js</include>
      });
    </adapter>
  </script>
  <depends>
    <module>require</module>
  </depends>
  <depends>
    <module>exports</module>
  </depends>
</module>
<module>
  <name>increment</name>
  <script>
```

```

<adapter>
  define(["require", "exports", "math"], function(require, exports) {
    <include>/commonjs/increment.js</include>
  });
</adapter>
</script>
<depends>
  <module>require</module>
</depends>
<depends>
  <module>exports</module>
</depends>
<depends>
  <module>math</module>
</depends>
</module>

```

For more information related to modules, see the RequireJS.org documentation available at <http://requirejs.org/docs/commonjs.html>.

[Report a bug](#)

43.8. Mustache.js module

[Mustache.js](#) is a popular JavaScript template engine. Mustache can be executed in different environments such as a global object, as a CommonJS module, or as a native AMD module.

If "module" and "exports" dependencies are available, Mustache registers it as a CommonJS module and it is adapted for the portal as per the adapter format as follows:

```

<module>
  <name>mustache</name>
  <script>
    <adapter>
      define(["require", "exports", "module"], function(require, exports,
        module) {
        <include>/requirejs/js/plugins/mustache.js</include>
      });
    </adapter>
  </script>
  <depends>
    <module>require</module>
  </depends>
  <depends>
    <module>exports</module>
  </depends>
  <depends>
    <module>module</module>
  </depends>
</module>

```

The **adapter** tag is used to declare *require*, *exports*, and *module* dependencies of the CommonJS module. Any module can have Mustache instance injected by declaring it in its dependencies list as follows:


```
<module>
  <name>foo</name>
  ...
  <depends>
    <module>mustache</module>
  </depends>
</module>
```

```
(function(mustache){
  //code that use Mustache
  mustache.render(template);
})(mustache);
```

[Report a bug](#)

43.9. Resource loading with Text.js

RequireJS supports loader plugin which allows a module to work as a plugin and use the AMD system for loading web resources efficiently.

Mustache.js is a javascript template engine. This engine need a template file to parse and generate html. In case of many templates or the template size is large, embedding a template in the page is not a good practice for front-end performance reason. In this case, use **Text.js** to load the separate template files and inject them as dependencies.

Text.js is a native AMD module, which depends on the *module* predefined dependency provided by the AMD loader. Due to the native AMD support in the portal, it is easy to declare and use Text.js:

```
<module>
  <name>text</name>
  <script>
    <path>/requirejs/js/plugins/text.js</path>
  </script>
  <depends>
    <module>module</module>
  </depends>
</module>
```

Use *mustache* and *text* modules to load templates and render them in your own module:

```
<portlet>
  <name>foo</name>
  <module>
    ...
    <depends>
      <module>mustache</module>
    </depends>
    <depends>
      <module>text</module>
      <as>tpl</as>
```

```
        <resource>/path/to/template.html</resource>
    </depends>
</module>
</portlet>
```

The *text* module is in the dependency list with a `<resource>` tag. **Text.js** loads that resource template and inject it with the name **tmpl**. The javascript of the portlet is as follows:

```
function(mustache, tmpl) {
    var html = mustache.render(tmpl);
    //append rendered html to DOM
})(mustache, tmpl);
```

[Report a bug](#)

Chapter 44. Script Code Examples

[Report a bug](#)

44.1. Accessing a module from a script

To access a module from a script, RequireJS uses the **require** function to execute a function in the managed context:

```
require(["SHARED/ModuleA"], function(a) {
  // Codes of interacting with module A
  a.doSomething();
});
```

As shown above, use the AMD module name of the module that depend on, **PORTLET/ModuleA**. The prefix in upper case is the module scope which has a value **SHARED**, **PORTLET**, or **PORTAL**.

[Report a bug](#)

44.2. Exposing jQuery Version Globally

The built-in jQuery is declared as an AMD module. By default jQuery is not available in the window object of the browser. This example shows how to make it available and allow to write code in a plain script.

The following script makes jQuery available by mounting the jQuery object in the window object:

```
require( ["SHARED/jquery"], function($) {
  // the '$' in window.$ is alias, you can make the other for yourself.
  window.$ = $;
});
```

This script must be integrated as a shared script:

```
<scripts>
<name>immediatejs</name>
<script>
<path>/myfolder/immediateJScript.js</path>
</script>
</scripts>
```

A portlet then provide its own script that depends on the script written above:

```
<portlet>
<name>foo</name>
<script>
<name>portletjs</name>
<path>/myfolder/portlet.js</path>
```

```

</script>
<depends>
<scripts>immediatejs</scripts>
</depends>
</scripts>

```

The JavaScript is as follows:

```
$("#foo").html("<h1>hello global jQuery</h1>");
```

[Report a bug](#)

44.3. Defining a custom global jQuery

Define a globally-available custom jQuery by reusing existing code, and combine them by:

- ✱ Using a custom jQuery version.
- ✱ Exposing the portal version of jQuery globally.

[Report a bug](#)

44.4. Using a Global jQuery plugin

To implement this example the global jQuery is available should be available before the global jQuery plugin is loaded.

Now, you know scoping a module to a portlet and that the module is loaded when the portlet is on a page using the **PORTLET** scope.

In this example change the scope and use **PORTAL**. The main difference is that the loading of your plugin is triggered on a specific portal instead of a specific portlet.

Procedure 44.1. Using a global jQuery plugin

1. Create a jQuery plugin as a script named **myPlugin.js** and integrate your plugin.

```

require(["SHARED/jquery"], function($) {
$.fn.myPluginFunction = function() {
// Your work here;
};
});

```

2. Bind the script in the portal and reuse the **immediate.js** script seen before:

```

<portal>
<name>classic</name>
<scripts>
<script>
  <name>myPlugin</name>
  <path>/myfolder/myPlugin.js</path>

```

```
</script>
<script>
  <name>immediatejs</name>
  <path>/myfolder/immediateJScript.js</path>
</script>
</scripts>
</portal>
```

3. The plugin is globally available to use:

```
<script type="text/javascript">
$('#foo').myPluginFunction();
</script>
```

[Report a bug](#)

Part VII. Advanced Development Concepts

The eXo Kernel

A.1. The eXo Kernel

The Red Hat JBoss Portal is built as a set of services on top of a dependency injection kernel. The kernel provides configuration, life-cycle handling, component scopes and some core services.

The portal kernel and the JCR used in the portal use the *Inversion of Control (IoC)* system to control service instantiation. This method is also known as 'dependency injection'.

In this system, services are not responsible for the instantiation of the components on which they depend. It prevents objects creating the instances of any objects referenced. This task is delegated to the container, as described in [Figure A.1, “Inversion of Control”](#):

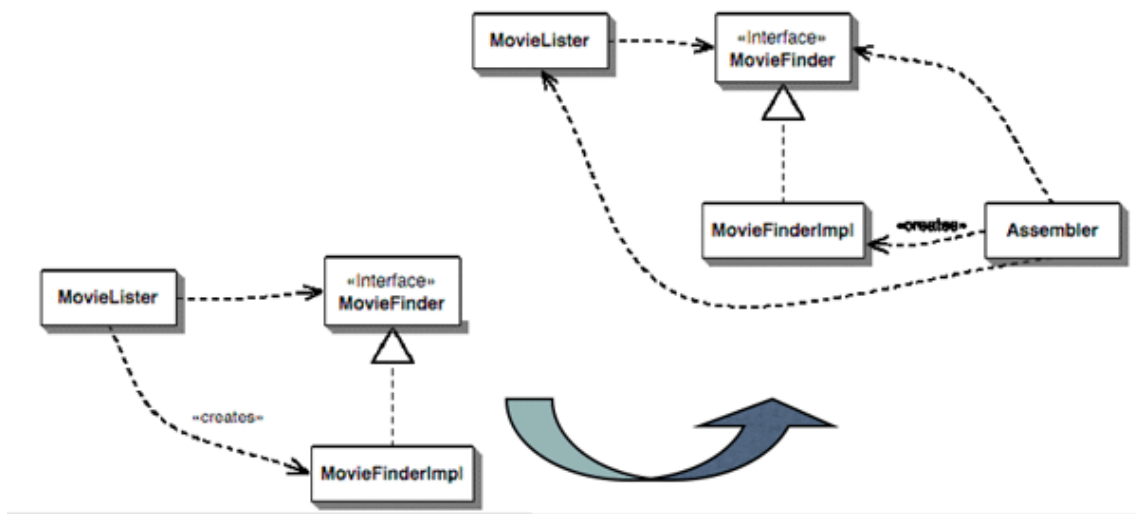


Figure A.1. Inversion of Control

There are two ways to inject a dependency :

Using a constructor:

```
public ServiceA(ServiceB serviceB)
```

Using setter methods:

```
public void setServiceB(ServiceB serviceB)
```

When a client service can not be stored in the container then the service locator pattern is used:

```
public ServiceA(){
    this.serviceB =Container.getInstance().getService(ServiceB.class);
}
```

The container package is responsible for building a hierarchy of containers. Each service is registered in a container according to the XML configuration file it is defined in. There can be several PortalContainer instances that all are children of the RootContainer.

The behavior of the hierarchy is similar to class loader behavior. When a portal application is

integrated with eXo services, these services are looked up through the **PortalContainer**. If the service cannot be found, then the class loader looks in the parent container. This means the reusable business logic components can be loaded in the same container (here the **RootContainer**) and differentiates the service implementation from one portal instance to the other by loading different service implementations in two sibling **PortalContainers**.

If the **Portal Container** is thought of as a service repository for all the business logic in a portal instance, it is easier to understand why several **PortalContainers** allows several portals to be managed (each one deployed as a single war) in the same server by changing XML configuration files.

The default configuration XML files are packaged in the service Java Archive (JAR). There are three **configuration.xml** files: one for each container type. This file defines the list of services and their init parameters that are loaded in the corresponding container.

Service components exist in two scopes:

- ✦ **RootContainer**, which is a base container and contains services that exist independently of any portal. The **RootContainer** can be accessed by all portals. This container plays an important role during startup, but must not be used directly.
- ✦ **PortalContainer**, in which each portal exists. This scope contains services that are common for a set of portals, and services which must remain unique to each portal instance. This service component is created when a portal web application (in the `init()` method of the **PortalController** servlet) starts.

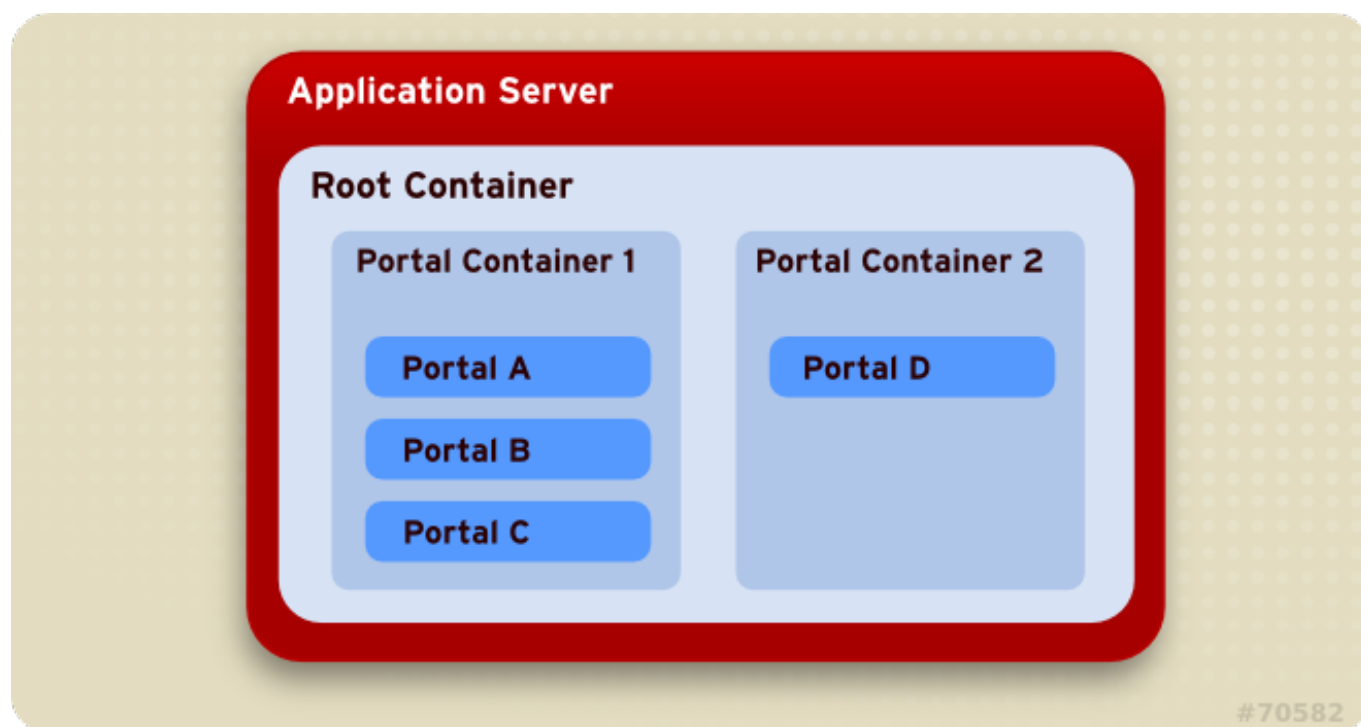


Figure A.2. Portal Containers

Because several portal container instances can exist for each JVM, configuring loaded services for each instance is an important feature. All the default configuration files located in the service impl JAR can be overridden from the **portal.war**.

Whenever a specific service is looked up through the **PortalContainer**, and the service is not available, the lookup is delegated further up to the **RootContainer**.

The portal can have default instances of a certain component in the **RootContainer**, and portal specific instances in some or all **PortalContainers**, that override the default instance.

Whenever a portal application has to be integrated more closely with eXo services, these services can be looked up through the **PortalContainer**.



Important

Only officially documented services should be accessed this way, and used according to their supporting documentation. Most services are an implementation detail of eXo, and subject to change without notice.

[Report a bug](#)

A.2. Kernel Configuration Namespace

To be effective, the namespace URI **`http://www.exoplatform.org/xml/ns/kernel_1_2.xsd`** must be the target namespace of the XML configuration file.

```
<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd
http://www.exoplatform.org/xml/ns/kernel_1_2.xsd"
xmlns="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd">

...
</configuration>
```

The eXo Kernel will resolve variables declared in configuration files. For example, the following configuration would be interpreted correctly:

```
<configuration
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd
http://www.exoplatform.org/xml/ns/kernel_1_2.xsd"
xmlns="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd">

<import>${db.configuration.path}/db.xml</import>
<import>${java.io.tmpdir}/bindfile.xml</import>
<import>simple.xml</import>

</configuration>
```

Supported variables include System properties, and variables specific to the portal container. Variables are described in the proceeding sections.

[Report a bug](#)

A.3. Configuration Retrieval

The container performs the following steps for configuration retrieval, depending on the container type.

This container is used by non-portal applications, and configurations are overloaded in the following lookup sequence:

1. Services default **RootContainer** configurations from JAR files **/conf/configuration.xml**.
2. Services default **PortalContainer** configurations from JAR files **/conf/portal/configuration.xml**.
3. Web applications configurations from WAR files **/WEB-INF/conf/configuration.xml**
4. Configuration URL for **PortalContainer**, based on the following caveats:
 - If the configuration URL was initialized to be added to services defaults using **PortalContainer.addConfigurationURL(containerConf)**; the configuration from the containerConf overrides services configured in the file only.
 - The **\$JPP_HOME/gatein/gatein.ear/portal.war/WEB-INF/conf/configuration.xml** is the only path that states the correct location for the configuration.

The search looks for a configuration file in each JAR/WAR available from the classpath using the current thread context classloader. During the search these configurations are added to a set.

If the service was configured previously and the current JAR contains a new configuration of that service, the latest (from the current JAR/WAR) will replace the previous one. The last one will be applied to the service during the services start phase.



Warning

No dependencies between JAR files contained in **/conf/portal/configuration.xml** and **/conf/configuration.xml** are loaded because there is no way of determining the JAR loading order prior to start.

To overload configuration information located in these files of a JAR file, load from another configuration file that is loaded after configurations from JAR files (for example, **/conf/portal/configuration.xml** in the portal itself).

After all configuration available in the system is processed, the container will initialize and start each service in order of the dependency injection (DI).



Important

Be careful when configuring the same service in different configuration files.

To ensure service configuration is loaded as expected, configure a service within its own JAR.

In the case of a portal configuration, strictly reconfigure the services in portal WAR files, or from an external configuration file.

There are services that are configured more than once, depending on the business logic of the service. A service may initialize the same resource (shared with other services), or may add a particular object to a set of objects (also shared with other services). For services that initialize the same resource, it is critical to determine which service configuration will be used (the last one to be loaded). For services that add objects to a set of objects, the initialization order is not important providing the parameter objects are independent.

In case of problems with service configuration, it is important to know from which JAR/WAR the configuration originates. The JVM system property

org.exoplatform.container.configuration.debug can be used to identify this information.

```
java -Dorg.exoplatform.container.configuration.debug ...
```

If the property is enabled, the container configuration manager logs the configuration adding process at *INFO* level.

The effective configuration of the `StandaloneContainer`, `RootContainer` and/or `PortalContainer` can be shown using the **getConfigurationXML()** method that is exposed through JMX at the container level. This method exposes the effective configuration, in XML format, that the kernel interpreted. This information could be helpful when analyzing how a given component or plug-in has been initialized.

[Report a bug](#)

A.4. PortalContainer Advanced Concepts

Since eXo JCR 1.12, a new set of features are available that extend portal applications.

[Report a bug](#)

A.4.1. Add new configuration files from a WAR file

A `ServletContextListener` called

org.exoplatform.container.web.PortalContainerConfigOwner notifies the application that a web application provides some configuration to the portal container, and this configuration file is the file `WEB-INF/conf/configuration.xml` available in the web application itself.

If the WAR file contains configuration to add to the **PortalContainer** add the following lines to the `web.xml` file.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!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>
  ...
  <!--
===== -->
  <!--          LISTENER
-->
  <!--
===== -->
  <listener>
    <listener-
```

```

class>org.exoplatform.container.web.PortalContainerConfigOwner</listener-
class>
  </listener>
...
</web-app>

```

[Report a bug](#)

A.4.2. Creating PortalContainers from a WAR File



Important

In Red Hat JBoss Portal, the `PortalContainerCreator` is already managed by the file `starter.war/ear`.

A `ServletContextListener` called `org.exoplatform.container.web.PortalContainerCreator` creates the current portal containers that have been registered. It is assumed that the web applications have already been loaded before calling `PortalContainerCreator.contextInitialized`.

[Report a bug](#)

A.4.3. Defining a PortalContainer with its dependencies and its settings

The **PortalContainerDefinition** contains the name of the portal container, the name of the rest context, the name of the realm, the web application dependencies ordered by loading priority, and the settings.

Example A.1. Defining a PortalContainerDefinition

Define a **PortalContainerConfig** at the **RootContainer** level to define a **PortalContainerDefinition**.

```

<component>
  <!-- The full qualified name of the PortalContainerConfig -->

  <type>org.exoplatform.container.definition.PortalContainerConfig</type>
  <
    <init-params>
      <!-- The name of the default portal container -->
      <value-param>
        <name>default.portal.container</name>
        <value>myPortal</value>
      </value-param>
      <!-- The name of the default rest ServletContext -->
      <value-param>
        <name>default.rest.context</name>
        <value>myRest</value>
      </value-param>
      <!-- The name of the default realm -->
      <value-param>
        <name>default.realm.name</name>

```

```

        <value>my-exo-domain</value>
    </value-param>
    <!-- Indicates whether the unregistered webapps have to be
ignored -->
    <value-param>
        <name>ignore.unregistered.webapp</name>
        <value>true</value>
    </value-param>
    <!-- The default portal container definition -->
    <!-- It can be used to avoid duplicating configuration -->
    <object-param>
        <name>default.portal.definition</name>
        <object
type="org.exoplatform.container.definition.PortalContainerDefinition">
            <!-- All the dependencies of the portal container ordered by
loading priority -->
            <field name="dependencies">
                <collection type="java.util.ArrayList">
                    <value>
                        <string>foo</string>
                    </value>
                    <value>
                        <string>foo2</string>
                    </value>
                    <value>
                        <string>foo3</string>
                    </value>
                </collection>
            </field>
            <!-- A map of settings tied to the default portal container -
->
            <field name="settings">
                <map type="java.util.HashMap">
                    <entry>
                        <key>
                            <string>foo5</string>
                        </key>
                        <value>
                            <string>value</string>
                        </value>
                    </entry>
                    <entry>
                        <key>
                            <string>string</string>
                        </key>
                        <value>
                            <string>value0</string>
                        </value>
                    </entry>
                    <entry>
                        <key>
                            <string>int</string>
                        </key>
                        <value>
                            <int>100</int>
                        </value>

```

```

        </entry>
      </map>
    </field>
    <!-- The path to the external properties file -->
    <field name="externalSettingsPath">

<string>classpath:/org/exoplatform/container/definition/default-
settings.properties</string>
    </field>
  </object>
</object-param>
</init-params>
</component>

```

PortalContainerConfig Parameters



Note

Parameter values marked with a (*) can be defined through System properties like any values in configuration files, and variables loaded by the *PropertyConfigurator*. For example, in the portal, it would be all the variables defined in the file *configuration.properties* by default.

default.portal.container (*)

The name of the default portal container. This field is optional.

default.rest.context (*)

The name of the default rest **ServletContext**. This field is optional.

default.realm.name (*)

The name of the default realm. This field is optional.

ignore.unregistered.webapp (*)

Indicates whether the unregistered webapps have to be ignored.

This field is optional and is set to *false* by default.

If a webapp has not been registered as a dependency of any portal container, the application will use the value of this parameter to determine how to classify the webapp:

- ✳ If set to *false*, the webapp will be considered by default as a dependency of all the portal containers.
- ✳ If set to *true*, the webapp will not be considered by default as a dependency of any portal container, and is ignored.

default.portal.definition

The definition of the default portal container. This field is optional. The expected type is **org.exoplatform.container.definition.PortalContainerDefinition** that is described below. Allow the parameters defined in this default **PortalContainerDefinition** will be the default values.

Example A.2. Defining a PortalContainerDefinition at the RootContainer Level Using

A new **PortalContainerDefinition** can be defined at the **RootContainer** level using an external plug-in:

```
<external-component-plugins>
  <!-- The full qualified name of the PortalContainerConfig -->
  <target-
component>org.exoplatform.container.definition.PortalContainerConfig</
target-component>
  <component-plugin>
    <!-- The name of the plugin -->
    <name>Add PortalContainer Definitions</name>
    <!-- The name of the method to call on the PortalContainerConfig
in order to register the PortalContainerDefinitions -->
    <set-method>registerPlugin</set-method>
    <!-- The full qualified name of the
PortalContainerDefinitionPlugin -->

<type>org.exoplatform.container.definition.PortalContainerDefinitionPl
ugin</type>
  <init-params>
    <object-param>
      <name>portal</name>
      <object
type="org.exoplatform.container.definition.PortalContainerDefinition">
        <!-- The name of the portal container -->
        <field name="name">
          <string>myPortal</string>
        </field>
        <!-- The name of the context name of the rest web
application -->
        <field name="restContextName">
          <string>myRest</string>
        </field>
        <!-- The name of the realm -->
        <field name="realmName">
          <string>my-domain</string>
        </field>
        <!-- All the dependencies of the portal container ordered
by loading priority -->
        <field name="dependencies">
          <collection type="java.util.ArrayList">
            <value>
              <string>foo</string>
            </value>
            <value>
              <string>foo2</string>
            </value>
            <value>
              <string>foo3</string>
            </value>
          </collection>
        </field>
        <!-- A map of settings tied to the portal container -->
        <field name="settings">
```

```

        <map type="java.util.HashMap">
          <entry>
            <key>
              <string>foo</string>
            </key>
            <value>
              <string>value</string>
            </value>
          </entry>
          <entry>
            <key>
              <string>int</string>
            </key>
            <value>
              <int>10</int>
            </value>
          </entry>
          <entry>
            <key>
              <string>long</string>
            </key>
            <value>
              <long>10</long>
            </value>
          </entry>
          <entry>
            <key>
              <string>double</string>
            </key>
            <value>
              <double>10</double>
            </value>
          </entry>
          <entry>
            <key>
              <string>boolean</string>
            </key>
            <value>
              <boolean>true</boolean>
            </value>
          </entry>
        </map>
      </field>
      <!-- The path to the external properties file -->
      <field name="externalSettingsPath">
        <string>classpath:/org/exoplatform/container/definition/settings.properties</string>
      </field>
    </object>
  </object-param>
</init-params>
</component-plugin>
</external-component-plugins>

```

PortalContainerDefinition Parameter Values

name (*)

The name of the portal container. This field is mandatory .

restContextName (*)

The name of the context name of the rest web application. This field is optional. The default value will be defined at the **PortalContainerConfig** level.

realmName (*)

The name of the realm. This field is optional. The default value will be defined at the **PortalContainerConfig** level.

dependencies

All the dependencies of the portal container ordered by loading priority. This field is optional. The default value will be defined at the **PortalContainerConfig** level.

The dependencies are in fact the list of the context names of the web applications from which the portal container depends on. The dependency order is crucial because it will be interpreted the same way by several components of the platform. All components will consider the first element in the list less important than the second element and so on.

Dependency order is used to define the following:

- ✧ Know the loading order of all the dependencies.
- ✧ If several **PortalContainerConfigOwner** dependencies exist, the unified **ServletContext** (*PortalContainer.getPortalContext()*) is called to get a resource. It will attempt to get the resource in the **ServletContext** of the most important **PortalContainerConfigOwner** (the last in the dependency list). If it is able to find the resource, it will try again with the second most important **PortalContainerConfigOwner** and so on.

settings

A **java.util.Map** of internal parameters to apply to the portal container. Parameters could have any type of value. This field is optional. If some internal settings are defined at the **PortalContainerConfig** level, the two settings maps are merged. If a setting with the same name is defined in both maps, it will keep the value defined at the **PortalContainerDefinition** level.

externalSettingsPath

The path of the external properties file to load as default settings to the portal container. This field is optional. If some external settings are defined at the **PortalContainerConfig** level, the two settings maps are merged. If a setting with the same name is defined in both maps, it will keep the value defined at the **PortalContainerDefinition** level. The external properties files can be either of type "properties" or of type "xml". The path will be interpreted as follows:

1. If the path doesn't contain any prefix of type "classpath:", "jar:" or "file:", it is assumed the file could be externalized. The following rules are applied:
 - a. If a file exists at `${exo-conf-dir}/portal/${portalContainerName}/${externalSettingsPath}` this file is loaded.
 - b. If no file exists, it is assumed that the path can be interpreted by the **ConfigurationManager**.

2. If the path contains a prefix, it is assumed that the path can be interpreted by the **ConfigurationManager**.

When a `PortalContainerDefinition` is used to define the default portal container, some extra logic is used in the fields.

PortalContainerDefinition Parameter Descriptions When Used for The Default Portal Container



Note

Parameter values marked with an asterisk (*) can be defined through System properties like any values in configuration files, and variables loaded by the *PropertyConfigurator*. For example, in the portal it would be all the variables defined in the file *configuration.properties* by default.

name (*)

The name of the portal container. This field is optional. The default portal name is determined by the following logic:

1. If this field is not empty, the default value is the value of this field.
2. If this field is empty and the value of the parameter *default.portal.container* is not empty, the default value will be the value of the parameter.
3. If this field and the parameter *default.portal.container* are both empty, the default value is "portal".

restContextName (*)

The name of the context name of the rest web application. This field is optional. The default value is determined by the following logic:

1. If this field is not empty, then the default value will be the value of this field.
2. If this field is empty and the value of the parameter *default.rest.context* is not empty, then the default value will be the value of the parameter.
3. If this field and the parameter *default.rest.context* are both empty, the default value will be "rest".

realmName (*)

The name of the realm. This field is optional. The default value is determined by the following logic:

1. If this field is not empty, then the default value is the value of this field.
2. If this field is empty and the value of the parameter *default.realm.name* is not empty, then the default value is the value of the parameter.
3. If this field and the parameter *default.realm.name* are both empty, the default value is "exo-domain".

dependencies

All dependencies of the portal container ordered by loading priority. This field is optional. If this field has a non empty value, it is interpreted as the default list of dependencies.

settings

A `java.util.Map` of internal parameters to associate with the default portal container. Parameters can be of any data type. This field is optional.

externalSettingsPath

The path of the external properties file to load as default settings to the default portal container. This field is optional. The external properties files can be either of type "properties" or of type "xml". The path is interpreted as follows:

1. If the path does not contain any prefix of type "classpath:", "jar:" or "file:", it is assumed that the file could be externalized. The following rule subset is applied:
 - a. If a file exists at `${exo-conf-dir}/portal/${externalSettingsPath}`, the file is loaded.
 - b. If no file exists at the previous path, it is assumed that the path can be interpreted by the **ConfigurationManager**.
2. If the path contains a prefix, it is assumed that the path can be interpreted by the **ConfigurationManager**.

Internal and external settings are both optional, however the application will merge the settings if a non empty value is provided. If the same setting name exists in both settings, the following rules apply:

1. If the value of the external setting is *null*, the value is ignored.
2. If the value of the external setting is not *null*, and the value of the internal setting is *null*, the final value is the external setting value that is of type **String**.
3. If both values are not **null**, the external setting value is converted into the target type (the internal setting value) through the static method `valueOf(String)`. The following rule subset is applied:
 - a. If the method cannot be found, the final value is the external setting value that is of type **String**.
 - b. If the method can be found and the external setting value is an empty **String**, the external setting value is ignored.
 - c. If the method can be found and the external setting value is not an empty **String** but the method call fails, the external setting value is ignored.
 - d. If the method can be found, the external setting value is not an empty **String**, and the method call succeeds, the final value is the external setting value that is the same type as the internal setting value.

[Report a bug](#)

A.4.4. PortalContainer Settings

The `portal.container.*` variable permits value injection into portal container configuration files. For example, to get the value of a setting called "name", use `${portal.container.name}`. The following internal variables are also valid:

PortalContainer Internal Variables

portal.container.name

Gives the name of the current portal container.

portal.container.rest

Gives the context name of the rest web application of the current portal container.

portal.container.realm

Gives the realm name of the current portal container.

Example A.3. portal.container.* Usage Example

```
<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.exoplaform.org/xml/ns/kernel_1_2.xsd
http://www.exoplaform.org/xml/ns/kernel_1_2.xsd"
  xmlns="http://www.exoplaform.org/xml/ns/kernel_1_2.xsd">
  <component>

<type>org.exoplatform.container.TestPortalContainer$MyComponent</type>
  <init-params>
    <!-- The name of the portal container -->
    <value-param>
      <name>portal</name>
      <value>${portal.container.name}</value>
    </value-param>
    <!-- The name of the rest ServletContext -->
    <value-param>
      <name>rest</name>
      <value>${portal.container.rest}</value>
    </value-param>
    <!-- The name of the realm -->
    <value-param>
      <name>realm</name>
      <value>${portal.container.realm}</value>
    </value-param>
    <value-param>
      <name>foo</name>
      <value>${portal.container.foo}</value>
    </value-param>
    <value-param>
      <name>before foo after</name>
      <value>before ${portal.container.foo} after</value>
    </value-param>
  </init-params>
</component>
</configuration>
```

Example A.4. Create a New Variable from Existing Variables

In the properties file corresponding to the external settings, variables previously defined in the internal or external settings can be reused to create a new variable. Contrary to [Example A.3, “portal.container.* Usage Example”](#) the prefix **portal.container.** is not needed.

```
my-var1=value 1
my-var2=value 2
complex-value=${my-var1}-${my-var2}
```

Example A.5. Settings Created Using System Parameters

External and internal settings can also be created based on the values of system parameters. The System parameters can be defined at launch time, or by using the **PropertyConfigurator**.

```
temp-dir=${java.io.tmpdir}${file.separator}my-temp
```



Note

System parameters of type **java.lang.String** can only be used when generating new internal setting variables.

Example A.6. Specify a Dynamically-changing Variable

A generic variable can be defined in the settings of the default portal container. The value of this variable will change dynamically, according to the current portal container.

```
my-generic-var=value of the portal container "${name}"
```

If this variable is defined at the default portal container level, the value of this variable for a portal container called **"sales"** is **value of the portal container "sales"**.

[Report a bug](#)

A.4.5. Dynamically Changing a PortalContainerDefinition

It is possible to use **component-plugin** elements to dynamically change a PortalContainerDefinition.

Example A.7.

The dependency **sales** is added to the default portal container, and to the portal containers named **shop1** and **shop2**:

```
<external-component-plugins>
  <!-- The full qualified name of the PortalContainerConfig -->
  <target-
component>org.exoplatform.container.definition.PortalContainerConfig</
target-component>
  <component-plugin>
```

```

<!-- The name of the plugin -->
<name>Change PortalContainer Definitions</name>
<!-- The name of the method to call on the PortalContainerConfig
in order to register the changes on the PortalContainerDefinitions -->
<set-method>registerChangePlugin</set-method>
<!-- The full qualified name of the
PortalContainerDefinitionChangePlugin -->

<type>org.exoplatform.container.definition.PortalContainerDefinitionCh
angePlugin</type>
<init-params>
  <value-param>
    <name>apply.default</name>
    <value>true</value>
  </value-param>
  <values-param>
    <name>apply.specific</name>
    <value>shop1</value>
    <value>shop2</value>
  </values-param>
  <object-param>
    <name>change</name>
    <object
type="org.exoplatform.container.definition.PortalContainerDefinitionCh
ange$AddDependencies">
      <!-- The list of name of the dependencies to add -->
      <field name="dependencies">
        <collection type="java.util.ArrayList">
          <value>
            <string>foo</string>
          </value>
        </collection>
      </field>
    </object>
  </object-param>
</init-params>
</component-plugin>
</external-component-plugins>

```

PortalContainerDefinitionChangePlugin Parameters



Note

Parameter values marked with a (*) can be defined through System properties like any values in configuration files, and variables loaded by the *PropertyConfigurator*. In the portal, an example of acceptable values would be all variables defined in the file **configuration.properties** by default.

apply.all (*)

Indicates whether the changes have to be applied to all the portal containers or not. The default value of this field is **false**. This field is a **ValueParam** and is not mandatory.

apply.default (*)

Indicates whether the changes have to be applied to the default portal container or not. The default value of this field is **false**. This field is a **ValueParam** and is not mandatory.

apply.specific (*)

A set of specific portal container names to apply the changes to. This field is a **ValuesParam** and is not mandatory.



Other Expected Parameters

The rest of the expected parameters are **ObjectParam** of type **PortalContainerDefinitionChange**. Those parameters are in fact the list of changes to apply to one or several portal containers. If the list of changes is empty, the component plug-in is ignored. The supported implementations of **PortalContainerDefinitionChange** are described later in this section.

The following algorithm is used to identify how changes are applied to portal containers:

1. If the parameter **apply.all** is set to **true** the corresponding changes are applied to all the portal containers and other parameters will be ignored.
2. If the parameter **apply.default** is set to **true** and the parameter **apply.specific** is **null**, the corresponding changes are applied to the default portal container only.

If **apply.specific** is not **null**, the corresponding changes are applied to the default portal container, and the list of specific portal containers.

3. If the parameter **apply.default** is set to **false** or has not been set, and the parameter **apply.specific** is **null**, the corresponding changes are applied to the default portal container only.

If **apply.specific** is not **null**, the corresponding changes are applied to the list of specific portal containers.

[Report a bug](#)

A.4.5.1. PortalContainerDefinitionChange Implementations

The modifications that can be applied to a **PortalContainerDefinition** must be a class of type **PortalContainerDefinitionChange**. The product ships implementations described in the following sections.

[Report a bug](#)

A.4.5.1.1. AddDependencies

This modification adds a list of dependencies at the end of the list of dependencies defined into the **PortalContainerDefinition**. The full qualified name is *org.exoplatform.container.definition.PortalContainerDefinitionChange\$AddDependencies*.

AddDependencies Parameters

dependencies

A *String* list, corresponding to the list of dependency names to add. If the field is unpopulated, the change is ignored.

Example A.8. Appending a Dependency

This example adds **foo** at the end of the dependency list of the default portal container.

```
<external-component-plugins>
  <!-- The full qualified name of the PortalContainerConfig -->
  <target-
component>org.exoplatform.container.definition.PortalContainerConfig</
target-component>
  <component-plugin>
    <!-- The name of the plugin -->
    <name>Change PortalContainer Definitions</name>
    <!-- The name of the method to call on the PortalContainerConfig
in order to register the changes on the PortalContainerDefinitions -->
    <set-method>registerChangePlugin</set-method>
    <!-- The full qualified name of the
PortalContainerDefinitionChangePlugin -->

    <type>org.exoplatform.container.definition.PortalContainerDefinitionCh
angePlugin</type>
    <init-params>
      <value-param>
        <name>apply.default</name>
        <value>true</value>
      </value-param>
      <object-param>
        <name>change</name>
        <object
type="org.exoplatform.container.definition.PortalContainerDefinitionCh
ange$AddDependencies">
          <!-- The list of name of the dependencies to add -->
          <field name="dependencies">
            <collection type="java.util.ArrayList">
              <value>
                <string>foo</string>
              </value>
            </collection>
          </field>
        </object>
      </object-param>
    </init-params>
  </component-plugin>
</external-component-plugins>
```

[Report a bug](#)

A.4.5.1.2. AddDependenciesBefore

This modification adds a list of dependencies before a given target dependency defined into the list of dependencies of the **PortalContainerDefinition**. The fully qualified name is **org.exoplatform.container.definition.PortalContainerDefinitionChange\$AddDependenciesBefore**.

AddDependenciesBefore Parameters

dependencies

A *String* list, corresponding to the list of dependency names to add. If the field is unpopulated, the change is ignored.

target

The name of the dependency that the new dependencies are added before. New dependencies are added in first position to the list if this field is **null**, or the target dependency cannot be found in the list of dependencies defined into the **PortalContainerDefinition**.

Example A.9. Adding Multiple Dependencies

This example adds **dependency1** before **dependency2** in the dependency list of the default portal container.

```
<external-component-plugins>
  <!-- The full qualified name of the PortalContainerConfig -->
  <target-
component>org.exoplatform.container.definition.PortalContainerConfig</
target-component>
  <component-plugin>
    <!-- The name of the plugin -->
    <name>Change PortalContainer Definitions</name>
    <!-- The name of the method to call on the PortalContainerConfig
in order to register the changes on the PortalContainerDefinitions -->
    <set-method>registerChangePlugin</set-method>
    <!-- The full qualified name of the
PortalContainerDefinitionChangePlugin -->

    <type>org.exoplatform.container.definition.PortalContainerDefinitionCh
angePlugin</type>
    <init-params>
      <value-param>
        <name>apply.default</name>
        <value>true</value>
      </value-param>
      <object-param>
        <name>change</name>
        <object
type="org.exoplatform.container.definition.PortalContainerDefinitionCh
ange$AddDependenciesBefore">
          <!-- The list of name of the dependencies to add -->
          <field name="dependencies">
            <collection type="java.util.ArrayList">
              <value>
                <string>dependency1</string>
              </value>
```

```

        </collection>
      </field>
      <!-- The name of the target dependency -->
      <field name="target">
        <string>dependency2</string>
      </field>
    </object>
  </object-param>
</init-params>
</component-plugin>
</external-component-plugins>

```

[Report a bug](#)

A.4.5.1.3. AddDependenciesAfter

This modification adds a list of dependencies before a given target dependency defined into the list of dependencies of the **PortalContainerDefinition**. The full qualified name is *org.exoplatform.container.definition.PortalContainerDefinitionChange\$AddDependenciesAfter*.

AddDependenciesAfter Parameters

dependencies

A list of *String* corresponding to the list of dependency names to add. If the field is unpopulated, the change is ignored.

target

The name of the dependency that the new dependencies are added before. New dependencies are added in first position to the list if this field is **null**, or the target dependency cannot be found in the list of dependencies defined into the **PortalContainerDefinition**.

Example A.10. Adding Dependencies After

This example adds **dependency1** after **dependency2** in the dependency list of the default portal container.

```

<external-component-plugins>
  <!-- The full qualified name of the PortalContainerConfig -->
  <target-
component>org.exoplatform.container.definition.PortalContainerConfig</
target-component>
  <component-plugin>
    <!-- The name of the plugin -->
    <name>Change PortalContainer Definitions</name>
    <!-- The name of the method to call on the PortalContainerConfig
in order to register the changes on the PortalContainerDefinitions -->
    <set-method>registerChangePlugin</set-method>
    <!-- The full qualified name of the
PortalContainerDefinitionChangePlugin -->

    <type>org.exoplatform.container.definition.PortalContainerDefinitionCh
angePlugin</type>
  </component-plugin>
</external-component-plugins>

```

```

<init-params>
  <value-param>
    <name>apply.default</name>
    <value>true</value>
  </value-param>
  <object-param>
    <name>change</name>
    <object
type="org.exoplatform.container.definition.PortalContainerDefinitionCh
ange$AddDependenciesAfter">
      <!-- The list of name of the dependencies to add -->
      <field name="dependencies">
        <collection type="java.util.ArrayList">
          <value>
            <string>dependency1</string>
          </value>
        </collection>
      </field>
      <!-- The name of the target dependency -->
      <field name="target">
        <string>dependency2</string>
      </field>
    </object>
  </object-param>
</init-params>
</component-plugin>
</external-component-plugins>

```

[Report a bug](#)

A.4.5.1.4. AddSettings

This modification adds new settings to a **PortalContainerDefinition**. The full qualified name is *org.exoplatform.container.definition.PortalContainerDefinitionChange\$AddSettings*.

AddSettings Parameters

settings

A map of *<String, Object>* corresponding to the settings to add. The change is ignored if the value of this field is empty.

Example A.11. Add Settings

This example adds the settings **string** and **stringX** to the settings of the default portal container.

```

<external-component-plugins>
  <!-- The full qualified name of the PortalContainerConfig -->
  <target-
component>org.exoplatform.container.definition.PortalContainerConfig</
target-component>
  <component-plugin>
    <!-- The name of the plugin -->
    <name>Change PortalContainer Definitions</name>

```

```

    <!-- The name of the method to call on the PortalContainerConfig
    in order to register the changes on the PortalContainerDefinitions -->
    <set-method>registerChangePlugin</set-method>
    <!-- The full qualified name of the
    PortalContainerDefinitionChangePlugin -->

<type>org.exoplatform.container.definition.PortalContainerDefinitionCh
angePlugin</type>
  <init-params>
    <value-param>
      <name>apply.default</name>
      <value>true</value>
    </value-param>
    <object-param>
      <name>change</name>
      <object
type="org.exoplatform.container.definition.PortalContainerDefinitionCh
ange$AddSettings">
        <!-- The settings to add to the to the portal containers -->
        <field name="settings">
          <map type="java.util.HashMap">
            <entry>
              <key>
                <string>string</string>
              </key>
              <value>
                <string>value1</string>
              </value>
            </entry>
            <entry>
              <key>
                <string>stringX</string>
              </key>
              <value>
                <string>value1</string>
              </value>
            </entry>
          </map>
        </field>
      </object>
    </object-param>
  </init-params>
</component-plugin>
</external-component-plugins>

```

[Report a bug](#)

A.4.6. Dynamically Disable a Portal Container

It is possible to use **component-plugin** elements to dynamically disable one or more portal containers.

Example A.12. Disabling a Portal Container

The example shows how to disable the portal container named **sales**.

```
<external-component-plugins>
  <!-- The full qualified name of the PortalContainerConfig -->
  <target-
component>org.exoplatform.container.definition.PortalContainerConfig</
target-component>
  <component-plugin>
    <!-- The name of the plugin -->
    <name>Disable a PortalContainer</name>
    <!-- The name of the method to call on the PortalContainerConfig
in order to register the changes on the PortalContainerDefinitions -->
    <set-method>registerDisablePlugin</set-method>
    <!-- The full qualified name of the
PortalContainerDefinitionDisablePlugin -->

<type>org.exoplatform.container.definition.PortalContainerDefinitionDis
ablePlugin</type>
    <init-params>
      <!-- The list of the name of the portal containers to disable -->
      <values-param>
        <name>names</name>
        <value>sales</value>
      </values-param>
    </init-params>
  </component-plugin>
</external-component-plugins>
```

PortalContainerDefinitionDisablePlugin Parameters

names (*)

Specifies a list of portal container names to disable.



Note

Values of parameters marked with an (*) can be defined through System properties like any values in configuration files, and variables loaded by the *PropertyConfigurator*. In the portal, parameters would be all the variables defined in the file *configuration.properties* by default.

To prevent access to a disabled web application corresponding to **PortalContainer**, ensure the following HTTP Filter (or a sub class of it) has been added to the **web.xml** file, at the beginning of the file.

```
<filter>
  <filter-name>PortalContainerFilter</filter-name>
  <filter-
class>org.exoplatform.container.web.PortalContainerFilter</filter-class>
</filter>
```

```
<filter-mapping>
  <filter-name>PortalContainerFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```



Note

It is only possible to disable a portal container when at least one `PortalContainerDefinition` has been registered.

[Report a bug](#)

A.5. Runtime Configuration Profiles

The kernel configuration is able to handle configuration profiles at runtime (as opposed to packaging time).

An active profile list is obtained when the root container is loaded, and is composed of the system property `exo.profiles` sliced according the "," delimiter and a server specific profile value (tomcat for tomcat, jboss for jboss, etc...).

```
# runs the portal with the profiles jboss, foo and bar
sh run.sh -Dexo.profiles=foo,bar
```

Profile activation occurs in XML to configure object unmarshalling time. It is based on a "profile" attribute that is present on some of the XML elements of the configuration files. The configuration is based on the following rules:

1. Any kernel element with the no *profiles* attribute will create a configuration object.
2. Any kernel element having a *profiles* attribute containing at least one of the active profiles will create a configuration object.
3. Any kernel element having a *profiles* attribute matching none of the active profiles will not create a configuration object
4. Resolution of duplicates (such as two components of the same type) is left up to the kernel.

A `<configuration>` element is profiles-capable when it carries a `<profiles>` element.

Profiles-capable Configuration Directives

`<component>`

The `<component>` element declares a component when activated. It will shadow any element with the same key declared before in the same configuration file:

```
<component>
  <key>Component</key>
  <type>Component</type>
</component>
```

```
<component profiles="foo">
  <key>Component</key>
  <type>FooComponent</type>
</component>
```

<component-plugin>

The <component-plugin> element is used to dynamically extend the configuration of a given component. Thanks to the profiles the <component-plugin> elements can be enabled or disabled:

```
<external-component-plugins>
  <target-component>Component</target-component>
  <component-plugin profiles="foo">
    <name>foo</name>
    <set-method>addPlugin</set-method>
    <type>type</type>
    <init-params>
      <value-param>
        <name>param</name>
        <value>empty</value>
      </value-param>
    </init-params>
  </component-plugin>
</external-component-plugins>
```

<import>

The <import> element imports a referenced configuration file when activated:

```
<import>empty</import>
<import profiles="foo">foo</import>
<import profiles="bar">bar</import>
```

<init-params>

The <init-params> element configures the parameter argument of the construction of a component service:

```
<component>
  <key>Component</key>
  <type>ComponentImpl</type>
  <init-params>
    <value-param>
      <name>param</name>
      <value>empty</value>
    </value-param>
    <value-param profiles="foo">
      <name>param</name>
      <value>foo</value>
    </value-param>
    <value-param profiles="bar">
      <name>param</name>
```

```

    <value>bar</value>
  </value-param>
</init-params>
</component>

```

<value-collection>

The <value-collection> element configures one of the value of collection data:

```

<object
  type="org.exoplatform.container.configuration.ConfigParam">
  <field name="role">
    <collection type="java.util.ArrayList">
      <value><string>manager</string></value>
      <value profiles="foo"><string>foo_manager</string></value>
      <value profiles="foo, bar"><string>foo_bar_manager</string>
    </value>
    </collection>
  </field>
</object>

```

<field-configuration>

The <field-configuration> element configures the field of an object:

```

<object-param>
  <name>test.configuration</name>
  <object
    type="org.exoplatform.container.configuration.ConfigParam">
    <field name="role">
      <collection type="java.util.ArrayList">
        <value><string>manager</string></value>
      </collection>
    </field>
    <field name="role" profiles="foo, bar">
      <collection type="java.util.ArrayList">
        <value><string>foo_bar_manager</string></value>
      </collection>
    </field>
    <field name="role" profiles="foo">
      <collection type="java.util.ArrayList">
        <value><string>foo_manager</string></value>
      </collection>
    </field>
  </object>
</object-param>

```

[Report a bug](#)

A.6. Component request life cycle

[Report a bug](#)

A.6.1. Component request life cycle contract

The component request life cycle is an interface that defines a contract for a component for being involved into a request:

```
public interface ComponentRequestLifecycle
{
    /**
     * Start a request.
     * @param container the related container
     */
    void startRequest(ExoContainer container);

    /**
     * Ends a request.
     * @param container the related container
     */
    void endRequest(ExoContainer container);
}
```

The container passed is the container to which the component is related. This contract is often used to setup a thread local based context that will be demarcated by a request.

For example, a component in the portal request life cycle is triggered for user requests. Another example is the initial data import in the portal that demarcates using callbacks made to that interface.

[Report a bug](#)

A.6.2. Request life cycle

The **RequestLifeCycle** class has several statics methods that are used to schedule the component request life cycle of components. Its main responsibility is to perform scheduling while respecting the constraint to execute the request life cycle of a component only once even if it can be scheduled several times.

[Report a bug](#)

A.6.2.1. Scheduling a component request life cycle

```
RequestLifeCycle.begin(component);
try
{
    // Do something
}
finally
{
    RequestLifeCycle.end();
}
```

[Report a bug](#)

A.6.2.2. Scheduling a container request life cycle

Scheduling a container triggers the component request life cycle of all the components that implement the interface **ComponentRequestLifeCycle**. If one of the components has already been scheduled before, the component is not scheduled again.

If the local value is **true**, the looked components are those of the container. When the value is **false**, the scheduler looks at the components in the ancestor containers.

```
RequestLifecycle.begin(container, local);
try
{
    // Do something
}
finally
{
    RequestLifecycle.end();
}
```

[Report a bug](#)

A.6.2.3. When request life cycle is triggered

[Report a bug](#)

A.6.2.3.1. Portal request life cycle

Each portal request triggers the life cycle of the associated portal container.

[Report a bug](#)

A.6.2.3.2. JMX request Life Cycle

When a JMX bean is invoked, the request life cycle of the container to which it belongs is scheduled. Indeed JMX is an entry point of the system that may need component to have a request life cycle triggered.

[Report a bug](#)

A.7. Configuring Services

The eXo Kernel uses dependency injection to create services based on **configuration.xml** configuration files. The location of the configuration files determines if services are placed into the **RootContainer** scope, or into the **PortalContainer** scope.

When creating a service, declare its existence to the Container by creating a **/conf/configuration.xml** configuration file in the service's base folder. The container looks for a **/conf/configuration.xml** file in each jar-file.

All files located at **/conf/configuration.xml** in the classpath (any directory, or any jar in the classpath) will have services configured in the **RootContainer** scope.

All **configuration.xml** files located in **/conf/portal/configuration.xml** of the classpath will have services configured at the **PortalContainer** scope.

Portal extensions can also use configuration information stored in **\$JPP_HOME/gatein/gatein.ear/portal.war/WEB-INF/conf/configuration.xml**, and will also have their services configured in the **PortalContainer** scope.

When eXo kernel reads a configuration, it loads the file from the kernel jar using the classloader, and does not use an internet connection to resolve the file.



Note

Portal extensions are described later in this document.

[Report a bug](#)

A.7.1. Configuration syntax

[Report a bug](#)

A.7.1.1. Components

A service component is defined in **configuration.xml** by using a `<component>` element.

Only one piece of information is required when defining a service; the service implementation class. This is specified using `<type>`

Every component has a `<key>` element that identifies it. If not explicitly set, a key defaults to the value of `<type>`. If a key can be loaded as a class, a class object is used as a key, otherwise a string is used.

The typical approach is to specify an interface as a key.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<configuration
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.exoplaform.org/xml/ns/kernel_1_2.xsd
http://www.exoplaform.org/xml/ns/kernel_1_2.xsd"
  xmlns="http://www.exoplaform.org/xml/ns/kernel_1_2.xsd">
  <component>
    <key>org.exoplaform.services.database.HibernateService</key>

    <type>org.exoplaform.services.database.impl.HibernateServiceImpl</type>

    ...

  </component>
</configuration>
```

The configuration found inside the JAR file is considered the default configuration. The default configuration can be overridden in different places inside the JAR. When the container finds several configurations for the same service, the *configuration override mechanism* always loads the most recently discovered configuration file.

All custom configuration is made in the **/webapps/portal/WEB-INF/conf/configuration.xml** file. Use `<component>` elements to contain the configuration information. The `<key>` element defines the interface, and the `<type>` tag defines the implementation. While the `<key>` tag is not mandatory, specifying it can result in a small performance improvement.

Example A.13. `<component>` Custom Configuration Block

Register plug-ins that can act as listeners or external plug-in to bundle some plug-in classes in other jar modules. The usual example is the hibernate service to which HBM mapping files can be added, even if those files are deployed in another maven artifact.

```
<!-- Portlet container hooks -->
<component>

<key>org.exoplatform.services.portletcontainer.persistence.PortletPreferencesPersister</key>

<type>org.exoplatform.services.portal.impl.PortletPreferencesPersisterImpl</type>
</component>
```

Example A.14. Define Services That Can Receive Plug-ins Without a Framework Interface

Target the HibernateService and call its addPlugin() method with an argument of the type AddHibernateMappingPlugin. Init parameters for the object are already filled.

```
<external-component-plugins>
  <target-component>org.exoplatform.services.database.HibernateService</target-component>
  <component-plugin>
    <name>add.hibernate.mapping</name>
    <set-method>addPlugin</set-method>

    <type>org.exoplatform.services.database.impl.AddHibernateMappingPlugin</type>
    <init-params>
      <values-param>
        <name>hibernate.mapping</name>

        <value>org/exoplatform/services/portal/impl/PortalConfigData.hbm.xml</value>

        <value>org/exoplatform/services/portal/impl/PageData.hbm.xml</value>

        <value>org/exoplatform/services/portal/impl/NodeNavigationData.hbm.xml</value>
      </values-param>
    </init-params>
  </component-plugin>
</external-component-plugins>
```

Example A.15. Add Listener to the OrganizationService

The OrganizationService calls the method **addListenerPlugin** with an object of type PortalUserEventListenerImpl. Each time a user is created (apart the predefined ones in the list) PortalUserEventListenerImpl methods are called by the service.

```
<external-component-plugins>
```

```

<target-
component>org.exoplatform.services.organization.OrganizationService</t
arget-component>
  <component-plugin>
    <name>portal.new.user.event.listener</name>
    <set-method>addListenerPlugin</set-method>

<type>org.exoplatform.services.portal.impl.PortalUserEventListenerImpl<
/type>
  <description>this listener create the portal configuration for the
new user</description>
  <init-params>
    <object-param>
      <name>configuration</name>
      <description>description</description>
      <object
type="org.exoplatform.services.portal.impl.NewPortalConfig">
        <field name="predefinedUser">
          <collection type="java.util.HashSet">
            <value><string>admin</string></value>
            <value><string>exo</string></value>
            <value><string>company</string></value>
            <value><string>community</string></value>
            <value><string>portal</string></value>
            <value><string>exotest</string></value>
          </collection>
        </field>
        <field name="templateUser"><string>template</string></field>
        <field name="templateLocation">
<string>war:/conf/users</string></field>
      </object>
    </object-param>
  </init-params>
</component-plugin>
...

```

From the examples above, it is evident there are several types of init parameters available, from a simple value param which binds a key with a value to a more complex object mapping that fills a JavaBean with the info defined in the XML.

[Report a bug](#)

A.7.1.1.1. RootContainer

RootContainer is a dependency of PortalContainer. It is important to understand how RootContainer works because it is referred to in other containers.

Configuration retrieval occurs from the following file locations:

1. Services default **RootContainer** configurations from JAR files
/conf/configuration.xml
2. External **RootContainer** configuration, to be found at
\$JPP_HOME/gatein/gatein.ear/portal.war/WEB-INF/conf/configuration.xml

The **RootContainer** creates a Java **HashTable** which contains key-value pairs for the services.

The qualified interface name of each service is used as key for the hash table. The **<key>** element of the configuration file contains the interface name. The value of each hash table pair is an object that contains the service configuration, which is the whole structure between the **<component>** tags of the **configuration.xml** file.

The **RootContainer** runs over all JAR files present in

\$JPP_HOME/modules/system/layers/gatein/org/gatein/lib and checks if there is a configuration file located in **/conf/configuration.xml**. If the file is present, the services configured in this file are added to the hash table. At the end of the boot process, the default configurations for all services are stored in the hash table.



Note

If the same service, recognized by the same qualified interface name, is configured in different JARs, the configuration defined in a JAR archive loaded after the previous JAR is used. Because the loading of or JAR files can be unpredictable, do not depend on the load order to load the correct configuration.

To provide configuration detail for one or more services, create a general configuration file located in **\$JPP_HOME/gatein/gatein.ear/portal.war/WEB-INF/conf/configuration.xml**. The same configuration rules that apply to JARs apply to this configuration file.

Further configuration retrieval depends on the container type.

[Report a bug](#)

A.7.1.1.2. PortalContainer

The PortalContainer takes the hash table filled by the RootContainer and looks in other locations. Here you get the opportunity to replace RootContainer configurations by those which are specific to your portal. Again, the configurations are overridden whenever necessary.

The PortalContainer configuration is retrieved in the following lookup sequence:

1. Inherit the configuration of the RootContainer.
2. Default PortalContainer configuration from all JAR files (**/conf/portal/configuration.xml**)
3. Web application configurations from the **portal.war** file, or the portal web app configuration file (**/WEB-INF/conf/configuration.xml**).
4. External configuration for services of a named portal, located in **exo-tomcat/exo-conf/portal/\$portal_name/configuration.xml**.

The **/conf/portal/configuration.xml** file of each JAR are loaded as they are discovered, the last configuration file loaded being the one used for all JARs. The next file the container loads is from within the portal.war archive (or the portal webapp folder). The **/WEB-INF/conf/configuration.xml** contains many import statements that point to other configuration files in the same portal.war (or portal webapp).

For multiple portals, be aware that each portal may use a unique **configuration.xml** file in the PortalContainer. From Red Hat JBoss Portal 6.1, configuration can be provided from outside the JARs and WARs or webapps. Place a configuration file in **exo-tomcat/exo-conf/portal/\$portal_name/configuration.xml** where **\$portal_name** is the name of the

portal to configure. In most cases, installations have one primary portal, named by default as "portal". In this scenario, the `exo - tomcat/exo - conf/portal/portal/configuration.xml` is used.



Note

`exo - conf` is looked up in directory specified by the Red Hat JBoss System property `jboss.server.config.url`. If the property is not found or empty, `exo-jboss/exo-conf` is loaded by default.



Note

As of Red Hat JBoss Portal 6.1 the system property `exo . conf . dir` can be used to override the external configuration location. If the property is defined, its value is used as path to the eXo configuration directory. This is an alternative method of loading configuration in the `exo - tomcat/exo - conf` folder.

When starting the portal, add the property in the command line: `java - Dexo . conf . dir=/path/to/exo/conf`, or use `eXo . bat` or `eXo . sh`.

In this particular use case, it is not necessary to use any prefixes in the configuration file to import other files. For example, if the standard configuration file is located in `exo - tomcat/exo - conf/portal/PORTAL_NAME/configuration.xml` and the target configuration file located in `exo - tomcat/exo - conf/portal/PORTAL_NAME/mySubConfDir/myConfig.xml` must be loaded, add `<import>mySubConfDir/myConfig.xml</import>` to the configuration file.

[Report a bug](#)

A.7.1.1.3. External Plug-ins

The eXo Kernel supports non-component objects that can be configured, instantiated, and injected into registered components using method calls. This plug-in method allows portal extensions to add additional configuration to core services.

An external plug-in is defined by using the `<external-component-plugin>` wrapper element, which contains one or more `<component-plugin>` definitions.

The `<external-component-plugin>` element uses `<target-component>` to specify a target service component that will receive injected objects.

Every `<component-plugin>` defines an implementation type, and a method on the target component to use for injection (`<set-method>`).

A plug-in implementation class has to implement the `org.exoplatform.container.component.ComponentPlugin` interface.

In the following example the `PortalContainerDefinitionPlugin` implements the `ComponentPlugin`:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
```

```

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.exoplaform.org/xml/ns/kernel_1_2.xsd
http://www.exoplaform.org/xml/ns/kernel_1_2.xsd"
xmlns="http://www.exoplaform.org/xml/ns/kernel_1_2.xsd">

  <external-component-plugins>
    <target-
component>org.exoplatform.container.definition.PortalContainerConfig</ta
rget-component>
    <component-plugin>
      <!-- The name of the plug-in -->
      <name>Add PortalContainer Definitions</name>

      <!-- The name of the method to call on the
PortalContainerConfig
      in order to register the PortalContainerDefinitions -->
      <set-method>registerPlugin</set-method>

      <!-- The fully qualified name of the
PortalContainerDefinitionPlugin -->

<type>org.exoplatform.container.definition.PortalContainerDefinitionPlug
in</type>

      ...

    </component-plugin>
  </external-component-plugins>
</configuration>

```

The `<target-component>` defines the service for which the plug-in is defined. The configuration is injected by the container using a method that is defined in `<set-method>`. The method has one argument of the type

`org.exoplatform.services.cms.categories.impl.TaxonomyPlugin`:

```

* addTaxonomyPlugin(org.exoplatform.services.cms.categories.impl.Taxonom
yPlugin plugin)

```

The content of `<init-params>` corresponds to the structure of the `TaxonomyPlugin` object.



Note

The `CategoriesService` component can be configured using the `addTaxonomyPlugin` as often as required. `addTaxonomyPlugin` can also be called in different configuration files. The method `addTaxonomyPlugin` is then called several times: everything else depends on the implementation of the method.

[Report a bug](#)

A.7.1.1.4. Service instantiation

All services are *singletons*: the container creates only one single instance of each container. The services are created by calling the constructors, the process of which is referred to as *constructor injection*).

Example A.16. Service Instantiation

The JDBC implementation of the `BaseOrganizationService` interface contained in `OrganizationServiceImpl.java` ^[1] file has one constructor only.

This service depends on two other services. To call this constructor, the container requires a `ListenerService` and a `DatabaseService`. These services must be instantiated before `BaseOrganizationService`, because `BaseOrganizationService` depends on them.

```
public OrganizationServiceImpl(ListenerService listenerService,
    DatabaseService dbService);
```

To execute the service instantiation required in [Example A.16, “Service Instantiation”](#), dependencies are injected by the container. The container first looks at the constructors of all services and creates a matrix of service dependencies in order to call the services in the correct order. If for any reason there are interdependencies or circular dependencies detected, a `java.lang.Exception` is thrown.



Note

If one service has more than one constructor, the container attempts to use the constructor with a maximum of arguments first. If this is not possible, the container continues step by step with constructors that have less arguments until arriving at the zero-argument constructor (if there is one).

[1] <https://anonsvn.jboss.org/repos/exo-jcr/core/trunk/exo.core.component.organization.jdbc/src/main/java/org/exoplatform/services/organization/jdbc/OrganizationServiceImpl.java>

[Report a bug](#)

A.7.1.1.5. Service Access

To maintain Inversion of Control (IoC) principles, a container must be used when accessing a service. Accessing a service directly does not conform with IoC requirements.

To get the current container, use the `ExoContainer myContainer = ExoContainerContext.getCurrentContainer();` command.

To retrieve a configured service, use the `myContainer.getComponentInstance(class)` method.

Example A.17. Service Access Command and Method Usage

```
ArticleStatsService statsService = (ArticleStatsService)
myContainer.getComponentInstance(ArticleStatsService.class);
```

Example A.18.

```

package com.laverdad.common;

import org.exoplatform.container.ExoContainer;
import org.exoplatform.container.ExoContainerContext;
import com.laverdad.services.*;

public class Statistics {

    public int makeStatistics(String articleText) {
        ExoContainer myContainer =
ExoContainerContext.getCurrentContainer();
        ArticleStatsService statsService = (ArticleStatsService)
            myContainer.getComponentInstance(ArticleStatsService.class);
        int numberOfSentences = statsService.calcSentences(articleText);
        return numberOfSentences;
    }

    public static void main( String args[]) {
        Statistics stats = new Statistics();
        String newText = "This is a normal text. The method only counts the
number of periods. "
            + "You can implement your own implementation with a more exact
counting. "
            + "Let`s make a last sentence.";
        System.out.println("Number of sentences: " +
stats.makeStatistics(newText));
    }
}

```

[Report a bug](#)

A.7.1.1.6. Includes, and special URLs

It is possible to divide the **configuration.xml** file into many smaller files, which are then included into the main configuration file.

The included files must be valid XML files: fragments of text in the referenced files will cause load issues.

Example A.19. Outsourcing Configuration to External Files

```

<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.exoplaform.org/xml/ns/kernel_1_2.xsd
http://www.exoplaform.org/xml/ns/kernel_1_2.xsd"
    xmlns="http://www.exoplaform.org/xml/ns/kernel_1_2.xsd">
    <!-- Comment #1 -->
    <import>war:/conf/sample-ext/jcr/jcr-configuration.xml</import>

```

```
<import>war:/conf/sample-ext/portal/portal-
configuration.xml</import>

</configuration>
```

Comment #1: This line specifies the location of another configuration file. The **war:** URL schema indicates the path is resolved relative to the current **PortalContainer**'s servlet context resource path, starting with **/WEB-INF** as the root.



Note

The current **PortalContainer** is a newly created **PortalContainer**, because **war:** URLs are used for **PortalContainer** scoped configuration only.

To have an 'include' path resolved relative to current classpath (context classloader), use a '**jar:**' URL schema.

Through the extension mechanism, the servlet context used for resource loading is a *unified servlet context* (this is explained in a later section).

[Report a bug](#)

A.7.1.1.7. Special variables

Configuration files may contain a **\${container.name.suffix}** special variable reference. This variable resolves to the name of the current portal container, prefixed by underscore (_).

This facilitates reuse of configuration files in situations where portal-specific unique names need to be assigned to certain resources: JNDI names, Database/DataSource names, and JCR repository names.

This variable is only defined when there is a current **PortalContainer** available, and is only available for **PortalContainer** scoped services.

Example A.20. \${container.name.suffix} Usage

```
<?xml version="1.0" encoding="ISO-8859-1"?>
...
<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.exoplaform.org/xml/ns/kernel_1_2.xsd
http://www.exoplaform.org/xml/ns/kernel_1_2.xsd"
xmlns="http://www.exoplaform.org/xml/ns/kernel_1_2.xsd">
...
  <component>
    <key>org.exoplaform.services.database.HibernateService</key>
    <jmx-name>database:type=HibernateService</jmx-name>

    <type>org.exoplaform.services.database.impl.HibernateServiceImpl</type>
  >
    <init-params>
      <properties-param>
```

```

        <name>hibernate.properties</name>
        <description>Default Hibernate Service</description>
        <property name="hibernate.cache.region.jbc2.query.localonly"
value="true" />
        <property name="hibernate.cache.region.factory_class"
value="org.hibernate.cache.jbc2.MultiplexedJBossCacheRegionFactory" />
        <property name="hibernate.transaction.manager_lookup_class"
value="org.hibernate.transaction.JBossTransactionManagerLookup" />
        <property name="hibernate.show_sql" value="false"/>
        <property name="hibernate.current_session_context_class"
value="thread"/>
        <property name="hibernate.cache.use_second_level_cache"
value="true"/>
        <property name="hibernate.cache.use_query_cache"
value="true"/>
        <property name="hibernate.connection.datasource"
value="${gatein.idm.datasource.name}${container.name.suffix}"/>
        <property name="hibernate.connection.autocommit"
value="true"/>
        <!--
        Should be automatically detected. Force otherwise
        <property name="hibernate.dialect"
value="org.hibernate.dialect.XXXDialect"/>
        -->
    </properties-param>
</init-params>
</component>
...
</configuration>

```

[Report a bug](#)

A.7.1.2. <init-params> configuration element

<init-params> is a configuration element that contains a map of key-value pairs, where the key is always a **String**, and the value can be any type that can be described using the kernel XML configuration.

Service components that form the portal infrastructure use <init-params> to define the configuration. A component can have one instance of <init-params> injected at most.

If the service component's constructor takes <init-params> as any of the parameters, it will be injected at component instantiation time.

The <init-params> structure (the names and types of entries) is specific for each service, as it is the code inside a service components' class that defines which entry names to look up and what types it expects to find.

The XML configuration for a service component that expects an <init-params> element must have an <init-params> element present, however this element can be left empty.

Example A.21. <init-params> XML Configuration and Parameter Descriptions

```
<component>
```

```

<key>org.exoplatform.services.naming.InitialContextInitializer</key>
<type>org.exoplatform.commons.InitialContextInitializer2</type>
<init-params>
  <properties-param>
    <name>default-properties</name>
    <description>Default initial context properties</description>
  </properties-param>
</init-params>
</component>

```

An `<init-params>` element can have zero or more children elements, each of which is one of the following:

- ✧ `<value-param>`
- ✧ `<values-param>`
- ✧ `<properties-param>`
- ✧ `<object-param>`

Each child element uses a `<name>` element that is used as a map entry key, and an optional `<description>` element. It also takes a type-specific **value** specification.

The value specification for the `<properties-param>` defines one or more `<property>` elements, each of which specifies two strings; a property name and a property value.

Each `<properties-params>` defines one **java.util.Properties** instance.

Example A.22. `<value-param>` Example

```

<component>
  <key>org.exoplatform.portal.config.UserACL</key>
  <type>org.exoplatform.portal.config.UserACL</type>
  <init-params>
...
    <value-param>
      <name>access.control.workspace</name>
      <description>groups with memberships that have the right to
access the User Control Workspace</description>

<value>*:/platform/administrators,*:/organization/management/executive
-board</value>
    </value-param>
...
  </component>

```

The UserACL class accesses to the `<value-param>` in its constructor.

```

package org.exoplatform.portal.config;
public class UserACL {

  public UserACL(InitParams params) {
    UserACLMetaData md = new UserACLMetaData();
    ValueParam accessControlWorkspaceParam =

```

```

params.getValueParam("access.control.workspace");
    if(accessControlWorkspaceParam != null)
md.setAccessControlWorkspace(accessControlWorkspaceParam.getValue());
...

```

Example A.23. <values-param> XML Configuration and Parameter Descriptions

The value specification for <value-param> elements is a <value> element which defines a **String** instance.

```

<component>
  <key>org.exoplatform.services.resources.ResourceBundleService</key>

  <type>org.exoplatform.services.resources.impl.SimpleResourceBundleService</type>
  <init-params>
    <values-param>
      <name>classpath.resources</name>
      <description>The resources that start with the following
package name should be load from file system</description>
      <value>locale.portlet</value>
    </values-param>

    <values-param>
      <name>init.resources</name>
      <description>Store the following resources into the db for the
first launch </description>
      <value>locale.test.resources.test</value>
    </values-param>

    <values-param>
      <name>portal.resource.names</name>
      <description>The properties files of the portal , those file
will be merged
      into one ResourceBundle properties </description>
      <value>local.portal.portal</value>
      <value>local.portal.custom</value>
    </values-param>
  </init-params>
</component>

```

The value specification for <values-param> requires one or more <value> elements. Each <value> represents one **String** instance. All **String** values are then collected into a **java.util.List** instance.

Example A.24. <properties-param> Hibernate Example

```

<component>
  <key>org.exoplatform.services.database.HibernateService</key>

  <type>org.exoplatform.services.database.impl.HibernateServiceImpl</type>
>

```

```

<init-params>
  <properties-param>
    <name>hibernate.properties</name>
    <description>Default Hibernate Service</description>
    <property name="hibernate.show_sql" value="false"/>
    <property name="hibernate.cglib.use_reflection_optimizer"
value="true"/>
    <property name="hibernate.connection.url"
value="jdbc:hsqldb:file:../temp/data/exodb"/>
    <property name="hibernate.connection.driver_class"
value="org.hsqldb.jdbcDriver"/>
    ...
  </properties-param>
</init-params>
</component>

```

In the `org.exoplatform.services.database.impl.HibernateServiceImpl`, the name `hibernate.properties` of the `<properties-param>` is used to access the properties.

```

package org.exoplatform.services.database.impl;

public class HibernateServiceImpl implements HibernateService,
ComponentRequestLifecycle {
  public HibernateServiceImpl(InitParams initParams, CacheService
cacheService) {
    PropertiesParam param =
initParams.getPropertiesParam("hibernate.properties");
    ...
  }
}

```

Example A.25. <object-param> XML Configuration and Parameter Descriptions

```

<component>
  <key>org.exoplatform.services.cache.CacheService</key>
  <jmx-name>cache:type=CacheService</jmx-name>
  <type>org.exoplatform.services.cache.impl.CacheServiceImpl</type>
  <init-params>
    <object-param>
      <name>cache.config.default</name>
      <description>The default cache configuration</description>
      <object type="org.exoplatform.services.cache.ExoCacheConfig">
        <field name="name">
          <string>default</string>
        </field>
        <field name="maxSize">
          <int>300</int>
        </field>
        <field name="liveTime">
          <long>300</long>
        </field>
        <field name="distributed">
          <boolean>false</boolean>
        </field>
        <field name="implementation">

```

```

<string>org.exoplatform.services.cache.concurrent.ConcurrentFIFOExoCache</string>
    </field>
  </object>
</object-param>
</init-params>
</component>

```

For <object-param> entries, the value specification consists of an <object> element which is used for plain Java style object specification (specifying an implementation *class* - <type>, and *property values* - <field>).

Example A.26. <object-param> and LDAP Example

```

<component>
  <key>org.exoplatform.services.LDAP.LDAPService</key>
  <type>org.exoplatform.services.LDAP.impl.LDAPServiceImpl</type>
  <init-params>
    <object-param>
      <name>LDAP.config</name>
      <description>Default LDAP config</description>
      <object
type="org.exoplatform.services.LDAP.impl.LDAPConnectionConfig">
        <field name="providerURL">
          <string>LDAPS://10.0.0.3:636</string>
        </field>
        <field name="rootdn">
          <string>CN=Administrator,CN=Users,DC=exoplatform,DC=org</string>
        </field>
        <field name="password">
          <string>exo</string>
        </field>
        <field name="version">
          <string>3</string>
        </field>
        <field name="minConnection">
          <int>5</int>
        </field>
        <field name="maxConnection">
          <int>10</int>
        </field>
        <field name="referralMode">
          <string>ignore</string>
        </field>
        <field name="serverName">
          <string>active.directory</string>
        </field>
      </object>
    </object-param>
  </init-params>
</component>

```


It is not important to understand LDAP for this example: only the parameters as they relate to <object-param> are discussed.

An <object-param> element is used to pass the parameters inside an object (a java bean). It consists of the following elements:

- ✱ <name>
- ✱ <description>
- ✱ <object>, which defines the type and a number of <field> elements.

The service accesses the object in the following way:

```
package org.exoplatform.services.LDAP.impl;

public class LDAPServiceImpl implements LDAPService {
    ...
    public LDAPServiceImpl(InitParams params) {
        LDAPConnectionConfig config = (LDAPConnectionConfig)
        params.getObjectParam("LDAP.config")

        .getObject();
    }
    ...
}
```

The passed object is LDAPConnectionConfig, which is a Java bean. It contains all fields and also the appropriate getters and setters (not listed here). Default values can also be provided. The container creates a new instance of the bean, and calls all setters whose values are configured in the configuration file.

```
package org.exoplatform.services.LDAP.impl;

public class LDAPConnectionConfig {
    private String providerURL      = "LDAP://127.0.0.1:389";
    private String rootdn;
    private String password;
    private String version;
    private String authenticationType = "simple";
    private String serverName        = "default";
    private int    minConnection;
    private int    maxConnection;
    private String referralMode      = "follow";
    ...
}
```

The types (String, int) of the fields in the configuration correspond with the bean. The kernel_1_2.xsd file also shows some simple types:

- ✱ **string, int, long, boolean, date, double**

The object.xml resources file shows the complete list of types, :

<https://anonsvn.jboss.org/repos/exo-jcr/kernel/trunk/exo.kernel.container/src/test/resources/object.xml>.

```
<object type="package.name">
  <field name="string"><string>This is a string</string></field>
  <field name="int"><int>1234</int></field>
```

```

<field name="long"><long>123456</long></field>
<field name="double"><double>1.1234</double></field>
<field name="boolean"><boolean>true</boolean></field>
<field name="name">
  <object version="1.0" type="package.name">
    <field name="nested 1"><string>value</string></field>
    <field name="nested 2"><int>1234</int></field>
  </object>
</field>
<field name="map">
  <map type="java.util.HashMap">
    <entry>
      <key><string>a key</string></key>
      <value><string>a value</string></value>
    </entry>
    <entry>
      <key><int>1234</int></key>
      <value><string>a value</string></value>
    </entry>
  </map>
</field>
<field name="list">
  <collection type="java.util.ArrayList">
    <value><string>a value</string></value>
  </collection>
</field>
</object>

```

[Report a bug](#)

A.7.1.2.1. Collection

Java collections can be used to configure a service. Open the **database-organization-configuration.xml** file as an example configuration. This file defines a default user organization (users, groups, memberships/roles) of the portal. The configuration uses `<component-plugins>` elements, and the `<object-param>` element is also used.

There are two collections: The first collection is an **ArrayList**, which contains one value: the object which defines the field of the `NewUserConfig$JoinGroup` bean.

The second collection is a **HashSet** that is a set of strings.

```

<component-plugin>
  <name>new.user.event.listener</name>
  <set-method>addListenerPlugin</set-method>

  <type>org.exoplatform.services.organization.impl.NewUserEventListener</type>

  <description>this listener assign group and membership to a new
  created user</description>
  <init-params>
    <object-param>
      <name>configuration</name>
      <description>description</description>
    </object-param>
  </init-params>
</component-plugin>

```

```

    <object
type="org.exoplatform.services.organization.impl.NewUserConfig">
    <field name="group">
        <collection type="java.util.ArrayList">
            <value>
                <object
type="org.exoplatform.services.organization.impl.NewUserConfig$JoinGroup"
>
                    <field name="groupId">
<string>/platform/users</string></field>
                    <field name="membership"><string>member</string>
</field>
                </object>
            </value>
        </collection>
    </field>
    <field name="ignoredUser">
        <collection type="java.util.HashSet">
            <value><string>root</string></value>
            <value><string>john</string></value>
            <value><string>marry</string></value>
            <value><string>demo</string></value>
            <value><string>james</string></value>
        </collection>
    </field>
</object>
</object-param>
</init-params>
</component-plugin>

```

The org.exoplatform.services.organization.impl.NewUserConfig bean is a good example to showcase the HashSet:

```

public class NewUserConfig {
    private List    role;
    private List    group;
    private HashSet ignoredUser;

    ...

    public void setIgnoredUser(String user) {
        ignoredUser.add(user);
    }

    ...

    static public class JoinGroup {
        public String groupId;
        public String membership;
    }

    ...
}

```

The values of the HashSet are set one by one by the container. It is the responsibility of the bean to add these values to its HashSet.

The JoinGroup object is an inner class and implements a bean of its own. It can be accessed like any other inner class using NewUserConfig.JoinGroup.

[Report a bug](#)

A.7.1.3. Component Plug-in Priority

It is possible to setup loading order for ComponentPlugin directives. Use the `<priority>` element to define the plug-in load priority. All plug-ins get priority '0' by default: they are loaded in the container's normal method. To load one plug-in later than the others, set the priority for it higher than zero.

Simple example of fragment of a **configuration.xml**.

Example A.27. `<priority>` Element

```
...
<component>
  <type>org.exoplatform.services.Component1</type>
</component>

<external-component-plugins>
  <target-component>org.exoplatform.services.Component1</target-
component>

  <component-plugin>
    <name>Plugin1</name>
    <set-method>addPlugin</set-method>
    <type>org.exoplatform.services.plugins.Plugin1</type>
    <description>description</description>
    <priority>1</priority>
  </component-plugin>

  <component-plugin>
    <name>Plugin2</name>
    <set-method>addPlugin</set-method>
    <type>org.exoplatform.services.plugins.Plugin2</type>
    <description>description</description>
    <priority>2</priority>
  </component-plugin>

</external-component-plugins>

<external-component-plugins>
  <target-component>org.exoplatform.services.Component1</target-
component>
  <component-plugin>
    <name>Plugin3</name>
    <set-method>addPlugin</set-method>
    <type>org.exoplatform.services.plugins.Plugin3</type>
    <description>description</description>
  </component-plugin>
</external-component-plugins>
...
```

The plug-in 'Plugin3' is loaded first because it has the default priority '0'. Plug-in 'Plugin1' is loaded and then plug-in 'Plugin2'.

[Report a bug](#)

A.7.1.4. Configuration Logging

The JVM system property **`org.exoplatform.container.configuration.debug`** can assist with identifying problems with service configuration, by identifying the JAR/WAR that is causing issues. Add the system property to the `eXo.bat` or `eXo.sh` file in the **`exo-tomcat/bin/`** directory.

```
set EXO_CONFIG_OPTS="-Dorg.exoplatform.container.configuration.debug"
```

If this property is set, the container configuration manager reports during startup the configuration retrieval process to the standard output (System.out).

```
.....
Add configuration jar:file:/D:/Projects/eXo/dev/exo-working/exo-
tomcat/lib/exo.kernel.container-trunk.jar!/conf/portal/configuration.xml
Add configuration jar:file:/D:/Projects/eXo/dev/exo-working/exo-
tomcat/lib/exo.kernel.component.cache-
trunk.jar!/conf/portal/configuration.xml
Add configuration jndi:/localhost/portal/WEB-INF/conf/configuration.xml
import jndi:/localhost/portal/WEB-INF/conf/common/common-
configuration.xml
import jndi:/localhost/portal/WEB-INF/conf/database/database-
configuration.xml import jndi:/localhost/portal/WEB-INF/conf/ecm/jcr-
component-plugins-configuration.xml
import jndi:/localhost/portal/WEB-INF/conf/jcr/jcr-configuration.xml
.....
```

[Report a bug](#)

A.7.1.5. Import

Use the **`import`** tag to reference other configuration files from within the **`configuration.xml`** file. The imported files can be placed anywhere on the system, providing the files can be accessed by the user configured on the host machine.



Important

If a default configuration file is packaged as part of the JAR/WAR, do not import files external to the JAR/WAR.

- ✱ **war:** Imports from **`portal.war/WEB-INF`**
- ✱ **jar** or **classpath:** Uses the classloader. Use this prefix in the default configuration for importing another configuration file accessible by the classloader.
- ✱ **file:** Uses an absolute path. A URL is also a valid value for this parameter.
- ✱ **without any prefix:**

The `$JPP_HOME/gatein/gatein.ear/portal.war/WEB-INF/conf/configuration.xml` file contains import statements primarily:

```

<import>war:/conf/common/common-configuration.xml</import>
<import>war:/conf/common/logs-configuration.xml</import>
<import>war:/conf/database/database-configuration.xml</import>
<import>war:/conf/jcr/jcr-configuration.xml</import>
<import>war:/conf/common/portlet-container-configuration.xml</import>
...

```

[Report a bug](#)

A.7.1.6. System properties

System properties can be used in literal values of component configuration metadata. This makes it possible to resolve properties at runtime instead of providing a value at packaging time.

Example A.28. System Property

The `/web/portal/src/main/webapp/WEB-INF/conf/database/database-configuration.tmp1.xml` shows how system properties are used in configuration.

```

<component>
  <key>org.exoplatform.services.database.HibernateService</key>
  <jmx-name>database:type=HibernateService</jmx-name>

  <type>org.exoplatform.services.database.impl.HibernateServiceImpl</type>
  <
    <init-params>
      <properties-param>
        <name>hibernate.properties</name>
        <description>Default Hibernate Service</description>
        ...
        <property name="hibernate.connection.url"
value="${connectionUrl}"/>
        <property name="hibernate.connection.driver_class"
value="${driverClass}"/>
        <property name="hibernate.connection.username"
value="${username}"/>
        <property name="hibernate.connection.password"
value="${password}"/>
        <property name="hibernate.dialect" value="${dialect}"/>
        ...
      </properties-param>
    </init-params>
  </component>

```



Note

System properties require the -D command in the start up command: **java -DconnectionUrl=jdbc:hsqldb:file:../temp/data/exodb -DdriverClass=org.hsqldb.jdbcDriver.**

Alternatively, specify the command in the .bat or .sh portal start script, which will load the configuration each time the portal is started: **set EXO_OPTS="-DconnectionUrl=jdbc:hsqldb:file:../temp/data/exodb -DdriverClass=org.hsqldb.jdbcDriver"**

[Report a bug](#)

A.8. Specific Services

[Report a bug](#)

A.8.1. ListenerService

Inside eXo, an event mechanism allows to trigger and listen to events under specific conditions. This mechanism is used in several places in eXo such as login/logout time.

Listeners must be subclasses of **org.exoplatform.services.listener.Listener** registered by the ListenerService.

[Report a bug](#)

A.8.1.1. Configuring a Listener

All listeners are a **ComponentPlugin**. The following configuration is required:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<configuration>
...
  <external-component-plugins>
    <!-- The full qualified name of the ListenerService -->
    <target-
component>org.exoplatform.services.listener.ListenerService</target-
component>

    <component-plugin>
      <!-- The name of the listener that is also the name of the target
event -->
      <name>${name-of-the-target-event}</name>
      <!-- The name of the method to call on the ListenerService in
order to register the Listener -->
      <set-method>addListener</set-method>
      <!-- The full qualified name of the Listener -->
      <type>${the-FQN-of-the-listener}</type>
```

```

    </component-plugin>

</external-component-plugins>
</configuration>

```

[Report a bug](#)

A.8.1.2. Registering a Listener

To register a listener, call the **addListener()** method.

```

/**
 * This method is used to register a listener with the service. The
 * method
 * will: 1. Check to see if there is a list of listener with the listener
 * name, create one if the listener list doesn't exist 2. Add the new
 * listener
 * to the listener list
 *
 * @param listener
 */
public void addListener(Listener listener) {
    ...
}

```

By convention, the listener name is declared as the name of the event to listen for.

[Report a bug](#)

A.8.1.3. Triggering an Event

To trigger an event, an application can call one of the **broadcast()** methods of ListenerService.

```

/**
 * This method is used to broadcast an event. This method will: 1. Check
 * if
 * there is a list of listener that listen to the event name. 2. If there
 * is a
 * list of listener, create the event object with the given name , source
 * and
 * data 3. For each listener in the listener list, invoke the method
 * onEvent(Event)
 *
 * @param <S> The type of the source that broadcast the event
 * @param <D> The type of the data that the source object is working on
 * @param name The name of the event
 * @param source The source object instance
 * @param data The data object instance
 * @throws Exception
 */
public <S, D> void broadcast(String name, S source, D data) throws
Exception {
    ...
}

```



```

/**
 * This method is used when a developer needs to implement his own event
 * object
 * and broadcast the event. The method will: 1. Check if there is a list
 * of
 * listener that listen to the event name. 2. If there is a list of the
 * listener, For each listener in the listener list, invoke the method
 * onEvent(Event)
 *
 *
 * @param <T> The type of the event object, the type of the event object
 * has
 *          to be extended from the Event type
 * @param event The event instance
 * @throws Exception
 */
public <T extends Event> void broadcast(T event) throws Exception {
    ...
}

```

The **broadcast()** methods retrieve the name of the event, finds the registered listeners with the same name, and calls the **onEvent()** method on each listener found.

Each listener is a class that extends **org.exoplatform.services.listener.Listener**, as described below:

```

public abstract class Listener<S, D> extends BaseComponentPlugin {

    /**
     * This method should be invoked when an event with the same name is
     * broadcasted
     */
    public abstract void onEvent(Event<S, D> event) throws Exception;
}

```



Warning

Generics are used to limit the source of the event to the type 'S', and the data of the event to the type 'D'. It is expected that listeners implement the **onEvent()** method with the corresponding types.

Each listener is also a ComponentPlugin with a name and a description. Therefore, the name of the listener is the name specified in the configuration file.

```

public interface ComponentPlugin {
    public String getName();

    public void setName(String name);

    public String getDescription();

    public void setDescription(String description);
}

```

[Report a bug](#)

A.8.1.4. Asynchronous Event Broadcast

In previous releases, ListenerService stored Listeners and broadcasted events to them. ListenerService event broadcasting takes destination listeners, and executes events on those listeners.

Because some events may take longer to complete, enabling event processing asynchronous can lead to a performance improvement.

To make a listener asynchronous, mark the Listener implementation as **@Asynchronous**.

```
@Asynchronous
class AsyncListenerWithException<S,D> extends Listener<S,D>
{
    @Override
    public void onEvent(Event<S,D> event) throws Exception
    {
        // some expensive operation
    }
}
```

The AsyncListener is now executed in a separate thread by **ExecutorService**.

By default, the **ExecutorService** is configured with thread pool size 1, which can be changed in the configuration file.

```
<component>
  <key>org.exoplatform.services.listener.ListenerService</key>
  <type>org.exoplatform.services.listener.ListenerService</type>

  <init-params>
    <value-param>
      <name>asynchPoolSize</name>
      <value>5</value>
    </value-param>
  </init-params>
</component>
```

[Report a bug](#)

A.8.1.5. ListenerService Example

The **org.exoplatform.services.security.ConversationRegistry** uses the ListenerService to notify that a user has signed in or left the application. When a user signs in, the following code is called:

```
listenerService.broadcast("exo.core.security.ConversationRegistry.register", this, state);
```

This code creates a new event named **exo.core.security.ConversationRegistry.register**, the source of which is the current

instance of `ConversationRegistry`, and the data is that of the given state. The `ListenerService` calls the method `onEvent(Event<ConversationRegistry, ConversationState> event)` on all listeners with the name `exo.core.security.ConversationRegistry.register`.

Example A.29. Define a `ConversationRegistry.register` Listener

Define a Listener that will listen for the event

`exo.core.security.ConversationRegistry.register`.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<configuration>
...
  <external-component-plugins>
    <!-- The full qualified name of the ListenerService -->
    <target-
component>org.exoplatform.services.listener.ListenerService</target-
component>

    <component-plugin>
      <!-- The name of the listener that is also the name of the target
event -->
      <name>exo.core.security.ConversationRegistry.register</name>
      <!-- The name of the method to call on the ListenerService in
order to register the Listener -->
      <set-method>addListener</set-method>
      <!-- The full qualified name of the Listener -->

    <type>org.exoplatform.forum.service.AuthenticationLoginListener</type>
    </component-plugin>

  </external-component-plugins>
</configuration>
...
```

[Report a bug](#)

A.8.2. Job Scheduler

Job scheduler defines a job to execute a given number of times during a given period. It is a service that is in charge of unattended background executions, commonly known for historical reasons as batch processing. It is used to create and run jobs automatically and continuously, to schedule event-driven jobs, and reports.

Jobs are scheduled to run when a given Trigger occurs. Triggers can be created with nearly any combination of the following directives:

- ✧ at a certain time of day (to the millisecond)
- ✧ on certain days of the week
- ✧ on certain days of the month
- ✧ on certain days of the year
- ✧ not on certain days listed within a registered Calendar (such as business holidays)

- » repeated a specific number of times
- » repeated until a specific time/date
- » repeated indefinitely
- » repeated with a delay interval

Jobs are given names by their creator and can also be organized into named groups. Triggers may also be given names and placed into groups, in order to easily organize them within the scheduler. Jobs can be added to the scheduler once, but registered with multiple Triggers. Within a J2EE environment, Jobs can perform their work as part of a distributed (XA) transaction.

[Report a bug](#)

A.8.2.1. Using the Job Scheduler Service in the Kernel

The kernel leverages Quartz for its scheduler service and wraps `org.quartz.Scheduler` in `org.exoplatform.services.scheduler.impl.QuartzScheduler` for easier service wiring and configuration. To work with Quartz in Kernel, the `org.exoplatform.services.scheduler.JobSchedulerService` class (implemented by `org.exoplatform.services.scheduler.impl.JobSchedulerServiceImpl` is primarily used.

To use `JobSchedulerService`, configure it as a component in the `configuration.xml` file. Because `JobSchedulerService` requires `QuartzScheduler` and `QueueTasks`, these two components must also be configured.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.exoplaform.org/xml/ns/kernel_1_2.xsd
http://www.exoplaform.org/xml/ns/kernel_1_2.xsd"
  xmlns="http://www.exoplaform.org/xml/ns/kernel_1_2.xsd">

  <component>
    <type>org.exoplatform.services.scheduler.impl.QuartzScheduler</type>
  </component>

  <component>
    <type>org.exoplatform.services.scheduler.QueueTasks</type>
  </component>

  <component>
    <key>org.exoplatform.services.scheduler.JobSchedulerService</key>
    <type>org.exoplatform.services.scheduler.impl.JobSchedulerServiceImpl</type>
  </component>

</configuration>
```

[Report a bug](#)

A.8.2.2. Samples

Work with `JobSchedulerService` by creating a sample project, and use the portal for testing.

The kernel makes it easier to work with job scheduler service. Implement the `org.quartz.Job` interface and add the job class to perform.

Procedure A.1. Prepare Job Scheduler Project

1. Download the project code from <https://github.com/hoatle/job-scheduler-service-tutorial>
2. Create a project by using the maven archetype plug-in. Run the `mvn archetype:generate` command to create a sample project.
 - ✳ For project type: select **maven-archetype-quickstart**.
 - ✳ For groupId: select **org.exoplatform.samples**.
 - ✳ For artifactId: select **exo.samples.scheduler**.
 - ✳ For version: select **1.0.0-SNAPSHOT**.
 - ✳ For package: select **org.exoplatform.samples.scheduler**.

```
<project
  xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <parent>
    <artifactId>exo.portal.parent</artifactId>
    <groupId>org.exoplatform.portal</groupId>
    <version>3.1.0-GA</version>
  </parent>

  <groupId>org.exoplatform.samples</groupId>
  <artifactId>exo.samples.scheduler</artifactId>
  <version>1.0.0-SNAPSHOT</version>
  <name>eXo Samples For Scheduler</name>
  <description>eXo Samples Code For Scheduler</description>
</project>
```

3. Generate an eclipse project by using maven eclipse plug-in and then import into eclipse:

```
mvn eclipse:eclipse
```

[Report a bug](#)

A.8.2.2.1. Define a Job

Define the job to be performed. For example, the job `SampleJob` is defined as follows:

```
package org.exoplatform.samples.scheduler.jobs;

import org.exoplatform.services.log.ExoLogger;
import org.exoplatform.services.log.Log;
import org.quartz.Job;
import org.quartz.JobExecutionContext;
```

```
import org.quartz.JobExecutionException;

/**
 * SampleJob for executing a defined sample job.
 */
public class SampleJob implements Job {

    /**
     * The logger
     */
    private static final Log LOG = ExoLogger.getLogger(SampleJob.class);

    /**
     * The job of the SampleJob will be done by executing this method.
     *
     * @param context
     * @throws JobExecutionException
     */
    public void execute(JobExecutionContext context) throws
    JobExecutionException {
        LOG.info("SampleJob is executing...");
    }
}
```

All jobs must implement the **execute** method from **org.quartz.Job** interface. This method is called whenever a job is performed. With **SampleJob**, logging is used to verify the job is working. By looking at the terminal, the log message "SampleJob is executing..." is displayed.

[Report a bug](#)

A.8.2.2.2. Configuring a Job

After defining the job, it must be configured using the `<external-component-plugin>` configuration for **org.exoplatform.services.scheduler.JobSchedulerService**. Use the following methods for setting the component plug-in:

```
public void addPeriodJob(ComponentPlugin plug-in) throws Exception;
```

The component plug-in for this method must be **org.exoplatform.services.scheduler.PeriodJob**. This type of job is used to perform actions that are executed over a period of time, by defining when the job is performed, when it ends, when it performs the first action, how many times it is executed and the period of time to perform the action.

Example A.30. PeriodJob <component-plugin>

```
<external-component-plugins>
  <target-
component>org.exoplatform.services.scheduler.JobSchedulerService</target-
component>
    <component-plugin>
      <name>PeriodJob Plugin</name>
      <set-method>addPeriodJob</set-method>
      <type>org.exoplatform.services.scheduler.PeriodJob</type>
      <description>period job configuration</description>
```

```

<init-params>
  <properties-param>
    <name>job.info</name>
    <description>dumb job executed periodically</description>
    <property name="jobName" value="DumbJob"/>
    <property name="groupName" value="DumbJobGroup"/>
    <property name="job"
value="org.exoplatform.samples.scheduler.jobs.DumbJob"/>
    <property name="repeatCount" value="0"/>
    <property name="period" value="60000"/>
    <property name="startTime" value="+45"/>
    <property name="endTime" value=""/>
  </properties-param>
</init-params>
</component-plugin>
</external-component-plugins>

```

Example A.31. addCronJob

The component plug-in for this method must be **org.exoplatform.services.scheduler.CronJob**. This type of job is used to perform actions at a specified time, using Unix 'cron-like' definitions. The plug-in uses the expression parameter to specify the 'cron-like' definitions to execute the job.

cron-line Examples

At 12pm every day

```
"0 0 12 * * ?"
```

10:15am every Monday, Tuesday, Wednesday, Thursday and Friday

```
"0 15 10 ? * MON-FRI"
```



Note

For more information about CRON expressions, see the article at http://en.wikipedia.org/wiki/CRON_expression.

```
public void addCronJob(ComponentPlugin plugin) throws Exception;
```

```

<external-component-plugins>
  <target-
component>org.exoplatform.services.scheduler.JobSchedulerService</target-
component>
  <component-plugin>
    <name>CronJob Plugin</name>
    <set-method>addCronJob</set-method>
    <type>org.exoplatform.services.scheduler.CronJob</type>
    <description>cron job configuration</description>
    <init-params>
      <properties-param>

```

```

        <name>job.info</name>
        <description>dumb job executed by cron
expression</description>
        <property name="jobName" value="DumbJob"/>
        <property name="groupName" value="DumbJobGroup"/>
        <property name="job"
value="org.exoplatform.samples.scheduler.jobs.DumbJob"/>
        <!-- The job will be performed at 10:15am every day -->
        <property name="expression" value="0 15 10 * * ?"/>
        </properties-param>
    </init-params>
</component-plugin>
</external-component-plugins>

```

Example A.32. addGlobalJobListener and addJobListener

The component plug-in for these methods must be `org.quartz.JobListener`. This job listener is informed when `org.quartz.JobDetail` executes.

```
public void addGlobalJobListener(ComponentPlugin plugin) throws
Exception;
```

```
public void addJobListener(ComponentPlugin plugin) throws Exception;
```

Example A.33. addGlobalTriggerListener and addTriggerListener

The component plug-in for these methods must be `org.quartz.TriggerListener`. This trigger listener is informed when a `org.quartz.Trigger` executes.

```
public void addGlobalTriggerListener(ComponentPlugin plugin) throws
Exception;
```

```
public void addTriggerListener(ComponentPlugin plugin) throws
Exception;
```

[Report a bug](#)

A.8.2.2.3. Run the Project

Create a `conf.portal` package in the sample project. Add the `configuration.xml` file with the content as follows:

```

<?xml version="1.0" encoding="UTF-8"?>
<configuration
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.exoplaform.org/xml/ns/kernel_1_2.xsd
http://www.exoplaform.org/xml/ns/kernel_1_2.xsd"
  xmlns="http://www.exoplaform.org/xml/ns/kernel_1_2.xsd">

  <component>
    <type>org.exoplatform.services.scheduler.impl.QuartzSheduler</type>

```



```

</component>
<component>
  <type>org.exoplatform.services.scheduler.QueueTasks</type>
</component>
<component>
  <key>org.exoplatform.services.scheduler.JobSchedulerService</key>

<type>org.exoplatform.services.scheduler.impl.JobSchedulerServiceImpl</type>
</component>

<external-component-plugins>
  <target-
component>org.exoplatform.services.scheduler.JobSchedulerService</target-
component>
    <component-plugin>
      <name>PeriodJob Plugin</name>
      <set-method>addPeriodJob</set-method>
      <type>org.exoplatform.services.scheduler.PeriodJob</type>
      <description>period job configuration</description>
      <init-params>
        <properties-param>
          <name>job.info</name>
          <description>sample job executed periodically</description>
          <property name="jobName" value="SampleJob"/>
          <property name="groupName" value="SampleJobGroup"/>
          <property name="job"
value="org.exoplatform.samples.scheduler.jobs.SampleJob"/>
          <property name="repeatCount" value="0"/>
          <property name="period" value="60000"/>
          <property name="startTime" value="+45"/>
          <property name="endTime" value=""/>
        </properties-param>
      </init-params>
    </component-plugin>
  </external-component-plugins>
</configuration>

```

mvn clean install the project, and copy the **.jar** file to the **/lib** directory in the portal directory. Run **bin/run.sh** to see the SampleJob to be executed on the terminal when portal containers are initialized. Review the terminal to see the log message of SampleJob.

From this point on, a job can be executed in the portal by defining the job and configuring it.

[Report a bug](#)

A.8.2.3. Job Scheduler Reference

See the following links for more information about Job Scheduler:

- * <http://www.quartz-scheduler.org/>
- * http://en.wikipedia.org/wiki/Job_scheduler
- * <http://www.theserverside.com/news/1364726/Job-Scheduling-in-J2EE-Applications>
- * <http://technet.microsoft.com/en-us/library/cc720070%28WS.10%29.aspx>

[Report a bug](#)

A.8.3. The data source provider

[Report a bug](#)

A.8.3.1. DataSourceProvider

The *DataSourceProvider* is a service used to give access to a data source in an uniform manner, to support data sources that are managed by the application server.

Parameters

getDataSource(String dataSourceName)

Queries the data source from a JNDI lookup. If the datasource is found and the data source is defined as managed, the service wraps the original *DataSource* instance in a new *DataSource* instance that is aware of its *managed* state. Otherwise it will return the original *DataSource* instance.

isManaged(String dataSourceName)

Indicates whether or not the given data source is managed.

[Report a bug](#)

A.8.3.2. Configuring DataSourceProvider

Configuration for *DataSourceProvider* is defined only if managed data sources are used. All data sources are considered “not managed” by default.

```
<configuration>
....
  <component>
    <key>org.exoplatform.services.jdbc.DataSourceProvider</key>

    <type>org.exoplatform.services.jdbc.impl.DataSourceProviderImpl</type>
    <init-params>
      <!-- Indicates that the data source needs to check if a tx is
active
to decide if the provided connection needs to be managed
or not.
If it is set to false, the data source will provide only
managed connections if the data source itself is managed.
-->
      <!--value-param>
        <name>check-tx-active</name>
        <value>true</value>
      </value-param-->
      <!-- Indicates that all the data sources are managed
If set to true the parameter never-managed and
managed-data-sources will be ignored -->
      <!--value-param>
        <name>always-managed</name>
        <value>true</value>
      </value-param-->
```

```

        <!-- Indicates the list of all the data sources that are
            managed, each value tag can contain a list of
            data source names separated by a comma, in the
            example below we will register ds-foo1, ds-foo2
            and ds-foo3 as managed data source. If always-managed
            and/or never-managed is set true this parameter is ignored
        -->

        <!--values-param>
            <name>managed-data-sources</name>
            <value>ds-foo1, ds-foo2</value>
            <value>ds-foo3</value>
        </values-param-->
    </init-params>
</component>
...
</configuration>

```

Field Descriptions

<check-tx-active>

Specifies the data source must check whether a transaction is active to determine if the provided connection must be managed or not. If it is set to **false**, the data source provides only managed connections if the data source itself is managed.

By default, this parameter is set to **true**. If set to **true**, the **TransactionService** is required in the configuration.

<always-managed>

Specifies all data sources are managed. If set to **true**, the parameters <never-managed> and <managed-data-sources> are ignored (marking all the data sources as managed). This parameter is set to *false* by default.

<managed-data-sources>

Specifies the list of all data sources that are managed. Each <value> element can contain a list of data source names separated by a comma. This parameter is ignored if <always-managed> and/or <never-managed> is set to **true**.

[Report a bug](#)

A.9. Configuring a portal container

A portal container is defined by several attributes:

Portal Container Name

The URL context to which the current portal is bound.

REST Context Name

The REST name used for access to portal application. Every portal has one unique REST context name.

Realm Name

The name of the security realm used for authentication when users log into the portal.

Dependencies

List of other web applications, whose resources are visible to the current portal (through the extension mechanism described later), and are searched for in the specified order.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.exoplaform.org/xml/ns/kernel_1_2.xsd
                      http://www.exoplaform.org/xml/ns/kernel_1_2.xsd"
  xmlns="http://www.exoplaform.org/xml/ns/kernel_1_2.xsd">

  <external-component-plugins>
    <!-- The full qualified name of the PortalContainerConfig -->
    <target-
component>org.exoplaform.container.definition.PortalContainerConfig</ta
rget-component>

    <component-plugin>
      <!-- The name of the plugin -->
      <name>Add PortalContainer Definitions</name>

      <!-- The name of the method to call on the
PortalContainerConfig
in order to register the PortalContainerDefinitions -->
      <set-method>registerPlugin</set-method>

      <!-- The full qualified name of the
PortalContainerDefinitionPlugin -->

<type>org.exoplaform.container.definition.PortalContainerDefinitionPlug
in</type>

      <init-params>
        <object-param>
          <name>portal</name>
          <object
type="org.exoplaform.container.definition.PortalContainerDefinition">
            <!-- The name of the portal container -->
            <field name="name"><string>portal</string></field>

            <!-- The name of the context name of the rest web
application -->
            <field name="restContextName"><string>rest</string>
</field>

            <!-- The name of the realm -->
            <field name="realmName"><string>exo-domain</string>
</field>

            <!-- All the dependencies of the portal container
ordered by loading priority -->
            <field name="dependencies">
              <collection type="java.util.ArrayList">
                <value>
                  <string>eXoResources</string>
```

```

        </value>
        <value>
            <string>portal</string>
        </value>
        <value>
            <string>dashboard</string>
        </value>
        <value>
            <string>exoadmin</string>
        </value>
        <value>
            <string>eXoGadgets</string>
        </value>
        <value>
            <string>eXoGadgetServer</string>
        </value>
        <value>
            <string>rest</string>
        </value>
        <value>
            <string>web</string>
        </value>
        <value>
            <string>wsrp-producer</string>
        </value>
        <!-- The sample-ext has been added at the end of
the dependency list
            in order to have the highest priority -->
        <value>
            <string>sample-ext</string>
        </value>
    </collection>
</field>
</object>
</object-param>
</init-params>
</component-plugin>
</external-component-plugins>
</configuration>

```

Dependencies are part of the extension mechanism. Every **PortalContainer** is represented by a **PortalContainer instance**, which contains the following configuration directives.

Portal Container Directives

eXoContainerContext

Contains information about the portal.

Unified Servlet Context

Controls web-archive-relative resource loading.

Unified Classloader

Classpath based resource loading.

Various methods for retrieving services

The *Unified servlet context* and *unified classloader* are part of the extension mechanism.

They provide the standard API (**ServletContext**, and **ClassLoader**) with specific resource loading behavior, such as visibility into associated web application archives, configured with the dependencies property of **PortalContainerDefinition**.

Resources from other web applications are queried in the order specified by the dependencies. The later entries in the list override the previous ones.

[Report a bug](#)

A.10. System property configuration

The property configuration service is designed to allow configuration of system properties from the in-line kernel configuration, or from specified property files.

The service is scoped at the root container level because it is used by all the services in the different portal containers in the application runtime.

System Property Definitions

properties

The properties init param takes a property declared to configure various properties.

```
<component>
  <key>PropertyManagerConfigurator</key>
  <type>org.exoplatform.container.PropertyConfigurator</type>
  <init-params>
    <properties-param>
      <name>properties</name>
      <property name="foo" value="bar"/>
    </properties-param>
  </init-params>
</component>
```

properties.url

The properties.url init param specifies an external file to load by URL reference. Both property and XML formats are supported (see the javadoc of the **java.util.Properties** class for more information).

When a property file is loaded the various property declarations are loaded in the order in which the properties are declared sequentially in the file.

```
<component>
  <key>PropertyManagerConfigurator</key>
  <type>org.exoplatform.container.PropertyConfigurator</type>
  <init-params>
    <value-param>
      <name>properties.url</name>
```

```

    <value>classpath:configuration.properties</value>
  </value-param>
</init-params>
</component>

```

In the properties file corresponding to the external properties, existing variables can be reused to create a new variable. Using this feature, the prefix "*portal.container.*" is not needed. For example:

```

my-var1=value 1
my-var2=value 2
complex-value=${my-var1}-${my-var2}

```

It is possible to replace the properties URL init param by a system property that overwrites it. The name of that property is `exo.properties.url`.

All variables can be defined through two possible syntaxes:

- » **`${variable-name}`**, which does not define a default value. If the variable has not be set, the value is `${variable-name}` to indicate that it could not be resolved.
- » **`${variable-name: default-value}`**, which allows the default value to be defined after the semicolon. If the variable has not be set, the value will be the given default value.

[Report a bug](#)

A.10.1. Properties `<init-param>`

The properties init param takes a property declared to configure various properties.

```

<component>
  <key>PropertyManagerConfigurator</key>
  <type>org.exoplatform.container.PropertyConfigurator</type>
  <init-params>
    <properties-param>
      <name>properties</name>
      <property name="foo" value="bar"/>
    </properties-param>
  </init-params>
</component>

```

[Report a bug](#)

A.10.2. Properties URL `<init-param>`

The properties URL init param allow to load an external file by specifying its URL. Both property and XML format are supported, see the javadoc of the **`java.util.Properties`** class for more information. When a property file is loaded the various property declarations are loaded in the order in which the properties are declared sequentially in the file.

```

<component>
  <key>PropertyManagerConfigurator</key>
  <type>org.exoplatform.container.PropertyConfigurator</type>
  <init-params>
    <value-param>

```

```

    <name>properties.url</name>
    <value>classpath:configuration.properties</value>
  </value-param>
</init-params>
</component>

```

In the properties file corresponding to the external properties, you can reuse variables before defining to create a new variable. In this case, the prefix *portal.container.* is not needed, see an example below:

```

my-var1=value 1
my-var2=value 2
complex-value=${my-var1}-${my-var2}

```

[Report a bug](#)

A.10.3. System Property configuration of the properties URL

It is possible to replace the properties URL init param by a system property that overwrites it. The name of that property is *exo.properties.url*.

[Report a bug](#)

A.10.4. Variable Syntaxes

All the variables that we described in the previous sections can be defined through 2 possible syntaxes which are *\${variable-name}* or *\${variable-name:default-value}*. The first syntax doesn't define any default value so if the variable has not be set the value will be *\${variable-name}* to indicate that it could not be resolved. The second syntax allows you to define the default value after the semi colon so if the variable has not be set the value will be the given default value.

[Report a bug](#)

A.11. The Extension Mechanism and Portal Extensions

The Extension Mechanism makes it possible to override portal resources in a way similar to hardware plug-and-play functionality.

Customizations can be implemented without unpacking and repacking the original portal **.war** archives by adding a **.war** archive to the resources and configuring its position in the portal's classpath. Custom **.war** archives can be created with new resources that override the resources in the original archive.

These archives, packaged for use through the extension mechanism, are called *portal extensions*.

Procedure A.2. Creating a Portal Extension

1. Declare the PortalConfigOwner servlet context listener in the **web.xml** of the web application.

This example shows a portal extension called **sample-ext**.

```

<?xml version="1.0" encoding="ISO-8859-1" ?>
<!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>

  <display-name>sample-ext</display-name>

  <listener>
    <listener-
class>org.exoplatform.container.web.PortalContainerConfigOwner</lis
tener-class>
    </listener>

    ...

</web-app>

```

2. Add the application's servlet context name to the **PortalContainerDefinition**'s list of dependencies. This must be done for each portal container that requires access to the new application.

The application's position in these lists dictates its priority when the portal loads resources. The later the application appears in the list, the higher its resource priority will be.

3. The new web archive is present on both the portal's unified classpath and unified servlet context resource path.

[Report a bug](#)

A.11.1. Running Multiple Portals

It is possible to run several independent portal containers, each bound to a different URL context, within the same JVM instance.

This method of deployment allows for efficient administration and resource consumption by allowing coexisting portals to reuse configuration arrangements through the extension mechanism.

Portals can inherit resources and configuration from existing web archives and add extra resources as needed, overriding those that need to be changed by including modified copies.

In order for a portal application to function correctly when deployed within a multiple portal deployment, it may have to dynamically query the information about the current portal container. The application must not make any assumptions about the name, and other information of the current portal, as there are now multiple different portals in play.

At any point during request processing, or life-cycle event processing, an application can retrieve this information through **org.exoplatform.container.eXoContainerContext**.

Sometimes an application must ensure that the proper **PortalContainer** is associated with the current **eXoContainerContext** call.

If the portal application contains servlets or servlet filters that need to access portal specific resources during their request processing, the servlet or filter must be associated with the current container.

A servlet in this instance must extend the **org.exoplatform.container.web.AbstractHttpServlet** class so as to properly initialize the current **PortalContainer**.

This will also set the current thread's context Classloader to one that looks for resources in associated web applications in the order specified by the dependencies configuration.

Filter classes need to extend the `org.exoplatform.container.web.AbstractFilter`.

Both **AbstractHttpServlet**, and **AbstractFilter** have a `getContainer()` method, which returns the current **PortalContainer**.

If your servlet handles the requests by implementing a `service()` method, that method must be renamed to match the following signature:

```
/**
 * Use this method instead of Servlet.service()
 */
protected void onService(ExoContainer container, HttpServletRequest req,
    HttpServletResponse res) throws ServletException, IOException;
```



Note

This ensures that **AbstractHttpServlet**'s `service()` interception is not overwritten.

An application may also need access to portal information within the **HttpSessionListener**. Ensure the abstract class `org.exoplatform.container.web.AbstractHttpSessionListener` is extended.

In this instance, modify the method signatures as follows:

```
/**
 * Use this method instead of HttpSessionListener.sessionCreated()
 */
protected void onSessionCreated(ExoContainer container, HttpSessionEvent
event);

/**
 * Use this method instead of HttpSessionListener.sessionDestroyed()
 */
protected void onSessionDestroyed(ExoContainer container,
HttpSessionEvent event);
```

Another method must also be implemented in this case:

```
/**
 * Method should return true if unified servlet context,
 * and unified classloader should be made available
 */
protected boolean requirePortalEnvironment();
```

If this method returns true the current thread's context Classloader is set up according to the dependencies configuration and availability of the associated web applications.

If it returns false the standard application separation rules are used for resource loading (effectively turning off the extension mechanism).

This method exists on both **AbstractHttpServlet** and **AbstractFilter**. This is a default implementation that automatically returns true when it detects there is a current **PortalContainer** present and false otherwise.

ServletContextListener-based initialization access to PortalContainer

The portal has no direct control over the deployment of application archives (.war and .ear files); it is the application server that performs the deployment.

However, applications in the dependencies configuration must be deployed before the portal that depends on them is initialized in order for the extension mechanism to work properly.

Conversely, some applications may require an already initialized **PortalContainer** to properly initialize themselves. This gives rise to a recursive dependency problem.

A mechanism of initialization tasks and task queues has been implemented in the portal to resolve this dependency issue.

Web applications that depend on a current **PortalContainer** to initialize must avoid performing their initialization directly on a **ServletContextListener** executed during their deployment (before any **PortalContainer** was initialized).

To ensure this, a web application must package its initialization logic into an **init** task of an appropriate type and only use **ServletContextListener** to insert the **init** task instance into the proper **init** tasks queue.

An example of this is the Gadgets application which registers Google gadgets with the current **PortalContainer**. This example uses **PortalContainerPostInitTask** which is executed after the portal container has been initialized.

```
public class GadgetRegister implements ServletContextListener
{
    public void contextInitialized(ServletContextEvent event)
    {
        // Create a new post-init task
        final PortalContainerPostInitTask task = new
PortalContainerPostInitTask() {

            public void execute(ServletContext context, PortalContainer
portalContainer)
            {
                try
                {
                    SourceStorage sourceStorage =
(SourceStorage)
portalContainer.getComponentInstanceOfType(SourceStorage.class);
                    ...
                }
                catch (RuntimeException e)
                {
                    throw e;
                }
                catch (Exception e)
                {
                    throw new RuntimeException("Initialization failed: ", e);
                }
            }
        };

        // Add post-init task for execution on all the portal containers
        // that depend on the given ServletContext according to
```

```
// PortalContainerDefinitions (via Dependencies configuration)
PortalContainer.addInitTask(event.getServletContext(), task);
}
}
```

In some situations initialization may be required after the portal container is instantiated but before it has initialized. **PortalContainerPreInitTask** can be used in that case.

Use **PortalContainerPostCreateTask** if initialization is required after all the **post-init** tasks have been executed.

LoginModules

If some custom **LoginModules** require the current **eXoContainer** for initialization ensure they extend **org.exoplatform.services.security.jaas.AbstractLoginModule**.

AbstractLoginModule enforces some basic configuration. It recognizes two initialization options; `portalContainerName` and `realmName`.

The values for these options can be accessed through protected fields of the same name.

See Also:

✱ [Section A.11, “The Extension Mechanism and Portal Extensions”](#)

[Report a bug](#)

A.12. Manageability

[Report a bug](#)

A.12.1. Introduction

The kernel has a framework for exposing a *management view* of the various sub systems of the platform.

The management view is a portal-specific term that means “defining how we can access relevant information about the system and how we can apply management operations”.

JMX is the default standard for exposing a management view in the Java Platform, but we take in consideration other kind of views such as REST web services. Therefore, the framework is not tied to JMX, yet it provides a JMX part to define more precisely details related to the JMX management view.

The legacy framework is still in use but is deprecated in favor of the new framework, as it is less tested and less efficient.

[Report a bug](#)

A.12.2. Managed framework API

The managed frameworks define an API for exposing a management view of objects. The API is targeted for internal use and is not a public API. The framework leverages Java 5 annotations to describe the management view from an object.

[Report a bug](#)

A.12.2.1. Annotations

The following annotations are available for use in the eXo Kernel.

Annotations List

@org.exoplatform.management.annotations.Managed annotation

The @Managed annotates elements that wants to expose a management view to a management layer.

- ✦ The @Managed annotation on objects exports a management view for the objects annotated.
- ✦ The @Managed annotation on getters and setters defines a managed property. An annotated getter defines a read property, an annotated setter defines a write property. If a matching getter or setter is annotated, the annotation defines a read-write property.
- ✦ The @Managed annotation on methods defines a managed operation.

@org.exoplatform.management.annotations.ManagedDescription

The @ManagedDescription annotation provides a description of a managed element. It is valid to annotated object or methods. It takes as sole argument a string that is the description value.

@org.exoplatform.management.annotations.ManagedName

The @ManagedName annotation provides an alternative name for managed properties. It is used to accomodate legacy methods of an object that can be renamed for compatibility reasons. It takes as sole argument a string that is the name value.

@org.exoplatform.management.annotations.ManagedBy

The @ManagedBy annotation defines a delegate class for exposing a management view. The sole argument of the annotation are class literals. The delegate class must provide a constructor with the managed object as argument.

[Report a bug](#)

A.12.2.1.1. @org.exoplatform.management.annotations.Managed annotation

The @Managed annotates elements that wants to expose a management view to a management layer.

@Managed for objects

The framework will export a management view for the objects annotated.

@Managed for getter/setter

Defines a managed property. An annotated getter defines a read property, an annotated setter defines a write property and if matching getter/setter are annotated it defines a read/write property.

@Managed on method

Defines a managed operation.

[Report a bug](#)

A.12.2.1.2. `@org.exoplatform.management.annotations.ManagedDescription`

The `@ManagedDescription` annotation provides a description of a managed element. It is valid to annotated object or methods. It takes as sole argument a string that is the description value.

[Report a bug](#)

A.12.2.1.3. `@org.exoplatform.management.annotations.ManagedName`

The `@ManagedName` annotation provides an alternative name for managed properties. It is used to accomodate legacy methods of an object that can be renamed for compatibility reasons. It takes as sole argument a string that is the name value.

[Report a bug](#)

A.12.2.1.4. `@org.exoplatform.management.annotations.ManagedBy`

The `@ManagedBy` annotation defines a delegate class for exposing a management view. The sole argument of the annotation are class literals. The delegate class must provide a constructor with the managed object as argument.

[Report a bug](#)

A.12.3. JMX Management View

[Report a bug](#)

A.12.3.1. JMX Annotations

The following JMX annotations are available for use in eXo Kernel.

JMX Annotations

`@org.exoplatform.management.jmx.annotations.Property`

The `@Property` annotation is used to within other annotations such as `@NameTemplate` or `@NamingContext`. It should be seen as a structural way for a list of properties. A property is made of a key and a value. The value can either be a string literal or it can be surrounded by curly braces (`{}`) to a dynamic property. A dynamic property is resolved against the instance of the object at runtime.

`@org.exoplatform.management.jmx.annotations.NameTemplate`

The `@NameTemplate` defines a template used at registration time of a managed object to create the JMX object name. The template is populated with properties.

```
@NameTemplate({
    @Property(key="container", value="workspace"),
    @Property(key="name", value="{Name}")})
```

`@org.exoplatform.management.jmx.annotations.NamingContext`

The `@NamingContext` annotation defines a set of properties which are used within a management context. It allows to propagate properties down to managed objects which are defined by an object implementing the `ManagementAware` interface. The goal is to scope different instances of the same class that would have the same object name otherwise.

```
@NamingContext(@Property(key="workspace", value="{Name}"))
```

[Report a bug](#)

A.12.3.1.1. @org.exoplatform.management.jmx.annotations.Property annotation

The @Property annotation is used to within other annotations such as @NameTemplate or @NamingContext. It should be seen as a structural way for a list of properties. A property is made of a key and a value. The value can either be a string literal or it can be surrounded by curly brace to be a dynamic property. A dynamic property is resolved against the instance of the object at runtime.

[Report a bug](#)

A.12.3.1.2. @org.exoplatform.management.jmx.annotations.NameTemplate annotation

The @NameTemplate defines a template that is used at registration time of a managed object to create the JMX object name. The template is formed of properties.

```
@NameTemplate({
    @Property(key="container", value="workspace"),
    @Property(key="name", value="{Name}")})
```

[Report a bug](#)

A.12.3.1.3. @org.exoplatform.management.jmx.annotations.NamingContext annotation

The @NamingContext annotation defines a set of properties which are used within a management context. It allows to propagate properties down to managed objects which are defined by an object implementing the ManagementAware interface. The goal is to scope different instances of the same class that would have the same object name otherwise.

```
@NamingContext(@Property(key="workspace", value="{Name}"))
```

[Report a bug](#)

A.12.4. Example

[Report a bug](#)

A.12.4.1. CacheService example

The cache service delegates most of the work to the CacheServiceManaged class by using the @ManagedBy annotation. At runtime when a new cache is created, it calls the CacheServiceManaged class in order to let the CacheServiceManaged object register the cache.

```
@ManagedBy(CacheServiceManaged.class)
public class CacheServiceImpl implements CacheService {

    CacheServiceManaged managed;
    ...
    synchronized private ExoCache createCacheInstance(String region)
    throws Exception {
        ...
        if (managed != null) {
```

```

        managed.registerCache(simple);
    }
    ...
}
}

```

The ExoCache interface is annotated to define its management view. The @NameTemplate is used to produce object name values when ExoCache instance are registered.

```

@Managed
@NameTemplate({@Property(key="service", value="cache"),
@Property(key="name", value="{Name}")})
@ManagedDescription("Exo Cache")
public interface ExoCache {

    @Managed
    @ManagedName("Name")
    @ManagedDescription("The cache name")
    public String getName();

    @Managed
    @ManagedName("Capacity")
    @ManagedDescription("The maximum capacity")
    public int getMaxSize();

    @Managed
    @ManagedDescription("Evict all entries of the cache")
    public void clearCache() throws Exception;

    ...
}

```

The CacheServiceManaged is the glue code between the CacheService and the management view. The main reason is that only exo services are registered automatically against the management view. Any other managed bean must be registered manually for now. Therefore, it needs to know about the management layer via the management context. The management context allows an object implementing the ManagementAware interface to receive a context to perform further registration of managed objects.

```

@Managed
public class CacheServiceManaged implements ManagementAware {

    /** . */
    private ManagementContext context;

    /** . */
    private CacheServiceImpl cacheService;

    public CacheServiceManaged(CacheServiceImpl cacheService) {
        this.cacheService = cacheService;

        //
        cacheService.managed = this;
    }
}

```



```
public void setContext(ManagementContext context) {  
    this.context = context;  
}  
  
void registerCache(ExoCache cache) {  
    if (context != null) {  
        context.register(cache);  
    }  
}  
}
```

[Report a bug](#)

eXo JCR

[Report a bug](#)

B.1. Introduction and Support Scope



eXo JCR usage

The portal is using a JCR API to store its information for internal usage. Red Hat does not support the JCR as an application information store.

The information contained in this appendix is provided to assist users to understand particular low level details on how the portal works, and how it can be fine-tuned.

The term JCR refers to the Java Content Repository. The JCR is the data store of the portal. All content is stored and managed using the JCR.

The eXo JCR included with the portal is a JSR-170 compliant implementation of the JCR 1.0 specification. The JCR provides the following features:

- » Revision control
- » Textual search
- » Access control
- » Content event monitoring
- » Text and binary data for internal portal usage

The back-end storage for the JCR can be provisioned as a file system or a database.

[Report a bug](#)

B.2. Concepts

[Report a bug](#)

B.2.1. Repository and Workspace

Repository

A repository is a form of data storage device. A repository differs from a database in the nature of the information contained. While a database holds hard data in rigid tables, a repository may access the data on a database by using less rigid meta-data. So, a repository operates as an interpreter between the database(s) and the user.



Note

The data model for the interface (the repository) is rarely the same as the data model used by the repository's underlying storage subsystems (such as a database) but the repository is able to make persistent data changes in the storage subsystem.

Workspace

The eXo JCR uses workspaces as the main data abstraction in its data model. The content is stored in a workspace as a hierarchy of items and each workspace has its own hierarchy of items.

Repositories access one or more workspaces. Persistent JCR workspaces consist of a directed acyclic graph of items where the edges represent the parent-child relation.

[Report a bug](#)

B.2.2. Items

An item is either a node or a property. Properties contain the data. Data can be simple values or binary data. The nodes give structure to a workspace and the properties hold the data.

Nodes

Nodes are identified using accepted namespacing conventions. Changed nodes are versioned through an associated version graph. This type of versioning helps to preserve data integrity.

Nodes can have various properties or child nodes associated to them.

Properties

Properties hold data as values of predefined types, such as, String, Binary, Long, Boolean, Double, Date, Reference and Path.

[Report a bug](#)

B.2.3. The Data Model

The core of any Content Repository is the data model. The data model defines the data elements such as fields, columns, attributes, and the relationships between these elements. These data elements are stored in the Content Repository.

Data elements can be singular pieces of information, for example the value 3.14, or compound values $\pi = 3.14$. A data model uses concepts like nodes, arrays and links to define relationships between data elements.

The use and structure of these elements forms the content repository's data model.

[Report a bug](#)

B.2.4. Data Abstraction

Data abstraction describes the separation between abstract and concrete properties of data, stored in a repository.

The concrete properties of the data implementation may be changed without affecting the abstract properties of the data, which are read by the data client.

Consider the presentation of data in a list, graph or table. While the information implementation may change, the data itself is unaffected, and readers to whom the data is presented can perform a mental abstraction to interpret it correctly, regardless of the implementation.

[Report a bug](#)

B.3. eXo JCR Repository Service Components

eXo JCR Implementation consist of an eXoContainer. The eXoContainer consist of Repository Service, Repository and the Workspace.

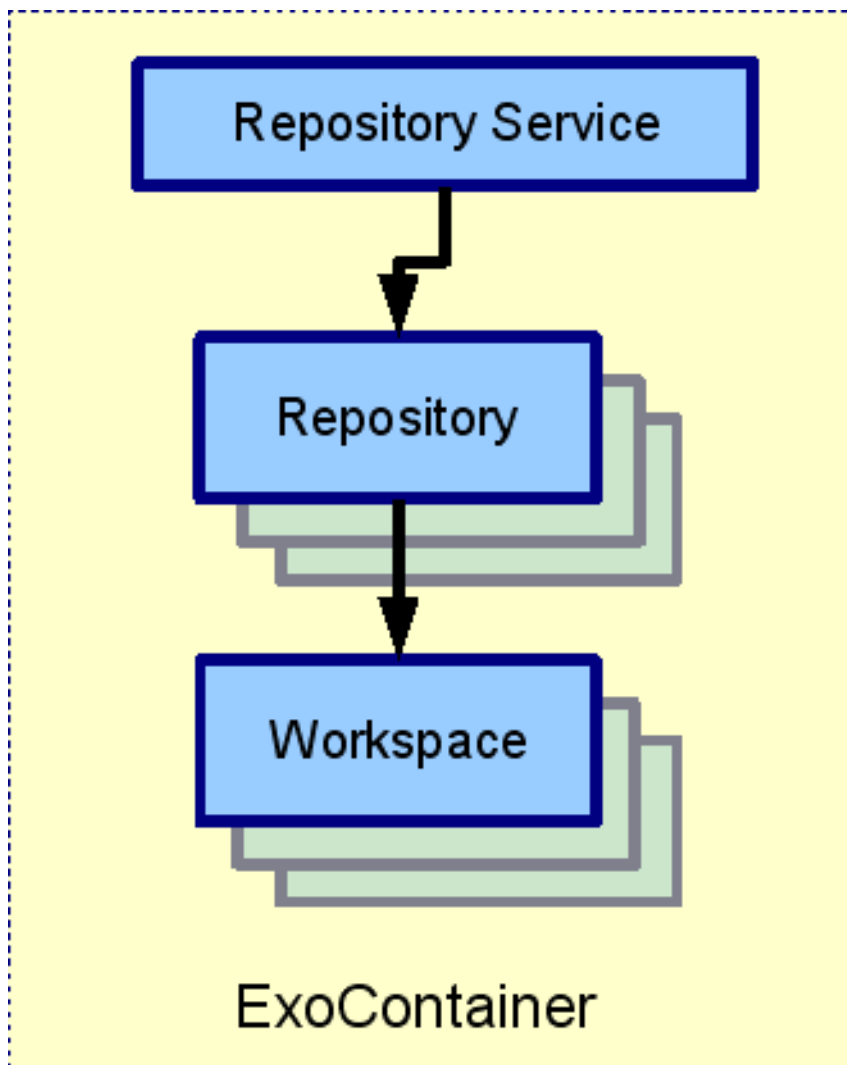


Figure B.1. eXo JCR Repository Service Components

Repository Service Component Definitions

ExoContainer:

A subclass of *org.exoplatform.container.ExoContainer* (*org.exoplatform.container.PortaContainer*) holds a reference to the Repository Service.

Repository Service

This contains information about repositories. eXo JCR is able to manage many Repositories.

Repository

An implementation of `javax.jcr.Repository`. It holds references to one or more Workspace(s).

Workspace

Container of a single rooted tree of Items.



Note

An ExoContainer Workspace is not the same as a `javax.jcr.Workspace`: it is not a per-session object.

The usual JCR application use-case includes two initial steps:

1. Obtain a repository object by getting Repository Service through a JNDI lookup if an eXo repository is bound to the naming context.
2. Create a `javax.jcr.Session` *object* that calls `Repository.login(..)`.

See Also:

» [Section B.4, "Template for JCR configuration file"](#)

[Report a bug](#)

B.3.1. Workspace Data Model

The Workspace working model shown here explains the components of eXo JCR implementation, that are used in the data flow to perform operations specified in the JCR API.

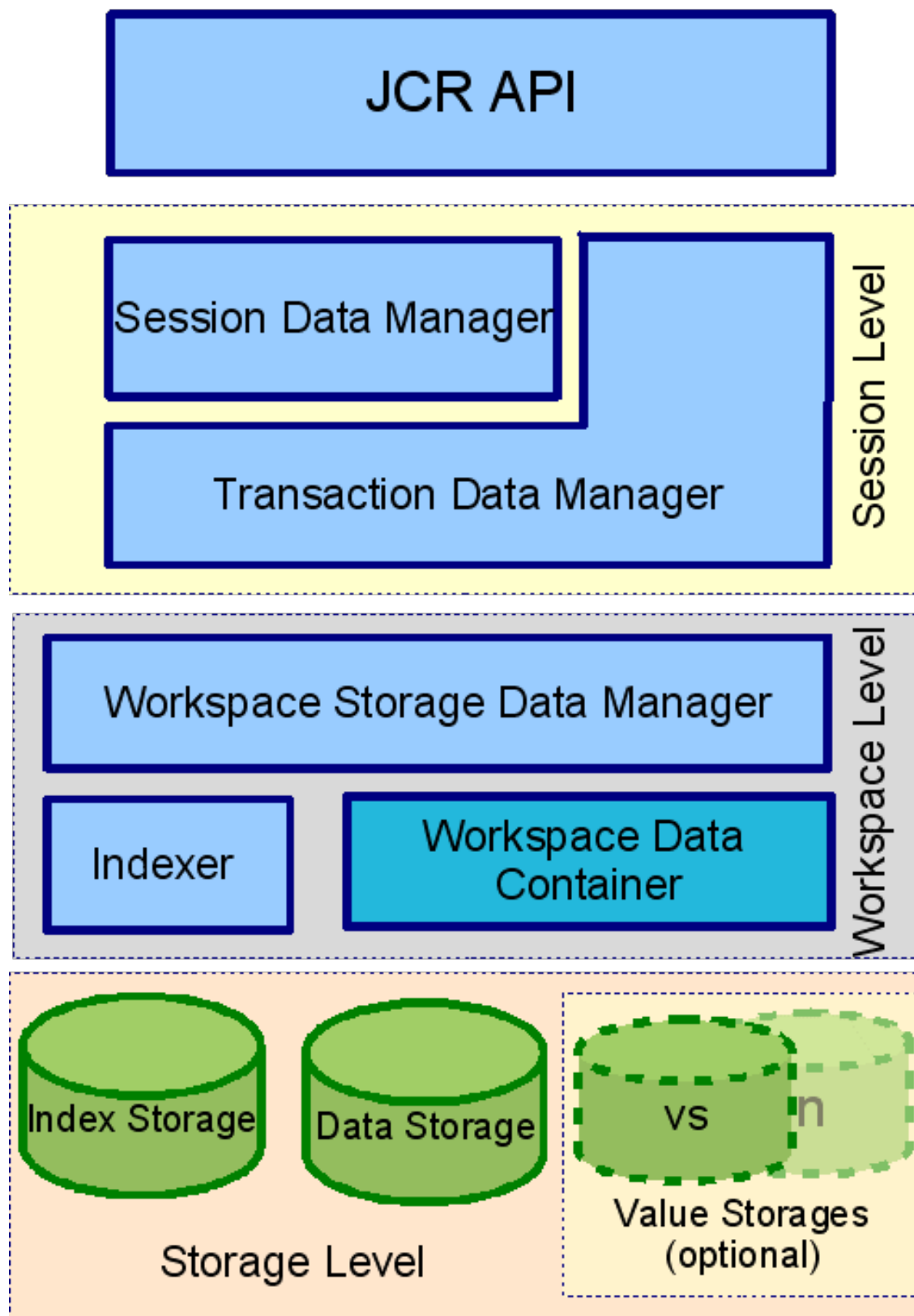


Figure B.2. Workspace Data Model

The Workspace Data Model has four levels. These levels are created based on data isolation and value from the JCR model perspective.

JCR API

The eXo JCR core implements JCR API interfaces. The JCR API interface includes Item, Node, Property and JCR logical view on stored data.

Session Level

Session Level isolates transient data viewable inside one JCR Session and interacts with API level using eXo JCR internal API.

Session Data Manager

Session Data Manager maintains transient session data. With data access/ modification/ validation logic, it contains Modified Items Storage to hold the data changed between subsequent **save()** calling and Session Items Cache.

Transaction Data Manager

Transaction Data Manager maintains session data between **save()** and transaction commit/ rollback if the current session is part of a transaction.

Workspace Level

Workspace Level operates for particular workspace shared data. It contains per-Workspace objects

Workspace Storage Data Manager

Workspace Storage Data Manager maintains workspace data, including final validation, events firing and caching.

Workspace Data Container

Workspace Data Container implements physical data storage. It allows different types of backend, such as RDB, FS files, and so on to be used as storage for JCR data. Along with the main Data Container, other storages for persisted Property Values can be configured and used.

Indexer

Indexer maintains workspace data indexing for further queries.

Storage Level

Storage Level provides persistent storages for:

- ✧ JCR Data
- ✧ Indexes (Apache Lucene)
- ✧ Values for BLOBs if they are different from the main Data Container

[Report a bug](#)

B.4. Template for JCR configuration file

The JCR configuration is defined in an XML file which is constructed as per the DTD show here:

```
!ELEMENT repository-service (repositories)
!ATTLIST repository-service default-repository NMTOKEN #REQUIRED
!ELEMENT repositories (repository)
!ELEMENT repository (security-domain,access-control,session-max-
age,authentication-policy,workspaces)
!ATTLIST repository
```

```

default-workspace NMTOKEN #REQUIRED
name NMTOKEN #REQUIRED
system-workspace NMTOKEN #REQUIRED
!ELEMENT security-domain (#PCDATA)
!ELEMENT access-control (#PCDATA)
!ELEMENT session-max-age (#PCDATA)
!ELEMENT authentication-policy (#PCDATA)
!ELEMENT workspaces (workspace+)
!ELEMENT workspace (container,initializer,cache,query-handler)
!ATTLIST workspace name NMTOKEN #REQUIRED
!ELEMENT container (properties,value-storages)
!ATTLIST container class NMTOKEN #REQUIRED
!ELEMENT value-storages (value-storage+)
!ELEMENT value-storage (properties,filters)
!ATTLIST value-storage class NMTOKEN #REQUIRED
!ELEMENT filters (filter+)
!ELEMENT filter EMPTY
!ATTLIST filter property-type NMTOKEN #REQUIRED
!ELEMENT initializer (properties)
!ATTLIST initializer class NMTOKEN #REQUIRED
!ELEMENT cache (properties)
!ATTLIST cache
    enabled NMTOKEN #REQUIRED
    class NMTOKEN #REQUIRED

!ELEMENT query-handler (properties)
!ATTLIST query-handler class NMTOKEN #REQUIRED
!ELEMENT access-manager (properties)
!ATTLIST access-manager class NMTOKEN #REQUIRED
!ELEMENT lock-manager (time-out,persister)
!ELEMENT time-out (#PCDATA)
!ELEMENT persister (properties)
!ELEMENT properties (property+)
!ELEMENT property EMPTY

```

[Report a bug](#)

B.4.1. Portal Configuration for JCR

JCR services are registered in the Portal container.

The DTD show here is an example configuration from the file `jcr-configuration.xml`. The configuration file is located at `$JPP_HOME/gatein/gatein.ear/portal.war/WEB-INF/conf/jcr/`.

```

<component>
  <key>org.exoplatform.services.jcr.RepositoryService</key>
  <type>org.exoplatform.services.jcr.impl.RepositoryServiceImpl</type>
  <component-plugins>
    <component-plugin>
      <name>add.namespaces</name>
      <set-method>addPlugin</set-method>
      <type>org.exoplatform.services.jcr.impl.AddNamespacesPlugin</type>
      <init-params>
    <properties-param>
      <name>namespaces</name>

```



```

    <property name="test" value="http://www.apache.org/jackrabbit/test"/>
    <property name="exojcrtest"
value="http://www.exoplatform.org/jcr/test/1.0"/>
    <property name="rma" value="http://www.rma.com/jcr"/>
    <property name="metadata"
value="http://www.exoplatform.com/jcr/metadata/1.1"/>
    <property name="dc" value="http://purl.org/dc/elements/1.1"/>
    <property name="publication"
value="http://www.exoplatform.com/jcr/publication/1.1"/>
  </properties-param>
  </init-params>
</component-plugin>
<component-plugin>
  <name>add.nodeType</name>
  <set-method>addPlugin</set-method>
  <type>org.exoplatform.services.jcr.impl.AddNodeTypePlugin</type>
  <init-params>
<values-param>
  <name>autoCreatedInNewRepository</name>
  <description>Node types configuration file</description>
  <value>jar:/conf/test/nodetypes-tck.xml</value>
  <value>jar:/conf/test/nodetypes-impl.xml</value>
  <value>jar:/conf/test/nodetypes-usecase.xml</value>
  <value>jar:/conf/test/nodetypes-config.xml</value>
  <value>jar:/conf/test/nodetypes-config-extended.xml</value>
  <value>jar:/conf/test/wcm-nodetypes.xml</value>
  <value>jar:/conf/test/nodetypes-publication-config.xml</value>
  <value>jar:/conf/test/publication-plugins-nodetypes-config.xml</value>
</values-param>

<values-param>
  <name>testInitNodeTypesRepository</name>
  <description>
    Node types configuration file for repository with name
testInitNodeTypesRepository
  </description>
  <value>jar:/conf/test/nodetypes-test.xml</value>
</values-param>

<values-param>
  <name>testInitNodeTypesRepositoryTest2</name>
  <description>
    Node types configuration file for repository with name
testInitNodeTypesRepositoryTest2
  </description>
  <value>jar:/conf/test/nodetypes-test2.xml</value>
</values-param>

<!--values-param>
  <name>testInitNodeTypesRepositoryTest3</name>
  <description>Node types from ext. Needed because core starup earlie than
ext</description>
  <value>jar:/conf/test/nodetypes-test3_ext.xml</value>
</values-param-->

  </init-params>

```

```

    </component-plugin>
  </component-plugins>
</component>

<component>

<key>org.exoplatform.services.jcr.config.RepositoryServiceConfiguration</key>

<type>org.exoplatform.services.jcr.impl.config.RepositoryServiceConfigurationImpl</type>
  <init-params>
    <value-param>
      <name>conf-path</name>
      <description>JCR configuration file</description>
      <value>jar:/conf/standalone/test-jcr-config-jbc.xml</value>
    </value-param>
    <properties-param>
      <name>working-conf</name>
      <description>working-conf</description>
      <property name="dialect" value="auto" />
      <property name="source-name" value="jdbcjcr"/>
      <property name="persister-class-name"
value="org.exoplatform.services.jcr.impl.config.JDBCConfigurationPersister"/>
    </properties-param>
  </init-params>
</component>

```

[Report a bug](#)

B.4.1.1. JCR Workspace Configuration

The JCR Service can use multiple Repositories and each repository can have multiple Workspaces. To configure the workspaces, you need to edit the repository configuration file.

Configure the workspaces by locating the workspace you need to modify. The workspace is found in **\$JPP_HOME/gatein/gatein.ear/portal.war/WEB-INF/conf/jcr/repository-configuration.xml**.

The repository configuration supports human-readable value as they are not case-sensitive.

Complete the appropriate element fields using the following value formats:

Number formats

- ✧ **K** or **KB** for kilobytes.
- ✧ **M** or **MB** for megabytes.
- ✧ **G** or **GB** for gigabytes.
- ✧ **T** or **TB** for terabytes.
- ✧ Examples: 200K or 200KB; 4M or 4MB; 1.4G or 1.4GB; 10T or 10TB.

Time formats:

- **ms** for milliseconds.
- **s** for seconds.
- **m** for minutes.
- **h** for hours.
- **d** for days.
- **w** for weeks.
- The default time format is seconds.
- Examples: 500ms or 500 milliseconds; 20, 20s or 20 seconds; 30m or 30 minutes; 12h or 12 hours; 5d or 5 days; 4w or 4 weeks.

[Report a bug](#)

B.4.1.2. JCR Repository Service Configuration

The JCR Repository service configuration includes declaring repository names, workspace names, security domain name, access control policy, authentication policy name, and so on.

```
<!-- Comment #1 -->
<repository-service default-repository="repository">
  <!-- The list of repositories. -->
  <repositories>
    <!-- Comment #2. -->
    <repository name="repository" system-workspace="system" default-
workspace="portal-system">
      <!-- Comment #3 -->
      <security-domain>gatein-domain</security-domain>
      <!-- Comment #4 -->
      <access-control>optional</access-control>
      <!-- Comment #5 -->
      <authentication-
policy>org.exoplatform.services.jcr.impl.core.access.JAASAuthenticator</a
uthentication-policy>
      ...
    <!-- Comment #6 -->
    <workspaces>
      ...
      <!-- Comment #7 -->
      <workspace name="portal-system">
        <!-- Comment #8 -->
        <container
class="org.exoplatform.services.jcr.impl.storage.jdbc.optimisation.CQJDBC
WorkspaceDataContainer">
          <properties>
            <property name="source-name"
value="${gatein.jcr.datasource.name}${container.name.suffix}"/>
            <property name="dialect"
value="${gatein.jcr.datasource.dialect}"/>
            <property name="multi-db" value="false"/>
            <property name="update-storage" value="true"/>
            <property name="max-buffer-size" value="204800"/>
            <property name="swap-directory"
```

```

value="${gatein.jcr.data.dir}/swap/portal-
system${container.name.suffix}"/>
    </properties>
    <value-storages>
        <value-storage id="portal-system"
class="org.exoplatform.services.jcr.impl.storage.value.fs.TreeFileValueSt
orage">
            <properties>
                <property name="path"
value="${gatein.jcr.storage.data.dir}/portal-
system${container.name.suffix}"/>
            </properties>
            <filters>
                <filter property-type="Binary"/>
            </filters>
        </value-storage>
    </value-storages>
</container>
<!-- Comment #9 -->
<initializer
class="org.exoplatform.services.jcr.impl.core.ScratchWorkspaceInitializer
">
    <properties>
        <property name="root-nodetype" value="nt:unstructured"/>
        <property name="root-permissions"
value="*:/platform/administrators read;*:/platform/administrators
add_node;*:/platform/administrators
set_property;*:/platform/administrators remove"/>
    </properties>
</initializer>
<!-- Comment #10 -->
<cache enabled="true"
class="org.exoplatform.services.jcr.impl.dataflow.persistent.jbosscache.JB
ossCacheWorkspaceStorageCache">
    <properties>
        <property name="jbosscache-configuration"
value="${gatein.jcr.cache.config}" />
        <property name="jgroups-configuration"
value="${gatein.jcr.jgroups.config}" />
        <property name="jgroups-multiplexer-stack" value="true" />
        <property name="jbosscache-cluster-name" value="jcr-
${container.name.suffix}-portal-system" />
    </properties>
</cache>
<!-- Comment #11 -->
<query-handler
class="org.exoplatform.services.jcr.impl.core.query.lucene.SearchIndex">
    <properties>
        <property name="index-dir"
value="${gatein.jcr.index.data.dir}/portal-
system${container.name.suffix}"/>
        <property name="changesfilter-class"
value="${gatein.jcr.index.changesfilterclass}" />
        <property name="jbosscache-configuration"
value="${gatein.jcr.index.cache.config}" />
        <property name="jgroups-configuration"

```

```

value="${gatein.jcr.jgroups.config}" />
    <property name="jgroups-multiplexer-stack" value="true" />
    <property name="jboss-cache-cluster-name"
value="jcrindexer-${container.name.suffix}-portal-system" />
    <property name="max-volatile-time" value="60" />
    </properties>
</query-handler>
<lock-manager
class="org.exoplatform.services.jcr.impl.core.lock.jboss-cache.CacheableLo
ckManagerImpl">
    <properties>
        <!-- Comment #12 -->
        <property name="time-out" value="15m" />
        <property name="jboss-cache-configuration"
value="${gatein.jcr.lock.cache.config}" />
        <property name="jgroups-configuration"
value="${gatein.jcr.jgroups.config}" />
        <property name="jgroups-multiplexer-stack" value="true" />
        <property name="jboss-cache-cluster-name" value="jcrlock-
${container.name.suffix}-portal-system" />
        <property name="jboss-cache-cl-cache.jdbc.table.name"
value="jcrlock_portal_system" />
        <property name="jboss-cache-cl-cache.jdbc.table.create"
value="true" />
        <property name="jboss-cache-cl-cache.jdbc.table.drop"
value="false" />
        <property name="jboss-cache-cl-cache.jdbc.table.primarykey"
value="pk" />
        <property name="jboss-cache-cl-cache.jdbc.fqn.column"
value="fqn" />
        <property name="jboss-cache-cl-cache.jdbc.node.column"
value="node" />
        <property name="jboss-cache-cl-cache.jdbc.parent.column"
value="parent" />
        <property name="jboss-cache-cl-cache.jdbc.datasource"
value="${gatein.jcr.datasource.name}${container.name.suffix}" />
    </properties>
</lock-manager>
</workspace>

```

Comment #1

The name of the default repository, the one returned by **RepositoryService.getRepository()**.

Comment #2

The parameters are name of a repository, the name of workspace where **/jcr:system** node is placed and the name of the workspace obtained using Session's **login()** or **login(Credentials)** methods (the ones without an explicit workspace name).

Comment #3

The name of a security domain for JAAS authentication

Comment #4

The name of an access control policy. There can be 3 types: optional - ACL is created on-

demand(default), disable - no access control, mandatory - an ACL is created for each added node(not supported yet).

Comment #5

The name of an authentication policy class

Comment #6

The list of workspaces

Comment #7

The name of the workspace

Comment #8

Workspace data container (physical storage) configuration

Comment #9

Workspace initializer configuration

Comment #10

Workspace storage cache configuration

Comment #11

Query handler configuration

Comment #12

The amount of time before the unused global lock is removed.



session-max-age

session-max-age This parameter is not shown in the example file above as it is optional. The session-max-age sets the time after which an idle session is removed (called logout). If it is not set up, an idle session will never be removed.

lock-remover-max-threads denotes the number of threads that can serve LockRemover tasks. The default value is of **lock-remover-max-threads** 1. Each workspace has a LockManager. JCR supports Locks with defined lifetime. The LockRemover removes a lock on expiration. LockRemovers is not an independent timer-thread, its a task that is executed every 30 seconds. Such a task is served by ThreadPoolExecutor which may use different number of threads.

[Report a bug](#)

B.4.1.3. Workspace Configuration Parameters

name

The name of a workspace.

auto-init-root-nodetype

DEPRECATED in JCR 1.9 (use initializer). The node type for root node initialization.

container

Workspace data container (physical storage) configuration.

initializer

Workspace initializer configuration.

cache

Workspace storage cache configuration.

query-handler

Query handler configuration.

auto-init-permissions

DEPRECATED in JCR 1.9 (use initializer). Default permissions of the root node. It is defined as a set of semicolon-delimited permissions containing a group of space-delimited identities and the type of permission.

For example, any read; **:/admin read**;:/admin add_node; **:/admin set_property**;:/admin remove means that users from group **admin** have all permissions and other users have only a 'read' permission.

[Report a bug](#)

B.4.1.4. Workspace Data Container Configuration Parameters**class**

A workspace data container class name.

properties

The list of properties (name-value pairs) for the concrete Workspace data container.

Parameter Description**trigger_events_for_descendents_on_rename**

Indicates the need to trigger events for descendants on rename or not. It allows to increase performance on rename operation but in same time Observation is not notified. The default value is true

lazy-node-iterator-page-size

The page size for lazy iterator. Indicates how many nodes can be retrieved from storage per request. The default value is 100

acl-bloomfilter-false-positive-probability

ACL Bloom-filter desired false positive probability. Range [0..1]. Default value 0.1d.

acl-bloomfilter-elements-number

Expected number of ACL-elements in the Bloom-filter. Default value 1000000.

**Note**

Bloom filters are not supported by all the cache implementations so far only the implementation for infinispn supports it.

value-storage

The list of value storage plug-ins.

[Report a bug](#)

B.4.1.5. Value Storage plug-in Configuration Parameters**Note**

The value-storage element is optional. If you do not include it, the values will be stored as BLOBs inside the database.

value-storage

Optional value Storage plug-in definition.

class

A value storage plug-in class name (attribute).

properties

The list of properties (name-value pairs) for a concrete Value Storage plug-in.

filters

The list of filters defining conditions when this plug-in is applicable.

[Report a bug](#)

B.4.1.6. Initializer Configuration Parameters**class**

Initializer implementation class.

properties

The list of properties (name-value pairs).

root-nodetype

The node type for root node initialization.

root-permissions

Default permissions of the root node. It is defined as a set of semicolon-delimited permissions containing a group of space-delimited identities such as user, group, and the type of permission. For example any read; **:/admin read; :/admin**

add_node;:/admin set_property;:/admin remove means that users from group admin have all permissions and other users have only a read permission.

Configurable initializer adds a capability to override workspace initial startup procedure used for Clustering. It also replaces workspace element parameters **auto-init-root-nodetype** and **auto-init-permissions** with **root-nodetype** and **root-permissions**.

[Report a bug](#)

B.4.1.7. Cache Configuration Parameters

enabled

Checks if workspace cache is enabled or not.

class

Cache implementation class. The default value is **org.exoplatform.services.jcr.impl.dataflow.persistent.LinkedWorkspaceStorageCacheImpl**. This parameter is optional from JCR version 1.9.

Cache can be configured to use concrete implementation of WorkspaceStorageCache interface. JCR core has two implementation as:

- **LinkedWorkspaceStorageCacheImpl**, which is the default implementation with configurable read behavior and statistic.
- **WorkspaceStorageCacheImpl** was used pre 1.9 and can still be used.

properties

The list of properties (name-value pairs) for Workspace cache.

max-size

Cache maximum size (maxSize prior to v.1.9).

live-time

Cached item live time (liveTime prior to v.1.9).

From 1.9 **LinkedWorkspaceStorageCacheImpl** supports additional optional parameters.

statistic-period

Period (time format) of cache statistic thread execution, the default value is 5 minutes.

statistic-log

If true cache statistic will be printed to default logger (log.info),

statistic-clean

,

cleaner-period

Time period to remove the oldest items. the default time is 20 minutes.

blocking-users-count

Number of concurrent users allowed to read cache storage. The default value is 0 which means unlimited.

[Report a bug](#)

B.4.1.8. Query Handler Configuration Parameters**class**

A Query Handler class name.

properties

The list of properties (name-value pairs) for a Query Handler (indexDir).

Properties and advanced features described in Search Configuration.

[Report a bug](#)

B.4.1.9. Lock Manager Configuration Parameters**time-out**

Time after which the unused global lock will be removed.

persister

A class for storing lock information for future use. For example, remove lock after jcr restart.

path

A lock folder. Each workspace has its own lock folder.

[Report a bug](#)

B.5. Multi-language Support

Whenever a relational database is used to store multilingual text data in the eXo Java Content Repository the configuration must be adapted to support UTF-8 encoding. Dialect is automatically detected for certified database. You can still enforce it in case of failure.

The following sections describe enabling UTF-8 support with various databases.

**Note**

- ✧ The configuration file to be modified for these changes is **`$JPP_HOME/gatein/gatein.ear/portal.war/WEB-INF/conf/jcr/repository-configuration.xml`**.
- ✧ The datasource **`jdbcjcr`** used in the following examples can be configured via the **`InitialContextInitializer`** component.

[Report a bug](#)

B.5.1. Oracle

In order to run multilanguage JCR on an Oracle backend Unicode encoding for characters set should be applied to the database. Other Oracle globalization parameters do not have any effect. The property to modify is **NLS_CHARACTERSET**.

The **NLS_CHARACTERSET = AL32UTF8** entry has been successfully tested with many European and Asian languages.

Example of database configuration:

| | |
|-------------------------|------------------------------|
| NLS_LANGUAGE | AMERICAN |
| NLS_TERRITORY | AMERICA |
| NLS_CURRENCY | \$ |
| NLS_ISO_CURRENCY | AMERICA |
| NLS_NUMERIC_CHARACTERS | ., |
| NLS_CHARACTERSET | AL32UTF8 |
| NLS_CALENDAR | GREGORIAN |
| NLS_DATE_FORMAT | DD-MON-RR |
| NLS_DATE_LANGUAGE | AMERICAN |
| NLS_SORT | BINARY |
| NLS_TIME_FORMAT | HH.MI.SSXFF AM |
| NLS_TIMESTAMP_FORMAT | DD-MON-RR HH.MI.SSXFF AM |
| NLS_TIME_TZ_FORMAT | HH.MI.SSXFF AM TZR |
| NLS_TIMESTAMP_TZ_FORMAT | DD-MON-RR HH.MI.SSXFF AM TZR |
| NLS_DUAL_CURRENCY | \$ |
| NLS_COMP | BINARY |
| NLS_LENGTH_SEMANTICS | BYTE |
| NLS_NCHAR_CONV_EXCP | FALSE |
| NLS_NCHAR_CHARACTERSET | AL16UTF16 |

Create database with Unicode encoding and use Oracle dialect for the Workspace Container:

```
<workspace name="collaboration">
  <container
class="org.exoplatform.services.jcr.impl.storage.jdbc.JDBCWorkspaceDataCo
ntainer">
    <properties>
      <property name="source-name" value="jdbcjcr" />
      <property name="dialect" value="oracle" />
      <property name="multi-db" value="false" />
      <property name="max-buffer-size" value="200k" />
      <property name="swap-directory"
value="target/temp/swap/ws" />
    </properties>
    . . . . .
```



Warning

JCR does not use NVARCHAR columns, so the value of the parameter NLS_NCHAR_CHARACTERSET does not matter for JCR.

[Report a bug](#)

B.5.2. DB2

DB2 Universal Database (DB2 UDB) supports [UTF-8 and UTF-16/UCS-2](#). When a Unicode database is created, **CHAR**, **VARCHAR** and **LONG VARCHAR** data are stored in UTF-8 form.

This enables JCR multi-lingual support.

Below is an example of creating a UTF-8 database using the **db2** dialect for a workspace container with DB2 version 9 and higher:

```
DB2 CREATE DATABASE dbname USING CODESET UTF-8 TERRITORY US
```

```
<workspace name="collaboration">
  <container
class="org.exoplatform.services.jcr.impl.storage.jdbc.JDBCWorkspaceDataCo
ntainer">
    <properties>
      <property name="source-name" value="jdbcjcr" />
      <property name="dialect" value="db2" />
      <property name="multi-db" value="false" />
      <property name="max-buffer-size" value="200k" />
      <property name="swap-directory"
value="target/temp/swap/ws" />
    </properties>
    . . . . .
```



Note

For DB2 version 8.x support change the property "dialect" to db2v8.

[Report a bug](#)

B.5.3. MySQL

To use JCR with MySQL database requires the dialect, [MySQL-UTF8](#) for internationalization support.

The default charset for the database is **latin1**. **Latin1** allows to use limited index space effectively, for example 1000 bytes for **MyISAM** engine and 767 for **InnoDB**.



Note

If the default charset of the database is multibyte, you will face a JCR database initialization error is concerning index creation failure.

JCR works on any single byte default charset of database, with UTF8 supported by MySQL server. However, it has only been tested using the **latin1** charset.

An example:

```
<workspace name="collaboration">
  <container
```

```

class="org.exoplatform.services.jcr.impl.storage.jdbc.JDBCWorkspaceDataCo
ntainer">
    <properties>
        <property name="source-name" value="jdbcjcr" />
        <property name="dialect" value="mysql-utf8" />
        <property name="multi-db" value="false" />
        <property name="max-buffer-size" value="200k" />
        <property name="swap-directory"
value="target/temp/swap/ws" />
    </properties>
    . . . . .

```

You need to indicate the charset name either at server level using the server parameter - - **character-set-server** or at datasource configuration level by adding a new property as shown here:

```

<property name="connectionProperties"
value="useUnicode=yes;characterEncoding=utf8;characterSetResults=UTF-8;"
/>

```

[Report a bug](#)

B.5.4. PostgreSQL

Multilingual support can be enabled with PostgreSQL in the following ways:

1. To use the locale features of the operating system to provide locale-specific collation order, number formatting, translated messages, and so on.

UTF-8 is widely used on Linux distributions and can be useful in such a scenario.

2. To provide different character sets defined in the PostgreSQL server, including multiple-byte character sets, to support storing text in any language and providing character set translation between client and server.

Using UTF-8 database charset is recommended as it allows any-to-any conversations and make this issue transparent for the JCR.

Example of a database with UTF-8 encoding using PgSQL dialect for the Workspace Container:

```

<workspace name="collaboration">
    <container
class="org.exoplatform.services.jcr.impl.storage.jdbc.JDBCWorkspaceDataCo
ntainer">
        <properties>
            <property name="source-name" value="jdbcjcr" />
            <property name="dialect" value="pgsql" />
            <property name="multi-db" value="false" />
            <property name="max-buffer-size" value="200k" />
            <property name="swap-directory"
value="target/temp/swap/ws" />
        </properties>
        . . . . .

```

[Report a bug](#)

B.6. Configuring Search

The search function in JCR can be configured to perform in specific ways. This section discusses configuring the search function to improve search performance and results.

The JCR index configuration file is located at

`$JPP_HOME/gatein/gatein.ear/portal.war/WEB-INF/conf/jcr/repository-configuration.xml`.

A code example is listed here with a list of the configuration parameters shown below:

```
<repository-service default-repository="db1">
  <repositories>
    <repository name="db1" system-workspace="ws" default-workspace="ws">
      ....
      <workspaces>
        <workspace name="ws">
          ....
          <query-handler
class="org.exoplatform.services.jcr.impl.core.query.lucene.SearchIndex">
            <properties>
              <property name="index-dir"
value="${java.io.tmpdir}/temp/index/db1/ws" />
              <property name="synonymprovider-class"
value="org.exoplatform.services.jcr.impl.core.query.lucene.PropertiesSyno
nymProvider" />
              <property name="synonymprovider-config-path"
value="/synonyms.properties" />
              <property name="indexing-config-path" value="/indexing-
configuration.xml" />
              <property name="query-class"
value="org.exoplatform.services.jcr.impl.core.query.QueryImpl" />
            </properties>
          </query-handler>
          ...
        </workspace>
      </workspaces>
    </repository>
  </repositories>
</repository-service>
```

The table outlines some of the Configuration Parameters. The table lists the default setting and the version of eXo JCR.

Table B.1. Configuration parameters

| Parameter | Default Setting | Description | JCR Version |
|------------------|-----------------|--|-------------|
| index-dir | none | The location of the index directory. This parameter is mandatory. It is called " indexDir " in versions prior to eXo JCR version 1.9. | 1.0 |
| use-compoundfile | true | Advises lucene to use compound files for the index files. | 1.9 |

| Parameter | Default Setting | Description | JCR Version |
|------------------------|-------------------|--|-------------|
| min-merge-docs | 100 | The minimum number of nodes in an index until segments are merged. | 1.9 |
| volatile-idle-time | 3 | Idle time in seconds until the volatile index part is moved to a persistent index even though minMergeDocs is not reached. | 1.9 |
| max-merge-docs | Integer.MAX_VALUE | The maximum number of nodes in segments that will be merged. The default value changed to Integer.MAX_VALUE in eXo JCR version 1.9. | 1.9 |
| merge-factor | 10 | Determines how often segment indices are merged. | 1.9 |
| max-field-length | 10000 | The number of words that are full-text indexed at most per property. | 1.9 |
| cache-size | 1000 | Size of the document number cache. This cache maps UUID to lucene document numbers. | 1.9 |
| force-consistencycheck | false | Runs a consistency check on every start up. If false, a consistency check is only performed when the search index detects a prior forced shutdown. | 1.9 |
| auto-repair | true | Errors detected by a consistency check are automatically repaired. If false, errors are only written to the log. | 1.9 |
| query-class | QueryImpl | <p>Classname that implements the javax.jcr.query.Query interface.</p> <p>This class must also extend from the class: org.exoplatform.services.jcr.impl.core.query.AbstractQueryImpl.</p> | 1.9 |
| document-order | true | If true and the query does not contain an 'order by' clause, result nodes will be in document order. For better performance set to 'false' when queries return many nodes. | 1.9 |
| result-fetch-size | Integer.MAX_VALUE | The number of results when a query is executed. Default value: Integer.MAX_VALUE. | 1.9 |

| Parameter | Default Setting | Description | JCR Version |
|------------------------------|---------------------------|---|-------------|
| excerptprovider-class | DefaultXMLExcerpt | The name of the class that implements org.exoplatform.services.jcr.impl.core.query.lucene.ExcerptProvider . This should be used for the rep:excerpt() function in a query. | 1.9 |
| support-highlighting | false | If set to true additional information is stored in the index to support highlighting using the rep:excerpt() function. | 1.9 |
| synonymprovider-class | none | The name of a class that implements org.exoplatform.services.jcr.impl.core.query.lucene.SynonymProvider . The default value is null. | 1.9 |
| synonymprovider-config-path | none | The path to the synonym provider configuration file. This path is interpreted relative to the path parameter. If there is a path element inside the SearchIndex element, then this path is interpreted relative to the root path of the path. Whether this parameter is mandatory depends on the synonym provider implementation. The default value is null. | 1.9 |
| indexing-configuration-path | none | The path to the indexing configuration file. | 1.9 |
| indexing-configuration-class | IndexingConfigurationImpl | The name of the class that implements org.exoplatform.services.jcr.impl.core.query.lucene.IndexingConfiguration . | 1.9 |
| force-consistencycheck | false | If set to true a consistency check is performed depending on the parameter forceConsistencyCheck . If set to false no consistency check is performed on start up, even if a redo log had been applied. | 1.9 |
| spellchecker-class | none | The name of a class that implements org.exoplatform.services.jcr.impl.core.query.lucene.SpellChecker . | 1.9 |
| errorlog-size | 50(KB) | The default size of error log file in KB. | 1.9 |

| Parameter | Default Setting | Description | JCR Version |
|---------------|--|--|-------------|
| upgrade-index | false | <p>Allows JCR to convert an existing index into the new format. It is also possible to set this property via system property.</p> <p>Indexes prior to eXo JCR 1.12 will not run with eXo JCR 1.12. You must run an automatic migration.</p> <p>Start eXo JCR with:</p> <div style="border: 1px solid #ccc; border-radius: 10px; padding: 5px; display: inline-block; margin: 10px 0;">-Dupgrade - index=true</div> <p>The old index format is then converted in the new index format. After the conversion the new format is used.</p> <p>On subsequent starts this option is no longer needed. The old index is replaced and a back conversion is not possible</p> <p>It is recommended that a backup of the index be made before conversion. (Only for migrations from JCR 1.9 and later.)</p> | 1.12 |
| analyzer | org.apache.lucene.analysis.standard.StandardAnalyzer | Class name of a lucene analyzer to use for full-text indexing of text. | 1.12 |

See Also:

✱ [Section B.6.1, “Global Search Index”](#)

[Report a bug](#)

B.6.1. Global Search Index

eXo JCR uses the Lucene standard Analyzer to index content. The Lucene standard Analyzer uses some standard filters, to analyze the content.

Example B.1. Standard Analyzed Filters

```
public TokenStream tokenStream(String fieldName, Reader reader) {
    StandardTokenizer tokenizer = new StandardTokenizer(reader,
        replaceInvalidAcronym);
```

```

    tokenStream.setMaxTokenLength(maxTokenLength);
    // Comment #1
    TokenStream result = new StandardFilter(tokenStream);
    // Comment #2
    result = new LowerCaseFilter(result);
    // Comment #3
    result = new StopFilter(result, stopSet);
    return result;
}

```

Comment #1: The first filter (StandardFilter) removes possessive apostrophes ('s) from the end of words and removes periods (.) from acronyms.

Comment #2: The second filter (LowerCaseFilter) normalizes token text to lower case.

Comment #3: The third filter (StopFilter) removes stop words from a token stream. The stop set is defined in the analyzer.

The global search index is configured in the **\$JPP_HOME/gatein/gatein.ear/portal.war/WEB-INF/conf/jcr/repository-configuration.xml** configuration file within the "query-handler" tag.

```

<query-handler
  class="org.exoplatform.services.jcr.impl.core.query.lucene.SearchIndex">

```

The same analyzer must be used for indexing and for querying in lucene otherwise results may be unpredictable. eXo JCR does this automatically. The StandardAnalyzer is the default analyzer. You can replace it with another analyzer.

[Report a bug](#)

B.6.1.1. Customized Search Indexes and Analyzers

eXo JCR uses the Lucene standard Analyzer to index contents. The Lucene standard analyzer uses some standard filters to analyze the content:

```

public TokenStream tokenStream(String fieldName, Reader reader) {
    StandardTokenizer tokenStream = new StandardTokenizer(reader,
    replaceInvalidAcronym);
    tokenStream.setMaxTokenLength(maxTokenLength);
    TokenStream result = new StandardFilter(tokenStream);
    result = new LowerCaseFilter(result);
    result = new StopFilter(result, stopSet);
    return result;
}

```

- The first filter (StandardFilter) removes ('s) from the end of words and removes dots from acronyms.
- The second filter (LowerCaseFilter) normalizes token text to lower case.
- The third filter (StopFilter) removes stop words from a token stream. The stop set is defined in the analyzer.

Additional filters are used in specific cases. For example, the ISOLatin1AccentFilter filter replaces accented characters in the ISO Latin 1 character set (ISO-8859-1) by their unaccented equivalents.

**Note**

The `ISOLatin1AccentFilter` is not present in the current lucene version used by eXo.

[Report a bug](#)

B.6.1.2. Creating a Customized Query Handler

To use a different filter, a new analyzer and a new search index must be created. A new search index is created to use the analyzer. These are packaged into a jar file, which is then deployed with the application.

Procedure B.1. Create a new filter, analyzer and search index

1. Create a new filter.

```
public final Token next(final Token reusableToken) throws
java.io.IOException
```

This defines how characters are read and used by the filter.

2. Create the analyzer.

The analyzer must extend

`org.apache.lucene.analysis.standard.StandardAnalyzer` and overload the method.

Use the following to use new filters.

```
public TokenStream tokenStream(String fieldName, Reader reader)
```

3. To create the new search index, extend **`org.exoplatform.services.jcr.impl.core.query.lucene.SearchIndex`** and write the constructor to set the correct analyzer.

Use the method below to return your analyzer:

```
public Analyzer getAnalyzer() {
return MyAnalyzer;
}
```

**Note**

In eXo JCR version 1.12 and later the analyzer can be directly during configuration. For users using this version, the creation of a new `SearchIndex` for new analyzers is redundant.

[Report a bug](#)

B.6.1.3. Configuring an application to use the new SearchIndex

To configure an application to use a new **`SearchIndex`**, replace the following code:

```
<query-handler
class="org.exoplatform.services.jcr.impl.core.query.lucene.SearchIndex">
```

in `$JPP_HOME/gatein/gatein.ear/portal.war/WEB-INF/conf/jcr/repository-configuration.xml` file with the new class:

```
<query-handler class="mypackage.indexation.MySearchIndex">
```

To configure an application to use a new analyzer, add the ***analyzer*** parameter to each query-handler configuration in `$JPP_HOME/gatein/gatein.ear/portal.war/WEB-INF/conf/jcr/repository-configuration.xml` file:

```
<query-handler
class="org.exoplatform.services.jcr.impl.core.query.lucene.SearchIndex">
  <properties>
    ...
    <property name="analyzer"
value="org.exoplatform.services.jcr.impl.core.MyAnalyzer"/>
    ...
  </properties>
</query-handler>
```

The new **SearchIndex** starts to index content with the specified filters once JCR is restarted.

[Report a bug](#)

B.6.2. Indexing Configuration

JCR version 1.9 onwards the default search index implementation in JCR allows the user to control the properties of a node that are indexed. Different analyzers can also be set for different nodes.

The configuration parameter for indexing is called **indexingConfiguration**. The indexing parameter is not set by default, which means all properties of a node are indexed.

To configure the indexing behavior, you have to add a parameter to the ***query-handler*** element in your configuration file as shown here:

```
<param name="indexing-configuration-path"
value="/indexing_configuration.xml"/>
```

[Report a bug](#)

B.6.2.1. Node Scope Limit

You can limit the scope of a node such that certain properties of a node type are indexed. This is useful to optimize the index size.

The configuration shown here indexes two properties named, **Text** for ***nt:unstructured*** node types. This configuration applies to all the nodes type extending from ***nt:unstructured***.

```
<?xml version="1.0"?>
```

```
<!DOCTYPE configuration SYSTEM "http://www.exoplatform.org/dtd/indexing-configuration-1.0.dtd">
<configuration xmlns:nt="http://www.jcp.org/jcr/nt/1.0">
  <index-rule nodeType="nt:unstructured">
    <property>Text</property>
  </index-rule>
</configuration>
```



Namespace Prefixes

The namespace prefixes must be declared throughout the XML file in the configuration element that is being used.

[Report a bug](#)

B.6.2.2. Configuring Index Boost Value

JCR allows to configure a boost value for the nodes that match the index rule. The default boost value is 1.0. Higher boost values in the range of 1.0 - 5.0 will yield a higher score value and appear as more relevant.

Example B.2. Configuring index boost value

```
<?xml version="1.0"?>
<!DOCTYPE configuration SYSTEM
"http://www.exoplatform.org/dtd/indexing-configuration-1.0.dtd">
<configuration xmlns:nt="http://www.jcp.org/jcr/nt/1.0">
  <index-rule nodeType="nt:unstructured"
              boost="2.0">
    <property>Text</property>
  </index-rule>
</configuration>
```

To configure certain properties, you can provide a boost value for the listed properties:

```
<?xml version="1.0"?>
<!DOCTYPE configuration SYSTEM "http://www.exoplatform.org/dtd/indexing-configuration-1.0.dtd">
<configuration xmlns:nt="http://www.jcp.org/jcr/nt/1.0">
  <index-rule nodeType="nt:unstructured">
    <property boost="3.0">Title</property>
    <property boost="1.5">Text</property>
  </index-rule>
</configuration>
```

[Report a bug](#)

B.6.2.3. Adding Condition to Index Rules

You can add a condition to the index rule and have multiple rules with the same nodeType. The first index rule that matches will apply and all remaining ones are ignored:

Example B.3. Adding condition to index nodes

```
<?xml version="1.0"?>
<!DOCTYPE configuration SYSTEM
"http://www.exoplatform.org/dtd/indexing-configuration-1.0.dtd">
<configuration xmlns:nt="http://www.jcp.org/jcr/nt/1.0">
  <index-rule nodeType="nt:unstructured"
             boost="2.0"
             condition="@priority = 'high'">
    <property>Text</property>
  </index-rule>
  <index-rule nodeType="nt:unstructured">
    <property>Text</property>
  </index-rule>
</configuration>
```

In the above example the first rule applies if the **nt:unstructured** node has a priority property with a value **high**. The condition syntax only supports the equals operator and a string literal.

Example B.4. Referencing properties

```
<?xml version="1.0"?>
<!DOCTYPE configuration SYSTEM
"http://www.exoplatform.org/dtd/indexing-configuration-1.0.dtd">
<configuration xmlns:nt="http://www.jcp.org/jcr/nt/1.0">
  <index-rule nodeType="nt:unstructured"
             boost="2.0"
             condition="ancestor::*/@priority = 'high'">
    <property>Text</property>
  </index-rule>
  <index-rule nodeType="nt:unstructured"
             boost="0.5"
             condition="parent::foo/@priority = 'low'">
    <property>Text</property>
  </index-rule>
  <index-rule nodeType="nt:unstructured"
             boost="1.5"
             condition="bar/@priority = 'medium'">
    <property>Text</property>
  </index-rule>
  <index-rule nodeType="nt:unstructured">
    <property>Text</property>
  </index-rule>
</configuration>
```

The indexing configuration allows to specify the type of a node in the condition.

Example B.5. Specify node type in the condition

```
<?xml version="1.0"?>
<!DOCTYPE configuration SYSTEM
```

```
"http://www.exoplatform.org/dtd/indexing-configuration-1.0.dtd">
<configuration xmlns:nt="http://www.jcp.org/jcr/nt/1.0">
  <index-rule nodeType="nt:unstructured"
              boost="2.0"
              condition="element(*, nt:unstructured)/@priority =
'high'">
    <property>Text</property>
  </index-rule>
</configuration>
```



Note

The type match of the node must be exact. It does not consider sub types of the specified node type.

[Report a bug](#)

B.6.2.4. Exclusion from the Node Scope Index

All configured properties of type string and included in the node scope index are full-text indexed by default.

A node scope search normally finds all nodes of an index. For example, `jcr:contains(., 'foo')` returns all nodes that have a string property containing the word 'foo'.

Example B.6. Excluding properties explicitly from the node scope index

```
<?xml version="1.0"?>
<!DOCTYPE configuration SYSTEM
"http://www.exoplatform.org/dtd/indexing-configuration-1.0.dtd">
<configuration xmlns:nt="http://www.jcp.org/jcr/nt/1.0">
  <index-rule nodeType="nt:unstructured">
    <property nodeScopeIndex="false">Text</property>
  </index-rule>
</configuration>
```

[Report a bug](#)

B.6.2.5. Characteristics of Node Scope Searches

There are chances of unexpected behavior while using analyzers to search within a property as compared to searching within a node scope, because the node scope always uses the global analyzer.

Example B.7. Query

For example, the property `"mytext"` contains the text, testing my analyzers but no analyzers have been configured for this property and the default analyzer in `SearchIndex` has not changed.

```
xpath = "//*[jcr:contains(mytext, 'analyzer')]"
```

where, **xpath** does not return a result in the node with the and default analyzers.

Example B.8. Searching in node scope

If a search is done on the node scope, no result is returned.

```
xpath = "//*[jcr:contains(., 'analyzer')]"
```

Only specific analyzers can be set on a node property. The node scope indexing and analyzing is done with the globally defined analyzer in the SearchIndex element.

Example B.9. Changing the analyzer from the Global Analyzer to German Analyzer

```
<analyzer class="org.apache.lucene.analysis.Analyzer.GermanAnalyzer">
<property>mytext</property>
</analyzer>
```

When the global analyzer used to index the property *mytext* is changed to a specific analyzer, for example German Analyzer, the search query return a result due to word stemming.

Example B.10. Search query after changing the analyzer

This search query returns a result because of the word stemming, for example, analyzers - analyzer.

```
xpath = "//*[jcr:contains(mytext, 'analyzer')]"
```

Example B.11. Search query after changing the analyzer

This search query will not give a result, since the node scope is indexed with the global analyzer. Because global analyzer ignores word stemming.

```
xpath = "//*[jcr:contains(., 'analyzer')]"
```



Warning

While using analyzers for specific properties, a result may be found in a property for certain search text, but the same search text in the node scope of the property may not find a result.



Note

Both index rules and index aggregates influence how content is indexed in JCR. If the configuration is changed, the existing content is not automatically re-indexed according to the new rules.

Content must be manually re-indexed when the configuration is changed.

[Report a bug](#)

B.6.3. Advanced features

eXo JCR supports some advanced features, which are not specified in JSR 170:

- ✳ Get a text excerpt with highlighted words that matches the query.
- ✳ Search a term and its synonyms.
- ✳ Search similar nodes.
- ✳ Check spelling of a full text query statement.
- ✳ Define index aggregates and rules for indexing configuration.

See Also:

- ✳ [Section B.13.9.1, “DefaultXMLExcerpt”](#)
- ✳ [Section B.13.8.3, “Synonym Search”](#)
- ✳ [Section B.13.11, “Similarity”](#)
- ✳ [Section B.13.10, “SpellChecker”](#)

[Report a bug](#)

B.7. Configuring the JDBC Data Container

[Report a bug](#)

B.7.1. Introduction

eXo JCR persistent data container can work in two configuration modes:

- ✳ Multi-database: One database for each workspace (used in standalone eXo JCR service mode).
- ✳ Single-database: All workspaces persist in one database (used in embedded eXo JCR service mode, for example, eXo portal).

The data container uses the JDBC driver to communicate with the actual database software, i.e. any JDBC-enabled data storage can be used with eXo JCR implementation.

[Report a bug](#)

B.7.2. Supported Databases

The data container is tested with the following RDBMS:

Table B.2. Supported databases

| Database | Driver Version |
|--------------------------------|---|
| IBM DB2 9.7 (FP5) | IBM DB2 JDBC Universal Driver Architecture 4.13.80 |
| Oracle 11g R1 (11.1.0.7.0) | Oracle JDBC Driver 11.1.0.7 |
| Oracle 11g R1 RAC (11.1.0.7.0) | Oracle JDBC Driver 11.1.0.7 |
| Oracle 11g R2 (11.2.0.3.0) | Oracle JDBC Driver v11.2.0.3.0 |
| Oracle 11g R2 RAC (11.2.0.3.0) | Oracle JDBC Driver v11.2.0.3.0 |
| MySQL 5.1 | MySQL Connector/J 5.1.21 |
| MySQL 5.5 | MySQL Connector/J 5.1.21 |
| Microsoft SQL Server 2008 | Microsoft SQL Server JDBC Driver 3.0.1301.101, Microsoft SQL Server JDBC Driver 4.0.2206.100 |
| Microsoft SQL Server 2008 R2 | Microsoft SQL Server JDBC Driver 3.0.1301.101, Microsoft SQL Server JDBC Driver 4.0.2206.100 |
| PostgreSQL 8.4.8 | JDBC4 Postgresql Driver, Version 8.4-703 |
| PostgreSQL 9.1.0 | JDBC4 Postgresql Driver, Version 9.1-903 |
| Sybase ASE 15.7 | Sybase jConnect JDBC driver v7 |



Isolation Levels

JCR requires at the parameter **READ_COMMITTED**, to read the isolation level. This parameter and other RDBMS configurations can cause issues. So, ensure that the proper isolation level is configured on database server side.



Note

A mandatory JCR requirement for underlying databases is case sensitive collation. Microsoft SQL Server 2005 and 2008 customers must configure their server with collation corresponding to operational requirements, while honoring case sensitivity. See the Microsoft SQL Server documentation page titled *Selecting a SQL Server Collation* at <http://msdn.microsoft.com/en-us/library/ms144250.aspx>



Warning

JCR does not support MyISAM storage engine for the MySQL relational database management system.

[Report a bug](#)

B.7.3. Configuring the database using SQL-script

Each database software supports ANSI SQL standards and its own specifics. Therefore, each database has its own configuration setting in the eXo JCR as a database dialect parameter.

SQL-script files allow a detailed configuration of the database. SQL-scripts are located in **conf/storage/** directory of the **\$JPP_HOME/modules/org/gatein/lib/main/exo.jcr.component.core-1.15.3.jar** file.

The following tables show the correspondence between the scripts and databases:

Table B.3. Single-database

| Database | Script |
|----------------------|---------------------------------------|
| MySQL DB | <code>jcr-sjdbc.mysql.sql</code> |
| MySQL DB with utf-8 | <code>jcr-sjdbc.mysql-utf8.sql</code> |
| PostgreSQL | <code>jcr-sjdbc.pqsql.sql</code> |
| Oracle DB | <code>jcr-sjdbc.ora.sql</code> |
| DB2 9.7 | <code>jcr-sjdbc.db2.sql</code> |
| Microsoft SQL Server | <code>jcr-sjdbc.mssql.sql</code> |
| Sybase | <code>jcr-sjdbc.sybase.sql</code> |
| HSQLDB | <code>jcr-sjdbc.sql</code> |

Table B.4. Multi-database

| Database | Script |
|----------------------|---------------------------------------|
| MySQL DB | <code>jcr-mjdbc.mysql.sql</code> |
| MySQL DB with utf-8 | <code>jcr-mjdbc.mysql-utf8.sql</code> |
| PostgreSQL | <code>jcr-mjdbc.pqsql.sql</code> |
| Oracle DB | <code>jcr-mjdbc.ora.sql</code> |
| DB2 9.7 | <code>jcr-mjdbc.db2.sql</code> |
| Microsoft SQL Server | <code>jcr-mjdbc.mssql.sql</code> |
| Sybase | <code>jcr-mjdbc.sybase.sql</code> |
| HSQLDB | <code>jcr-mjdbc.sql</code> |

[Report a bug](#)

B.7.4. Multilanguage support database configuration

If a non-ANSI node name is used, you must use a database with MultiLanguage support. Some JDBC drivers need additional parameters for establishing a Unicode friendly connection. For example, under mysql it is necessary to add an additional parameter for the JDBC driver at the end of JDBC URL:

There are pre-configured configuration files for HSQLDB located in the **/conf/portal/** and **/conf/standalone/** folders of the **exo.jcr.component.core-1.15.3.jar** file, or within the source distribution of the eXo JCR implementation.

Example B.12. Parameter

```
jdbc:mysql://exoua.dnsalias.net/portal?characterEncoding=utf8
```

The configuration files are located in service jars `/conf/portal/configuration.xml` (eXo services including JCR Repository Service) and `exo-jcr-config.xml` (repositories configuration) by default.

In the portal, the JCR is configured in a portal web application `portal/WEB-INF/conf/jcr/jcr-configuration.xml` (JCR Repository Service and related services) and `repository-configuration.xml` (repositories configuration).

See Also:

» [Section B.4, "Template for JCR configuration file"](#)

[Report a bug](#)

B.7.5. Isolated-database Configuration

Isolated-database configuration allows configuring single database for repository, but separate database tables for each workspace.

Procedure B.2. Configuring isolated databases

1. Configure the data container in the `org.exoplatform.services.naming.InitialContextInitializer` service. It is the JNDI context initializer, which registers (binds) naming resources (DataSources) for data containers.

```
<external-component-plugins>
  <target-
component>org.exoplatform.services.naming.InitialContextInitializer
</target-component>
  <component-plugin>
    <name>bind.datasource</name>
    <set-method>addPlugin</set-method>

<type>org.exoplatform.services.naming.BindReferencePlugin</type>
  <init-params>
    <value-param>
      <name>bind-name</name>
      <value>jdbcjcr</value>
    </value-param>
    <value-param>
      <name>class-name</name>
      <value>javax.sql.DataSource</value>
    </value-param>
    <value-param>
      <name>factory</name>

<value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
    </value-param>
    <properties-param>
      <name>ref-addresses</name>
      <description>ref-addresses</description>
      <property name="driverClassName"
value="org.postgresql.Driver"/>
      <!-- MVCC configured to prevent possible deadlocks when a
```

```

global Tx is active -->
    <property name="url"
value="jdbc:postgresql://exoua.dnsalias.net/portal"/>
    <property name="username" value="exoadmin"/>
    <property name="password" value="exo12321"/>
    </properties-param>
</init-params>
</component-plugin>

</external-component-plugins>

```

The following database connection parameters are configured:

- ✱ **driverClassName**. For example: "org.hsqldb.jdbcDriver", "com.mysql.jdbc.Driver", "org.postgresql.Driver"
- ✱ **url**. For example: "jdbc:hsqldb:file:target/temp/data/portal", "jdbc:mysql://exoua.dnsalias.net/jcr"
- ✱ **username**. For example: "sa", "exoadmin"
- ✱ **password**. For example: "exo12321"

2. Configure the repository service.

Each workspace is configured for the same data container.

In this step, you are configuring two workspaces which will persist in different database tables.

```

<workspaces>
  <workspace name="ws" >

    <container
class="org.exoplatform.services.jcr.impl.storage.jdbc.optimisation.
CQJDBCWorkspaceDataContainer">
      <properties>
        <property name="source-name" value="jdbcjcr"/>
        <property name="db-structure-type" value="isolated"/>

      </properties>
    </container>

  </workspace>
  <workspace name="ws1" >
    <container
class="org.exoplatform.services.jcr.impl.storage.jdbc.optimisation.
CQJDBCWorkspaceDataContainer">
      <properties>
        <property name="source-name" value="jdbcjcr"/>
        <property name="db-structure-type" value="isolated"/>

      </properties>
    </container>

```

```
</workspace>
</workspaces>
```

[Report a bug](#)

B.7.6. Multi-database Configuration

You need to configure each workspace in a repository as part of multi-database configuration.

Databases may reside on remote servers as required.

Procedure B.3. Multi-database configuration

This procedure configures two workspace which will persistent in two different databases , ws in HSQLDB and ws1 in MySQL.

1. Configure the data containers in the **org.exoplatform.services.naming.InitialContextInitializer** service.

It's the JNDI context initializer which registers (binds) naming resources (DataSources) for data containers. For example, two data containers **jdbcjcr** - local HSQLDB, and **jdbcr1** - remote MySQL are shown here.

```
<external-component-plugins>
  <target-
component>org.exoplatform.services.naming.InitialContextInitializer
  </target-component>
  <component-plugin>
    <name>bind.datasource</name>
    <set-method>addPlugin</set-method>

    <type>org.exoplatform.services.naming.BindReferencePlugin</type>
    <init-params>
      <value-param>
        <name>bind-name</name>
        <value>jdbcjcr</value>
      </value-param>
      <value-param>
        <name>class-name</name>
        <value>javax.sql.DataSource</value>
      </value-param>
      <value-param>
        <name>factory</name>

        <value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
      </value-param>
      <properties-param>
        <name>ref-addresses</name>
        <description>ref-addresses</description>
        <property name="driverClassName"
value="$${all.driverClassName:org.hsqldb.jdbcDriver}"/>
        <!-- MVCC configured to prevent possible deadlocks when a
global Tx is active -->
```

```

        <property name="url"
value="${jdbcjcr.url:jdbc:hsqldb:file:target/temp/data/portal;hsqldb.tx=mvcc}"/>
        <property name="username"
value="${jdbcjcr.username:sa}"/>
        <property name="password" value="${jdbcjcr.password:}"/>
    </properties-param>
</init-params>
</component-plugin>
<component-plugin>
    <name>bind.datasource</name>
    <set-method>addPlugin</set-method>

<type>org.exoplatform.services.naming.BindReferencePlugin</type>
    <init-params>
        <value-param>
            <name>bind-name</name>
            <value>jdbcjcr1</value>
        </value-param>
        <value-param>
            <name>class-name</name>
            <value>javax.sql.DataSource</value>
        </value-param>
        <value-param>
            <name>factory</name>

<value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
        </value-param>
        <properties-param>
            <name>ref-addresses</name>
            <description>ref-addresses</description>
            <property name="driverClassName"
value="${all.driverClassName:org.hsqldb.jdbcDriver}"/>
            <property name="url"
value="${jdbcjcr1.url:jdbc:hsqldb:file:target/temp/data/jcr}"/>
            <property name="username"
value="${jdbcjcr1.username:sa}"/>
            <property name="password" value="${jdbcjcr1.password:}"/>
        </properties-param>
    </init-params>
</component-plugin>
    <!-- Unnecessary plugins not relevant to this section removed
for clarity -->
</external-component-plugins>

```

a. Configure the following database connection parameters:

- ✱ **driverClassName**, for example, "org.hsqldb.jdbcDriver", "com.mysql.jdbc.Driver", "org.postgresql.Driver"
- ✱ **url**, for example, "jdbc:hsqldb:file:target/temp/data/portal", "jdbc:mysql://exoua.dnsalias.net/jcr"
- ✱ **username**, for example, "sa", "exoadmin"
- ✱ **password**, for example, "", "exo12321"

According to apache DBCP configuration, there can be connection pool configuration parameters, for example ***org.apache.commons.dbcp.BasicDataSourceFactory***

- * ***maxActive***, for example, 50
- * ***maxIdle***, for example, 5
- * ***initialSize***, for example, 5
- * To access the list of parameters refer to <http://jakarta.apache.org/commons/dbcp/configuration.html>

2. Configure the repository service.

Each workspace will be configured for its own data container. For example, two workspaces ***ws*** - jdbcjcr and ***ws1*** - jdbcjcr1 are configured.

```
<workspaces>
  <workspace name="ws" auto-init-root-nodetype="nt:unstructured">
    <container
class="org.exoplatform.services.jcr.impl.storage.jdbc.JDBCWorkspaceD
ataContainer">
      <properties>
        <property name="source-name" value="jdbcjcr"/>
        <property name="dialect" value="hsqldb"/>
        <property name="multi-db" value="true"/>
        <property name="max-buffer-size" value="200K"/>
        <property name="swap-directory" value="target/temp/swap/ws"/>
      </properties>
    </container>
    <cache enabled="true">
      <properties>
        <property name="max-size" value="10K"/><!-- 10Kbytes -->
        <property name="live-time" value="30m"/><!-- 30 min -->
      </properties>
    </cache>
    <query-handler
class="org.exoplatform.services.jcr.impl.core.query.lucene.SearchIn
dex">
      <properties>
        <property name="index-dir" value="target/temp/index"/>
      </properties>
    </query-handler>
    <lock-manager>
      <time-out>15m</time-out><!-- 15 min -->
      <persister
class="org.exoplatform.services.jcr.impl.core.lock.FileSystemLockPer
sister">
        <properties>
          <property name="path" value="target/temp/lock/ws"/>
        </properties>
      </persister>
    </lock-manager>
  </workspace>
  <workspace name="ws1" auto-init-root-nodetype="nt:unstructured">
    <container
```



```

class="org.exoplatform.services.jcr.impl.storage.jdbc.JDBCWorkspaced
ataContainer">
  <properties>
    <property name="source-name" value="jdbcjcr1"/>
    <property name="dialect" value="mysql"/>
    <property name="multi-db" value="true"/>
    <property name="max-buffer-size" value="200K"/>
    <property name="swap-directory"
value="target/temp/swap/ws1"/>
  </properties>
</container>
<cache enabled="true">
  <properties>
    <property name="max-size" value="10K"/>
    <property name="live-time" value="5m"/>
  </properties>
</cache>
<query-handler
class="org.exoplatform.services.jcr.impl.core.query.lucene.SearchIn
dex">
  <properties>
    <property name="index-dir" value="target/temp/index"/>
  </properties>
</query-handler>
<lock-manager>
<time-out>15m</time-out><!-- 15 min -->
<persister
class="org.exoplatform.services.jcr.impl.core.lock.FileSystemLockPer
sister">
  <properties>
    <property name="path" value="target/temp/lock/ws1"/>
  </properties>
</persister>
</lock-manager>
</workspace>
</workspaces>

```

Parameters and their description

source-name

A javax.sql.DataSource name configured in InitialContextInitializer component (was **sourceName** prior JCR 1.9).

dialect

A database dialect, one of **hsqldb**, **mysql**, **mysql-utf8**, **pgsql**, **oracle**, **oracle-oci**, **mssql**, **sybase**, **derby**, **db2**, **db2v8** or **auto** for dialect auto detection.

multi-db

Enable multi-database container with this parameter (set value "true").

max-buffer-size

Threshold value, in bytes, after which the **javax.jcr.Value** content is swapped to a file in a temporary storage. A swap for pending changes, for example.

✎ **swap-directory** A path in the file system used to swap the pending changes.

[Report a bug](#)

B.7.7. Single-database Configuration

Configuring a single-database data container is easier than configuring a multi-database data container as only one naming resource must be configured.

Example B.13. jdbcjcr Data Container

```
<external-component-plugins>
  <target-
component>org.exoplatform.services.naming.InitialContextInitializer</t
arget-component>
  <component-plugin>
    <name>bind.datasource</name>
    <set-method>addPlugin</set-method>

<type>org.exoplatform.services.naming.BindReferencePlugin</type>
  <init-params>
    <value-param>
      <name>bind-name</name>
      <value>jdbcjcr</value>
    </value-param>
    <value-param>
      <name>class-name</name>
      <value>javax.sql.DataSource</value>
    </value-param>
    <value-param>
      <name>factory</name>

<value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
    </value-param>
    <properties-param>
      <name>ref-addresses</name>
      <description>ref-addresses</description>
      <property name="driverClassName"
value="org.postgresql.Driver"/>
      <property name="url"
value="jdbc:postgresql://exoua.dnsalias.net/portal"/>
      <property name="username" value="exoadmin"/>
      <property name="password" value="exo12321"/>
      <property name="maxActive" value="50"/>
      <property name="maxIdle" value="5"/>
      <property name="initialSize" value="5"/>
    </properties-param>
  </init-params>
</component-plugin>
</external-component-plugins>
```

To configure repository workspaces with one database, the ***multi-db*** parameter must be set as **false**.

For example, (two workspaces ***ws - jdbcjcr***, ***ws1 - jdbcjcr***):

Example B.14. Setting two workspaces in a single database

This step configures two persistent workspaces in one database (PostgreSQL).

```
<workspaces>
  <workspace name="ws" auto-init-root-nodetype="nt:unstructured">
    <container
class="org.exoplatform.services.jcr.impl.storage.jdbc.JDBCWorkspaceData
Container">
      <properties>
        <property name="source-name" value="jdbcjcr"/>
        <property name="dialect" value="pgsql"/>
        <property name="multi-db" value="false"/>
        <property name="max-buffer-size" value="200K"/>
        <property name="swap-directory" value="target/temp/swap/ws"/>
      </properties>
    </container>
    <cache enabled="true">
      <properties>
        <property name="max-size" value="10K"/>
        <property name="live-time" value="30m"/>
      </properties>
    </cache>
    <query-handler
class="org.exoplatform.services.jcr.impl.core.query.lucene.SearchIndex"
>
      <properties>
        <property name="index-dir" value="../temp/index"/>
      </properties>
    </query-handler>
    <lock-manager>
    <time-out>15m</time-out>
    <persister
class="org.exoplatform.services.jcr.impl.core.lock.FileSystemLockPersist
er">
      <properties>
        <property name="path" value="target/temp/lock/ws"/>
      </properties>
    </persister>
    </lock-manager>
  </workspace>
  <workspace name="ws1" auto-init-root-nodetype="nt:unstructured">
    <container
class="org.exoplatform.services.jcr.impl.storage.jdbc.JDBCWorkspaceData
Container">
      <properties>
        <property name="source-name" value="jdbcjcr"/>
        <property name="dialect" value="pgsql"/>
        <property name="multi-db" value="false"/>
```

```

        <property name="max-buffer-size" value="200K"/>
        <property name="swap-directory" value="target/temp/swap/ws1"/>
    </properties>
</container>
<cache enabled="true">
    <properties>
        <property name="max-size" value="10K"/>
        <property name="live-time" value="5m"/>
    </properties>
</cache>
<lock-manager>
    <time-out>15m</time-out>
    <persister
class="org.exoplatform.services.jcr.impl.core.lock.FileSystemLockPersist
er">
        <properties>
            <property name="path" value="target/temp/lock/ws1"/>
        </properties>
    </persister>
</lock-manager>
</workspace>
</workspaces>

```

[Report a bug](#)

B.7.7.1. Configuration Without DataSource

It is possible to configure the repository without binding **javax.sql.DataSource** in the JNDI service if you have a dedicated JDBC driver implementation with special features such as, XA transactions, statements/connections pooling and so on.

Procedure B.4. Configuring the repository without the data source

1. Remove the configuration in **InitialContextInitializer** for your database and configure a new one directly in the workspace container.
2. Remove parameter **source-name** and add next lines. Describe your values for a JDBC driver, database URL and username.



Connection Pooling

Ensure the JDBC driver provides connection pooling. Connection pooling is strongly recommended for use with JCR to prevent a database overload.

```

<workspace name="ws" auto-init-root-nodetype="nt:unstructured">
    <container
class="org.exoplatform.services.jcr.impl.storage.jdbc.JDBCWorkspaceDataCo
ntainer">
        <properties>
            <property name="dialect" value="hsqldb"/>
            <property name="driverliteral" value="org.hsqldb.jdbcDriver"/>
            <property name="url"

```

```
value="jdbc:hsqldb:file:target/temp/data/portal"/>
  <property name="username" value="su"/>
  <property name="password" value=""/>
  . . . . .
```

[Report a bug](#)

B.7.7.2. Dynamic Workspace Creation

Workspaces can be added dynamically during runtime.

This can be performed in two steps:

Procedure B.5. Adding workspace at runtime

1. Register a new configuration in RepositoryContainer and create a WorkspaceContainer.
ManageableRepository.configWorkspace(WorkspaceEntry wsConfig).
2. Create a new workspace **ManageableRepository.createWorkspace(String workspaceName).**

[Report a bug](#)

B.7.8. Simple and Complex queries

eXo JCR provides two ways to interact with the database,

JDBCStorageConnection

This method uses simple queries. Simple queries do not use sub queries, left or right joins. They are implemented to support maximum number of database dialects.

CQJDBCStorageConnection

This method uses complex queries. Complex queries are optimized to reduce the number of database calls.

Simple queries are used if you choose

org.exoplatform.services.jcr.impl.storage.jdbc.JDBCWorkspaceDataContainer.

```
<workspaces>
  <workspace name="ws" auto-init-root-nodetype="nt:unstructured">
    <container
      class="org.exoplatform.services.jcr.impl.storage.jdbc.JDBCWorkspaceDataCo
      ntainer">
      . . .
    </workspace>
  </worksapces>
```

Complex queries are used if you chose

**org.exoplatform.services.jcr.impl.storage.jdbc.optimisation.CQJDBCWorkspa
ceDataContainer.**

```
<workspaces>
  <workspace name="ws" auto-init-root-nodetype="nt:unstructured">
    <container
```

```
class="org.exoplatform.services.jcr.impl.storage.jdbc.optimisation.CQJDBC
WorkspaceDataContainer">
    ...
</workspace>
</worksapces>
```

[Report a bug](#)

B.7.9. Force Query Hints

Some databases, such as Oracle and MySQL, support hints to increase query performance. The eXo JCR has separate Complex Query implementations for the Oracle database dialect, which uses query hints to increase performance for few important queries.

To enable this option, use the following configuration property:

```
<workspace name="ws" auto-init-root-nodetype="nt:unstructured">
  <container
class="org.exoplatform.services.jcr.impl.storage.jdbc.JDBCWorkspaceDataCo
ntainer">
    <properties>
      <property name="dialect" value="oracle"/>
      <property name="force.query.hints" value="true" />
      .....
    </properties>
  </container>
</workspace>
```



Note

Query hints are only used for Complex Queries with the Oracle dialect. For all other dialects this parameter is ignored.

[Report a bug](#)

B.7.10. Notes for Microsoft Windows Users

The current configuration of eXo JCR uses [Apache DBCP](#) connection pool (`org.apache.commons.dbcp.BasicDataSourceFactory`).

It is possible to set a high value for the **maxActive** parameter in the **configuration.xml** file. This creates a high use of TCP/IP ports from a client machine inside the pool, for example, JDBC driver. As a result, the data container can throw exceptions like Address already in use.

To solve this problem, you must configure the client's machine networking software to use shorter timeouts for open TCP/IP ports.

The solution is to edit two registry keys within the **HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters** node. Both of these keys are unset by default.

Procedure B.6. Set the registry keys

1. Set the **MaxUserPort** registry key to **=dword:00001b58**. This sets the maximum of open ports to 7000 or higher (the default is 5000).
2. Set the **TcpTimedWaitDelay** registry key to **=dword:0000001e**. This sets **TIME_WAIT**

parameter to 30 seconds (the default is 240).

Example B.15. Sample Registry File

```
Windows Registry Editor Version 5.00
```

```
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters]
"MaxUserPort"=dword:00001b58
"TcpTimedWaitDelay"=dword:0000001e
```

[Report a bug](#)

B.8. External Value Storages

JCR values are stored in the Workspace Data container by default. The eXo JCR offers an additional option of storing JCR values separately from the Workspace Data container which can help keep Binary Large Objects (BLOBs) separate.

Tree-based storage is recommended in most cases. For example, if you run an application on Amazon EC2 the S3 option is useful for architecture. Simple flat storage is good in speed of creation/deletion of values, it might be a compromise for a small storages.

Value storage configuration is a part of Repository configuration.

See Also:

✎ [Section B.4.1.5, “Value Storage plug-in Configuration Parameters”](#)

[Report a bug](#)

B.8.1. Tree File Value Storage

Tree File Value Storage holds values in tree-like file system files. Path property points to the root directory where the files are stored.

This is a recommended type of external storage because it can contain large amount of files limited only by disk/volume free space.

However, using Tree File Value Storage can result in a higher time on value deletion, due to the removal of unused tree-nodes.

Example B.16. Tree File Value Storage Configuration

```
<workspace name="backup">
  <container
class="org.exoplatform.services.jcr.impl.storage.jdbc.optimisation.CQJD
BCWorkspaceDataContainer">
    <properties>
      <property name="source-name" value="jdbcjcr" />
      <property name="multi-db" value="false" />
      <property name="update-storage" value="false" />
      <property name="max-buffer-size" value="200k" />
```

```

        <property name="swap-directory" value="../../temp/swap/backup" />
    </properties>
    <value-storages>
        <value-storage id="draft"
class="org.exoplatform.services.jcr.impl.storage.value.fs.TreeFileValue
Storage">
<Comment #1>
        <properties>
            <property name="path" value="../../temp/values/backup" />
        </properties>
    <Comment #2>
        <filters>
            <filter property-type="Binary" />
        </filters>
    </value-storage>
    </value-storages>
</container>
<initializer
class="org.exoplatform.services.jcr.impl.core.ScratchWorkspaceInitializ
er">
    <properties>
        <property name="root-nodetype" value="nt:unstructured" />
    </properties>
</initializer>
<cache enabled="true"
class="org.exoplatform.services.jcr.impl.dataflow.persistent.LinkedWorks
paceStorageCacheImpl">
    <properties>
        <property name="max-size" value="10k" />
        <property name="live-time" value="1h" />
    </properties>
</cache>
<query-handler
class="org.exoplatform.services.jcr.impl.core.query.lucene.SearchIndex"
>
    <properties>
        <property name="index-dir" value="../../temp/jcrlucenedb/backup"
/>
    </properties>
</query-handler>
</workspace>
</workspaces>
</repository>
</repositories>
</repository-service>

```

Comment #1: **id** denotes the value storage unique identifier. It is used for linking with properties stored in a workspace container.

Comment #2: **path** denotes the location of value files.

Each file value storage has **filters** for incoming values. A filter can match values by property-type, property-name, ancestor-path. It can also match the size of values stored by min-value-size in bytes.

In the example a filter with property-type and min-value-size has been used. This results in storage for binary values with size greater of 1MB.



Note

It is recommended that properties with large values are stored in file value storage only.

Example B.17. Value storage for large files

This example shows a value storage with different locations for large files, for example, min-value-size of a 20Mb-sized filter.

A value storage uses ORed logic in the process of filter selection. This means the first filter in the list is called first and if it is not matched the next filter is called, and so on.

In this example a value matches the 20MB filter min-value-size and is stored in the path **data/20Mvalues**. All other filters are stored in **data/values**.

```
<value-storages>
  <value-storage id="Storage #1"
class="org.exoplatform.services.jcr.impl.storage.value.fs.TreeFileValue
Storage">
    <properties>
      <property name="path" value="data/20Mvalues"/>
    </properties>
    <filters>
      <filter property-type="Binary" min-value-size="20M"/>
    </filters>
  </value-storage>
  <value-storage id="Storage #2"
class="org.exoplatform.services.jcr.impl.storage.value.fs.TreeFileValue
Storage">
    <properties>
      <property name="path" value="data/values"/>
    </properties>
    <filters>
      <filter property-type="Binary" min-value-size="1M"/>
    </filters>
  </value-storage>
</value-storages>
```

[Report a bug](#)

B.8.2. Disabling Value Storage

The JCR allows you to disable value storage by adding the following property into its configuration.

```
<property name="enabled" value="false" />
```



Warning

It is recommended that this functionality be used for internal and testing purpose only, and with caution, as all stored values will be inaccessible.

[Report a bug](#)

B.9. Workspace Data Container

Each Workspace of the JCR has its own persistent storage to hold that workspace's items data. The eXo JCR can be configured so that it can use one or more workspaces that are logical units of the repository content.

The physical data storage mechanism is configured using mandatory **container** element. The type of container is described in the attribute **class** by specifying the fully qualified name of the `org.exoplatform.services.jcr.storage` subclass.

Example B.18. Physical Data Storage Configuration

```

<container
  class="org.exoplatform.services.jcr.impl.storage.jdbc.optimisation.CQJD
  BCWorkspaceDataContainer">
  <properties>
    <property name="source-name" value="jdbcjcr1"/>
    <property name="dialect" value="hsqldb"/>
    <property name="multi-db" value="true"/>
    <property name="max-buffer-size" value="200K"/>
    <property name="swap-directory" value="target/temp/swap/ws"/>
    <property name="lazy-node-iterator-page-size" value="50"/>
    <property name="acl-bloomfilter-false-positive-probability"
  value="0.1d"/>
    <property name="acl-bloomfilter-elements-number" value="1000000"/>
    <property name="check-sns-new-connection" value="false"/>
    <property name="batch-size" value="1000"/>
  </properties>

```

Standard Workspace Data Container Properties

max-buffer-size

Default value is **200K**.

The maximum buffer size for the container. If a value size is greater than this setting, it is spooled to a temporary file as defined in `<swap-directory>`.

swap-directory

Default value is `java.io.tmpdir`.

Specifies the location where the value is spooled, if no value storage is configured but a **max-buffer-size** is exceeded.

lazy-node-iterator-page-size

Default value is **100**.

Specifies the page size and the number of nodes that are retrieved from persistent storage at once.

acl-bloomfilter-false-positive-probability

Default value is **0.1d**.

Specifies the ACL Bloom-filter desired false positive probability. Range [0..1].

acl-bloomfilter-elements-number

Default value is **1000000**.

Specifies the expected number of ACL-elements in the Bloom-filter.

**Note**

Bloom filters are used to avoid reading nodes that do not have ACL. Infinispan is the only cache implementation that currently supports Bloom filters. See http://en.wikipedia.org/wiki/Bloom_filter for an overview of bloom filters.

check-sns-new-connection

Default value is **false**.

Specifies whether to create a connection for checking if an older same-name sibling exists.

max-descendant-nodes-allowed-on-move

Default value is **100**.

Specifies the maximum amount of descendant nodes allowed before determining whether descendant items are included into the changes log. This allows best possible performance, regardless of the total amount of sub-nodes.

This parameter is only used when **trigger-events-for-descendants-on-move** or **trigger-events-for-descendants-on-rename** is not set.

trigger-events-for-descendants-on-rename

Specifies whether each descendant item must be included in the changes log in case of a rename.

When this parameter is not set, the application will rely on the **max-descendant-nodes-allowed-on-move** parameter to handle whether or not descendant items are added to the changes log. If this parameter is not set but the parameter **trigger-events-for-descendants-on-move** is set, it will have the same value.

If set to **false**, performance on rename operations will increase on source parent nodes with a large number of sub-nodes but will decrease on parent nodes with a small number of sub-nodes.

If set to **true**, performance will decrease for a large number of sub-nodes and increase for a small number of sub-nodes.

trigger-events-for-descendants-on-move

When this parameter is not set, the application will rely on the **max-descendant-nodes-allowed-on-move** parameter to handle whether or not descendant items are added to the changes log.

Specifies whether each descendant item must be included in the change logs, in case of a move.

If set to **false**, performance on move operations will increase on source parent nodes with a large number of sub-nodes but will decrease on parent nodes with a small number of sub-nodes.

If set to **true**, performance will decrease for a large number of sub-nodes and increase for a small number of sub-nodes.

The eXo JCR has a JDBC-based relational database which is production ready Workspace Data Container.

JDBC Workspace Data Container Properties**source-name**

Mandatory parameter, which specifies the JDBC data source name (registered in JDNI by InitialContextInitializer).

dialect

Default value is **auto**.

The database dialect can be one of the following values: "auto", "hsqldb", "h2", "mysql", "mysql-myisam", "mysql-utf8", "mysql-myisam-utf8", "pgsql", "pgsql-scs", "oracle", "oracle-oci", "mssql", "sybase", "derby", "db2", "db2-mys", "db2v8" hsqldb, mysql, mysql-utf8, pgsql, oracle, oracle-oci.

db-structure-type**Note**

This parameter supersedes multi-db.

Mandatory parameter, which specifies the structure of the database container. Supported values include: "isolated", "multi", and "single".

db-tablename-suffix

Specifies the workspace name appended to tables. If db-structure-type is set to **isolated**, tables used by the repository service have the following values:

- **JCR_I\${db-tablename-suffix}** for items.
- **JCR_V\${db-tablename-suffix}** for values.
- **JCR_R\${db-tablename-suffix}** for references.

Workspace Data Container *may* support external storages for **javax.jcr.Value** (for example, BLOB values) using the optional element **value-storages**.

The Data Container attempts to read or write a value using the underlying value storage plug-in if the filter criteria matches the current property.

Example B.19. External Value Storage Configuration

```
<value-storages>
  <value-storage id="Storage #1"
class="org.exoplatform.services.jcr.impl.storage.value.fs.TreeFileValue
Storage">
    <properties>
      <property name="path" value="data/values"/>
    </properties>
    <filters>
      <filter property-type="Binary" min-value-size="1M"/><!-- Values
greater than 1Mbyte -->
    </filters>
  <!-- content removed for readability -->
</value-storages>
```

value-storage is the subclass of **org.exoplatform.services.jcr.storage.value.ValueStoragePlugin** and **properties** are optional plug-in specific parameters.

filters: Each file value storage can have the filter(s) for incoming values. If there are several filter criteria, they all have to match (AND-Condition).

A filter can match values by property type (property-type), property name (property-name), ancestor path (ancestor-path) and/or the size of values stored (min-value-size, e.g. 1M, 4.2G, 100 (bytes)).

In a code sample, we use a filter with property-type and min-value-size only. That means that the storage is only for binary values whose size is greater than 1Mbyte.

It is recommended that you store properties with large values in a file value storage only.

[Report a bug](#)

B.10. Configuring the Cluster

[Report a bug](#)

B.10.1. Launching Cluster

[Report a bug](#)

B.10.1.1. Configuring JCR to use external configuration

1. Create a new configuration file, for example, **exo-jcr-configuration.xml** as follows:

Example B.20. Creating an external Configuration file

```
<repository-service default-repository="repository1">
  <repositories>
```

```

    <repository name="repository1" system-workspace="ws1"
default-workspace="ws1">
    <security-domain>exo-domain</security-domain>
    <access-control>optional</access-control>
    <authentication-
policy>org.exoplatform.services.jcr.impl.core.access.JAASAuthenti
cator</authentication-policy>
    <workspaces>
    <workspace name="ws1">
    <container
class="org.exoplatform.services.jcr.impl.storage.jdbc.optimisatio
n.CQJDBCWorkspaceDataContainer">
    <properties>
    <property name="source-name"
value="jdbcjcr" />
    <property name="dialect" value="oracle" />
    <property name="multi-db" value="false" />
    <property name="update-storage"
value="false" />
    <property name="max-buffer-size"
value="200k" />
    <property name="swap-directory"
value="../temp/swap/production" />
    </properties>
    <value-storages>
    <!-- Comment #1 -->
    </value-storages>
    </container>
    <initializer
class="org.exoplatform.services.jcr.impl.core.ScratchWorkspaceIni
tializer">
    <properties>
    <property name="root-nodetype"
value="nt:unstructured" />
    </properties>
    </initializer>
    <cache enabled="true"
class="org.exoplatform.services.jcr.impl.dataflow.persistent.jboss
cache.JBossCacheWorkspaceStorageCache">
    <!-- Comment #2 -->
    </cache>
    <query-handler
class="org.exoplatform.services.jcr.impl.core.query.lucene.Search
Index">
    <!-- Comment #3 -->
    </query-handler>
    <lock-manager
class="org.exoplatform.services.jcr.impl.core.lock.jbossCache.Cac
heableLockManagerImpl">
    <!-- Comment #4 -->
    </lock-manager>
    </workspace>
    <workspace name="ws2">
    ...
    </workspace>
    <workspace name="wsN">

```

```

        ...
    </workspace>
</workspaces>
</repository>
</repositories>
</repository-service>

```

Comment #1: Configure the Value Storage

Comment #2: Configure cache

Comment #3: Configure Indexer

Comment #4: Configure Lock Manager

2. Update the ***RepositoryServiceConfiguration*** configuration in the **exo-configuration.xml** to reference your file.

```

<component>

<key>org.exoplatform.services.jcr.config.RepositoryServiceConfiguration</key>

<type>org.exoplatform.services.jcr.impl.config.RepositoryServiceConfigurationImpl</type>
  <init-params>
    <value-param>
      <name>conf-path</name>
      <description>JCR configuration file</description>
      <value>exo-jcr-configuration.xml</value>
    </value-param>
  </init-params>
</component>

```

See Also:

- ✦ [Section B.10.2.2, "Configuration Guidelines"](#)

[Report a bug](#)

B.10.2. Requirements

[Report a bug](#)

B.10.2.1. Environment requirements

- ✦ Every node of the cluster must have the same mounted Network File System (NFS) with the read and write permissions.
- ✦ Every node of cluster must use the same database.
- ✦ The same clusters on different nodes must have the same names.

For example, if the Indexer cluster in the production workspace on the first node is named **production_indexer_cluster**, then indexer clusters in the production workspace on all other nodes must also be named **production_indexer_cluster**.

[Report a bug](#)

B.10.2.2. Configuration Guidelines

The configuration of every workspace in the repository must contain the following elements:

Example B.21. Value Storage configuration

```
<value-storages>
  <value-storage id="system"
class="org.exoplatform.services.jcr.impl.storage.value.fs.TreeFileValue
Storage">
    <properties>
      <property name="path"
value="/mnt/tornado/temp/values/production" />      <!-- path within NFS
where ValueStorage will hold it's data-->
    </properties>
    <filters>
      <filter property-type="Binary" />
    </filters>
  </value-storage>
</value-storages>
```

Example B.22. Cache Configuration

```
<cache enabled="true"
class="org.exoplatform.services.jcr.impl.dataflow.persistent.jbosscache
.JBossCacheWorkspaceStorageCache">
  <properties>
    <property name="jbosscache-configuration"
value="jar:/conf/portal/test-jbosscache-data.xml" />      <!--      path
to JBoss Cache configuration for data storage -->
    <property name="jgroups-configuration"
value="jar:/conf/portal/udp-mux.xml" />      <!--
path to JGroups configuration -->
    <property name="jbosscache-cluster-name"
value="JCR_Cluster_cache_production" />      <!--      JBoss
Cache data storage cluster name -->
    <property name="jgroups-multiplexer-stack" value="true" />
  </properties>
</cache>
```

Example B.23. Indexer Configuration

```
<query-handler
class="org.exoplatform.services.jcr.impl.core.query.lucene.SearchIndex"
>
  <properties>
    <property name="changesfilter-class"
value="org.exoplatform.services.jcr.impl.core.query.jbosscache.JBossCac
heIndexChangesFilter" />
```



```

    <property name="index-dir"
value="/mnt/tornado/temp/jcrlucenedb/production" />
<!--      path within NFS where ValueStorage will hold it's data -->
    <property name="jboss-cache-configuration"
value="jar:/conf/portal/test-jboss-cache-indexer.xml" />      <!--      path
to JBoss Cache configuration for indexer -->
    <property name="jgroups-configuration"
value="jar:/conf/portal/udp-mux.xml" />                        <!--
path to JGroups configuration -->
    <property name="jboss-cache-cluster-name"
value="JCR_Cluster_indexer_production" />                      <!--
JBoss Cache indexer cluster name -->
    <property name="jgroups-multiplexer-stack" value="true" />
  </properties>
</query-handler>

```

Example B.24. Lock Manager Configuration

```

<lock-manager
class="org.exoplatform.services.jcr.impl.core.lock.jboss-cache.Cacheable
LockManagerImpl">
  <properties>
    <property name="time-out" value="15m" />
    <property name="jboss-cache-configuration"
value="jar:/conf/portal/test-jboss-cache-lock.xml" />      <!--      path
to JBoss Cache configuration for lock manager -->
    <property name="jgroups-configuration"
value="jar:/conf/portal/udp-mux.xml" />                      <!--
path to JGroups configuration -->
    <property name="jgroups-multiplexer-stack" value="true" />
    <property name="jboss-cache-cluster-name"
value="JCR_Cluster_lock_production" />                      <!--
JBoss Cache locks cluster name -->

    <property name="jboss-cache-cl-cache.jdbc.table.name"
value="jcrlocks_production"/>                                <!--      the name of
the DB table where lock's data will be stored -->
    <property name="jboss-cache-cl-cache.jdbc.table.create"
value="true"/>
    <property name="jboss-cache-cl-cache.jdbc.table.drop"
value="false"/>
    <property name="jboss-cache-cl-cache.jdbc.table.primarykey"
value="jcrlocks_production_pk"/>
    <property name="jboss-cache-cl-cache.jdbc.fqn.column"
value="fqn"/>
    <property name="jboss-cache-cl-cache.jdbc.node.column"
value="node"/>
    <property name="jboss-cache-cl-cache.jdbc.parent.column"
value="parent"/>
    <property name="jboss-cache-cl-cache.jdbc.datasource"
value="jdbcjcr"/>
  </properties>
</lock-manager>

```

[Report a bug](#)

B.11. Configuring JBoss Cache

[Report a bug](#)

B.11.1. Indexer lock manager and data container configuration

Indexer, lock manager and data container uses instances of the JBoss Cache product for caching in clustered environment. So, every element has its own transport and has to be configured correctly.

Workspaces have similar configuration with different cluster names and parameters. The simplest way to configure is to define a configuration file for each component in each workspace.

```
<property name="jboss-cache-configuration" value="conf/standalone
/test-jboss-cache-lock-db1-ws1.xml" />
```

To configure workspaces, eXo JCR offers a template-based configuration for JBoss Cache instances. You can have one template for each Lock Manager, Indexer and Data container.

To use these templates define the map of substitution parameters in a main configuration file by using define `${jboss-cache-<parameter name>}` inside xml template and list correct value in JCR configuration file just below *jboss-cache-configuration*.

Example B.25. Template for configuring workspaces

```
...
<clustering mode="replication" clusterName="${jboss-cache-cluster-
name}">
  <stateRetrieval timeout="20000" fetchInMemoryState="false" />
...
```

Example B.26. JCR configuration file

```
...
<property name="jboss-cache-configuration"
value="jar:/conf/portal/jboss-cache-lock.xml" />
<property name="jboss-cache-cluster-name" value="JCR-cluster-locks-db1-
ws" />
...
```

[Report a bug](#)

B.11.2. JGroups configuration

JGroups is used by JBoss Cache for network communication and transport in a clustered environment. If the property is defined in component configuration, it is injected in the JBoss Cache instance on start up.

```
<property
name="jgroups-configuration" value="your/path/to/modified-udp.xml"
/>
```

Lock manager, data container and query handler component for each workspace requires its own clustered environment with unique names.

Each cluster should perform multi-casts on a separate port. This configuration leads to much unnecessary overhead on cluster. JGroups provides a multiplexer feature providing ability to use one single channel for set of clusters.

The multiplexer reduces network overheads and increase performance and stability of application. To enable multiplexer stack, you should define appropriate configuration file (**udp-mux.xml** is pre-shipped with eXo JCR) and set **jgroups-multiplexer-stack** as **true**.

```
<property
name="jgroups-configuration" value="jar:/conf/portal/udp-mux.xml" />
<property name="jgroups-multiplexer-stack" value="true"
/>
```

[Report a bug](#)

B.11.3. Sharing JBoss Cache instances

A single JBoss Cache instance consumes large resources, and the default setup has an instance for the indexer, the lock manager and the data container on each workspace. So, an environment that uses multiple workspace can benefit from sharing a JBoss Cache instance between several instances of the same type, for example, the lock manager instance. .

Sharing feature is disabled by default. To enable sharing at the component configuration level, you need to set the **jboss-cache-shareable** property to **true**:

Example B.27. Configuring sharing between JBoss cache instances

```
<property name="jboss-cache-shareable" value="true" />
```

This feature allows the JBoss Cache instance, that is used by a component to be re-used by another components of the same type with the same JBoss Cache configuration. So, all the parameters of type **jboss-cache-*<PARAM_NAME>*** must be identical between the components of same type of different workspaces. Therefore, if you use the same values for the parameters in each workspace, you need three JBoss Cache instances, one instance each for the indexer, lock manager and data container running at once. This approach frees resource significantly.



Note

In eviction configuration, reusing JBoss cache instance is handled differently.

[Report a bug](#)

B.11.4. Shipped JBoss Cache configuration templates

The eXo JCR implementation is shipped with ready-to-use JBoss Cache configuration templates for JCR's components. They are located in **\$JPP_HOME/gatein/gatein.ear/portal.war/WEB-INF/conf/jcr/jbossccache** directory, inside either the **cluster** or **local** directory.

[Report a bug](#)

B.11.4.1. Data container template

The data container template is **jbossccache-data.xml**.

Example B.28. Data container template

```
<?xml version="1.0" encoding="UTF-8"?>
<jbossccache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:jboss:jbossccache-core:config:3.1">

  <locking useLockStriping="false" concurrencyLevel="50000"
lockParentForChildInsertRemove="false"
    lockAcquisitionTimeout="20000" />

  <clustering mode="replication" clusterName="${jbossccache-cluster-
name}">
    <stateRetrieval timeout="20000" fetchInMemoryState="false" />
    <jgroupsConfig multiplexerStack="jcr.stack" />
    <sync />
  </clustering>

  <!-- Eviction configuration -->
  <eviction wakeUpInterval="5000">
    <default algorithmClass="org.jboss.cache.eviction.LRUAlgorithm"
actionPolicyClass="org.exoplatform.services.jcr.impl.dataflow.persisten
t.jbossccache.ParentNodeEvictionActionPolicy"
      eventQueueSize="1000000">
      <property name="maxNodes" value="1000000" />
      <property name="timeToLive" value="120000" />
    </default>
  </eviction>
</jbossccache>
```

Template Variables

jbossccache-cluster-name

Unique cluster name.

[Report a bug](#)

B.11.4.2. Lock manager template

The lock manager template is **jbossccache-lock.xml**.

Example B.29. Lock manager template

```

<jboss-cache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:jboss:jboss-cache-core:config:3.1">

  <locking useLockStriping="false" concurrencyLevel="500"
lockParentForChildInsertRemove="false"
    lockAcquisitionTimeout="20000" />

  <loaders passivation="false" shared="true">
    <!-- All the data of the JCR locks needs to be loaded at startup -->
    <preload>
      <node fqname="/" />
    </preload>
    <!--
For another cache-loader class you should use another template with
cache-loader specific parameters
-->
    <loader
class="org.exoplatform.services.jcr.impl.core.lock.jboss-cache.JDBCCache
Loader" async="false" fetchPersistentState="false"
    ignoreModifications="false" purgeOnStartup="false">
      <properties>
        cache.jdbc.table.name=${jboss-cache-cl-
cache.jdbc.table.name}
        cache.jdbc.table.create=${jboss-cache-cl-
cache.jdbc.table.create}
        cache.jdbc.table.drop=${jboss-cache-cl-
cache.jdbc.table.drop}
        cache.jdbc.table.primarykey=${jboss-cache-cl-
cache.jdbc.table.primarykey}
        cache.jdbc.fqname.column=${jboss-cache-cl-
cache.jdbc.fqname.column}
        cache.jdbc.fqname.type=${jboss-cache-cl-cache.jdbc.fqname.type}
        cache.jdbc.node.column=${jboss-cache-cl-
cache.jdbc.node.column}
        cache.jdbc.node.type=${jboss-cache-cl-cache.jdbc.node.type}
        cache.jdbc.parent.column=${jboss-cache-cl-
cache.jdbc.parent.column}
        cache.jdbc.datasource=${jboss-cache-cl-
cache.jdbc.datasource}
      </properties>
    </loader>
  </loaders>
</jboss-cache>

```



Note

To prevent inconsistency related to the lock data, ensure that your cache loader is **org.exoplatform.services.jcr.impl.core.lock.jboss-cache.JDBCCacheLoader** and your database engine is transactional.

Template Variables

jboss-cache-cluster-name

Table name.

jboss-cache-cl-cache.jdbc.table.name

Unique cluster name.

jboss-cache-cl-cache.jdbc.table.create

Indicates whether to create the table during startup. Value can be true or false. If true, the table is created if it doesn't already exist. The default value is true.

jboss-cache-cl-cache.jdbc.table.drop

Indicates whether to drop the table during shutdown. Value can be true or false. The default value is true.

jboss-cache-cl-cache.jdbc.table.primarykey

The name of the primary key for the table.

jboss-cache-cl-cache.jdbc.fqn.column

FQN column name. The default value is 'fqn'.

jboss-cache-cl-cache.jdbc.fqn.type

FQN column type. The default value is **varchar(255)**.

jboss-cache-cl-cache.jdbc.node.column

Node contents column name. The default value is **node**.

jboss-cache-cl-cache.jdbc.node.type

node contents column type. The default value is **blob**. This type must specify a valid binary data type for the database used.

jboss-cache-cl-cache.jdbc.parent.column

Parent column name. The default value is **parent**.

jboss-cache-cl-cache.jdbc.datasource

JNDI name of the DataSource.

[Report a bug](#)

B.11.4.3. Query Handler Template

The query handler template is called **jboss-cache-indexer.xml**.

Example B.30. Indexer template

```
<jboss-cache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:jboss:jboss-cache-core:config:3.1">

  <locking useLockStriping="false" concurrencyLevel="500"
lockParentForChildInsertRemove="false"
```

```

        lockAcquisitionTimeout="20000" />
    <!-- Configure the TransactionManager -->
    <transaction
transactionManagerLookupClass="org.jboss.cache.transaction.JBossStandaloneJTAManagerLookup" />

        <clustering mode="replication" clusterName="${jboss-cache-cluster-name}">
            <stateRetrieval timeout="20000" fetchInMemoryState="false" />
            <sync />
        </clustering>
    </jboss-cache>

```

Template Variable

jboss-cache-cluster-name

Unique cluster name.

[Report a bug](#)

B.12. LockManager

The LockManager stores lock objects. It can lock or release objects as required. It is also responsible for removing stale locks. The parameter to remove stale locks is configured with **time-out** property.

The LockManager in the portal is implemented with **org.exoplatform.services.jcr.impl.core.lock.jboss-cache.CacheableLockManagerImpl**.

[Report a bug](#)

B.12.1. CacheableLockManagerImpl

CacheableLockManagerImpl stores lock objects in JBoss-cache, which implements **JDBCCacheLoader** to store locks in a database. Locks can be replicated. Locks can affect an entire cluster rather than a single node.

JBoss-cache has JDBCCacheLoader, so locks are stored in the database.

You can enable LockManager by adding lock-manager-configuration to workspace-configuration.

Example B.31. Enabling Lockmanager

```

<workspace name="ws">
    ...
    <lock-manager
class="org.exoplatform.services.jcr.impl.core.lock.jboss-cache.CacheableLockManagerImpl">
        <properties>
            <property name="time-out" value="15m" />
        ...
    </lock-manager>
</workspace>

```

```

    </properties>
  </lock-manager>
  ...
</workspace>

```

The parameter **time-out** represents interval to remove expired Locks. LockRemover separates threads, that periodically ask LockManager to remove stale locks.

See Also:

» [Section B.12.2, “JBoss Cache Configuration”](#)

[Report a bug](#)

B.12.2. JBoss Cache Configuration

A simple method to configure the LockManager is to place the JBoss Cache configuration file path into **CacheableLockManagerImpl** class. This method is useful to configure a single LockManager, for a specific purpose.



Note

This is not an efficient method for configuring the LockManager as it requires a JBoss Cache configuration file for each LockManager configuration in each workspace of each repository. The configuration set up can subsequently become quite difficult to manage.

[Report a bug](#)

B.12.3. Configuration of JBoss Cache for LockManager

A simple LockManager configuration is shown here.

Example B.32. LockManager Configuration file test-jboss-cache-lock.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<jboss-cache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="urn:jboss:jboss-cache-core:config:3.1">

  <locking useLockStriping="false" concurrencyLevel="500"
    lockParentForChildInsertRemove="false"
    lockAcquisitionTimeout="20000" />

  <clustering mode="replication" clusterName="{jboss-cache-cluster-
    name}">
    <stateRetrieval timeout="20000" fetchInMemoryState="false" />
    <sync />
  </clustering>

  <loaders passivation="false" shared="true">
    <!-- All the data of the JCR locks needs to be loaded at startup
  -->

```



```

<preload>
  <node fqn="/" />
</preload>
<!--
For another cache-loader class you should use another template
with
cache-loader specific parameters
->
<loader
class="org.exoplatform.services.jcr.impl.core.lock.jbosscache.JDBCCache
Loader" async="false" fetchPersistentState="false"
  ignoreModifications="false" purgeOnStartup="false">
  <properties>
    cache.jdbc.table.name=${jbosscache-cl-
cache.jdbc.table.name}
    cache.jdbc.table.create=${jbosscache-cl-
cache.jdbc.table.create}
    cache.jdbc.table.drop=${jbosscache-cl-
cache.jdbc.table.drop}
    cache.jdbc.table.primarykey=${jbosscache-cl-
cache.jdbc.table.primarykey}
    cache.jdbc.fqn.column=${jbosscache-cl-
cache.jdbc.fqn.column}
    cache.jdbc.fqn.type=${jbosscache-cl-cache.jdbc.fqn.type}
    cache.jdbc.node.column=${jbosscache-cl-
cache.jdbc.node.column}
    cache.jdbc.node.type=${jbosscache-cl-cache.jdbc.node.type}
    cache.jdbc.parent.column=${jbosscache-cl-
cache.jdbc.parent.column}
    cache.jdbc.datasource=${jbosscache-cl-
cache.jdbc.datasource}
  </properties>
</loader>
</loaders>
</jbosscache>

```



Note

To prevent any consistency issue regarding the lock data, please ensure that your cache loader is

org.exoplatform.services.jcr.impl.core.lock.jbosscache.JDBCCacheLoader and your database engine is transactional.

[Report a bug](#)

B.12.4. LockManager Configuration Template

All configurable parameters are filled by templates and replaced by LockManagers configuration parameters:

Example B.33. LockManager configuration template

```

<lock-manager
class="org.exoplatform.services.jcr.impl.core.lock.infinispan.ISPNCache
ableLockManagerImpl">
  <properties>
    <property name="time-out" value="15m" />
    <property name="infinispan-configuration"
value="conf/standalone/cluster/test-infinispan-lock.xml" />
    <property name="jgroups-configuration" value="udp-mux.xml" />
    <property name="infinispan-cluster-name" value="JCR-cluster" />
    <property name="infinispan-cl-cache.jdbc.table.name" value="lk"
/>
    <property name="infinispan-cl-cache.jdbc.table.create"
value="true" />
    <property name="infinispan-cl-cache.jdbc.table.drop"
value="false" />
    <property name="infinispan-cl-cache.jdbc.id.column" value="id"
/>
    <property name="infinispan-cl-cache.jdbc.data.column"
value="data" />
    <property name="infinispan-cl-cache.jdbc.timestamp.column"
value="timestamp" />
    <property name="infinispan-cl-cache.jdbc.datasource"
value="jdbcjcr" />
    <property name="infinispan-cl-cache.jdbc.connectionFactory"
value="org.exoplatform.services.jcr.infinispan.ManagedConnectionFactory
" />
  </properties>
</lock-manager>

```

Configuration requirements

- ✱ **infinispan-cl-cache.jdbc.id.type**, **infinispan-cl-cache.jdbc.data.type** and **infinispan-cl-cache.jdbc.timestamp.type** are injected in the Infinispan configuration into the property respectively **idColumnType**, **dataColumnType** and **timestampColumnType**.

You can set the data types according to your database type or set it as AUTO or do not set at all. Data type is detected automatically.

[Report a bug](#)

B.12.5. Creating udp-mux.xml

jgroups-configuration is moved to separate the configuration file - **udp-mux.xml**.

The **udp-mux.xml** file is a common JGroup configuration file for all components such as QueryHandler, Cache, and LockManager, but you can create your own configuration.

Example B.34. udp-mux.xml

```

<config>
  <UDP
    singleton_name="JCR-cluster"
    mcast_addr="${jgroups.udp.mcast_addr:228.10.10.10}"
  >

```

```

mcast_port="${jgroups.udp.mcast_port:45588}"
tos="8"
ucast_recv_buf_size="20000000"
ucast_send_buf_size="640000"
mcast_recv_buf_size="25000000"
mcast_send_buf_size="640000"
loopback="false"
discard_incompatible_packets="true"
max_bundle_size="64000"
max_bundle_timeout="30"
use_incoming_packet_handler="true"
ip_ttl="${jgroups.udp.ip_ttl:2}"
enable_bundling="false"
enable_diagnostics="true"
thread_naming_pattern="cl"

use_concurrent_stack="true"

thread_pool.enabled="true"
thread_pool.min_threads="2"
thread_pool.max_threads="8"
thread_pool.keep_alive_time="5000"
thread_pool.queue_enabled="true"
thread_pool.queue_max_size="1000"
thread_pool.rejection_policy="discard"

oob_thread_pool.enabled="true"
oob_thread_pool.min_threads="1"
oob_thread_pool.max_threads="8"
oob_thread_pool.keep_alive_time="5000"
oob_thread_pool.queue_enabled="false"
oob_thread_pool.queue_max_size="100"
oob_thread_pool.rejection_policy="Run" />

<PING timeout="2000"<config xmlns="urn:org:jgroups"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:org:jgroups
http://www.jgroups.org/schema/JGroups-3.2.xsd">
  <UDP
    singleton_name="JCR-cluster"
    mcast_port="${jgroups.udp.mcast_port:45588}"
    tos="8"
    ucast_recv_buf_size="20M"
    ucast_send_buf_size="640K"
    mcast_recv_buf_size="25M"
    mcast_send_buf_size="640K"
    loopback="true"
    max_bundle_size="64K"
    max_bundle_timeout="30"
    ip_ttl="${jgroups.udp.ip_ttl:8}"
    enable_bundling="true"
    enable_diagnostics="true"
    thread_naming_pattern="cl"

    timer_type="old"
    timer.min_threads="4"

```

```

        timer.max_threads="10"
        timer.keep_alive_time="3000"
        timer.queue_max_size="500"

        thread_pool.enabled="true"
        thread_pool.min_threads="2"
        thread_pool.max_threads="8"
        thread_pool.keep_alive_time="5000"
        thread_pool.queue_enabled="true"
        thread_pool.queue_max_size="10000"
        thread_pool.rejection_policy="discard"

        oob_thread_pool.enabled="true"
        oob_thread_pool.min_threads="1"
        oob_thread_pool.max_threads="8"
        oob_thread_pool.keep_alive_time="5000"
        oob_thread_pool.queue_enabled="false"
        oob_thread_pool.queue_max_size="100"
        oob_thread_pool.rejection_policy="Run"/>

<PING timeout="2000"
        num_initial_members="20"/>
<MERGE2 max_interval="30000"
        min_interval="10000"/>
<FD_SOCKET/>
<FD_ALL/>
<VERIFY_SUSPECT timeout="1500" />
<BARRIER />
<pbcast.NAKACK2 xmit_interval="1000"
        xmit_table_num_rows="100"
        xmit_table_msgs_per_row="2000"
        xmit_table_max_compaction_time="30000"
        max_msg_batch_size="500"
        use_mcast_xmit="false"
        discard_delivered_msgs="true"/>
<UNICAST xmit_interval="2000"
        xmit_table_num_rows="100"
        xmit_table_msgs_per_row="2000"
        xmit_table_max_compaction_time="60000"
        conn_expiry_timeout="60000"
        max_msg_batch_size="500"/>
<pbcast.STABLE stability_delay="1000" desired_avg_gossip="50000"
        max_bytes="4M"/>
<pbcast.GMS print_local_addr="true" join_timeout="3000"
        view_bundling="true"/>
<UFC max_credits="2M"
        min_threshold="0.4"/>
<MFC max_credits="2M"
        min_threshold="0.4"/>
<FRAG2 frag_size="60K" />
<RSVP resend_interval="2000" timeout="10000"/>
<pbcast.STATE_TRANSFER />
<!-- pbcast.FLUSH /-->
</config>

```

[Report a bug](#)

B.12.6. FQN type and node type in different Databases

Table B.5. Data Types in Different Databases

| DataBase name | Node data type | FQN data type |
|---------------|----------------|---------------|
| default | BLOB | VARCHAR(512) |
| HSSQL | OBJECT | VARCHAR(512) |
| MySQL | LONGBLOB | VARCHAR(512) |
| ORACLE | BLOB | VARCHAR2(512) |
| PostgreSQL | bytea | VARCHAR(512) |
| MSSQL | VARBINARY(MAX) | VARCHAR(512) |
| DB2 | BLOB | VARCHAR(512) |
| Sybase | IMAGE | VARCHAR(512) |

[Report a bug](#)

B.12.7. Lock Migration

There are three methods for lock migration.

Lock Migration Methods

New Shareable Cache feature is not used and all locks are kept after migration.

Procedure B.7. Shareable Cache is not used and locks are kept

1. Ensure that the same lock tables are used in configuration.
2. Start the server.

New Shareable Cache feature is not used and all locks are removed after migration.

Procedure B.8. Shareable Cache is not used and locks are removed

1. Ensure that the same lock tables is used in configuration.
2. Start the sever with system property -
Dorg.exoplatform.jcr.locks.force.remove=true.
3. Stop the server
4. Start the server without the system property -
Dorg.exoplatform.jcr.locks.force.remove.

New Shareable Cache feature is used and all locks are removed after migration.

Procedure B.9. Shareable Cache is used

1. Start the sever with system property -
Dorg.exoplatform.jcr.locks.force.remove=true.
2. Stop the server.

3. Start the server without system property -
`Dorg.exoplatform.jcr.locks.force.remove.`

4. **Optional**

Manually remove old tables for lock.

[Report a bug](#)

B.13. JCR Indexing

JCR offers indexing strategies for standalone and clustered environments. This ensure that JCR use the advantages of running in a single JVM and ensure efficient use of resources available in cluster.

JCR uses Lucene library as underlying search and indexing engine, but it has several limitations that greatly reduce possibilities and limits the usage of cluster advantages. So, eXo JCR offers two strategies that are suitable for the use-cases. These use-cases are clustered with shared index and local indexes.

[Report a bug](#)

B.13.1. Standalone Index

Standalone strategy provides a stack of indexes for greater performance within a single JVM.

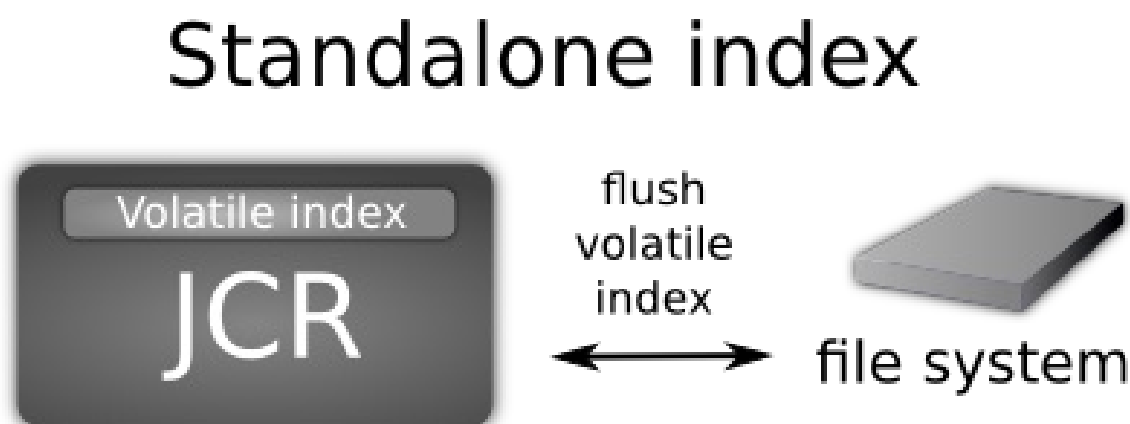


Figure B.3. Standalone Index Diagram

Standalone Index combines in-memory buffer index directory with delayed file-system flushing. This index is called Volatile and it is invoked in searches. Under specific conditions volatile index is flushed to the persistent storage as new index directory. This allows to achieve great results for write operations.

[Report a bug](#)

B.13.2. Local Index

Clustered implementation with local indexes combines in-memory buffer index directory with delayed file-system flushing. This index is called Volatile and is invoked in searches. Under specific conditions volatile index is flushed to the persistent storage (file system) as new index directory. This enables high performance for write operations.

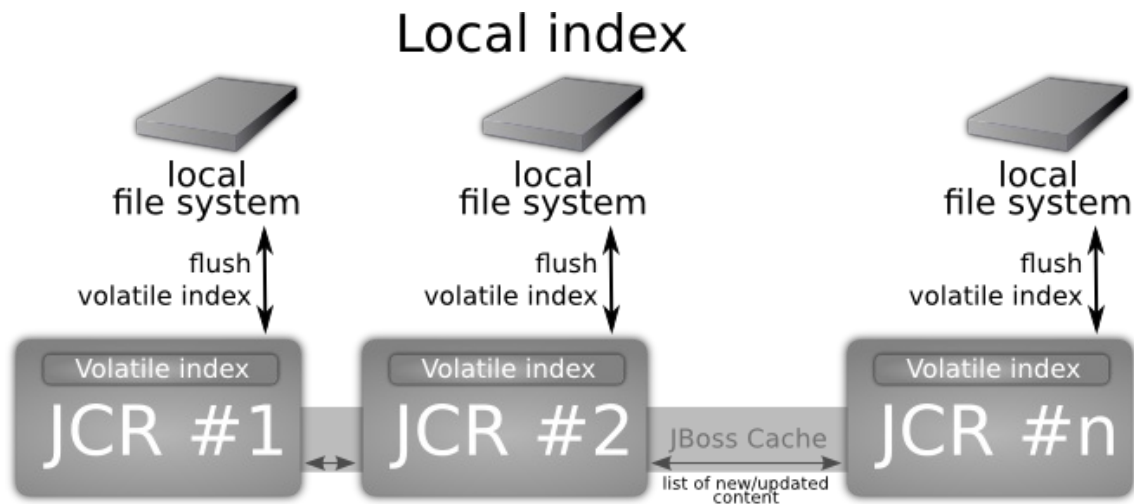


Figure B.4. Local Index Diagram

Clustered Index is designed for clustered environment. It has additional mechanisms for data delivery within cluster.

Text extraction and content operations, such as write operation are done on the same node. The **documents** (Lucene term that means block of data ready for indexing) prepared are replicated within cluster nodes and processed by local indexes. So each cluster instance has the same index content

. When new node joins the cluster, the index is created.



Warning

To create the index, you can copy the index manually but this is not intended for use.

If no initial index is found JCR uses automated scenarios. They are controlled via configuration parameter ***index-recovery-mode***. This parameter does the re-indexing from database or copying from another cluster node.



Note

Due to certain reasons having a multiple index copies on each instance is costly. So shared index is used instead.

[Report a bug](#)

B.13.3. Shared Index

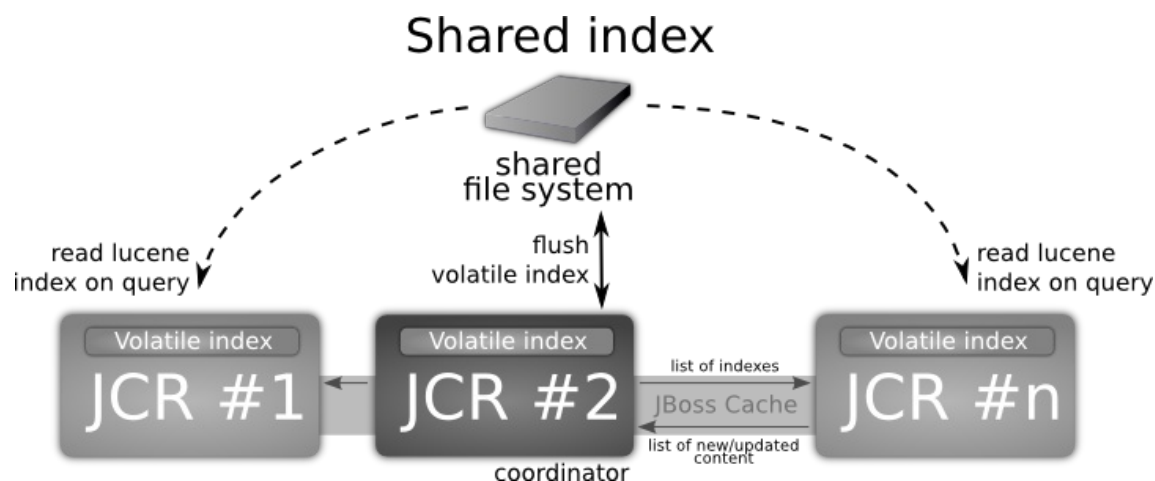


Figure B.5. Shared Index Diagram

Shared indexing combines advantages of in-memory index and shared persistent index providing near real time search capabilities. This strategy allows nodes to index data in their own volatile (in-memory) indexes, but persistent indexes are managed by single coordinator node.

Each cluster instance has a read access for shared index to perform queries combining search results found in the in-memory index. For example, a shared folder must be configured in your system environment, which is mounted NFS folder.

In rare instances, this strategy can have different volatile indexes within cluster instances for a fraction of time, and in a few seconds the index is updated.

Shared index is consistent, stable, and slow. Local index is fast and takes time for re-synchronization, when cluster node is leaving a cluster for a small period of time. RSync-based index solves this problem along with local file system advantages in term of speed.

[Report a bug](#)

B.13.4. RSync-based Index

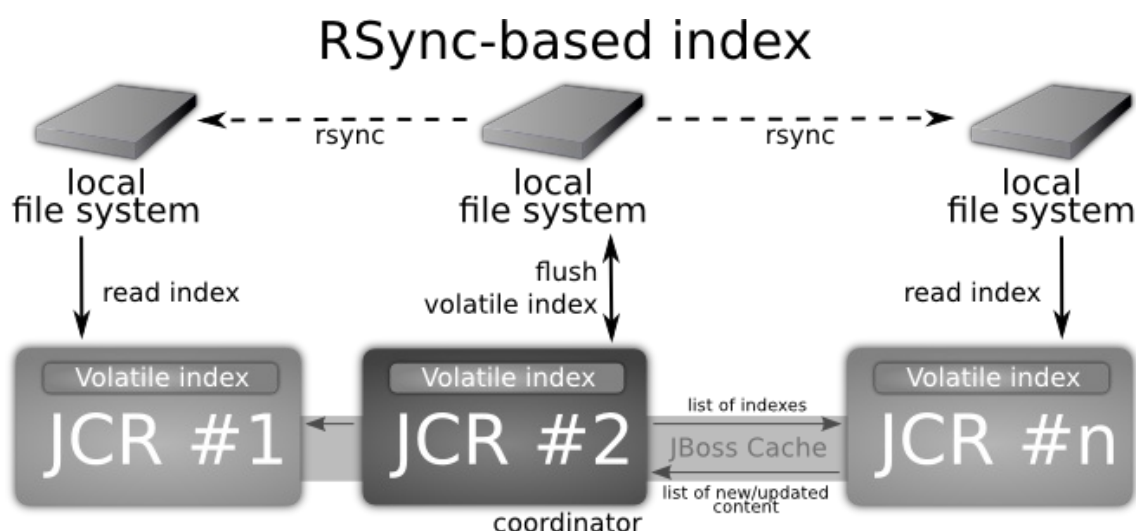


Figure B.6. RSync-based Index Diagram

RSync-based indexing is the same as shared indexing, but stores actual data on local file system, instead of shared. This triggers a synchronization job, that works on the level of file blocks, synchronizing modified data.

The Coordinator node in the cluster modifies index files. When data persists, the corresponding command is evoked and synchronization jobs start over the cluster.

See Also:

✎ [Section B.6, “Configuring Search”](#)

[Report a bug](#)

B.13.5. Query-handler configuration

Example B.35. Sample configuration file

```
<workspace name="ws">
  <query-handler
class="org.exoplatform.services.jcr.impl.core.query.lucene.SearchIndex"
>
    <properties>
      <property name="index-dir" value="sharedir/index/db1/ws" />
      <property name="changesfilter-class"
value="org.exoplatform.services.jcr.impl.core.query.jboss-cache.JBossCacheIndexChangesFilter" />
      <property name="jboss-cache-configuration" value="jboss-cache-indexer.xml" />
      <property name="jgroups-configuration" value="udp-mux.xml" />
      <property name="jgroups-multiplexer-stack" value="true" />
      <property name="jboss-cache-cluster-name" value="JCR-cluster-indexer-ws" />
      <property name="max-volatile-time" value="60" />
      <property name="rdbms-reindexing" value="true" />
      <property name="reindexing-page-size" value="1000" />
      <property name="index-recovery-mode" value="from-coordinator" />
    />
    <property name="index-recovery-filter"
value="org.exoplatform.services.jcr.impl.core.query.lucene.DocNumberRecoveryFilter" />
      <property name="indexing-thread-pool-size" value="16" />
    </properties>
  </query-handler>
</workspace>
```

[Report a bug](#)

B.13.5.1. Configuration properties

Table B.6.

| Property name | Description |
|---------------------------|---|
| index-dir | path to index |
| changesfilter-class | template of JBoss-cache configuration for all query-handlers in repository |
| jboss-cache-configuration | template of JBoss-cache configuration for all query-handlers in repository |
| jgroups-configuration | jgroups-configuration is template configuration for all components (search, cache, locks) [Add link to document describing template configurations] |
| jgroups-multiplexer-stack | [TODO about jgroups-multiplexer-stack - add link to JBoss doc] |
| jboss-cache-cluster-name | cluster name (must be unique) |
| max-volatile-time | max time to live for Volatile Index |
| rdbms-reindexing | indicate that need to use rdbms reindexing mechanism if possible, the default value is true |
| reindexing-page-size | maximum amount of nodes which can be retrieved from storage for re-indexing purpose, the default value is 100 |
| index-recovery-mode | If the parameter has been set to from-indexing , so a full indexing will be automatically launched (default behavior), if the parameter has been set to from-coordinator , the index will be retrieved from coordinator |
| index-recovery-filter | Defines implementation class or classes of RecoveryFilters, the mechanism of index synchronization for Local Index strategy. |
| async-reindexing | Controls the process of re-indexing on JCR's startup. If this flag is set, indexing will be launched asynchronously, without blocking the JCR. Default is " false ". |
| indexing-thread-pool-size | Defines the total amount of indexing threads. |
| max-volatile-size | The maximum volatile index size in bytes until it is written to disk. The default value is 1048576 (1MB). |

[Report a bug](#)

B.13.5.2. Improve Query Performance with postgresQL and rdbms-reindexing

The performance of the queries used while indexing can be improved by using **postgresQL** parameter and setting **rdbms-reindexing** parameter value as **true**.

Procedure B.10. Improve query performance

1. Set the parameter **enable_seqscan** to **off**.

OR

Set **default_statistics_target** to at least **50**.

2. Restart DB server and analyze the JCR_SVALUE or JCR_MVALUE table.

[Report a bug](#)

B.13.5.3. Improve Query Performance with DB2 and rdbms-reindexing

The performance of the queries used while indexing can be improved by using **DB2** and setting ***rdbms-reindexing*** to **true**.

Procedure B.11. Improve query performance

- ✱ Collect statistics on tables by running the following query for **JCR_SITEM** (or **JCR_MITEM**) and **JCR_SVALUE** (or **JCR_MVALUE**) tables:

```
RUNSTATS ON TABLE <scheme>.<table> WITH DISTRIBUTION AND INDEXES ALL
```

[Report a bug](#)

B.13.5.4. Cluster-ready indexing for shared index

For cluster-ready implementations JBoss Cache, JGroups and Changes Filter values must be defined.

Shared index requires a remote or shared file system, for example NFS, SMB. Indexing directory ***indexDir*** value must point to the file system.

Example B.36. Enable shared indexing

To enable shared index implementation, set ***changesfilter-class*** to ***org.exoplatform.services.jcr.impl.core.query.jbosscache.JBossCacheIndexChangesFilter***

```
<workspace name="ws">
  <query-handler
    class="org.exoplatform.services.jcr.impl.core.query.lucene.SearchIndex"
  >
    <properties>
      <property name="index-dir" value="/mnt/nfs_drive/index/db1/ws"
    />
      <property name="changesfilter-class"
        value="org.exoplatform.services.jcr.impl.core.query.jbosscache.JBossCacheIndexChangesFilter" />
      <property name="jbosscache-configuration" value="jbosscache-indexer.xml" />
      <property name="jgroups-configuration" value="udp-mux.xml" />
      <property name="jgroups-multiplexer-stack" value="true" />
      <property name="jbosscache-cluster-name" value="JCR-cluster-indexer-ws" />
      <property name="max-volatile-time" value="60" />
      <property name="rdbms-reindexing" value="true" />
      <property name="reindexing-page-size" value="1000" />
      <property name="index-recovery-mode" value="from-coordinator"
    />
    </properties>
  </query-handler>
</workspace>
```

[Report a bug](#)

B.13.5.5. System Requirements for RSync Index

- Rsync-based indexing strategy is an installed and properly configured RSync utility.
- Rsync-based indexing must be accessible by calling "rsync" without defining it's full path.
- Each cluster node must have a running RSync Server supporting "rsync://" protocol.
- Path for index for each workspace must be the same across the cluster, **/var/data/index/repository-name/workspace-name**.
- Each cluster node must have a running RSync Server supporting "rsync://" protocol.
- RSync Server configuration must share some of index's parent folders. For example, **/var/data/index**. In other words, index is stored inside of RSync Server shared folder.

[Report a bug](#)

B.13.5.6. RSync Index Configuration

RSync configuration is similar to Shared Index, it just requires some additional parameters for RSync options. If they are present, JCR switches from shared to RSync-based index.

Example B.37. RSync configuration

```
<query-handler
class="org.exoplatform.services.jcr.impl.core.query.lucene.SearchIndex"
>
  <properties>
    <property name="index-dir"
value="/var/data/index/repository1/production" />
    <property name="changesfilter-class"
value="org.exoplatform.services.jcr.impl.core.query.ispn.ISPNIndexChang
esFilter" />
    <property name="infinispan-configuration"
value="jar:/conf/portal/cluster/infinispan-indexer.xml" />
    <property name="jgroups-configuration"
value="jar:/conf/portal/cluster/udp-mux.xml" />
    <property name="infinispan-cluster-name" value="JCR-cluster" />
    <property name="max-volatile-time" value="60" />
    <property name="rsync-entry-name" value="index" />
    <property name="rsync-entry-path" value="/var/data/index" />
    <property name="rsync-port" value="8085" />
    <property name="rsync-user" value="rsyncexo" />
    <property name="rsync-password" value="exo" />
  </properties>
</query-handler>
```

RSync uses **rsync-user** and **rsync-password** for authentication.

They are optional and can be skipped if RSync Server configured to accept anonymous identity.

Example B.38. RSync Server configuration

```
uid = nobody
gid = nobody
use chroot = no
port = 8085
log file = rsyncd.log
pid file = rsyncd.pid
[index]
    path = /var/data/index
    comment = indexes
    read only = true
    auth users = rsyncexo
    secrets file= rsyncd.secrets
```

This sample configuration shares folder **/var/data/index** as an entry . The parameters should match **rsync-entry-name**, **rsync-entry-path**, and **rsync-port** properties in JCR configuration.

**Note**

index-dir is a descendant folder of RSync shared folder and those paths are the same on each cluster node.

[Report a bug](#)

B.13.5.7. Cluster-ready indexing for local index**Example B.39. Enable local indexing**

To use cluster-ready strategy based on local indexes, when each node owns a copy of index on local file system, the indexing directory must point to any folder on local file system.

The **changesfilter-class** must be set to **org.exoplatform.services.jcr.impl.core.query.jbosscache.LocalIndexChangesFilter**.

```
<workspace name="ws">
  <query-handler
class="org.exoplatform.services.jcr.impl.core.query.lucene.SearchIndex"
  >
    <properties>
      <property name="index-dir" value="/mnt/nfs_drive/index/db1/ws"
/>
      <property name="changesfilter-class"
value="org.exoplatform.services.jcr.impl.core.query.jbosscache.LocalIndexChangesFilter" />
      <property name="jbosscache-configuration" value="jbosscache-indexer.xml" />
      <property name="jgroups-configuration" value="udp-mux.xml" />
      <property name="jgroups-multiplexer-stack" value="true" />
      <property name="jbosscache-cluster-name" value="JCR-cluster-
```

```

indexer-ws" />
    <property name="max-volatile-time" value="60" />
    <property name="rdbms-reindexing" value="true" />
    <property name="reindexing-page-size" value="1000" />
    <property name="index-recovery-mode" value="from-coordinator"
/>
    </properties>
</query-handler>
</workspace>

```

[Report a bug](#)

B.13.5.8. Local Index Recovery Filters

All nodes that are joining a cluster for the first time or nodes joining after downtime, must be in a synchronized state.

When using shared value storages, databases and indexes, cluster nodes are synchronized at any given time. But this is not the case in a local index strategy.

If a new node joins a cluster, without an index it is retrieved or recreated. Nodes can be restarted and thus the index is not empty. By default, even though the existing index is looks updated, it can be outdated.

The portal JCR offers a mechanism called **RecoveryFilters** that automatically retrieve index for the joining node startup. This feature uses a set of filters that are defined in **QueryHandler** configuration.

Example B.40. QueryHandler configuration

```

<property name="index-recovery-filter"
value="org.exoplatform.services.jcr.impl.core.query.lucene.DocNumberRe
coveryFilter" />

```

Filter numbers are not limited so they can be combined:

```

<property name="index-recovery-filter"
value="org.exoplatform.services.jcr.impl.core.query.lucene.DocNumberRe
coveryFilter" />
    <property name="index-recovery-filter"
value="org.exoplatform.services.jcr.impl.core.query.lucene.SystemProper
tyRecoveryFilter" />

```

If any one returns fires, the index is re-synchronized. This feature uses standard index recovery mode defined by previously described parameter (can be "from-indexing" (default) or "from-coordinator")

```

<property name="index-recovery-mode" value="from-coordinator" />

```

[Report a bug](#)

B.13.5.9. Filter Implementations

There are multiple filter implementations.

org.exoplatform.services.jcr.impl.core.query.lucene.DummyRecoveryFilter

Always returns true, when the index must be forcibly resynchronized each time.

org.exoplatform.services.jcr.impl.core.query.lucene.SystemPropertyRecoveryFilter

Returns value of system property

org.exoplatform.jcr.recoveryfilter forcereindexing. Index recovery is controlled from the top without changing documentation using system properties.

org.exoplatform.services.jcr.impl.core.query.lucene.ConfigurationPropertyRecoveryFilter

Returns value of **QueryHandler** configuration property **index-recovery-filter-forcereindexing**. So index recovery is controlled from configuration, separately for each workspace. For example:

```
<property name="index-recovery-filter"
value="org.exoplatform.services.jcr.impl.core.query.lucene.ConfigurationPropertyRecoveryFilter" />
  <property name="index-recovery-filter-forcereindexing"
value="true" />
```

org.exoplatform.services.jcr.impl.core.query.lucene.DocNumberRecoveryFilter

Checks the number of documents in index on coordinator side and self-side. It returns **true** if the count differs.

The advantage of this filter, is that it skips reindexing for workspaces where the index is not modified.

For example, if there are ten repositories with three workspaces each and only one is heavily used in the cluster, this filter reindexes those workspaces that have changed, without affecting other indexes.

This reduces start up time.

[Report a bug](#)

B.13.5.10. JBoss-Cache template configuration

JBoss-Cache template configuration for query handler is same for both clustered strategies. The configuration file is **jboss-cache-indexer.xml**

Example B.41. JBoss-Cache configuration for query handler

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss-cache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:jboss:jboss-cache-core:config:3.1">
  <locking useLockStriping="false" concurrencyLevel="50000"
lockParentForChildInsertRemove="false"
  lockAcquisitionTimeout="20000" />
  <!-- Configure the TransactionManager -->
  <transaction
transactionManagerLookupClass="org.jboss.cache.transaction.JBossStandalone
```

```

        JTAManagerLookup" />
    <clustering mode="replication" clusterName="${jboss-cache-cluster-
name}">
        <stateRetrieval timeout="20000" fetchInMemoryState="false" />
        <jgroupsConfig multiplexerStack="jcr.stack" />
        <sync />
    </clustering>
    <!-- Eviction configuration -->
    <eviction wakeUpInterval="5000">
        <default algorithmClass="org.jboss.cache.eviction.FIFOAlgorithm"
eventQueueSize="1000000">
            <property name="maxNodes" value="10000" />
            <property name="minTimeToLive" value="60000" />
        </default>
    </eviction>
</jboss-cache>

```

See Also:

✱ [Section B.11, “Configuring JBoss Cache”](#)

[Report a bug](#)

B.13.6. Asynchronous Re-indexing

To manage large data set using JCR in production environment requires special operations with Indexes. These operations include recreation of index, which is called re-indexing.

Re-indexing is required to recover from hardware faults, hard restarts, data-corruption, migrations, and JCR updates that brings new features related to index. Index re-creation is requested on server startup or during runtime.

[Report a bug](#)

B.13.6.1. On startup indexing

eXo JCR supports RDBMS re-indexing, which is faster than ordinary indexing. To configure RDBMS reindexing, the **QueryHandler** parameter *rdbms-reindexing* is set to **true**.

[Report a bug](#)

B.13.6.2. Asynchronous Indexing on startup

The server startup is blocked when the indexing process is in progress during server startup. Asynchronous indexing on startup unblocks the server at startup .

In asynchronous indexing on startup, all indexing operations are performed in the background without blocking the repository.

To configure asynchronous indexing on startup the value of the *async-reindexing* parameter in **QueryHandler** configuration is set to **true**.

With active asynchronous indexation, the JCR starts without active indexes. You can execute queries on JCR without exceptions but no results are returned until index creation is completed.

The following code shows the usage of the parameter `QueryManagerImpl` to verify the state of the index, the return value of `isOnline()` is true.:

- ✱ The OFFLINE state indicates that the index is currently re-creating. When the state changes, a corresponding log event is printed.
 - When the background index task starts the index is switched to OFFLINE, with following log event:

```
[INFO] Setting index OFFLINE (repository/production[system]).
```

- ✱ When the indexing process is completed, the following two events are logged:

```
[INFO] Created initial index for 143018 nodes
(repository/production[system]).
[INFO] Setting index ONLINE (repository/production[system]).
```

These two log lines indicates the end of process for workspace named system.

[Report a bug](#)

B.13.6.3. Hot Asynchronous Workspace Re-indexing using JMX

Due to hard system faults, system upgradation errors, migration issues and so on the index gets corrupted. Hot Asynchronous Workspace Re-indexing allows service administrators to launch the process in the background without stopping or blocking the application. Hot Asynchronous Workspace Re-indexing uses a JMX-compatible console.

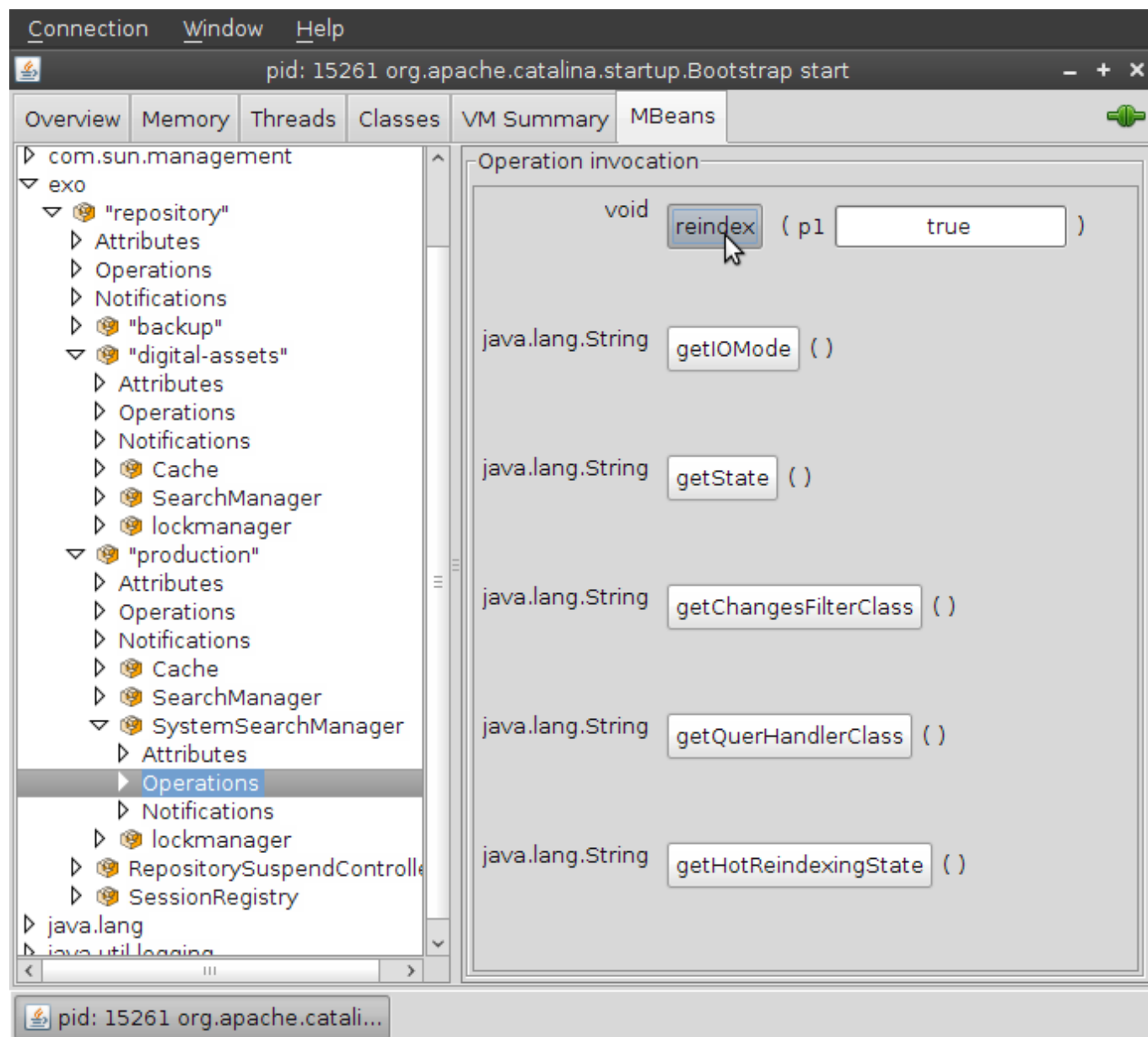


Figure B.7. JMX Jconsole

The server can continue working as expected while the index is recreated.

This depends on the flag ***allow queries*** being passed via JMX interface to the reindex operation invocation. If the flag is set, the application continues working.



Important

In hot asynchronous workspace re-indexing method, the index is frozen while the background task is running.

This means that queries are performed on a version of the index present at the moment the indexing task is started, and that data written into the repository after startup will not be available through the search until process completes.

Data added during re-indexation is also indexed. The data is available when reindexing is complete. The JCR makes a snapshot of indexes at the invocation of the asynchronous indexing task and uses that snapshot for searches.

When the operation is finished, the stale index is replaced by the newly created index, which included any newly added data.

If the ***allow queries*** flag is set to **false**, then all queries will throw an exception while task is running. The current state can be acquired using the following JMX operation:

- `getHotReindexingState()` - returns information about latest invocation: start time, if in progress or finish time if done.

[Report a bug](#)

B.13.6.4. Notices

You cannot launch hot re-indexing using JMX cannot if the index is in offline mode. This means that the index is currently busy in other operations, such as re-indexing at startup, copying in cluster to another node and so on.

Hot Asynchronous Reindexing via JMX and **on startup** reindexing are different features. You cannot get the state of startup reindexing using command **`getHotReindexingState`** in JMX interface.

Common JMX operations

- `getIOMode` - returns current index IO mode (`READ_ONLY` / `READ_WRITE`), belongs to clustered configuration states.
- `getState` - returns current state, (`ONLINE` / `OFFLINE`).

[Report a bug](#)

B.13.7. Lucene tuning

JCR Indexing is based on the Lucene indexing library. JCR Indexing uses directories to store index and manages access to index by Lock Factories.

By default, the JCR implementation uses optimal combination of Directory implementation and Lock Factory implementation.

- **`SimpleFSDirectory`** is used in Windows environments and the **`NIOFSDirectory`** implementation is used in non-Windows systems.
- **`NativeFSLockFactory`** is an optimal solution for a wide variety of cases including clustered environment with NFS shared resources.

You can override the default settings in the system properties.

- The properties, **`org.exoplatform.jcr.lucene.store.FSDirectoryLockFactoryClass`** and **`org.exoplatform.jcr.lucene.FSDirectory.class`** control the default behavior.
 - **`org.exoplatform.jcr.lucene.store.FSDirectoryLockFactoryClass`** defines the implementation of abstract Lucene **`LockFactory`** class.
 - **`org.exoplatform.jcr.lucene.FSDirectory.class`** sets the implementation class for **`FSDirectory`** instances.



Important

JCR allows you to change the implementation classes of Lucene internals but it does not guarantee the stability and functionality of the changes.

For more information, see the Lucene documentation located at <http://lucene.apache.org/core/documentation.html>.

[Report a bug](#)

B.13.7.1. JBossTransactionsService

JBossTransactionsService implements eXo TransactionService and provides access to JBoss Transaction Service (JBossTS) JTA implementation using eXo container dependency.

TransactionService is used in JCR cache

`org.exoplatform.services.jcr.impl.dataflow.persistent.jbosscache.JBossCacheWorkspaceStorageCache` implementation.

Example B.42. JBossTransactionsService Configuration

```

<component>
  <key>org.exoplatform.services.transaction.TransactionService</key>

  <type>org.exoplatform.services.transaction.jbosscache.JBossTransactionsService</type>
  <init-params>
    <value-param>
      <name>timeout</name>
      <value>3000</value>
    </value-param>
  </init-params>
</component>

```

timeout - XA transaction timeout in seconds.

[Report a bug](#)

B.13.7.2. JCR Query Use-cases

The JCR supports two query languages; JCR and XPath. A query, whether XPath or SQL, specifies a subset of nodes within a workspace, called the result set. The result set constitutes all the nodes in the workspace that meet the constraints stated in the query.

Example B.43. SQL Query Creation and Execution

```

// get QueryManager
QueryManager queryManager = workspace.getQueryManager();
// make SQL query
Query query = queryManager.createQuery("SELECT * FROM nt:base ",

```

```
Query.SQL);
// execute query
QueryResult result = query.execute();
```

Example B.44. XPath Query Creation and Execution

```
// get QueryManager
QueryManager queryManager = workspace.getQueryManager();
// make XPath query
Query query = queryManager.createQuery("//element(*,nt:base)",
Query.XPATH);
// execute query
QueryResult result = query.execute();
```

Example B.45. Query Result Processing

```
// fetch query result
QueryResult result = query.execute();
```

To fetch the nodes:

```
NodeIterator it = result.getNodes();
```

The results can be formatted in a table:

```
// get column names
String[] columnNames = result.getColumnNames();
// get column rows
RowIterator rowIterator = result.getRows();
while(rowIterator.hasNext()){
    // get next row
    Row row = rowIterator.nextRow();
    // get all values of row
    Value[] values = row.getValues();
}
```

[Report a bug](#)

B.13.8. Searching Repository Content

JCR configuration file is located at:

\$JPP_HOME/gatein/gatein.ear/portal.war/portal/WEB-INF/conf/jcr/repository-configuration.xml.

You can search the JCR content repository using various search techniques such as bi-directional range iteration, fuzzy search, synonym search and so on.

See Also:

» [Section B.6, “Configuring Search”](#)

[Report a bug](#)

B.13.8.1. Bi-directional RangIterator

Bi-directional **NodeIterator** is implemented using **QueryResult.getNodes()**.

The **TwoWayRangeIterator** interface is shown in the following example.

Example B.46. TwoWayRangeIterator interface

```
/**
 * Skip a number of elements in the iterator.
 *
 * @param skipNum the non-negative number of elements to skip
 * @throws java.util.NoSuchElementException if skipped past the first
 * element
 *           in the iterator.
 */
public void skipBack(long skipNum);
```

Example B.47. Usage of TwoWayRangeIterator

```
NodeIterator iter = queryResult.getNodes();
while (iter.hasNext()) {
    if (skipForward) {
        iter.skip(10); // Skip 10 nodes in forward direction
    } else if (skipBack) {
        TwoWayRangeIterator backIter = (TwoWayRangeIterator) iter;
        backIter.skipBack(10); // Skip 10 nodes back
    }
    .....
}
```



Note

Bi-directional NodeIterator is **not supported** in two cases:

1. SQL query: select * from nt:base
2. XPath query: //* .

[Report a bug](#)

B.13.8.2. Fuzzy Searches

JCR supports Lucene Fuzzy Searches. The following query **q** performs a fuzzy search:

```

QueryManager qman = session.getWorkspace().getQueryManager();
Query q = qman.createQuery("select * from nt:base where contains(field,
'ccccc~')", Query.SQL);
QueryResult res = q.execute();

```

[Report a bug](#)

B.13.8.3. Synonym Search

Searching with synonyms is integrated in the **jcr:contains()** function and uses the same syntax as synonym searches in web search engines such as Google. If a search term is prefixed by a tilde symbol (~), synonyms of the search term are taken into consideration.

Example B.48. Usage of the tilde symbol in search

```
SQL: select * from nt:resource where contains(., '~parameter')
```

```
XPath: //element(*, nt:resource)[jcr:contains(., '~parameter')]
```

Example B.49. Enabling Synonym Search

To enable synonym search you need to add a configuration parameter to the **query-handler** element in your JCR configuration file.

```

<param name="synonymprovider-config-path" value="..you path to
configuration file....."/>
<param name="synonymprovider-class"
value="org.exoplatform.services.jcr.impl.core.query.lucene.PropertiesSy
nonymProvider"/>

```

Example B.50. Synonym Provider Interface

```

/**
 * <code>SynonymProvider</code> defines an interface for a component
that
 * returns synonyms for a given term.
 */
public interface SynonymProvider {

    /**
     * Initializes the synonym provider and passes the file system
resource to
     * the synonym provider configuration defined by the configuration
value of
     * the <code>synonymProviderConfigPath</code> parameter. The
resource may be
     * <code>null</code> if the configuration parameter is not set.
     *
     * @param fsr the file system resource to the synonym provider
configuration.
     * @throws IOException if an error occurs while initializing the

```

```

synonym
    *
    *           provider.
    */
    public void initialize(InputStream fsr) throws IOException;

    /**
     * Returns an array of terms that are considered synonyms for the
    given
     * <code>term</code>.
     *
     * @param term a search term.
     * @return an array of synonyms for the given <code>term</code> or
    an empty
     *         array if no synonyms are known.
     */
    public String[] getSynonyms(String term);
}

```

[Report a bug](#)

B.13.9. Highlighting

An **ExcerptProvider** retrieves text excerpts for a node in the query result and marks up the words in the text that match the query terms.

By default, match highlighting is disabled because as it requires that additional information is written to the search index.

To enable this feature, you need to add a configuration parameter to the **query-handler** element in your JCR configuration file:

```
<param name="support-highlighting" value="true"/>
```

Additionally, there is a parameter that controls the format of the excerpt created. In JCR 1.9, the default is set to

org.exoplatform.services.jcr.impl.core.query.lucene.DefaultHTMLExcerpt. The configuration parameter for this setting is:

```
<param name="excerptprovider-class"
value="org.exoplatform.services.jcr.impl.core.query.lucene.DefaultXMLExcerpt"/>
```

[Report a bug](#)

B.13.9.1. DefaultXMLExcerpt

This excerpt provider creates an XML fragment of the following form:

```

<excerpt>
  <fragment>
    <highlight>exoplatform</highlight> implements both the mandatory
    XPath and optional SQL <highlight>query</highlight> syntax.
  </fragment>
</fragment>

```



```

    Before parsing the XPath <highlight>query</highlight> in
    <highlight>exoplatform</highlight>, the statement is surrounded
  </fragment>
</excerpt>

```

[Report a bug](#)

B.13.9.2. DefaultHTMLExcerpt

This excerpt provider creates an HTML fragment of the following form:

```

<div>
  <span>
    <strong>exoplatform</strong> implements both the mandatory XPath
    and optional SQL <strong>query</strong> syntax.
  </span>
  <span>
    Before parsing the XPath <strong>query</strong> in
    <strong>exoplatform</strong>, the statement is surrounded
  </span>
</div>

```

[Report a bug](#)

B.13.9.3. Usage

If you are using XPath, you must use the **rep:excerpt()** function in the last location step, just like you would select properties:

```

QueryManager qm = session.getWorkspace().getQueryManager();
Query q = qm.createQuery("//*[jcr:contains(.,
'exoplatform')]/(@Title|rep:excerpt(.))", Query.XPATH);
QueryResult result = q.execute();
for (RowIterator it = result.getRows(); it.hasNext(); ) {
  Row r = it.nextRow();
  Value title = r.getValue("Title");
  Value excerpt = r.getValue("rep:excerpt(.));
}

```

The above code searches for nodes that contain the word *exoplatform* and then gets the value of the **Title** property and an excerpt for each resultant node.

It is also possible to use a relative path in the call **Row.getValue()** while the query statement still remains the same. Also, you may use a relative path to a string property. The returned value will then be an excerpt based on string value of the property.

Both available excerpt providers will create fragments of about 150 characters and up to three fragments.

In SQL, the function is called **excerpt()** without the rep prefix, but the column in the **RowIterator** will nonetheless be labelled **rep:excerpt(.)**.

```

QueryManager qm = session.getWorkspace().getQueryManager();
Query q = qm.createQuery("select excerpt(.) from nt:resource where
contains(., 'exoplatform')", Query.SQL);

```

```

QueryResult result = q.execute();
for (RowIterator it = result.getRows(); it.hasNext(); ) {
    Row r = it.nextRow();
    Value excerpt = r.getValue("rep:excerpt(.)");
}

```

[Report a bug](#)

B.13.10. SpellChecker

The lucene based query handler implementation supports a pluggable spell-checker mechanism. By default, spell checking is not available, so you have to configure it.

The JCR currently provides an implementation class which uses the lucene-spellchecker.

The dictionary is derived from the fulltext, indexed content of the workspace and updated periodically. You can configure the refresh interval by selecting the available inner classes of `org.exoplatform.services.jcr.impl.core.query.lucene.spell.LuceneSpellChecker`:

Inner Classes available in SpellChecker

- ✧ `OneMinuteRefreshInterval`
- ✧ `FiveMinutesRefreshInterval`
- ✧ `ThirtyMinutesRefreshInterval`
- ✧ `OneHourRefreshInterval`
- ✧ `SixHoursRefreshInterval`
- ✧ `TwelveHoursRefreshInterval`
- ✧ `OneDayRefreshInterval`

For example, for a refresh interval of six hours, the class name is, `org.exoplatform.services.jcr.impl.core.query.lucene.spell.LuceneSpellChecker$SixHoursRefreshInterval`.

If you use `org.exoplatform.services.jcr.impl.core.query.lucene.spell.LuceneSpellChecker`, the refresh interval will be one hour.

The spell checker dictionary is stored as a lucene index under `<index-dir>/spellchecker`. If this index does not exist, a background thread will create it on start up. Similarly, the dictionary refresh is also done in a background to avoid blocking of regular queries.

See Also:

- ✧ [Section B.6, “Configuring Search”](#)

[Report a bug](#)

B.13.10.1. Spell check Usage

You can spell check a fulltext statement either with an XPath or a SQL query:

|

Example B.51. Spell check using XPath

```
// rep:spellcheck('explatform') will always evaluate to true
Query query =
qm.createQuery("/jcr:root[rep:spellcheck('explatform')]/(rep:spellcheck
())", Query.XPATH);
RowIterator rows = query.execute().getRows();
// the above query will always return the root node no matter what
string we check
Row r = rows.nextRow();
// get the result of the spell checking
Value v = r.getValue("rep:spellcheck()");
if (v == null) {
    // no suggestion returned, the spelling is correct or the spell
    checker
    // does not know how to correct it.
} else {
    String suggestion = v.getString();
}
```

Example B.52. Spell check using SQL

```
// SPELLCHECK('explatform') will always evaluate to true
Query query = qm.createQuery("SELECT rep:spellcheck() FROM nt:base
WHERE jcr:path = '/' AND SPELLCHECK('explatform')", Query.SQL);
RowIterator rows = query.execute().getRows();
// the above query will always return the root node no matter what
string we check
Row r = rows.nextRow();
// get the result of the spell checking
Value v = r.getValue("rep:spellcheck()");
if (v == null) {
    // no suggestion returned, the spelling is correct or the spell
    checker
    // does not know how to correct it.
} else {
    String suggestion = v.getString();
}
```

[Report a bug](#)

B.13.11. Similarity

JCR, version 1.12 and onwards, allows you to search nodes that are similar to an existing node.

Similarity is determined by looking up terms that are common to nodes. There are conditions that must be met for a term to be considered. This is required to limit the number of relevant terms.

For a term to be considered relevant, the term must meet the following conditions.

- » The term must be at least four characters long.
- » The term must occur at least twice in the source node.

- » The term must occur in at least five other nodes.



Note

The similarity function requires that the support Highlighting is enabled. You must have the following parameter set for the query handler in your **workspace.xml**.

```
<param name="support-highlighting" value="true"/>
```

The functions (**rep:similar()** in XPath and **similar()** in SQL) have two arguments:

relativePath

A relative path to a descendant node or a period (.) for the current node.

absoluteStringPath

A string literal that contains the path to the node for which, you are finding similar nodes.



Warning

Relative path is not supported yet.

Example B.53. Query to find similar nodes

The following query finds **nt:resource** nodes, which are similar to node by path **/parentnode/node.txt/jcr:content**.

```
//element(*, nt:resource)[rep:similar(.,  
'/parentnode/node.txt/jcr:content')]
```

[Report a bug](#)

B.14. Full Text Search And Affecting Settings

Each indexable property of a node is processed with the Lucene analyzer and stored in the Lucene index. This is called indexing of a property. It allows fulltext searching of the indexed properties.

[Report a bug](#)

B.14.1. Lucene Analyzers

The purpose of analyzers is to transform all strings stored in the index into a well-defined condition. The same analyzer(s) is/are used when searching in order to adapt the query string to the index reality.

Therefore, performing the same query using different analyzers can return different results.

This example illustrates how the same string is transformed by different analyzers.

Table B.7. "The quick brown fox jumped over the lazy dogs"

| Analyzer | Parsed |
|--|---|
| org.apache.lucene.analysis.WhitespaceAnalyzer | [The] [quick] [brown] [fox] [jumped] [over] [the] [lazy] [dogs] |
| org.apache.lucene.analysis.SimpleAnalyzer | [the] [quick] [brown] [fox] [jumped] [over] [the] [lazy] [dogs] |
| org.apache.lucene.analysis.StopAnalyzer | [quick] [brown] [fox] [jumped] [over] [lazy] [dogs] |
| org.apache.lucene.analysis.standard.StandardAnalyzer | [quick] [brown] [fox] [jumped] [over] [lazy] [dogs] |
| org.apache.lucene.analysis.snowball.SnowballAnalyzer | [quick] [brown] [fox] [jump] [over] [lazi] [dog] |
| org.apache.lucene.analysis.standard.StandardAnalyzer (configured without stop word - jcr default analyzer) | [the] [quick] [brown] [fox] [jumped] [over] [the] [lazy] [dogs] |

Table B.8. "XY&Z Corporation - xyz@example.com"

| Analyzer | Parsed |
|--|--|
| org.apache.lucene.analysis.WhitespaceAnalyzer | [XY&Z] [Corporation] [-] [xyz@example.com] |
| org.apache.lucene.analysis.SimpleAnalyzer | [xy] [z] [corporation] [xyz] [example] [com] |
| org.apache.lucene.analysis.StopAnalyzer | [xy] [z] [corporation] [xyz] [example] [com] |
| org.apache.lucene.analysis.standard.StandardAnalyzer | [xy&z] [corporation] [xyz@example] [com] |
| org.apache.lucene.analysis.snowball.SnowballAnalyzer | [xy&z] [corpor] [xyz@exampl] [com] |
| org.apache.lucene.analysis.standard.StandardAnalyzer (configured without stop word - jcr default analyzer) | [xy&z] [corporation] [xyz@example] [com] |



Note

StandardAnalyzer is the default analyzer in the portal JCR search engine. But it does not use stop words.

See Also:

- » [Section B.6, "Configuring Search"](#)
- » [Section B.11.1, "Indexer lock manager and data container configuration"](#)

[Report a bug](#)

B.14.2. Property Indexing

Different properties are indexed in different ways and this affects whether it can be searched using fulltext by property or not.

Two property types are indexed as fulltext searchable, which are **STRING** and **BINARY**.

Table B.9. Fulltext search by different properties

| Property Type | Fulltext search by all properties | Fulltext search by exact property |
|---------------|-----------------------------------|-----------------------------------|
| STRING | YES | YES |
| BINARY | YES | NO |

For example, the **jcr:data** property (which is **BINARY**) is not be found with a query structured as follows because, **BINARY** is not searchable by fulltext search by exact property.

```
SELECT * FROM nt:resource WHERE CONTAINS(jcr:data, 'some string')
```

However, the following query return some results, provided the node contains the targeted data.

```
SELECT * FROM nt:resource WHERE CONTAINS( * , 'some string')
```

[Report a bug](#)

B.14.3. Different Analyzers

This topic shows the different types of analyzers. To create examples to analyze, first you have to fill repository by nodes with mixin type `mix:title` and different values of `jcr:description` property.

```
root
├─ document1 (mix:title) jcr:description = "The quick brown fox jumped over the lazy dogs"
├─ document2 (mix:title) jcr:description = "Brown fox live in forest."
└─ document3 (mix:title) jcr:description = "Fox is a nice animal."
```

Example B.54. Usage of analyzer

The first instance uses base JCR settings, so the string, The quick brown fox jumped over the lazy dogs is transformed to the set; `{[the] [quick] [brown] [fox] [jumped] [over] [the] [lazy] [dogs]}`.

```
// make SQL query
QueryManager queryManager = workspace.getQueryManager();
String sqlStatement = "SELECT * FROM mix:title WHERE CONTAINS(jcr:description, 'the')";
// create query
Query query = queryManager.createQuery(sqlStatement, Query.SQL);
// execute query and fetch result
QueryResult result = query.execute();
```

The **NodeIterator** returns *document1*.

If the default analyzer is changed to **org.apache.lucene.analysis.StopAnalyzer**, the repository is populated again (the new Analyzer must process node properties) and the same query runs to return nothing, because stop words like *the* are excluded from parsed string set.

[Report a bug](#)

B.15. WebDAV

WebDAV protocol enables you to use external tools to communicate with hierarchical content servers using the HTTP protocol. It is possible to add and remove documents or a set of documents from a path on the server.

DeltaV is an extension of the WebDav protocol that allows managing document versioning. The *Locking* feature guarantees protection against multiple access when writing resources. The ordering support allows changing the position of the resource in the list and sort the directory to make the directory tree viewed conveniently. The full-text search makes it easy to find the necessary documents. You can search by using two languages: SQL and XPATH.

In eXo JCR, the WebDAV layer is plugged on top of your JCR implementation. This setup enables to browse a workspace using external tools regardless of operating system environments. You can use a Java WebDAV client, such as **DAVExplorer** or **Internet Explorer** using **File** → **Open as a Web Folder**.

WebDav is an extension of the REST service. To get the WebDav server ready, you must deploy the REST application. Then, you can access any workspaces of your repository by using the following URL:

<http://host:port/portal/rest/private/jcr/{RepositoryName}/{WorkspaceName}/{Path}>

When accessing the WebDAV server via <http://localhost:8080/rest/jcr/repository/production>, you can substitute [production](#) with [collaboration](#).

Enter your login credentials. These are checked using the organization service that can be implemented using a dummy InMemory module, DB module or an LDAP. The JCR user session is created with the correct JCR Credentials.



Note:

If you try the "in ECM" option, add "@ecm" to the user's password. You can modify jaas.conf by adding the **domain=ecm** option as follows:

```
exo-domain {
    org.exoplatform.services.security.jaas.BasicLoginModule
    required domain=ecm;
};
```

[Report a bug](#)

B.15.1. WebDAV Configuration

The WebDAV configuration file:

```
<component>
  <key>org.exoplatform.services.webdav.WebDavServiceImpl</key>
  <type>org.exoplatform.services.webdav.WebDavServiceImpl</type>
  <init-params>
```

```

<!-- this parameter indicates the default login and password values
      used as credentials for accessing the repository -->
<!-- value-param>
      <name>default-identity</name>
      <value>admin:admin</value>
</value-param -->

<!-- this is the value of WWW-Authenticate header -->
<value-param>
      <name>auth-header</name>
      <value>Basic realm="eXo-Platform Webdav Server 1.6.1"</value>
</value-param>

<!-- default node type which is used for the creation of collections
-->
<value-param>
      <name>def-folder-node-type</name>
      <value>nt:folder</value>
</value-param>

<!-- default node type which is used for the creation of files -->
<value-param>
      <name>def-file-node-type</name>
      <value>nt:file</value>
</value-param>

<!-- if MimeTypesResolver can't find the required mime type,
      which conforms with the file extension, and the mimeType header
is absent
      in the HTTP request header, this parameter is used
      as the default mime type-->
<value-param>
      <name>def-file-mimetype</name>
      <value>application/octet-stream</value>
</value-param>

<!-- This parameter indicates one of the three cases when you update
the content of the resource by PUT command.
      In case of "create-version", PUT command creates the new version
of the resource if this resource exists.
      In case of "replace" - if the resource exists, PUT command
updates the content of the resource and its last modification date.
      In case of "add", the PUT command tries to create the new
resource with the same name (if the parent node allows same-name
siblings).-->

<value-param>
      <name>update-policy</name>
      <value>create-version</value>
      <!--value>replace</value -->
      <!-- value>add</value -->
</value-param>

<!--
      This parameter determines how service responds to a method that

```


attempts to modify file content.

In case of "checkout-checkin" value, when a modification request is applied to a checked-in version-controlled resource, the request is automatically preceded by a checkout and followed by a checkin operation.

In case of "checkout" value, when a modification request is applied to a checked-in version-controlled resource, the request is automatically preceded by a checkout operation.

```
-->
<value-param>
  <name>auto-version</name>
  <value>checkout-checkin</value>
  <!--value>checkout</value -->
</value-param>

<!--
  This parameter is responsible for managing Cache-Control header
  value which will be returned to the client.
  You can use patterns like "text/*", "image/*" or wildcard to
  define the type of content.
-->
<value-param>
  <name>cache-control</name>
  <value>text/xml,text/html:max-age=3600;image/png,image/jpg:max-
age=1800;*/*:no-cache;</value>
</value-param>

<!--
  This parameter determines the absolute path to the folder icon
  file, which is shown
  during WebDAV view of the contents
-->
<value-param>
  <name>folder-icon-path</name>
  <value>/absolute/path/to/file</value>
</value-param>

</init-params>
</component>
```

[Report a bug](#)

B.15.2. WebDAV and JCR Actions

Table B.10. Corresponding WebDAV and JCR Actions

| WebDav | JCR |
|--------|---|
| COPY | Workspace.copy(...) |
| DELETE | Node.remove() |
| GET | Node.getProperty(...); Property.getValue() |
| HEAD | Node.getProperty(...); Property.getLength() |
| MKCOL | Node.addNode(...) |
| MOVE | Session.move(...) or Workspace.move(...) |

| WebDav | JCR |
|-----------------|--|
| PROPFIND | Session.getNode(...); Node.getNode(...); Node.getNodes(...); Node.getProperties() |
| PROPPATCH | Node.setProperty(...); Node.getProperty(...).remove() |
| PUT | Node.addNode("node","nt:file"); Node.setProperty("jcr:data", "data") |
| CHECKIN | Node.checkin() |
| CHECKOUT | Node.checkout() |
| REPORT | Node.getVersionHistory(); VersionHistory.getAllVersions(); Version.getProperties() |
| RESTORE | Node.restore(...) |
| UNCHECKOUT | Node.restore(...) |
| VERSION-CONTROL | Node.addMixin("mix:versionable") |
| LOCK | Node.lock(...) |
| UNLOCK | Node.unlock() |
| ORDERPATCH | Node.orderBefore(...) |
| SEARCH | Workspace.getQueryManager(); QueryManager.createQuery(); Query.execute() |

[Report a bug](#)

B.15.3. WebDAV Limitation on Windows

When attempting to set up a web folder through **Add a Network Location** or **Map a Network Drive** through **My Computer**, an error message stating **The folder you entered does not appear to be valid. Please choose another** or **Windows cannot access ... Check the spelling of the name. Otherwise, there might be ...** may be encountered.

These errors may appear when you are using SSL or non-SSL.

To fix the error, perform the following steps:

1. Go to Windows Registry Editor.
2. Find a key:
`\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\WebClient\Parameters\BasicAuth`
3. Change the value to 2.

[Report a bug](#)

B.15.4. WebDAV Limitation for Microsoft Office 2010

If you have Microsoft Office 2007/2010 applications installed on a client computer and the client computer is connected to a web server configured for basic authentication using a connection that does not use Secure Sockets Layer (SSL) and you try to access an MS Office file that is stored on the remote server. You might experience the following symptoms when you try to open or to download the file:

- » The Office file does not open or download.

- ✱ You do not receive a Basic authentication password prompt when you try to open or to download the file.
- ✱ You do not receive an error message when you try to open the file. The associated Office application starts. However, the selected file does not open.

These outcomes can be circumvented by enabling Basic authentication on the client machine.

Procedure B.12. Enabling Basic Authentication on the Client Computer

1. Click Start, type **regedit** in the Start Search box, and then press Enter.
2. Locate and then click the following registry subkey:
HKEY_CURRENT_USER\Software\Microsoft\Office\14.0\Common\Internet
3. On the **Edit** menu, point to **New**, and then click **DWORD Value**.
4. Type **BasicAuthLevel**, and then press **Enter**.
5. Right-click **BasicAuthLevel**, and then click **Modify**.
6. In the Value data box, type **2**, and then click **OK**.

[Report a bug](#)

B.16. FTP

The JCR-FTP Server operates as an FTP server with access to a content stored in JCR repositories in the form of **nt:file/nt:folder** nodes or their successors. The client of an executed Server can be any FTP client. The FTP server is supported by a standard configuration which can be changed as required.

Configuration Parameters

command-port:

```
<value-param>
  <name>command-port</name>
  <value>21</value>
</value-param>
```

The value of the command channel port. The value '**21**' is default.

If you have already other FTP server installed in your system, this parameter needs to be changed (to **2121**, for example) to avoid conflicts or if the port is protected.

data-min-port and data-max-port

```
<value-param>
  <name>data-min-port</name>
  <value>52000</value>
</value-param>
```

```
<value-param>
  <name>data-max-port</name>
  <value>53000</value>
</value-param>
```

These two parameters indicate the minimum and maximum values of the range of ports, used by the server. The usage of the additional data channel is required by the FTP protocol, which is used to transfer the contents of files and the listing of catalogues. This range of ports should be free from listening by other server-programs.

system

```
<value-param>
  <name>system</name>

  <value>Windows_NT</value>
  or
  <value>UNIX Type: L8</value>
</value-param>
```

Types of formats of listing of catalogues which are supported.

client-side-encoding

```
<value-param>
  <name>client-side-encoding</name>

  <value>windows-1251</value>
  or
  <value>KOI8-R</value>

</value-param>
```

This parameter specifies the coding which is used for dialogue with the client.

def-folder-node-type

```
<value-param>
  <name>def-folder-node-type</name>
  <value>nt:folder</value>
</value-param>
```

This parameter specifies the type of a node, when an FTP-folder is created.

def-file-node-type

```
<value-param>
  <name>def-file-node-type</name>
  <value>nt:file</value>
</value-param>
```

This parameter specifies the type of a node, when an FTP-file is created.

def-file-mime-type

```
<value-param>
  <name>def-file-mime-type</name>
  <value>application/zip</value>
</value-param>
```

The mime type of a created file is chosen by using its file extension. In case, a server cannot find the corresponding mime type, this value is used.

cache-folder-name

```
<value-param>
  <name>cache-folder-name</name>
  <value>../temp/ftp_cache</value>
</value-param>
```

The Path of the cache folder.

upload-speed-limit

```
<value-param>
  <name>upload-speed-limit</name>
  <value>20480</value>
</value-param>
```

Restriction of the upload speed. It is measured in bytes.

download-speed-limit

```
<value-param>
  <name>download-speed-limit</name>
  <value>20480</value>
</value-param>
```

Restriction of the download speed. It is measured in bytes.

timeout

```
<value-param>
  <name>timeout</name>
  <value>60</value>
</value-param>
```

Defines the value of a timeout.

[Report a bug](#)

B.17. Use External Backup Tool

[Report a bug](#)

B.17.1. Repository Suspending

To have the repository content consistent with the search index and value storage, the repository must be suspended. This means all working threads are suspended until a resume operation is performed and the index is flushed.

JCR provides ability to suspend repository via JMX.

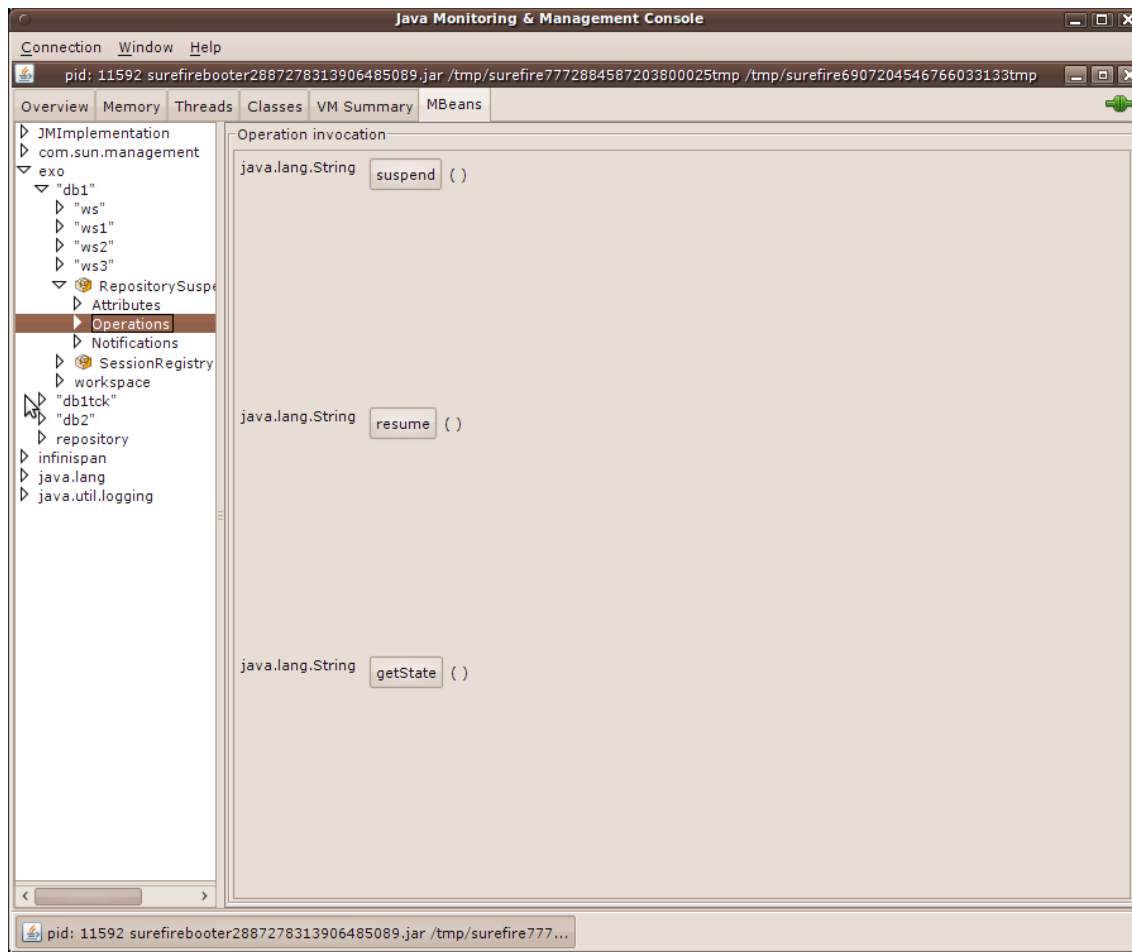


Figure B.8. Repository Suspend Controller

To suspend repository you need to invoke the **suspend ()** operation. The returned result is *"suspended"* if everything passes successfully.

An *"undefined"* result means not all components are successfully suspended. Review the stack traces in the console to identify the cause.

[Report a bug](#)

B.17.2. Backup Considerations

You can backup your content manually or by using third part software. You should back up:

- ✱ Database.
- ✱ Lucene index.
- ✱ Value storage (if configured).

[Report a bug](#)

B.17.3. Repository Resuming

Once a backup is done you need to invoke the **resume()** operation to switch the repository back to on-line. The returned result will be *"on-line"*.

[Report a bug](#)

B.18. eXo JCR statistics

[Report a bug](#)

B.18.1. Statistics on the Database Access Layer

The statistics on database access layer help you to identify, what is slow in this layer and help diagnose and fix the problem.

You can get statistics on the time spent into the database access layer using the environment variables such as,

org.exoplatform.services.jcr.impl.storage.jdbc.optimisation.CQJDBCWorkspaceDataContainer or
org.exoplatform.services.jcr.impl.storage.jdbc.JDBCWorkspaceDataContainer
 as **WorkspaceDataContainer**.

[Report a bug](#)

B.18.1.1. Database Access Layer Data

The database access layer is represented by the methods of the interface **org.exoplatform.services.jcr.storage.WorkspaceStorageConnection**, so for all the methods defined in this interface have the following data:

- ✧ The minimum time spent into the method.
- ✧ The maximum time spent into the method.
- ✧ The average time spent into the method.
- ✧ The total amount of time spent into the method.
- ✧ The total amount of time the method has been called.

These figures are also available globally for all the methods which gives the global behavior of this layer.

[Report a bug](#)

B.18.1.2. Enabling Statistics for Database Access Layer

To enable the statistics, set the JVM parameter called **JDBCWorkspaceDataContainer.statistics.enabled** to *true*. The corresponding CSV file is **StatisticsJDBCStorageConnection- $\{creation-timestamp\}$.csv**.

The format of each column header is $\{method-alias\}$ - $\{metric-alias\}$. The metric alias are described in the statistics manager section.

The name of the category of statistics corresponding to these statistics is **JDBCStorageConnection**, this name is mostly needed to access to the statistics through JMX.

[Report a bug](#)

B.18.1.3. Database Access Layer Methods and their alias

Table B.11. Method Alias

| | |
|----------------------------------|---|
| global | This is the alias for all the methods. |
| getItemDataById | This is the alias for the method <i>getItemData(String identifier)</i> . |
| getItemDataByNodeDataNQPathEntry | This is the alias for the method <i>getItemData(NodeData parentData, QPathEntry name)</i> . |
| getChildNodesData | This is the alias for the method <i>getChildNodesData(NodeData parent)</i> . |
| getChildNodesCount | This is the alias for the method <i>getChildNodesCount(NodeData parent)</i> . |
| getChildPropertiesData | This is the alias for the method <i>getChildPropertiesData(NodeData parent)</i> . |
| listChildPropertiesData | This is the alias for the method <i>listChildPropertiesData(NodeData parent)</i> . |
| getReferencesData | This is the alias for the method <i>getReferencesData(String nodeIdentifier)</i> . |
| commit | This is the alias for the method <i>commit()</i> . |
| addNodeData | This is the alias for the method <i>add(NodeData data)</i> . |
| addPropertyData | This is the alias for the method <i>add(PropertyData data)</i> . |
| updateNodeData | This is the alias for the method <i>update(NodeData data)</i> . |
| updatePropertyData | This is the alias for the method <i>update(PropertyData data)</i> . |
| deleteNodeData | This is the alias for the method <i>delete(NodeData data)</i> . |
| deletePropertyData | This is the alias for the method <i>delete(PropertyData data)</i> . |
| renameNodeData | This is the alias for the method <i>rename(NodeData data)</i> . |
| rollback | This is the alias for the method <i>rollback()</i> . |
| isOpened | This is the alias for the method <i>isOpened()</i> . |
| close | This is the alias for the method <i>close()</i> . |

[Report a bug](#)

B.18.2. Statistics on the JCR API Accesses

To understand the usage of eXo JCR, you have register all the JCR API access requests and create a real life test scenario based on pure JCR calls and tune the JCR to better fit operational requirements.

The Load-time Weaving proposed by AspectJ allows you to specify which part of eXo JCR to monitor without applying any changes to existing code. To enable this feature, you have to add the following jar files in your classpath.

- » *exo.jcr.component.statistics-X.Y.Z.jar* corresponding to your eXo JCR version that you get from the JBoss maven repository
<https://repository.jboss.org/nexus/content/groups/public/org/exoplatform/jcr/exo.jcr.component.statistics/>.
- » **aspectjrt-1.6.8.jar**, which can be obtained from
<http://repo2.maven.org/maven2/org/aspectj/aspectjrt>.
- » The **aspectjweaver-1.6.8.jar** is also required. This file is available from the main maven repository located at <http://repo2.maven.org/maven2/org/aspectj/aspectjweaver>.

[Report a bug](#)

B.18.2.1. Enabling Statistics on the JCR API Accesses

Add the JVM parameter **-javaagent:\${path to}/aspectjweaver-1.6.8.jar** to the command line. See <http://www.eclipse.org/aspectj/doc/released/devguide/tw-configuration.html> for more information.

By default, the configuration collects statistics on all methods of the internal interfaces **org.exoplatform.services.jcr.core.ExtendedSession** and **org.exoplatform.services.jcr.core.ExtendedNode**, and the JCR API interface **javax.jcr.Property**.

To add or remove monitored interfaces, two configuration files bundled into the jar **exo.jcr.component.statistics-X.Y.Z.jar** must be changed. The files are **conf/configuration.xml** and **META-INF/aop.xml**.

Example B.55. Configuration.xml

The file content below is the content of **conf/configuration.xml** that must be modified to add or remove the fully qualified name of the interfaces to monitor, into the list of parameter values of the init param called **targetInterfaces**.

```
<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd
http://www.exoplatform.org/xml/ns/kernel_1_2.xsd"
xmlns="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd">

  <component>

    <type>org.exoplatform.services.jcr.statistics.JCRAPIAspectConfig</type>
    <init-params>
      <values-param>
        <name>targetInterfaces</name>
        <value>org.exoplatform.services.jcr.core.ExtendedSession</value>
        <value>org.exoplatform.services.jcr.core.ExtendedNode</value>
        <value>javax.jcr.Property</value>
      </values-param>
    </init-params>
  </component>
</configuration>
```

Example B.56. aop.xml

The file content below is the content of **META-INF/aop.xml** that must be modified to add or remove the fully qualified name of the interfaces to monitor, into the expression filter of the pointcut called **JCRAPIPointcut**.

By default only JCR API calls from the **exoplatform** packages are taken into account. This filter can be modified to add other package names.

```
<aspectj>
  <aspects>
    <concrete-aspect
      name="org.exoplatform.services.jcr.statistics.JCRAPIAspectImpl"
      extends="org.exoplatform.services.jcr.statistics.JCRAPIAspect">
      <pointcut name="JCRAPIPointcut"
        expression="
(target(org.exoplatform.services.jcr.core.ExtendedSession) ||
target(org.exoplatform.services.jcr.core.ExtendedNode) ||
target(javax.jcr.Property)) && call(public * *(..))" />
    </concrete-aspect>
  </aspects>
  <weaver options="-XnoInline">
    <include within="org.exoplatform.*" />
  </weaver>
</aspectj>
```



Warning

This feature will affect the performance of eXo JCR. It must be used with caution.

[Report a bug](#)

B.18.2.2. CSV files used in Statistics

The corresponding CSV files are of type **Statistics\${interface-name}-\${creation-timestamp}.csv**

The format of each column header is *\$(method-alias)-\$(metric-alias)*. The method alias is of type *\$(method-name)(semicolon-delimited-list-of-parameter-types-to-be-compatible-with-the-CSV-format)*.

The name of the category of statistics corresponding to these statistics is the name of the monitored interface, for example, **ExtendedSession** for **org.exoplatform.services.jcr.core.ExtendedSession**. This name is required to access statistics through JMX.

[Report a bug](#)

B.18.3. Statistics Manager

The statistics manager manages all the statistics provided by eXo JCR. It is responsible for printing the data to the CSV files and exposing the statistics through JMX or Rest interface.

The statistics manager creates all the CSV files for each category of statistics. The format of the CSV files is *Statistics\${category-name}-\${creation-timestamp}.csv*.

The CSV files are created in the user directory or the temporary directory. The format of those files is **CSV** (Comma-Separated Values). One new line is added regularly, every 5 seconds by default and one last line is added at JVM exit. Each line is composed of the 5 figures described below for each method and globally for all the methods.

Table B.12. Metric Alias

| | |
|-------|---|
| Min | The minimum time spent into the method expressed in milliseconds. |
| Max | The maximum time spent into the method expressed in milliseconds. |
| Total | The total amount of time spent into the method expressed in milliseconds. |
| Avg | The average time spent into the method expressed in milliseconds. |
| Times | The total amount of times the method has been called. |



Note

You can disable the persistence of the statistics by setting the JVM parameter called ***JCRStatisticsManager.persistence.enabled*** to **false**. It is set to **true** by default.

You can define the time interval between each record by setting the JVM parameter called ***JCRStatisticsManager.persistence.timeout*** to your expected value in milliseconds. It is set to **5000** by default.

[Report a bug](#)

B.18.3.1. Accessing Statistics using JMX

You can access to the statistics through JMX. The available methods are:

Table B.13. JMX Methods

| | |
|--------|---|
| getMin | Give the minimum time spent into the method corresponding to the given category name and statistics name. The expected arguments are the name of the category of statistics (JDBCStorageConnection for example) and the name of the expected method or global for the global value. |
| getMax | Give the maximum time spent into the method corresponding to the given category name and statistics name. The expected arguments are the name of the category of statistics and the name of the expected method or global for the global value. |

| | |
|----------|--|
| getTotal | Give the total amount of time spent into the method corresponding to the given category name and statistics name. The expected arguments are the name of the category of statistics and the name of the expected method or global for the global value. |
| getAvg | Give the average time spent into the method corresponding to the given category name and statistics name. The expected arguments are the name of the category of statistics and the name of the expected method or global for the global value. |
| getTimes | Give the total amount of times the method has been called corresponding to the given category name and statistics name. The expected arguments are the name of the category of statistics (e.g. JDBCStorageConnection) and the name of the expected method or global for the global value. |
| reset | Reset the statistics for the given category name and statistics name. The expected arguments are the name of the category of statistics and the name of the expected method or global for the global value. |
| resetAll | Reset all the statistics for the given category name. The expected argument is the name of the category of statistics (e.g. JDBCStorageConnection). |

The full name of the related MBean is **xo:service=statistic, view=jcr**.

[Report a bug](#)

B.19. Checking Repository Integrity and Consistency

[Report a bug](#)

B.19.1. JMX-based consistency tool

It is important to check the integrity and consistency of system regularly, especially if there is no, or stale, backups. The portal JCR implementation offers an innovative JMX-based complex checking tool.

During an inspection, the tool checks every major JCR component, such as persistent data layer and the index. The persistent layer includes JDBC Data Container and Value-Storages if they are configured.

The database is verified using the set of complex specialized domain-specific queries. The Value Storage tool checks the existence of, and access to, each file.

Access to the check tool is exposed via the JMX interface, with the following operations available:

Table B.14. Available methods

| | |
|---|---|
| checkRepositoryDataConsistency() | Inspect full repository data (db, value storage and search index) |
|---|---|

| | |
|---|---------------------------|
| <code>checkRepositoryDataBaseConsistency()</code> | Inspect only DB |
| <code>checkRepositoryValueStorageConsistency()</code> | Inspect only ValueStorage |
| <code>checkRepositorySearchIndexConsistency()</code> | Inspect only SearchIndex |

All inspection activities and corrupted data details are stored in a file in the **app** directory and named as per the following convention: **report-*<repository name>-dd-MMM-yy-HH-mm.txt***.

The path to the file will be returned in result message also at the end of the inspection.



Note

There are three types of inconsistency (Warning, Error and Index) and two of them are critical (Errors and Index):

- ✦ Index faults are marked as "Reindex" and can be fixed by re-indexing the workspace.
- ✦ Errors can only be fixed manually.
- ✦ Warnings can be a normal situation in some cases and usually production system will still remain fully functional.

[Report a bug](#)

B.20. JCR Performance Tuning Guide

This section will show you various ways of improving JCR performance.

It is intended for Administrators and others who want to use the JCR features more efficiently.

[Report a bug](#)

B.20.1. Cluster configuration

[Table B.15, "EC2 network: 1Gbit"](#) contains details about the configuration of the cluster used in benchmark testing.



Note

NFS and statistics (cacti snmp) server were located on one physical server.

Table B.15. EC2 network: 1Gbit

| Servers hardware | Specification |
|------------------|---|
| RAM | 7.5 GB |
| Processors | 4 EC2 Compute Units (2 virtual cores with 2 EC2 Compute Units each) |
| Storage | 850 GB (2×420 GB plus 10 GB root partition) |
| Architecture | 64-bit |

| Servers hardware | Specification |
|------------------|-----------------|
| I/O Performance | High |
| API name | m1.large |

The configuration used to achieve this benchmark is as follows: **JAVA_OPTS:** -
Dprogram.name=run.sh -server -Xms4g -Xmx4g -XX:MaxPermSize=512m -
Dorg.jboss.resolver.warning=true -Dsun.rmi.dgc.client.gcInterval=3600000
-Dsun.rmi.dgc.server.gcInterval=3600000 -XX:+UseParallelGC -
Djava.net.preferIPv4Stack=true

[Report a bug](#)

B.20.2. JCR Clustered Performance

Benchmark test using WebDAV (Complex read/write load test (benchmark)) with 20K same file. To obtain per-operation results we have used custom output from the test case threads to CSV file.

Read operation:

Warm-up iterations: 100

Run iterations: 2000

Background writing threads: 25

Reading threads: 225

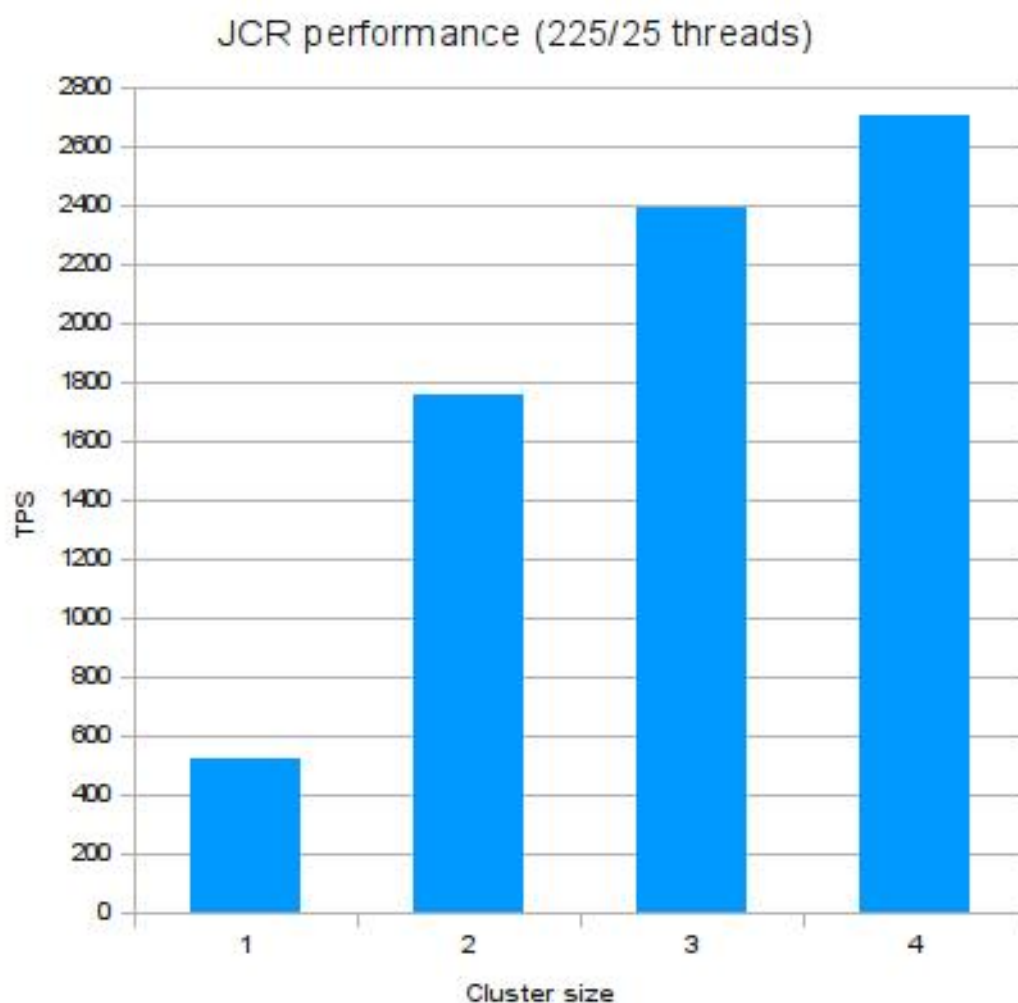


Figure B.9. EC2 Performance Results - 225/25 threads

Table B.16. EC2 Performance Results - 225/25 Bar Graph Details

| Nodes count | TPS | Responses >2s | Responses >4s |
|-------------|------|---------------|---------------|
| 1 | 523 | 6.87% | 1.27% |
| 2 | 1754 | 0.64% | 0.08% |
| 3 | 2388 | 0.49% | 0.09% |
| 4 | 2706 | 0.46% | 0.1% |

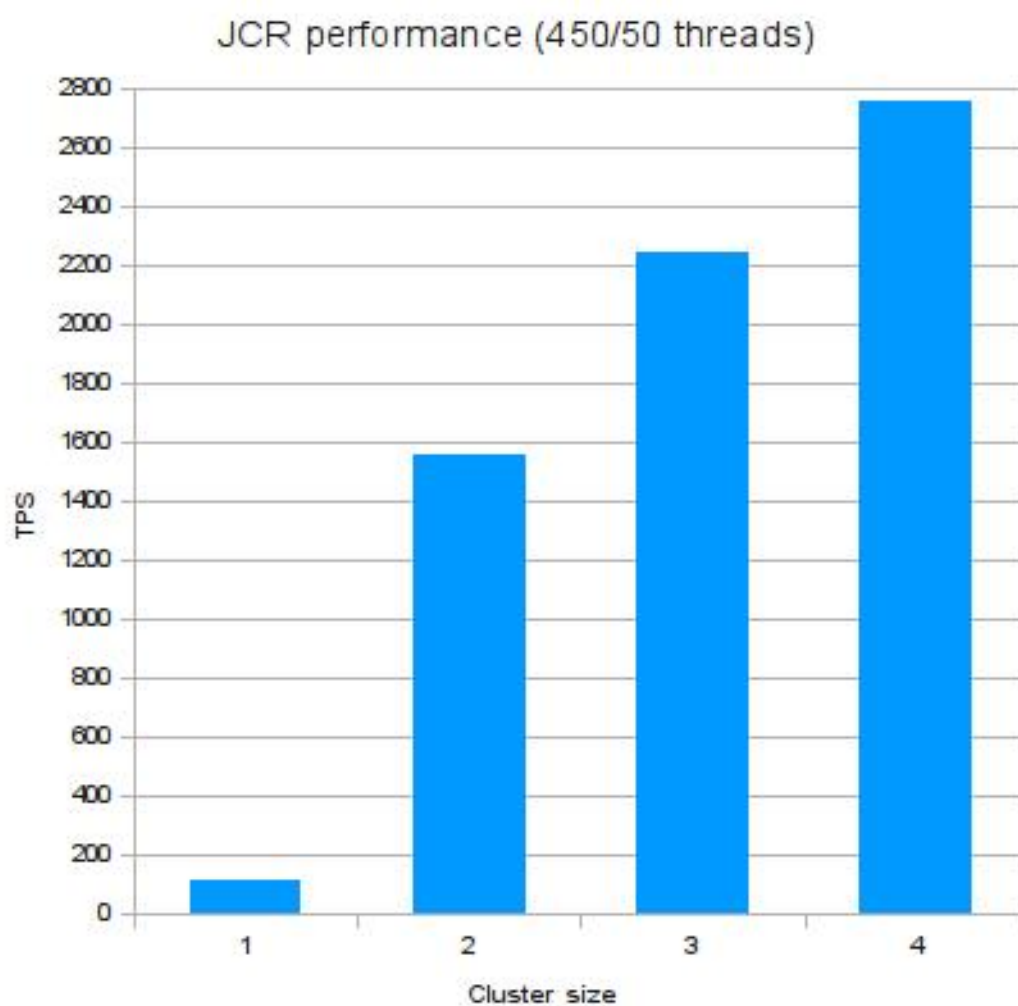
Read operation with more threads:

Warm-up iterations: 100

Run iterations: 2000

Background writing threads: 50

Reading threads: 450

**Figure B.10. EC2 Performance Results - 450/50 threads****Table B.17. EC2 Performance Results - 450/50 Bar Graph Details**

| Nodes count | tps | Responses >2s | Responses >4s |
|-------------|------|---------------|---------------|
| 1 | 116 | ? | ? |
| 2 | 1558 | 6.1% | 0.6% |
| 3 | 2242 | 3.1% | 0.38% |

| Nodes count | tps | Responses >2s | Responses >4s |
|-------------|------|---------------|---------------|
| 4 | 2756 | 2.2% | 0.41% |

[Report a bug](#)

B.20.3. JBoss Enterprise Application Platform 6 Tuning

You can use ***maxThreads*** parameter to increase maximum amount of threads that can be launched in AS instance. This can improve performance if you need a high level of concurrency. also you can use **-XX: +UseParallelGC** java directory to use parallel garbage collector.



Note

Beware of setting ***maxThreads*** too big, this can cause **OutOfMemoryError**. We've got it with ***maxThreads=1250*** on such machine:

- 7.5 GB memory
- 4 EC2 Compute Units (2 virtual cores with 2 EC2 Compute Units each)
- 850 GB instance storage (2×420 GB plus 10 GB root partition)
- 64-bit platform
- I/O Performance: High
- API name: m1.large
- java -Xmx 4g

[Report a bug](#)

B.20.4. JCR Cache Tuning

Cache size

JCR-cluster implementation is built using JBoss Cache as distributed, replicated cache. But there is one particularity related to remove action in it. Speed of this operation depends on the actual size of cache. As many nodes are currently in cache as much time is needed to remove one particular node (subtree) from it.

Eviction

Manipulations with eviction ***wakeUpInterval*** value does not affect on performance. Performance results with values from 500 up to 3000 are approximately equal.

Transaction Timeout

Using short timeout for long transactions such as Export/Import, removing huge subtree defined timeout may cause **TransactionTimeoutException**.

[Report a bug](#)

B.20.5. Clustering Tuning

For performance it is better to have a load-balancer, DB server and shared NFS on different computers. If in some reasons you see that one node gets more load than others you can decrease this load using load value in load balancer.

JGroups configuration

It's recommended to use "multiplexer stack" feature present in JGroups. It is set by default in eXo JCR and offers higher performance in cluster, using less network connections also. If there are two or more clusters in your network, please check that they use different ports and different cluster names.

Write performance in cluster

Exo JCR implementation uses Lucene indexing engine to provide search capabilities. But Lucene brings some limitations for write operations: it can perform indexing only in one thread. That is why write performance in cluster is not higher than in singleton environment. Data is indexed on coordinator node, so increasing write-load on cluster may lead to `ReplicationTimeout` exception. It occurs because writing threads queue in the indexer and under high load timeout for replication to coordinator will be exceeded.

Taking in consideration this fact, it is recommended to exceed **`replTimeout`** value in cache configurations in case of high write-load.

Replication timeout

Some operations may take too much time. So if you get **`ReplicationTimeoutException`** try increasing replication timeout:

```
<clustering mode="replication" clusterName="${jboss-cache-cluster-name}">
    ...
    <sync replTimeout="60000" />
</clustering>
```

value is set in milliseconds.

[Report a bug](#)

B.20.6. Declaring the Datasources in the Application Server

To declare the datasources using a JBoss application server, deploy a **`ds`** file (**`XXX-ds.xml`**) into the *deploy* directory of the appropriate server profile (**`/server/PROFILE/deploy`**).

Example B.57. Declaring datasources

The file `ds.xml` configures all datasources which the portal requires. There are four data sources *`jdbcjcr_portal`*, *`jdbcjcr_portal-sample`*, *`jdbcidm_portal`* and *`jdbcidm_sample-portal`* that are configured.

```
<?xml version="1.0" encoding="UTF-8"?>
<datasources>
  <no-tx-datasource>
    <jndi-name>jdbcjcr_portal</jndi-name>
    <connection-
url>jdbc:hsqldb:${jboss.server.data.dir}/data/jdbcjcr_portal</connection-
url>
    <driver-class>org.hsqldb.jdbcDriver</driver-class>
    <user-name>sa</user-name>
    <password></password>
  </no-tx-datasource>

  <no-tx-datasource>
    <jndi-name>jdbcjcr_sample-portal</jndi-name>
    <connection-
```

```

url>jdbc:hsqldb:${jboss.server.data.dir}/data/jdbcjcr_sample-
portal</connection-url>
  <driver-class>org.hsqldb.jdbcDriver</driver-class>
  <user-name>sa</user-name>
  <password></password>
</no-tx-datasource>

<no-tx-datasource>
  <jndi-name>jdbcidm_portal</jndi-name>
  <connection-
url>jdbc:hsqldb:${jboss.server.data.dir}/data/jdbcidm_portal</connectio
n-url>
    <driver-class>org.hsqldb.jdbcDriver</driver-class>
    <user-name>sa</user-name>
    <password></password>
  </no-tx-datasource>

  <no-tx-datasource>
    <jndi-name>jdbcidm_sample-portal</jndi-name>
    <connection-
url>jdbc:hsqldb:${jboss.server.data.dir}/data/jdbcidm_sample-
portal</connection-url>
      <driver-class>org.hsqldb.jdbcDriver</driver-class>
      <user-name>sa</user-name>
      <password></password>
    </no-tx-datasource>
  </datasources>

```

The properties can be set for datasource can be found at

http://docs.jboss.org/jbossas/docs/Server_Configuration_Guide/4/html/Connectors_on_JBoss-Configuring_JDBC_DataSources.html#Configuring_JDBC_DataSources-The_non_transactional_DataSource_configuration_schema

[Report a bug](#)

B.20.6.1. Binding Datasources



Warning

Do not let the portal explicitly bind datasources.

To prevent the portal from binding datasources explicitly perform the following steps:

1. Edit the **`$JPP_HOME/standalone/configuration/gatein/configuration.properties`** and comment out the following rows in the JCR section:

```

#gatein.jcr.datasource.driver=org.hsqldb.jdbcDriver
#gatein.jcr.datasource.url=jdbc:hsqldb:file:${gatein.db.data.dir}/d
ata/jdbcjcr_${name}
#gatein.jcr.datasource.username=sa
#gatein.jcr.datasource.password=

```

2. Comment out the following lines in the IDM section:

```
#gatein.idm.datasource.driver=org.hsqldb.jdbcDriver
#gatein.idm.datasource.url=jdbc:hsqldb:file:${gatein.db.data.dir}/data/jdbcidm_${name}
#gatein.idm.datasource.username=sa
#gatein.idm.datasource.password=
```

3. Open the **jcr-configuration.xml** and **idm-configuration.xml** files and comment out references to the plug-in **InitialContextInitializer**.

```
<!-- Commented because, Datasources are declared and bound by AS,
not in eXo -->
<!--
<external-component-plugins>
    [...]
</external-component-plugins>
-->
```

[Report a bug](#)

Quickstarts

C.1. Quickstarts

Quickstarts provide small, specific, working examples that can be used as a reference for your own project.

The Quickstarts can be obtained using the methods described in this appendix.

[Report a bug](#)

C.2. Quickstarts Downloads Page

A zip archive containing all Quickstarts can be downloaded from the Red Hat JBoss Portal Downloads page, located at <https://access.redhat.com/jbossnetwork/restricted/listSoftware.html?downloadType=distributions&product=jbportal&version=6.1.0>.



Note

For information on using the WOLF repository, see the **readme** file packaged with the quickstarts.

[Report a bug](#)

C.3. JBoss Developer Studio or Eclipse with JBoss Tools

Procedure C.1. Launching JBoss Tools with Eclipse or JBoss Developer Studio

1. In the Workbench, go to **File+New+Example...+JBoss Tools+Project Examples**.
2. In the New Project Example dialog, select the Site **JBoss Community JBoss Portal Platform Examples** from the drop down menu.
3. Select one of the available Quickstarts, for example **Simplest Hello World Portlet**.
4. Click **Finish** and the example project will be imported into your workspace.



Note

For more information on how to use and deploy quickstarts, see the **readme** file packaged with the quickstart.

[Report a bug](#)

Revision History

| Revision | Date | Author |
|--------------------------------|-----------------|-----------------|
| 6.2.0-0 | Mon May 11 2015 | Aakanksha Singh |
| Prepared the guide for 6.2 GA. | | |